**MATHEUS BARROS MANINI**

# CoEP: Uma Camada para Comunicação Segura para Dispositivos Computacionais de Grão Fino em Redes de Sensores sem Fio

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Ciências.

São Paulo
2019

**MATHEUS BARROS MANINI**

# CoEP: Uma Camada para Comunicação Segura para Dispositivos Computacionais de Grão Fino em Redes de Sensores sem Fio

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Ciências.

Área de concentração:
Sistemas Eletrônicos

Orientador:
Prof. Dr. Marcelo Knorich Zuffo

São Paulo
2019

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

Manini, Matheus Barros
CoEP: Uma Camada para Comunicação Segura para Dispositivos Computacionais de Grão Fino em Redes de Sensores sem Fio / M. B. Manini - versão corr. -- São Paulo, 2019.
151 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.INTERNET DAS COISAS 2.SEGURANÇA DE REDES 3.SENSORIAMENTO REMOTO I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

À Darlene, minha mãe

## AGRADECIMENTOS

Agradeço à equipe SwarmOS do CITI-USP, em especial meu orientador, que me guiou e auxiliou no desenvolvimento da ciência e tecnologia e que também proporcionaram todas condições que puderam para eu conciliar da vida com academia. Também agradeço à minha mãe Darlene e ao meu pai Alfredo, aos irmãos Erika e Renan, à minha parceira na vida Gabriela, e à família pelo apoio pessoal. Agradeço meus amigos, colegas de trabalho e de faculdade por opiniões e bons momentos. Por fim, agradeço à omni-electronica, que disponibilizou comentários e oportunidades para avaliação dessa dissertação em aplicações reais. Finalmente agradeçemos a CAPES Programa Pró-Defesa Processo: 23038.009307/2013-7.

# RESUMO

O trabalho desenvolvido nessa dissertação é a concepção de uma camada de segurança nomeada de CoEP (*Constrained Extensible Protocol*), nome similar à camada de aplicação CoAP (*Constrained Aplication Protocol*), por razões de sua compatibilidade em utilização, como apresentado em detalhes no texto. Essa camada, por sua vez, é criada por conta da identificação de lacuna de prover tecnologia de segurança adequada e compacta em dispositivos restritos; isto é, dispositivos com baixa capacidade de processamento, comunicação e armazenamento, mas alto volume de dipositivos em rede que, como justificado posteriormente, chamamos de dispositivo de grão fino. Além disso, esses volumes, que são de dezenas a centenas de dispositivos, dão-se em sistemas conhecidos como redes de sensores sem fio, onde processamento paralelo e distribuído ocorre de forma autônoma e de forma a criar-se uma rede de comunicação. Enquanto diversas tecnologias habilitadoras permitiram que redes de sensores sem fio se tornassem cada vez mais possíveis tecnicamente e economicamente - devido ao avanço da tecnologia em áreas como baterias, supercapacitores, circuitos de consumo ultra-baixo, sensores mais econômicos e com novas técnicas de medição, e outras -, o mesmo tem sido feito em questões de software e implementação de protocolos e padrões para que se utilize todo esse avanço de *hardware* de forma inteligente e eficiente. Isto foi feito através de reestruturação e adaptação de protocolos para esses dispositivos, porém percebe-se que foram feitas poucas contribuições para implementar-se segurança de forma eficiente; como identificado na dissertação, esses dispositivos têm utilizado versões parcialmente implementadas do protocolo DTLS, um protocolo extenso e que foi desenvolvido para dispositivos de recursos abundantes, isto é, que possuem recursos de sobra para a tecnologia. O CoEP é, nesse contexto, uma camada de segurança que substituiria o DTLS em dispositivos de grão fino para realizar tarefas de segurança para protocolos de aplicação de forma inteligente e eficiente. Além do CoEP realizar sua tarefa de segurança, a camada também foi arquitetada para utilizar recursos de forma eficiente e prover segurança como um serviço para protocolos, retirando necessidades de implementação de segurança individualmente em cada protocolo e demais trabalhos de desenvolvimento. Como resultado, a camada foi implementada em sistema restrito e pode reduzir consideravelmente a utilização de seus recursos, bem como sua utilização obriga todos dispositivos conectados nessa mesma rede a utilizem segurança, o que é essencial para redes de sensores sem fio, que tem o potencial de ter o mesmo impacto na sociedade que a própria Internet teve, como detalhado no texto.

**Palavras-chave:** Segurança, Protocolo de aplicação, redes de sensores sem fio, internet das coisas.

## ABSTRACT

The developed work in this dissertation is the design of a security layer named Constrained Extensible Protocol (CoEP), similar name to the Constrained Application Protocol (CoAP) application layer, for reasons of its compatibility in use, as presented in detail in the text. This layer, on the other hand, is created because of the identification of the gap to provide adequate and lean security technology in restricted devices; that is, with low processing, communication and storage capacity, but with high volume which, as justified later, we call fine grain device. In addition, these device volumes are reachable in systems known as wireless sensor networks, where parallel and distributed processing occurs autonomously and in a way to create a communication network. While several enabling technologies have made wireless sensor networks to become increasingly possible technically and economically - due to the advancement of technology, areas such as batteries, supercapacitors, ultra-low consumption circuits, more economical sensors and new measurement techniques, and more -, the same has been done in software, protocols and standards implementation so that all this advance of hardware can be used intelligently and efficiently. This was done through the restructuring and adaptation of protocols for these devices, but it is noticed that few contributions were made to implement security efficiently; as identified in the dissertation, these devices have used partially implemented versions of the DTLS protocol, an extensive protocol that was developed for relatively unlimited resource devices. CoEP is, in this context, a security layer that would replace DTLS in fine grain devices to perform security tasks for application protocols in an intelligent and efficient manner. In addition to CoEP performing its security task, the layer was also architected to efficiently utilize resources and provide security as a service for protocols, removing individual security implementation needs in each protocol and other development work. As a result, the layer has been deployed on a lean system and can greatly reduce the utilization of its resources, as well as its use obliges all connected devices to use security, which is essential for wireless sensor networks, which has the potential to have the same impact on society that the Internet itself had, as detailed in the text.



**Keywords:** Security, Application layer protocol, wireless sensor networks, internet of things.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIAÇÕES

| | |
|---|---|
| 6LoWPAN | *IPv6 over Low-Power Wireless Personal Area Networks* |
| ADC | *Analog-to-digital converter* |
| AES | *Advanced Encryption Standard* |
| API | *Application Programming Interface* |
| BLE | *Bluetooth Low Energy* |
| BSD | *Berkeley Software Distribution* |
| CoAP | *Constrained Application Protocol* |
| DDoS | *Distributed Denial-Of-Service* |
| DTLS | *Datagram Transport Layer Security* |
| ECC | *Elliptic-Curve Cryptography* |
| ECDH | *Elliptic-curve Diffie–Hellman* |
| FFD | *Full-Function Device* |
| FLOPS | *Floating Point Operations Per Second* |
| HAL | *Hardware Abstraction Layer* |
| I2C | *Inter-Integrated Circuit* |
| I2S | *Inter-IC Sound* |
| IEEE | *Institute of Electrical and Electronics Engineers* |
| M2M | *Machine to Machine* |
| MCU | *Microcontroller* / Microcontrolador |
| MQTT | *Message Queue Telemetry Transport* |
| MTU | *Maximum Transmission Unit* |
| NIST | *National Institute of Standards and Technology* |
| PAT | *Port Address Translation* |

| | |
|---|---|
| PSK | *Pre-shared key* |
| POSIX | *Portable Operating System Interface* |
| QoS | *Quality of Service* |
| RAM | *Random-access memory* |
| RDC | *Radio Duty Cycle* |
| RFD | *Reduced-Function Device* |
| ROM | *Read-only memory* |
| RTC | *Real-Time Clock* |
| SHA-2 | *Secure Hash Algorithm 2* |
| SO | Sistema Operacional |
| SoC | *System on a Chip* |
| SSI | *Synchronous Serial Interface* |
| TLS | *Transport Layer Security* |
| TRNG | *True Random Number Generator* |
| UART | *Universal asynchronous receiver-transmitter* |
| USP | Universidade de São Paulo |
| WPAN | *Wireless Personal Area Network* |
| WSN | *Wireless Sensor Networks* |
| XMPP | *Extensible Messaging and Presence Protocol* |

## LISTA DE SÍMBOLOS

| | |
|---|---|
| $\mu$ | micro |

**SUMÁRIO**

# 1 CONTEXTUALIZAÇÃO

O tema interdisciplinar de Redes de sensores sem fio abrange a utilização de inúmeras tecnologias de engenharia de computação, eletrônica, materiais e muitas outras; por isso, seu desenvolvimento é contínuo e já se prolonga por anos. Porém, apenas nos últimos anos, a evolução da tecnologia passou a permitir novas possibilidades nessas redes. Com meios mais eficientes de se extrair energia do ambiente; baterias melhores e menores; chips de consumo ultrabaixo; programação e comunicação especializada; sensores de menor tamanho e consumo; e com novas tecnologias em geral, paradigmas estão sendo quebrados e aplicações complexas no passado já são realidade.

Um exemplo é a implementação de soluções de redes de sensores sem fio sem ponto de energia ou bateria (PISCARI, 2018), essa solução que há poucos anos seria impraticável, principalmente em termos de viabilidade financeira, hoje é viável e apresenta vantagens e protótipos funcionais. Esse progresso culmina de avanços em diversas áreas da tecnologia em dois tipos de esforços, o primeiro é a criação de nova tecnologia, o segundo é a otimização e inovações nas tecnologias existentes. Em criação de nova tecnologia, pode-se destacar:

1. Processos de fabricação mais precisos e melhor caracterização de transistores para trabalho em consumo ultrabaixo (MARKOVIC et al., 2010), região de trabalho que é próxima da sua tensão de *threshold* (CHANDRAKASAN; SHENG; BRODERSEN, 1992), ou ainda em melhorias de velocidade e tipo de operação de transistores (SASAKI et al., 2014);

2. Melhorias em baterias e novas tecnologias como baterias de nano fios de Silício (CHAN et al., 2008) e supercapacitores de baixo custo;

3. Avanços em tecnologias de materiais que permitem sensores menores, mais baratos e muito mais econômicos em termos de consumo energético;

4. Integração de diversos sistemas em um único Chip - do inglês *System on a Chip* (SoC);

5. Muitas outras inovações que têm reduzido custos e restrições de recursos.

Mas os avanços não podem depender apenas de novas tecnologias, e, por isso, também dependem de melhorias das tecnologias existentes. Uma área da tecnologia que permitiu grandes avanços foi a utilização eficiente de recursos para:

- **operação**, quando o sistema está tratando de processamento e atividades internas; existem sistemas operacionais embarcados de baixo consumo, orientados a eventos (isto é, estão inativos maior parte do tempo), criptografia em hardware ou algoritmos criptográficos velozes e compactos.

- **comunicação**, quando o sistema está trocando informações entre seus nós; como na disponibilização de criptografia com chaves de tamanho menores com mesma ou maior força, como é o caso de curvas elípticas (MILLER, 1985) em relação a ao padrão usado até o momento que é RSA (JONSSON et al., 2016); e trabalhos realizados na própria comunicação, como criação de padrões econômicos como 6LoWPAN (THUBERT; HUI, 2011), *Bluetooth Low Energy* (GOMEZ; OLLER; PARADELLS, 2012), todos em camada 802.15.4 (GROUP et al., 2011) e outros em diversas camadas distintas do modelo OSI (ZIMMERMANN, 1980).

Apesar das melhorias em comunicação destacadas e ainda outras que existem, ainda há lacunas em tecnologias de comunicação por não existir um padrão de camada de segurança para redes de sensores sem fio de baixo consumo e, por isso, as essas redes de sensores acabam utilizando a mesma solução que é implementada em sistemas computacionais comuns, isto é, aqueles sistemas de recursos que podem ser considerados ilimitados. O mesmo ocorre com protocolos de aplicação que não foram desenvolvidos para funcionar especificamente em redes que têm MTU de 127 *bytes*, como o caso do CoAP (Z. HARTKE K., 2014).

É por isso que se verifica a necessidade de implementar uma camada de segurança para protocolos de aplicação simples de se utilizar, com as funcionalidades que são necessárias especificamente para esse tipo de rede de sensores sem fio; e também a necessidade de propor esquema de criptografia específico para baixo consumo. A comunicação é responsável por cerca de 45% do consumo de energia de um sistema de baixo consumo (SCHANDY; STEINFELD; SILVEIRA, 2015) e, portanto, deve

ser tratada com maior prioridade quando considera-se reduzir a utilização desnecessária da mesma, podendo-se ainda realizar algumas reduções de segurança que se julguem necessários para garantir melhorias de eficiência. Maior descrição e detalhes das tecnologias envolvidas estão apresentadas na seção 2 de Estado da Arte.

## 1.1 Relavância da Proposta

A proposta dessa dissertação possui relevância no âmbito das aplicações, no âmbito científico, e no âmbito tecnológico apresentados respectivamente abaixo.

### 1.1.1 Relevância em Aplicações

Redes de sensores sem fio têm inúmeras aplicações e seu potencial tornar-se-ia especulativo se considerar-se todas opções. Por essa razão, aqui apresenta-se a relevância sob a perspectiva de existência que uma camada de segurança na comunicação dos dispositivos da redes de sensores e especificamente o potencial de uso por entidades que fornecerão ambientes laboratoriais e reais para testes do mesmo. Em suas aplicações, os parceiros trabalham com sensoriamento e extração de dados; inteligência de controle e controle; atuação e sensoriamento *wireless*. Por conta disso, existem três domínios de aplicação, que acreditamos relevantes:

- Sensores avançados, para sensoriamento de parâmetros mais específicos, como qualidade do ar ou umidade do solo;
- Integração de dados prediais, para sistemas que unem informações de diversos sensores ou sistemas para tratamento e relatório; e
- Controle wireless, que visa trocar comunicação cabeada de sensores e atuadores por redes wireless, que apresentam custo pouco maior e custo de instalação milhares de vezes inferior ou inexistente;

Tais aplicações, como relatado em estudos da Navigant Research e MarketsandMarkets (MARETKSANDMARKETS, ) (ROHAN, ) (BUSINESSWIRE, 2014) (LORENZ, 2016) (MARTIN, 2014) (R., 2015), estão diretamente relacionadas a mercados de

grande expansão para os próximos anos e de grande importância para a expansão de redes de sensores sem fio e sua derivada Internet das Coisas (do inglês *Internet of Things*, ou IoT). Os mercados de automação predial inteligente e sistemas avançados de sensores têm expectativa de crescimento anual de pouco mais de 38% a.a., por exemplo. Os outros dois mercados ainda apresentam expectativa de crescimento de 30% para tratamento e acumulação de dados e 18% para sistemas de controle wireless. Para confirmar essas tendências, uma pesquisa realizada pela empresa IHS ainda afirma que empresas que fornecem produtos de automação predial inteligente podem esperar crescimento anual de 150% a partir de 2017. Além de enorme tendência de alto crescimento de todos mercados que o protocolo pode atuar - e também que os produtos dos parceiros fazem parte -, o próprio tamanho do mercado é grande: estima-se uma receita anual global de U$ 45 bilhões para 2021; os outros três mercados ainda apresentam tendências de crescimento alta, chegando a ficar 10 vezes maior que o valor atual em apenas 10 anos, que é o caso de controle via wireless. Estes mercados representam atualmente U$ 1 bilhão, U$ 100 milhões, U$ 90 milhões para sensores avançados, controle wireless e tratamento de dados, respectivamente.

### 1.1.2 Relevância Científica e Tecnológica

A importância de redes de sensores sem fios e suas aplicações, desafios, soluções e avanços em geral é facilmente representada pela quantidade de publicações e trabalhos realizados apenas sobre a mesma. A ferramenta de busca de artigos da IEEE (*Institute of Electrical and Electronics Engineers*), *IEEExplore*, apresenta 16.008 artigos sobre redes de sensores sem fio publicados desde 2001, em relação a sua biblioteca de 4.302.785 artigos desde 1907 no momento do desenvolvimento dessa dissertação. Isso significa que aproximadamente a cada 269 artigos científicos produzidos no mundo, um será especificamente sobre redes de sensores sem fio. Ou ainda que a cada dia são publicados 2.5 artigos em redes de sensores sem fio.

Essa tema ainda é relacionado a avanços em diversas áreas de ciência e tecnologia por ser multidisciplinar: se relaciona com química de baterias, física dos materiais, circuitos, *energy harvesting* (PRIYA; INMAN, 2009), sensores e sensores avançados, comunicação sem fio, criptografia, e inúmeros outros avanços para cada aplicação que é implementada utilizando essas redes de sensores sem fio. Ainda, além de sen-

soriamento, se um sistema tiver controle de atuação, o conceito torna-se ainda mais abrangente, envolvendo técnicas de controle e inteligência artificial em sistemas chamados Cyber-Físicos (LEE et al., 2014). Este refere-se a um sistema com grande quantidade de sensores interconectados, que pode oferecer grande quantidade de recursos de dados e medições, que seriam fontes sem precedentes para aprendizagem e modelagem de ambientes físicos, isto é, dinâmicos.

Tecnologicamente, as redes de sensores sem fio vão prover uma nova Internet, a Internet das Coisas conectadas (GUBBI et al., 2013). Estima-se que a mesma causará revolução na maneira com que interage-se com o mundo físico e como nos relacionaremos, da mesma forma que a Internet ou *Smartphones* fizeram. Para que tal revolução aconteça, é primordial que sua base tecnológica seja segura para acomodar toda inovação.

## 1.2 Premissas da Dissertação

Os objetivos para desenvolvimento dessa dissertação são baseados em premissas, levantadas pelo autor, a partir da revisão da literatura e observação empírica. Estas premissas são apresentadas nessa seção.

1. É possível desenvolver camada de segurança específica para redes de sensores sem fio de baixo consumo que atenda todas necessidades e abstrações necessárias por aplicações e serviços;
2. O estado atual de algumas tecnologias - como criptografia, por exemplo, permitem redução de gastos de processamento e, consequentemente, energia;
3. É possível reduzir quantidade de mensagens trocadas entre nós, principalmente quanto ao *handshake* inicial de criptografia;
4. É possível inovar cientificamente para conseguir novo esquema para segurança (que substituiria o DTLS, como será discutido ao longo da dissertação), integrado no protocolo de aplicação com:
   (a) *handshake* - troca inicial de mensagens de conexão;
   (b) autenticidade - quando se sabe que a origem da mensagem é verdadeira;

(c) privacidade - quando a mensagem é legível apenas pelo seu destino e não terceiros; e

(d) e integridade - quando a mensagem pode ser dita como válida.

Tudo isso de forma a se aumentar eficiência energética;

5. É possível reduzir tamanho de código na memória não-volátil (popularmente conhecida como ROM ou Flash) e dados necessários em memória volátil (popularmente conhecida como RAM) com camada de segurança mais integrada e específica para redes de sensores sem fio e dispositivos de baixa capacidade e consumo.

## 1.3 Objetivos da dissertação

Com base nas premissas levantadas, o objetivo principal dessa dissertação é:

**Propor, Desenvolver e avaliar uma camada de segurança para protocolos de comunicação de dispositivos de grão fino com amplo espectro de utilização.**

Para atingir o objetivo principal, objetivos específicos são necessários, descritos a seguir:

1. Levantar o estado da arte na área de protocolos de comunicação em redes de sensores e suas aplicações;
2. Discutir e propor estratégias de segurança suficientes para garantir autenticidade, confidencialidade e integridade;
3. Propor uma arquitetura da camada;
4. Desenvolver camada junto às necessidades e recursos disponíveis em padrão IEEE 802.15.4 - detalhado na seção 2;
5. Testar e validar protocolo empiricamente bem como descrevê-lo comparativamente seguindo as premissas da seção 3.2.2.

# 2 ESTADO DA ARTE

Neste capítulo, é apresentado o estado da arte, baseado em uma revisão da literatura científica referente aos problemas identificados nas soluções disponíveis na ciência e tecnologia; e relevante às necessidades de desenvolvimento de uma camada que provê segurança em dispositivos de grão fino.

## 2.1 Classificação de Categorias de Dispositivos

Nessa seção se discute como podem ser categorizados dispositivos utilizando suas características de recursos disponíveis e se justifica a utilização do termo "grão fino", proposto no título da dissertação.

### 2.1.1 Classificação de Dispositivos Quanto à granularidade

Inicialmente, é preciso definir quais dispositivos que fazem parte escopo do desenvolvimento de uma nova camada de segurança como descrita nos objetivos e detalhada na seção 3. Existem distintas formas de se categorizar um dispositivo. Um exemplo é pelo seu consumo, outro pela sua arquitetura. Por exemplo, um dispositivo de consumo ultra baixo (*Ultra Low Power*) é categorizado segundo seu design, se seus transistores trabalham perto da região de limiar (*threashold*), para reduzir perdas (MARKOVIC et al., 2010). Em outro caso, um supercomputador não tem uma definição de design, mas de arquitetura de paralelismo. Em suma, considerando a pesquisa realizada, não existe um padrão na literatura para se categorizar qualquer dispositivo.

Para abordar o tipo de dispositivo que a camada de segurança proposta tem melhor compatibilidade e vantagens na utilização, será utilizado o termo **dispositivo de grão fino**. A definição de granularidade segundo (STONE, 1993) é utilizada para computação paralela e segue a formulação: 2.1.

$$G_M = R_M / C_M \tag{2.1}$$

Onde $R_M$ é o tempo de execução de uma tarefa e $C_M$ é o *overhead* de comunicação

para realização dessa tarefa. Porém, esse tipo de relação pode ser abstraído para sistemas de redes de sensores, onde a computação paralela acontece, porém de forma distribuída e auto coordenada. A comunicação também é parte de redes de sensores, assim como de computação paralela, porém entre dispositivos. Além da analogia da arquitetura, o próprio nome de grão fino faz-se entender volumes coordenados de dispositivos enxutos, desta forma se justifica a utilização do termo grão fino em redes de sensores sem fio.

### 2.1.2 Classificação de Dispositivos Quanto às Restrições

Em definição padronizada, existe a categorização dada pela RFC 7228 (BORMANN; ERSUE; KERANEN, 2014) para *constrained devices*, que define classes de dispositivos que possuem memória volátil e não-volátil restrita. As categorias abrangem poucos dispositivos, pois têm foco naqueles que são, geralmente, embarcados. A tabela 1 demonstra as categorias.

| Classe | Memória Volátil | Memória Não-Volátil |
|---|---|---|
| Classe 0, C0 | «10 KiB | «100 KiB |
| Classe 1, C1 | ~10 KiB | ~100 KiB |
| Classe 2, C2 | ~50 KiB | ~250 KiB |

Tabela 1 - Tabela de classes da RFC7228 (BORMANN; ERSUE; KERANEN, 2014)

Dessa forma, define-se dispositivos de grão fino como dispositivos equivalentes àqueles, que no padrão de *constrained devices*, são de classe 0 e classe 1.

Essa seção de classificação de dispositivos teve como objetivo apresentar o tipo de dispositivo que será alvo da camada de segurança proposta nessa dissertação. Portanto, apesar desses conteúdos não serem utilizados para o desenvolvimento, serão constantemente utilizados como nomenclatura durante o restante do texto.

## 2.2 Sistemas Operacionais Embarcados

Sistemas operacionais embarcados têm grande importância nessa dissertação por comporem a base que permite a comunicação e operação das camadas e dos protoco-

| Nome | Características | Memória Volátil | Memória Não-Volátil | Classe |
|---|---|---|---|---|
| CC2630 | 2.4GHz<br>802.15.4 | 20kb | 128kb | Classe 1 |
| CC2650 | 2.4GHz<br>Multiprotocolo | 20kb | 128kb | Classe 1 |
| CC1310 | Sub 1GHz<br>802.15.4 | 20kb | 128kb | Classe 1 |
| CC2640 | 2.4GHz<br>Bluetooth Low Energy | 20kb | 128kb | Classe 1 |
| CC3220 | 2.4GHz<br>WiFi | 256kb | 1Mb | Classe 2 |
| MC12311 | Sub 1GHz<br>802.15.4 | 2kb | 32kb | Classe 0 |
| KW31Z | Sub 1GHz<br>Bluetooth Low Energy | 128kb | 512kb | Classe 2 |
| KW20Z | 2.4GHz<br>802.15.4 | 20kb | 160kb | Classe 1 |
| STM32 | MCU de uso genérico<br>sem rede | 128kb | 1Mb | Classe 2 |

Tabela 2 - Exemplos de dispositivos de baixo consumo populares e competitivos no mercado

los utilizados, testados e desenvolvidos, além disso, o sistema será o motor de comunicação e operação do código que funcionará como camada de segurança. Conforme supracitado, o *hardware* que é alvo para utilização do desenvolvimento aqui proposto é aquele caracterizado por grão fino ou ainda de classe 0 ou 1, segundo a classificação da RFC 7228. Dessa forma, apresentam-se sistemas operacionais embarcados conforme feito em trabalho realizado por Hahm (HAHM et al., 2016): reproduz-se uma tabela comparativa de sistemas operacionais embarcados na tabela 3, porém limitada a apenas quatro, que são aquelas que atendem as seguintes características:

- Possuam uma Pilha de Rede mais completa - isto é, implementações de camadas de transporte, rede, IP e outras que se julguem necessárias - integrado ao sistema operacional para desenvolvimento imediato;
- possuam abstração de *hardware* para algum dos dispositivos de *hardware* descritos, isto porque seria necessária outra dissertação para abstrair ou incorporar uma nova plataforma de *hardware* a um sistema operacional desse porte;
- Que seja *opensource*, para poder alterar código base em caso de necessidade

sem limitações e também para promover a utilização de código aberto;

- Que apresentem maturidade de código, pois será necessário aplicação estável para teste e validação dos resultados.

Dessa forma, segue ainda um detalhamento de cada um dos quatro sistemas operacionais que se enquadram nessas necessidades e estão apresentados na tabela 3.

1. **Contiki**, um sistema operacional (SO) desenvolvido desde 2002 (PAPADOPOULOS; MONTAVONT, 2002). Contiki é um dos primeiros a possuir pilha de rede completa como parte do próprio sistema ao implementar abstrações de hardware para SoC. O SO ainda é orientado a eventos e possui *protothreads*, que são *threads* de contexto mais compacto e de baixa latência, o que o torna muito econômico e veloz. Como exposto por Dunkels (DUNKELS, 2011), uma grande conquista científica do Contiki foi seu algoritmo *Radio Duty Cycle* (RDC) chamado de ContikiMAC que apresenta redução de energia gasta para comunicação em até 10 vezes. O Contiki ainda apresenta camada uIP que é um equivalente do IP apenas com funções mais necessárias para tais dispositivos comportarem seu binário sem grande custo de memória não-volátil.

2. **Contiki-NG**, ou *Contiki Next Generation* é um *Branch* - isto é, uma cópia com desenvolvimento próprio - do Contiki que teve nascimento recente para continuar a dar suporte ao sistema operacional de forma aberta em código, uma vez que os fundadores do Contiki passaram a dar suporte comercial ao sistema operacional original, o que levou o sistema original a ter problemas de instabilidade não resolvidos e implementações de padrões não atualizados para as versões mais recentes. O Contiki-NG resolveu todos problemas encontrados do sistema operacional original, atualizou suas implementações - algumas com completa reescrita do código, como no caso da camada de roteamento - e implementou novos protocolos como 6TiSCH e outros. Além disso, o Contiki-NG removeu suporte a protocolos não padrões como o próprio ContikiMAC, que apesar de econômico, trouxe dificuldades de integração com outros sistemas.

3. **RIOT**, desenvolvido desde 2012 (BACCELLI et al., 2012), é outro SO moderno

para sistemas embarcados, porém com foco específico em Internet das Coisas e suas necessidades e limitações. RIOT ainda foi desenvolvido para ter ambiente de desenvolvimento similar ao sistema operacional Linux, pois inclui a possibilidade de programação em C++ e possui recursos modernos como multithreading. Isso o faz um sistema robusto mas também custoso em processamento e memória, e, conforme citado por Hahm, está nos limites entre classe 1 e 2, ou seja, nos limite se ser considerado um sistema para um dispositivo de grão fino como definido aqui. O OS ainda tem camada de comunicação com 6LoWPAN e 6TiSCH além de outras portabilidades em constante desenvolvimento.

4. **FreeRTOS** teve desenvolvimento iniciado no mesmo ano que o Contiki, em 2002, porém com objetivos distintos. O SO (BARRY, 2003) tem funcionamento preemptivo e suporte para *multithreading* como o RIOT, o que o aproxima do modelo de programação convencional. Seu sucesso deve-se à grande quantidade de hardware para qual ele foi portado, maior parte dos microprocessadores embarcados populares têm suporte no FreeRTOS. Suas funcionalidades são modulares, portanto ele não possui 6LoWPAN no seu código base (*Kernel*), por exemplo, mas possuí bibliotecas de terceiros que permitem essa funcionalidade além de diversas outras como o próprio 6TiSCH.

5. **TinyOS** é o sistema operacional mais distinto dentre os citados. Isso porque o SO têm própria linguagem de programação chamada de *nestC* e estrutura de codificação complexa. Essas dificuldades são *trade-offs* escolhidos pelos desenvolvedores para garantir a menor quantidade possível de código de *linker*, melhor eficiência para funções e gerenciamento de processos e memória. Portanto, o TinyOS é um SO enxuto, que normalmente vai consumir menor memória volátil e não-volátil que seus concorrentes e apresentará grande eficiência. Por outro lado, sua dificuldade de aprendizado e programação limitou sua comunidade de desenvolvedores (LEVIS, 2012) e o sistema têm obtido avanços lentos.

Em suma, existem diversos tipos de sistemas operacionais com diversas características distintas, desde metodologia de programação até o nível de abstração. O estado da arte em sistemas operacionais permite ao autor tomar a escolha da tecno-

| Nome | Arquitetura | Agendamento | Modelo de programação | Classe do dispositivo mais apropriado |
|---|---|---|---|---|
| Contiki(-NG) | Monolítico | Cooperativo | Orientado a eventos *Protothreads* | Classes 0 e 1 |
| RIOT | *Microkernel* | Preemptivo, *tickless* | *multi-threading* | Classes 1 e 2 |
| FreeRTOS | *Microkernel* | Preemptivo | *multi-threading* | Classe 1 e 2 |
| TinyOS | Monolítico | Cooperativo | Orientado a eventos | Classe 0 |

Tabela 3 - Reprodução de tabela de referência de (HAHM et al., 2016) com apenas pontos destacados como relavantes para essa dissertação

logia que apresenta-se mais adequada para o desenvolvimento de uma camada de segurança de baixa utilização de recursos e, como citado posteriormente, o sistema escolhido foi o Contiki-NG.

## 2.3 Redes de Sensores sem Fio

A pesquisa de Redes de sensores sem fio, do inglês *wireless sensor networks* (WSN), não teve uma data bem determinada para seu início, é possível encontrar referências de telemetria desde o início da publicação de artigos científicos na IEEE, por exemplo. Porém, a utilização de sensores e seus desafios em específico têm sido abordada cientificamente com mais assertividade a partir dos anos 2000. Em 2004, Stankovic (STANKOVIC, 2004) discutiu, então de forma específica, quais eram os desafios para tornar WSN realidade científica, técnica e mercadológica. Stankovic fez discussões em diversos tópicos que estão reproduzidos a seguir, acrescentados avanços científicos realizados desde então.

- **Protocolos do mundo real**. Nesse tópico, o autor discute como WSN precisaria de adaptações em seus protocolos para o funcionamento em condições reais, ou seja, não simuladas e não teóricas como feito até o momento da sua publicação. Comparou ainda com outras tecnologias promissoras na época e como em WSN eram inviáveis pelo consumo elétrico ou custo, diferentemente de como é hoje. Em comparação, com o avanço da tecnologia e desenvolvimento das diversas camadas do modelo OSI (ZIMMERMANN, 1980), houve grande melhoria na eficiência da comunicação. Em camadas mais baixas, como enlace, protoco-

los MAC e RDC foram desenvolvidos especificamente para WSN para isso. Um exemplo é o ContikiMAC (DUNKELS, 2011) do sistema operacional Contiki.

- **Tempo real**. Em WSN, tempo real tornou-se uma necessidade em diversas aplicações. Em processos onde os dados sensoriados são críticos, a informação deve ser transmitida rapidamente. Neste contexto, considera-se tempo real o envio de dados sem aguardo em filas e no formato de *stream*, isto é, dados contínuos. O autor cita que os desafios encontrados eram em relação ao roteamento rápido das mensagens, feito em tempo real para que a mensagem encontre um caminho de baixa latência. No próprio artigo ele propõe duas soluções: a utilização de protocolo RAP (LU et al., 2002) ou SPEED (HE et al., 2003) de roteamento de dados. Atualmente, apesar do grande sucesso inicial do protocolo SPEED e seus resultados teóricos apresentarem vantagens sobre outras opções, sistemas modernos não possuem a implementação. Ao invés disso, sistemas modernos buscaram estabelecer o roteamento anteriormente ao tempo real para que essa não seja uma necessidade local no tempo. Os protocolos implementados para roteamento em sistemas operacionais modernos são RPL, Trickle e LOAD, que também já são padrões.

- **Gerenciamento de energia**. Nesse tópico o Stankovic discute as dificuldades de se energizar um nó de uma rede de sensores sem fio, ressaltando dois problemas: as limitações de energia das baterias e a dificuldade da utilização dessas baterias em ambientes sensíveis ou controlados. As baterias sofreram muitas melhorias desde o ano da publicação do artigo de Stankovic, na época era inicial a produção de baterias do tipo Lítio-Ion (SAHI, 2016), enquanto hoje já estão no limiar de serem substituídas por baterias de nano fios de silício (CHAN et al., 2008). Apesar do avanço das baterias, os avanços mais consideráveis foram em economia de energia, utilizando design de consumo ultrabaixo e reduzindo desperdícios com sensores, sistemas operacionais e protocolos (de diversas camadas) com melhor eficiência energética. Essa dissertação propõe melhorar também eficiência energética, porém de camada de segurança e consequentemente aplicação.

- **Abstração de Programação**. Stankovic também descreve a necessidade de

abstrair programação, mais especificamente as funções de *hardware* que precisariam ser reescritas para todo novo projeto. Isso auxiliaria desenvolvedores a produzirem mais e mais rapidamente, além de ser vantajoso para o desenvolvimento da tecnologia. Nessa mesma época, nos anos 2000, desenvolvedores perceberam essa necessidade e passaram a criar bibliotecas, *frameworks* e sistemas operacionais, como descrito nas seções acima. Atualmente, tanto desenvolvedores independentes quanto os próprios fabricantes de *hardware* têm fornecido sistema operacional ou bibliotecas de *drivers* para agilizar e facilitar o desenvolvimento. Porém ainda não existe um padrão para camada de abstração de *hardware*, do inglês *hardware abstraction layer* (HAL) e isso tem sido implementado pelos sistemas operacionais *opensource*. O fato de fabricantes quererem diferenciar e promover seus produtos é um dos fatores que prejudicam a existência de um HAL.

- **Segurança e privacidade**. Nesse último tópico, são abordados três tipos de situações distintas em WSN:

  1. a segurança física dos nós, que geralmente estão atrelados a medições locais e ao fato dessas medições terem representatividade. Isto é, um sensor de temperatura não deve fazer medições que podem ser não reais ou falsificadas por um isqueiro ligado nas proximidades, por exemplo;
  2. a segurança da comunicação, para evitar informações perdidas ou compartilhadas com terceiros, bem como integridade da informação na camada de comunicação; e
  3. a segurança contra ataques, como *jamming* - quando um sinal potente é iniciado nas proximidades para ocupar o canal -, ou *distributed denial of service* (DDoS) - equivalentemente ocupa comunicação mas enviando mensagens processáveis - e outros.

  Atualmente esses tópicos de segurança ainda levantam discussões de como serem abordados e sanados, respectivamente:

  1. Piedra (PIEDRA et al., 2013) por exemplo, discute sobre limitações do próprio ambiente e avalia como os trabalhos exemplificados em seu artigo foram realizados e como poderiam ter sido efetivamente mais protegidos.

Neste caso, a segurança não é mais voltada a desenvolvimento de aplicações, mas de peças físicas que protejam o equipamento, bem como níveis de acesso ao equipamento;

2. É o tópico dessa dissertação em nível de camada de segurança para aplicação e também é parte de extenso estado da arte que será discutida na seção 2.3.5.

3. Também ainda é alvo de pesquisa, porém ainda não existe solução, pois essas tipos de ataques apresentam desafios quando ao canal de comunicação; esses ataques afetariam quaisquer redes Wireless, caracterizando-os como desafios de comunicação Wireless em geral.

### 2.3.1 O padrão IEEE 802.15.4

O padrão IEEE 802.15.4 (GROUP et al., 2011) foi desenvolvido pelo grupo *IEEE 802.15 WPAN Task Group* com intuito de disponibilizar uma nova camada de enlace com conexão ou conversão fácil para padrões de mercado e ao mesmo tempo evitar interferências de frequência com redes sem fio convencionais do padrão 802.11, além de oferecer grande redução do consumo de energia nas camadas físicas e de enlace. É um padrão para redes sem fio locais particulares (do inglês *Wireless Personal Area Network*, ou WPAN) que especifica com precisão as necessidades das redes de sensores sem fio atuais. Algumas de suas características são:

- Camada física nas frequências 868.0 - 868.6 MHz (europeia), 902–928 MHz (norte americana) e 2400–2483.5 MHz para uso geral. Ainda existem adições ao padrão para outras frequências especiais de funcionamento, principalmente para implementação na Ásia e em atualizações recentes do padrão, porém, aqui se demonstra as frequências convencionais;

- Camada física com respectivamente 1, 13 e 16 canais, para as frequências acima;

- Taxa de transferência entre 100 e 250 kpbs para qualquer frequência;

- Banda do canal de 2MHz e espaçamento de 5MHz;

- Alcance projetado de 10 metros no mínimo, porém a tecnologia de *hardware* já permite alcance entre 100 metros e 10 quilômetros;
- Limite de *maximum transmission unit* (MTU) de 127 *bytes*.

O padrão ainda define tipos de dispositivos:

1. *full-function device* (FFD), que é um dispositivo que pode se comunicar com outros FFD, com nós mais restritos e também exercer outras funções como *root* ou coordenador da rede.
2. Os dispositivos mais restritos são os *reduced-function devices* (RFD) e podem apenas se comunicar com FFDs, além de não poderem assumir nenhum outro papel na rede, são conhecidos em literatura pelo nome de *leaf*.

Ainda com importância para a comunicação, o padrão define tipos de topologia que podem ser estrela ou ponto-a-ponto - que também é conhecido como *mesh*, quando envolve saltos por dispositivos intermediários.

Existe grande esforço para redução de gastos de energia na utilização do padrão 802.15.4, além disso, é um padrão que trata apenas das camadas mais baixas da comunicação no modelo OSI, as camadas de enlace e física e, portanto, foi o passo inicial necessário para o desenvolvimento de aplicações de consumo baixo. Redução de gastos com energia em camadas mais altas foram feitas utilizando outros padrões como 6LoWPAN (THUBERT; HUI, 2011), ZigBee (ALLIANCE, 2007), EnOcean (PLOENNIGS; RYSSEL; KABITZSCH, 2010), *Bluetooth Low Energy* (GOMEZ; OLLER; PARADELLS, 2012). Apesar dos últimos serem padrões numerosos, o padrão que terá uso no desenvolvimento na dissertação será o 6LoWPAN, este está descrito a seguir e justificado seu uso nas próximas seções.

### 2.3.2 6LoWPAN

6LoWPAN vem do inglês *IPv6 over Low-Power Wireless Personal Area Networks*. O padrão, de camadas médias do modelo OSI, especificamente da camada de rede e transporte, faz uso de técnicas e regras para redução de *headers* para compatibilidade com protocolos do mercado, como IP, mas em padrão IEEE 802.15.4. O Padrão

foi projetado e proposto pelo grupo *Network Working Group* do IETF. A seguir estão descritas algumas características do padrão:

- Funcionamento com IPv4 e IPv6: como o padrão 802.15.4 limita o MTU a 127 *bytes*, o padrão fornece regras para compressão de *header* IP, como utilização de IPv6 de 16 *bits* para rede local;
- O padrão permite utilização de algoritmos de descobrimento de vizinhos com otimizações;
- o padrão ainda descreve o uso de redes *mesh* e endereçamento, mas a execução deve ser feita por outras camadas (como camadas de roteamento).

Apesar de aparência mais simples, o padrão teve grande impacto científico pois possibilitou o uso do padrão 802.15.4 integrado a endereçamento IP. Diferentemente de outros padrões como ZigBee e BLE que sacrificaram o uso do mesmo. O protocolo IP é a base para o funcionamento da Internet e sua implementação em redes de baixo consumo promoveu a comunicação com redes 802.15.4 de forma quase direta, sem precisar de sistemas que precisem interpretar todo pacote e reprocessá-lo para outro padrão. A Exceção se dá ao BLE que em RFC 7668 (NIEMINEN et al., 2015), faz implementação de *IPv6 over BLE*, ou seja, utilização de protocolo IP em BLE, porém isso só torna a codificação mais extensa, não funcionando da mesma forma que o 6LoWPAN foi projetado.

### 2.3.3 Protocolos de aplicação para Redes de Sensores sem Fio

Nessa seção apresentam-se os protocolos de aplicação para redes de sensores sem fio de baixo consumo disponíveis no mercado.

#### 2.3.3.1 O protocolo MQTT

Acrônimo de *Message queue telemetry transport*, é um padrão ISO (ISO/IEC PRF 20922) (ISO, ) desenvolvido para funcionamento junto a pilha de rede TCP/IP e tem código aberto e intencionalmente feito para aplicações *machine to machine* (M2M). Sua proposta é ter baixa utilização de memória não-volátil e volátil, o que o torna ideal

para dispositivos embarcados. O protocolo funciona com modelo conhecido como *publisher/subcriber*, onde um ponto funciona como publicador de informações e os pontos *subcriber* captam essas informações publicadas. Sua maior vantagem é o *header* reduzido e mensagens que acontecem apenas com eventos, o que reduz drasticamente o consumo e ocupação da rede como apresentado por (YASSEIN; SHATNAWI et al., 2016). Vale ressaltar que essa melhoria pode trazer problemas de rajada de pacotes, porém isso não foi avaliado pelos trabalhos referenciados.

O protocolo tem performance melhor que o CoAP - apresentado nas próximas seções -, conforme exposto por (YASSEIN; SHATNAWI et al., 2016), principalmente em cenários onde o tráfego de dados é intenso. Ademais, segundo (COLLINA et al., 2014), o protocolo tem maior *throughput* e menor latência que o CoAP, tem suporte para *Quality of Service* (QoS), tornando-o ideal para aplicações M2M, que é objetivo do seu desenvolvimento. Esse resultado também é confirmado em aplicação real, conforme (KAYAL et al., 2016). Apesar do baixo consumo e facilidade de implementação, (YASSEIN; SHATNAWI et al., 2016) ainda cita que não é um protocolo de tempo real por conta da latência dos seus dados, o que impossibilita seu uso em operações críticas. Por fim, não é um protocolo que comporta muito tipos de dados. Sua maior desvantagem é necessidade de utilização da camada TCP/IP, visto que é uma camada extensa para se implementar em dispositivos de grão fino, portanto, apesar de algumas vantagens, seu consumo de recursos locais de memória e impactante.

#### 2.3.3.2 O protocolo XMPP

Acrônimo de *Extensible Messaging and Presence Protocol*, é um padrão IETF RFC 7622 (PETER, 2015) (RFC 6122 (PETER, 2011) até 2015) desenvolvido pelo *XMPP working group*. O protocolo tem uso intenso em sistemas de IoT por algumas de suas vantagens, uma delas por ter comunicação convencional do tipo cliente-servidor e ainda suporta modelo *publisher/subcriber* para comunicação bidirecional ou multidirecional, tudo sobre o protocolo TCP. O protocolo é completo e apresenta muitas funcionalidades em relação aos seus concorrentes, como alta escalabilidade e mensagens pequenas e estruturadas baseadas em XML com baixa latência. O protocolo ainda tem implementações de segurança baseadas em TLS e de seu nome, *Extensible*, significa que vários módulos podem ser utilizados como plug-in, o que possibilitou

o protocolo a realizar operações como VoIP, transferência de arquivos, mensagens instantâneas, chamadas de vídeo ou outros (KAYAL et al., 2016). Por outro lado, suas mensagens são em texto puro e não tem nenhum tipo de QoS. Além disso, segundo (YASSEIN; SHATNAWI et al., 2016), o protocolo tem grande consumo de CPU e, portanto, de energia. O XMPP apresenta ainda menor latência dentre todos os protocolos apresentados (KAYAL et al., 2016). Segundo os trabalhos de YASSEIN e KAYAL, o protocolo também não seria ideal para aplicações em dispositivos que aqui caracterizam-se como dispositivos de grão fino por ter implementação com grande consumo de memória e energia.

#### 2.3.3.3 O protocolo CoAP

Acrônimo de *Constrained Application Protocol*, o CoAP é padrão IETF RFC 7252 (SHELBY; HARTKE; BORMANN, 2014) desenvolvido pelo grupo Constrained *RESTful Environments Working Group (CoRE)* para funcionamento em dispositivos de baixo consumo de classe 0, 1 ou 2, conforme definido na RFC 7228, detalhada na seção 2.1.2.

O CoAP foi desenvolvido pensando nas necessidades das redes de sensores sem fio de dispositivos restritos, isto é: redes com altas taxas de perdas e desconexões; e dispositivos com baixa quantidade de recursos para processamento. Dessa forma, o CoAP seria o protocolo ideal para funcionamento de aplicações M2M IoT que tenham seus recursos limitados. Outra vantagem do CoAP é sua fácil tradução para o protocolo HTTP, fazendo sua interoperabilidade mais simples com sistemas de grande porte que estão por trás desse protocolo.

Ademais, o CoAP, dentre todos protocolos apresentados, foi o único especificado para funcionar em UDP especificamente, uma camada de transporte muito mais enxuta e fácil de se desenvolver, o que é grande vantagem quando tratando dos dispositivos restritos da RFC 7228 (OH; KIM; FOX, 2010).

O CoAP funciona com modelo de cliente-servidor, para responder mensagens instantaneamente, mas também implementa o modelo *publisher/subscriber*, isso permite maior performance e escalabilidade segundo (KARAGIANNIS et al., 2015). Ainda segundo (KARAGIANNIS et al., 2015), as maiores vantagens do CoAP são simplicidade

e confiabilidade (de entrega de mensagens): o protocolo administra a troca de mensagens via UDP, o que o faz leve, rápido e permite ainda implementação de mensagens de confirmação de entrega (*acknowledgement*). O CoAP tem alguns tipos de mensagens, esses são:

- ***Conformable Message***, uma mensagem referente a um serviço confirmado, ou seja, o seu envio é garantido e será confirmado o recebimento com retorno de uma mensagem de *Acknowledgement*;
- ***Acknowledgement Message***, que serve para identificar uma mensagem de *Acknowledgement*, é a resposta dada quando uma mensagem ***Conformable Message*** é recebida e processada com sucesso;
- ***Non-conformable Message***, uma mensagem que não é confirmável, ou seja, o seu envio é não é garantido por uma resposta de uma mensagem de *Acknowledgement*;
- **Reset Message**, No caso de uma mensagem, tanto *Conformable Message* quanto *Non-conformable Message*, for recebida com erros ou faltando informações ou partes, a mensagem *Reset* é propagada de volta para informar a fonte;
- ***Piggybacked Response*** é a mensagem de confirmação de recebimento de mensagems de *Acknowledgement*, respondida imediatamente após o recebimento da última;
- ***Separate Response*** utilizada quando a resposta para a mensagem enviada inicialmente é enviada separadamente da confirmação de recebimento, enviada em sequência após envio da mensagem de *Acknowledgement* quando for necessária; e
- ***Empty Message*** para mensagens vazias.

Por outro lado, o CoAP tem alta latência em relação aos protocolos de aplicação avaliados, como demonstrado por (KAYAL et al., 2016) em comparação com outros protocolos para mesma aplicação, além de sofrer de problemas com escalabilidade, que, segundo ainda os resultados de (KAYAL et al., 2016), tornam o protocolo lento

quando sob grande tráfego, portanto sua utilização é favorável em redes ou dispositivos com pouca utilização ou poucos nós, segundo a tecnologia utilizada pelas referências. Por conta da comparação entre protocolos supracitados, como descrito na seção 3.2.1, o CoAP será utilizado em *testbed* para avaliação da performance de camadas de segurança em dispositivos com recursos restritos, visto que é o protocolo de menor utilização de recursos e, portanto, apto a trabalhar nesses tipos de dispositivos. Além disso, apesar de algumas desvantagens em relação a outros protocolos, o CoAP foi escolhido por seu tamanho em memória menor e por utilizar de técnicas (como de filas) e modelos de comunicação que o tornam um protocolo de baixo impacto de recursos da rede como um todo. Por fim, o CoAP foi desenvolvido especificamente para esses tipos de rede que possuem baixas taxas de transferência, maior latência e menores filas para armazenamento de informações.

Nessa seção das camadas de aplicação, foram listados protocolos de utilização do mercado em soluções que estejam vínculadas à tecnologia de redes de sensores sem fio. Todos protocolos têm vantagens e desvantagens para cada tipo de utilização, mas tratando-se de baixa utilização de memória na implementação e em suas dependências, o CoAP apresenta melhor adequação para utilização em dispotivos restritos e por isso foi escolhido como protocolo para avaliação da camada de segurança, como reafirmado posteriormente.

### 2.3.4 Camada de segurança

Na maioria das implementações de código, a camada de segurança tem suas funcionalidades logo abaixo da camada de aplicação e funciona de forma imperceptível aos protocolos superiores. Apresentam-se aqui duas implementações e padrões principais de segurança, utilizados abrangentemente.

#### 2.3.4.1 TLS sobre TCP

TLS é acrônimo para *Transport Layer Security*. E é uma entidade da camada de aplicação que utiliza diretamente a camada de transporte para garantir segurança na comunicação entre dois pontos. A definição mais atual do protocolo segue a RFC 5246 (DIERKS, 2008), que define sua funcionalidade, campos de *header*, algoritmos de

criptografia aceitos e demais parâmetros para o funcionamento desse tipo de camada.

O TLS já está na versão 1.2 e recebe atualizações que provocam melhorias de segurança e otimização, porém o que define o nível de segurança do protocolo é o algoritmo criptográfico utilizado e o tamanho de sua chave. Atualmente, segundo a DigiCert (DIGICERT, 2018) o algoritmo mais adotado para troca de chaves na Internet é o RSA (RIVEST; SHAMIR; ADLEMAN, 1978). O RSA é um algoritmo que foi inovador na época de sua criação, por ser um algoritmo de chave pública/privada de grande dificuldade para ser quebrado, por ser baseado em multiplicação de números primos.

A NIST (*U.S. National Institute for Standards and Technology*) (BARKER; DANG, 2016) já recomenda a utilização de chaves de tamanho de 2048 *bits* para aplicações seguras com RSA.

Para realização de uma conexão, além da própria chave privada de 2048 *bits*, um cliente deve ter espaço para pelo menos mais 2048 *bits* de uma chave pública e mais 2048 *bits* para cálculo do segredo comum entre as chaves. Utilizando o máximo de eficiência em espaço, isso resulta em um espaço de memória volátil de 768 *bytes* apenas para buffers, ou cerca de 5-10% do espaço disponível em um SoC de Classe 0 ou 1. Por isso, algoritmos como esse são de difícil adaptação em sistemas computacionais de grão fino. A figura 1 mostra como funciona ainda a comunicação do *handshake* de inicialização de uma sessão criptografada TLS.

#### 2.3.4.2 DTLS sobre UDP

Utilizando-se a camada UDP como transporte necessitou a criação de um outro padrão similar ao TLS para segurança. O chamado DTLS (*Datagram TLS*) em camada de transporte UDP é especificado pela RFC 6347 (RESCORLA; MODADUGU, 2012) e aprovado pela IETF como padrão. Assim como no TLS, o padrão oferece flexibilidade para escolha de serviços de segurança e mecanismos de criptografia. Segundo o padrão, ocorre a troca de três mensagens (três *round-trip*) para estabelecimento da conexão DTLS, conforme figura 2. O processo é descrito a seguir:

1. O cliente envia mensagem de *Hello* contendo informações do tipo de criptografia a utilizar;

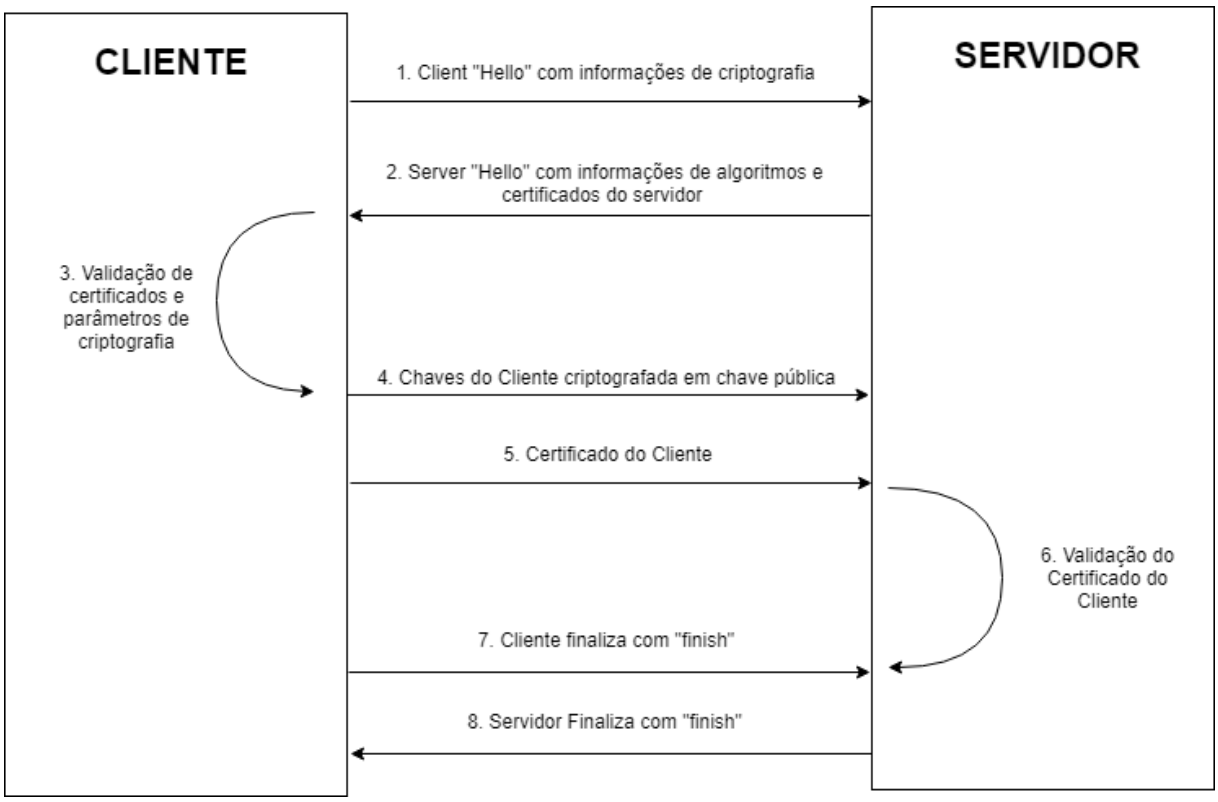



Figura 1 - Gráfico para *handshake* TLS em TCP

2. Se o servidor estiver de acordo com a criptografia, responde com confirmação de *Hello*;

3. O cliente então confirma o *Hello*, enviando-o novamente;

4. O servidor então envia seu *Hello* junto com certificado e chaves públicas;

5. O cliente então responde com mensagem de certificado com suas chaves, especificações da criptografia utilizada e, em seguida, uma mensagem de 1 byte acordando com os termos de segurança;

6. Por fim o servidor finaliza o *handshake* verificando certificados e enviando mensagem de 1 byte acordando com os termos de segurança.

No fim do processo, ambos pontos estão conectados com segurança de uma chave pública e podem por exemplo gerar chaves simétricas para comunicação baseado no segredo trocado, esquema de troca de chaves descrito em detalhes nas próximas seções.

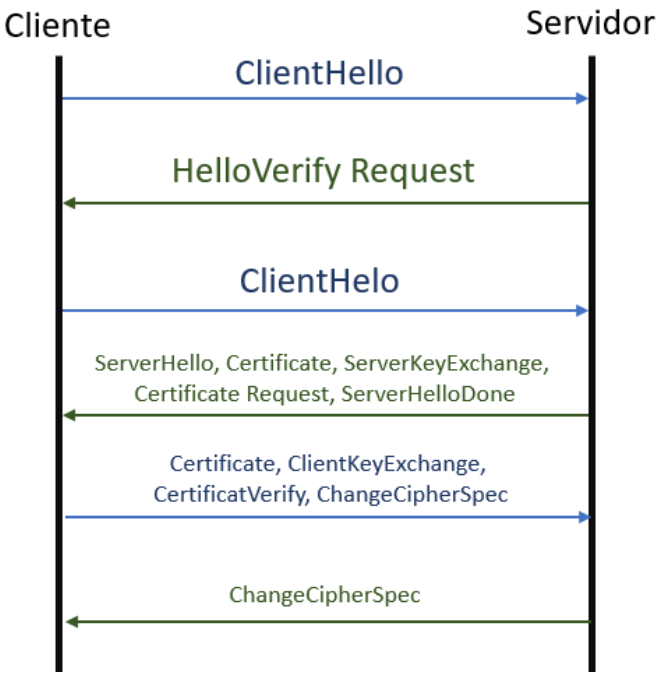



Figura 2 - Gráfico para *handshake* DTLS em UDP

### 2.3.5 Camadas de segurança em Protocolos de Aplicação

As camadas de segurança, como será discutido posteriormente, são demasiadamente extensas para aplicações de WSN. Mesmo o UDP, que é um protocolo mais simples que TCP com segurança mais enxuta, tem quantidade de funcionalidades e extensão de implementações de segurança fazem com que sua implementação não seja possível ou apresente grande consumo de energia e memória em sistemas de dispositivos de grão fino.

Por conta disso, já existe esforço em pesquisa para unificar essas camadas seguras com as camadas de aplicação que o desenvolvedor venha a utilizar, de forma a integrá-las de forma aa utilizar menos recursos, tanto em termos de memória não-volátil quanto de tamanho de pacote. Como o CoAP é o protocolo que será utilizado para avaliação comparativa da camada de segurança, segue uma descrição de trabalhos realizados para integrar CoAP e segurança de forma mais adequada a WSN.

#### 2.3.5.1 DTLS e CoAP

O DTLS foi desenvolvido inicialmente para segurança de aplicações na Internet e jamais foi intencional sua utilização com protocolos de baixo consumo como CoAP

ou outros de redes de sensores sem fio. Por isso, sua utilização causa um *overhead* grande, como descrito por Capossele (CAPOSSELE et al., 2015). Ademais, como discutido por Capossele, a implementação de DTLS requer que o protocolo de aplicação faça ordenação e confiabilidade de entrega o que não existe em DTLS e adiciona ainda mais funcionalidades para o CoAP, o tornando mais complexo e com maior consumo em termos de memória volátil e não-volátil, o que para dispositivos das classes 0, 1 e 2 é fator limitante.

Além do supracitado, ocorre grande desperdício de energia apenas para conexão que pode ser facilmente perdida em redes de sensores sem fio de baixo consumo: são necessários quatro *round-trips* para utilização do CoAP e DTLS, como apresentado por Alghamdi (ALGHAMDI; LASEBAE; AIASH, 2013). Existem ainda outras desvantagens do uso de CoAP com DTLS como não existir *multicast* ou *broadcast* sobre DTLS, os desafios de fragmentação de pacotes com uso de certas criptografias, e o fato de o cliente se conectar sem necessariamente fazer algum tipo de verificação de autenticidade e muitas outras levantadas por Alghamdi; mas sendo a única opção padronizada, sua implementação para redes de sensores sem fio tornou-se necessidade.

Para sobrepor esses problemas, autores têm evidenciado esforços para otimizar a utilização do CoAP com DTLS ou adaptá-los totalmente para um protocolo novo unificado com suas propriedades específicas: em trabalho de Alghamdi (ALGHAMDI; LASEBAE; AIASH, 2013), o autor fez o uso do protocolo IPSec como substituto; Capossele, utiliza de outros avanços para desenvolver um mecanismo que integra o DTLS no CoAP utilizando a camada de mensagens do mesmo para as necessidades de ordenação e fragmentação do DTLS, além disso, utiliza as próprias mensagens do CoAP para o *handshake* de segurança e já garante entrega de mensagens utilizando as mensagens *Confirmable messages* e *Acknowledgement message*, a figura 3 mostra como funciona o mecanismo. Raza (RAZA et al., 2013), tomou decisão de melhorar a eficiência por compressão do DTLS junto ao CoAP, reduzindo o tamanho das mensagens pois, a mensagem é o item que vai consumir mais energia, devido à potência necessária para sua transmissão, resultando em pacotes consideravelmente menores e com menor fragmentação (e portanto menos pacotes necessários), reduzindo o consumo energético em 15%, sem resultados de comparação de uso de memória.

De forma geral, há esforço científico para se adaptar o DTLS ao CoAP, porque o

próprio DTLS não foi feito para funcionar em sistemas com recursos restritos. Apesar de positivos, tais esforços ainda não possibilitaram melhoria significativa do desafio de implementar-se segurança em dispositivos de grão fino.

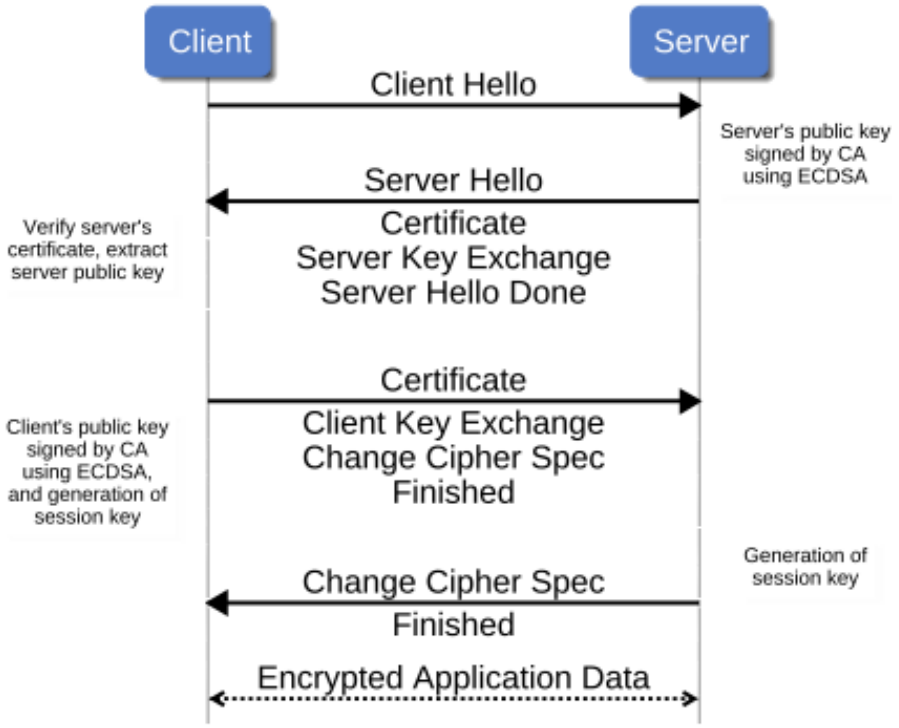



Figura 3 - Gráfico para *handshake* DTLS em UDP com integração no CoAP, implementado por Capossele et al. em (CAPOSSELE et al., 2015) e reproduzido aqui

#### 2.3.5.2 TinyDTLS

Devido a essas necessidades de implementação de DTLS, foi criado o TinyDTLS (BEATON, 2016), que é uma implementação limitada do DTLS mantida e desenvolvida pelo grupo Eclipse. O TinyDTLS foi desenvolvido para auxiliar a utilização do DTLS em dispositivos restritos de recursos como memória volátil e não-volátil. A biblioteca de implementação do TinyDTLS é autocontida, isto é, não têm dependências externas que os compiladores padrão não possuam e implementa apenas dois modos de funcionamento do DTLS:

- Troca de chaves assimétricas do tipo curvas elípticas para criptografia AES-128;
- Utilização direta de chaves pré-programadas (algoritmo *Pre-shared Key* ou PSK) com criptografia AES-128.

O TinyDTLS é implementação padrão na camada de segurança do CoAP no sistema operacional Contiki-NG e em outros sistemas que desejem e tenham recursos suficientes para utilizar o padrão mais enxuto do DTLS, além disso, tem se tornado recurso padrão de segurança para dispositivos de recursos restritos. Como será discutido nas próximas seções o *testbed* utiliza o TinyDTLS comparativamente.

## 2.4 Esquemas e Algoritmos criptográficos

Esquemas criptográficos são base para comunicação confidencial. Relevantes a essa dissertação, destacam-se algoritmos que são parte da solução implementada para os objetivos propostos.

### 2.4.1 Algoritmos de *Hashing*

Algoritmos de *hashing* são algoritmos que transformam algum tipo de informação em uma sequência única de *bytes* irreversivelmente, ou seja, cuja função inversa não existe. Algoritmos de *hashing* são comumente utilizados para validação de informações, visto que podem gerar um pequeno identificador baseado em uma mensagem maior. Além disso, são utilizados para guardar informações sigilosas de forma a serem irreversíveis. Essa última técnica é utilizada, por exemplo, em bancos de dados, para que mesmo se alguém tiver acesso a este banco de dados, as informações sensíveis armazenadas que passaram por *hashing* estariam todas irreversivelmente codificadas, o que impediria acesso a essas informações. Um exemplo de algoritmo de *hashing* é o SHA-256.

**SHA-256** O algoritmo SHA-256 faz parte do conjunto de algoritmos *Secure Hash Algorithm 2* (SHA-2) e possui tamanho de 256 *bits* ou seja, 32 *bytes*. Isso significa que mensagens serão processadas de 32 em 32 *bytes*, e terão sempre saída de *hashs* 32 *bytes*, que é o chamado tamanho de bloco. O processo de *hashing* do SHA-256 depende de combinações de operações como SHIFT circular, XOR, AND, OR e uma extensa tabela de inicialização para embaralhar os *bits* de forma a obter-se o *hash*. O processo completo é complexo e é descrito pelo documento oficial do NIST (NIST, 1995).

### 2.4.2 Algoritmos de Criptografia

Diferentemente dos algoritmos de *hashing*, algoritmos de criptografia conseguem fazer sua função inversa para extrair a informação criptografada. Por conta desse funcionamento, esses algoritmos dependem de um chave de criptografia, pois dessa forma uma mesma informação pode ser criptografada de diversas formas distintas, o que impossibilita, na prática, a descriptografia dessa informação sem a chave em questão. Os dois tipos de criptografia utilizados nos esquemas de criptografia dessa dissertação são:

1. Criptografia de chave simétrica, que depende que os *endpoints* de conexão tenham uma chave comum;

2. Criptografia de chaves assimétricas, cujos pares de *endpoints* não precisam saber a chave comum, mas precisam disponibilizar uma chave pública e operar com sua própria chave privada.

#### 2.4.2.1 Criptografia de Chave Simétrica: AES-128

O *Advanced Encryption Standard* (AES) é um algoritmo de criptografia de chave simétrica, ou seja, é uma algoritmo que apenas pode criptografar e descriptografar informações com uma mesma chave única, que é padronizado pelo NIST (NIST, 2001). A chave é escolhida pelo usuário e o nível de segurança depende diretamente do tamanho da chave. Apesar disso, como demonstrado através da figura 4, chaves simétricas pequenas têm naturalmente grande nível de segurança, pois sua origem pode ser qualquer. Em específico, o algoritmo AES-128 trabalha com blocos de 128 *bits*, ou 16 *bytes*. O processo de criptografia AES está descrito no seu documento de padronização e é complexo, envolvendo etapa de geração de chaves chamadas de *round keys* utilizando outro algoritmo, chamado *Rijndael's key schedule* como descrito pelos seus autores em (DAEMEN; RIJMEN, 1999); envolvendo em seguida diversas operações em etapas, sendo uma delas não linear e dependente de uma tabela de valores pré-configurados.

#### 2.4.2.2 Criptografia de Chaves Assimétricas: Curvas Elípticas

A criptografia de curvas elípticas (do inglês *Elliptic-curve cryptography*, ou ECC) é um tipo de criptografia de chaves assimétricas. Isto é, como explicado por Miller (MILLER, 1985), a criptografia depende não apenas de uma chave comum entre os pontos, mas de que cada ponto possua sua chave pública, que é a chave que é enviada aos pares que desejam se conectar; e a chave privada, que é usada internamente para cálculo de algum segredo.

Diferentemente das chaves simétricas, as chaves assimétricas não são utilizadas para criptografar informação, visto que é muito difícil prever o resultado de uma troca de chaves. Ao invés, as chaves assimétricas são utilizadas para troca de um segredo comum aleatório que então pode ser utilizado para geração de uma chave simétrica para utilização em algoritmo criptográfico. Esse processo é comum visto que são chaves simétricas que possibilitam trocas de informações de forma rápida e segura.

Em específico, o algoritmo de curvas elípticas pode funcionar como um provedor de criptografia assimétrica utilizando pontos em curvas elípticas pré-determinadas. Um exemplo de curva elíptica é a chamada *secp256r1*, definida pelo IETF (BLAKE-WILSON et al., 2006), que define os parâmetros da curva. De forma simples, com a curva e seu ponto inicial na curva definidos, é escolhido um número grande - que nesse caso é a chave privada -, para multiplicar-se esse ponto. A multiplicação desse ponto é rápida devido a propriedades inerentes da curva, que levam a curva a um novo ponto. As coordenadas desse novo ponto é a chave pública. A realização da função inversa dessas operações, porém, é de grande dificuldade, pois envolve aproximação e erros de funções logarítmicas, o que torna mesmo chaves pequenas, muito seguras. Dessa forma, com a troca chaves públicas, basta seguir a mesma metodologia de multiplicação da chave pública recebida com a privada guardada para encontrar-se um segredo comum que é, para ambos, a mesma multiplicação, como exposto em detalhes na figura 5 que é parte da explicação dessa técnica, conhecida como Diffie–Hellman, detalhada a seguir.

### 2.4.3 Esquema de troca de chaves Diffie–Hellman

O esquema de troca de chaves assimétricas Diffie–Hellman é resultado dos trabalhos de Whitfield Diffie e Martin Hellman, detalhados em (DIFFIE; HELLMAN, 1976). Esse tipo de esquema permite a transmissão de chaves públicas em canal não conhecido ou até mesmo não confiável sem que haja prejuízo a segurança, ao utilizar métodos matemáticos para determinação de um segredo comum apenas entre dois pontos. O tipo de chave, processo matemático e resultado definem do tipo de criptografia utilizado. Um possível esquema de criptografia é o de curvas elípticas, que quando usado com a troca de chaves Diffie–Hellman é conhecido como *Elliptic-curve Diffie–Hellman* (ECDH), especificado pelo NIST (BARKER et al., 2013).

O esquema ECDH tem grandes vantagens por ser matematicamente mais robusto, isto é, mais difícil de se descriptografar, o que torna possível a utilização de chaves menores, de tamanho 256 *bits*, por exemplo. O NIST fornece relatório para comparação dos tamanhos adequados de chaves, nele é exibido que uma chave de curvas elípticas de 256 *bits* tem segurança equivalente a uma chave do esquema RSA - algoritmo padrão de criptografia de chave assimétrica em diversos sistemas - de 3072 *bits*, 12 vezes mais enxuta. A figura 4 reproduz uma tabela do relatório NIST *Recomendação para Transição do Uso de Algoritmos Criptográficos e Comprimentos de Chave* por (BARKER; ROGINSKY, 2011) que demonstra esse caso.

| Symmetric Key Size (bits) | RSA and Diffie-Hellman Key Size (bits) | Elliptic Curve Key Size (bits) |
|---|---|---|
| 80 | 1024 | 160 |
| 112 | 2048 | 224 |
| 128 | 3072 | 256 |
| 192 | 7680 | 384 |
| 256 | 15360 | 521 |

Table 1: NIST Recommended Key Sizes

Figura 4 - Reprodução da tabela de comparação de tamanho de chaves criptográficas do relatório de recomendações de segurança do NIST por (BARKER; ROGINSKY, 2011)

Um exemplo de funcionamento do esquema de troca de chaves Diffie–Hellman é demonstrado na figura 5, utilizando números primos. Seguindo as seguintes etapas:

1. Inicia-se o algoritmo com um parâmetro comum, como um número primo **C**. Esse padrão geralmente é pré-definido ou determinado durante um processo de *handshake*.

2. Tanto Alice, quanto Bob, cada um gera um outro número primo, chamado de chave privada; **p1** para Alice e **p2** para Bob.

3. Faz-se um procedimento matemático, que para números primos pode ser uma multiplicação, já que a fatoração de números primos é trabalhosa e é base dessa criptografia exemplo, gerando as chamadas chaves públicas **C*p1** e **C*p2**.

4. As chaves então são trocadas entre o par Alice e Bob.

5. Faz-se novamente outra operação matemática sob as chaves públicas. No caso do exemplo, uma multiplicação pelas suas chaves privadas.

6. Nessa etapa, ambos possuem o mesmo resultado, **C*p1*p2**, um número fator de três primos. Esse número é um segredo comum que nenhum outro ponto terá acesso e pode ser utilizado para geração de chaves simétricas ou outros esquemas de segurança.

### 2.4.4 Gerador de números pseudoaleatórios do tipo Mersenne Twister

Algoritmos conhecidos como *Mersenne Twister* são algoritmos para geração de números pseudoaleatórios, isto é, geram sequências de números que se assimilam a uma sequência aleatória, mas, em verdade, seguem sempre uma sequência pré-definida baseada no seu estado de inicialização, o que é comumente chamado de *seed* ou semente. A origem do nome *Mersenne* se dá por serem geradores de números aleatórios que tem sequência de tamanho de um número primo de Mersenne $P$, da forma

$$P = 2^n - 1$$

.

Diversas soluções de mercado e até mesmo bibliotecas de programação de código aberto utilizam variantes do *Mersenne Twister* para geração de números aleatórios, cujo estado inicial geralmente é dado pelo horário. Diversas soluções de renome

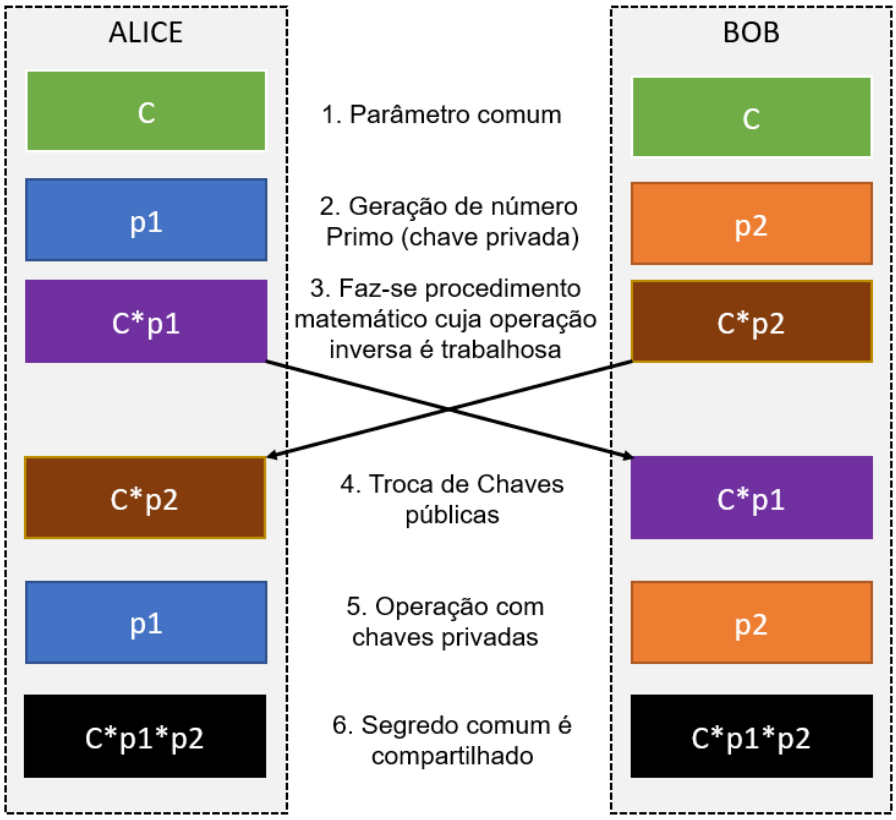



Figura 5 - Exemplo de troca de chaves o esquema Diffie–Hellman

utilizam essa tecnologia, como: Microsoft Excel como descrito por Melard (MÉLARD, 2014), GNU Multiple Precision Arithmetic Library conforme sua especificação (GNU, 2019), GNU Octave conforme sua especificação (OCTAVE, 2019), MATLAB conforme seu manual (MATLAB, 2019) e diversas outras soluções.

Algoritmos de *Mersenne Twister* dependem principalmente de seu estado inicial e isso pode ocupar bastante espaço de memória volátil para torná-lo mais aleatório, pois mais estados possíveis significa mais espaço necessário para armazenar uma semente maior. Para implementação especifica em dispositivos com restrições, Saito, desenvolveu um *Mersenne Twister* de baixo consumo de memória, o TinyMT (SAITO; MATSUMOTO, 2011), que tem apenas 128 *bits* de estado e que faz uso da sequência de comprimento do número primo de Mersenne $2^{127} - 1$. O TinyMT tem implementação enxuta em código, o que o torna ideal para utilização em sistemas com recursos restritos como os de grão fino e será parte da arquitetura da camada de segurança proposta, descrita a seguir.

## 2.5 Resumo

Nesta seção são reproduzidas as conclusões que o levantamento do estado da arte gerou, como discutido individualmente em cada seção.

- **Classificação de Dispositivos Quanto às Restrições -** Esta seção auxiliou na definição da nomenclatura utilizada no decorrer do texto, onde será utilizado o termo "grão fino"para classificação do dispositivo que tem restrições de recursos;
- **IEEE 802.15.4 -** A seção apresentou o protocolo de camada física e enlace utilizado para tecnologias de baixo consumo de memória e energia;
- **6LoWPAN -** Esta seção apresentou o padrão 6LoWPAN, que é um conjunto de algoritmos e definições para compressão de cabeçalho que permitem a intercomunicação de redes de dispositivos restritos com redes IPv6 convencionais.
- **Protocolos de aplicação para Redes de Sensores sem Fio -** Nesta seção foram apresentados diversos protocolos e algumas de suas propriedades que geram maior impacto na comunicação. Também foi demonstrado como o CoAP seria o protocolo mais adequado para teste e avaliação da camada de segurança proposta e desenvolvida.
- **Camada de segurança -** Nesta seção, apresentou-se quais são e o que fazem as camadas de segurança em redes de comunicação, além disso argumentou-se que DTLS é uma camada compacta para utilização em dispositivos restritos.
- **Camadas de segurança em Protocolos de Aplicação -** Nesta seção, apresentaram-se os esforços de diversos autores em se comprimir camadas de aplicação, como o CoAP, junto a camadas de segurança como o DTLS. Além disso, foi apresentado o TinyDTLS, que é a implementação parcial do DTLS especialmente desenvolvida para dispositivos restritos e, por isso, consequentemente escolhida para servir de comparação à camada CoEP desenvolvida neste trabalho.
- **Algoritmos de *Hashing* -** Nesta seção, demonstra-se como é a tecnologia de *hashing* e qual algoritmo - no caso, SHA-256 - foi escolhido para utilização na camada de segurança CoEP.

- **Algoritmos de Criptografia -** Nesta seção, demonstra-se como é a tecnologia de criptografia de dados para troca de mensagens arbitrárias seguras. O Algoritmo escolhido foi o AES-128, dada sua pequena chave e relativa boa segurança.

- **Esquema de troca de chaves Diffie–Hellman -** Nesta seção, explicou-se como é feita a troca de chaves assimétricas para geração de um segredo comum, que pode ser utilizado para geração de chaves simétricas, que é processo utilizado no CoEP.

- **Gerador de números pseudoaleatórios do tipo Mersenne Twister -** Finalmente, na última seção do estado da arte, apresentou-se o TinyMT, um gerador de números pseudoaleatórios do tipo Mersenne Twister que, por ter baixo consumo de memória, tem utilização no CoEP de gerador de números de autenticação da mensagem, como melhor detalhado nas seções posteriores.

# 3 COEP: ARQUITETURA PROPOSTA

Nesta seção, expõe-se inicialmente as necessidades levantadas para desenvolvimento de uma proposta com valor científico, tecnológico e no âmbito de utilização em aplicações reais. Além disso, essa seção também aborda como foram feitas metodologias de desenvolvimento e avaliação da proposta, bem como de extração de resultados. Em seguida, é exposta a proposta de camada de segurança CoEP dividida por seus elementos de forma a facilitar o entendimento. Cada elemento então tem seções de justificativa e metodologia como forma de mostrar como foram tomadas decisões que levaram ao desenvolvimento do elemento em questão.

## 3.1 Requisitos levantados para a camada

A partir das pesquisas realizadas até o momento, pode-se levantar especificações mínimas para a camada de segurança proposta de forma que possa substituir opções disponíveis na literatura sem perda de funcionalidades. O levantamento foi feito levando em conta as melhores características de todos protocolos estudados bem como suas estratégias de comunicação e segurança para utilização em dispositivos de grão fino.

Primeiramente, descrevem-se os requisitos mínimos de desenvolvimento do protocolo, divididos em requisitos funcionais, isto é, aqueles que levam a criação de serviços, e requisitos não funcionais:

### 3.1.1 Requisitos Não Funcionais

Requisitos que impactam na arquitetura da camada, mas são abstratos às suas funcionalidades.

#### 3.1.1.1 Simplificação da arquitetura de comunicação

Para esse requisito, considerou-se as seguintes necessidades:

- *Handshake* de conexão segura com número mínimo de *round-trips*;
- O *header* da camada deve ser compacto (em relação ao TinyDTLS) e garantir todas funcionalidades descritas nessa seção e garantir o mínimo de *overhead* possível respeitando as demais necessidades da camada.

#### 3.1.1.2 Menor utilização de memória

Para o último item, que considera menor consumo de memória volátil e não-volátil, a camada proposta:

- deve ser simples em complexidade, mas garantir todas funcionalidades necessárias para a utilização segura e imperceptível da camada de aplicação teste proposta (CoAP);
- deve ter código compilado (não-volátil) e dados (volátil) iguais ou menores à camada TinyDTLS.

### 3.1.2 Requisitos Funcionais

Requisitos que levam a criação de serviços na camada de segurança.

#### 3.1.2.1 Maior segurança

Para esse quesito, sem prejudicar o primeiro requisito de simplificação, as necessidades mínimas são:

- A camada deve implementar esquema de *handshake* de igual ou melhor segurança que a camada de comparação, TinyDTLS;
- A camada deve implementar autenticação, confidencialidade e integridade.

#### 3.1.2.2 Maior generalização

Além das características já definidas, ainda deseja-se acrescentar implementações que possam incentivar a utilização da camada de forma diversificada, inovativa ou mais simples para desenvolvedores:

- A camada deve oferecer possibilidade de *handshakes* personalizados;
- A camada deve oferecer possibilidade de utilizar múltiplas camadas de aplicação;
- A camada deve abstrair o máximo possível o *hardware*, para ter baixa dependência do mesmo;
- A camada deve abstrair o máximo suas funções, para que o trabalho de personalização e implementação em plataformas de *hardware* ou SOs seja mínimo;
- A camada deve possibilitar a coleta de informações e estatísticas de mensagens perdidas e outros;

Dessa forma, concluem-se as especificações mínimas da camada; como descrito em seções futuras, mais funcionalidades foram adicionadas durante desenvolvimento, porém ressalta-se que se deseja atingir todos requisitos supracitados.

## 3.2 Metodologias para pesquisa, desenvolvimento e avaliação

Uma vez definidos os requisitos, é necessário definir como será a metodologia de etapas para pesquisa, desenvolvimento e avaliação. Esta seção descreve estas metodologias.

### 3.2.1 Metodologia de Pesquisa

A metodologia de pesquisa envolve o levantamento de estado da arte em redes de sensores sem fio, levando sempre em conta o impacto das publicações, através de seu índice de impacto ou de importância dos congressos ou *journals* nas quais as pesquisas levantadas foram publicadas. Por fim, o levantamento também leva em conta a popularidade do assunto em questão, que no caso de protocolos, por exemplo, pode ser evidenciado por quantidade de seguidores do código publicado em plataforma Git. Após o levantamento das tecnologias, é feito mapeamento das publicações e seleciona-se aquelas que acrescentam informações mais relevantes para redes de sensores sem fio compostas de dispositivos restritos.

Para o desenvolvimento de uma nova camada de segurança para camadas de aplicação, conhecer e estudar outros protocolos similares, de camadas superiores ou seus mecanismos é de grande importância para efetivamente propor um esquema de segurança que possa ser melhor de acordo com as dificuldades e desafios encontrados nos protocolos padrões. Por isso, na seção de estado da arte, foram apresentados diversos protocolos de aplicação, camadas de segurança mais comuns e como todos funcionam, afim de que eles possam ser utilizados como base para propor novos mecanismos e uma nova arquitetura. Além disso, leva-se em conta outros trabalhos de redes de sensores sem fio que levantem pontos importantes para o desenvolvimento de uma camada de segurança melhor e mais compacta.

Para a determinação das necessidades e requisitos do protocolo, foi utilizada parceria com empresa privada que fez utilização, avaliação e forneceu opiniões de necessidades para que esta camada de segurança possa atender aplicações reais.

### 3.2.2 Metodologia de Avaliação

Para avaliação da qualidade e funcionamento da camada de segurança, todas demais camadas da pilha deverão ser mantidas as mesmas entre os testes e devem ter implementação especializada para dispositivos de grão fino. Além disso, a camada de aplicação deverá refletir uma camada de aplicação comum em redes de sensores sem fio. Para isso, o *testbed* foi definido e apresenta-se na tabela 4, com cada item justificando sua escolha.

No *testbed*, existem duas camadas de segurança definidas: TinyDTLS e CoEP. Isso porque, para avaliação e análise de resultados, foi realizada avaliação comparativa de dois *testbeds*, um com a camada de segurança proposta CoEP e um com a camada de segurança padrão TinyDTLS. Ainda para o *testbed* proposto, foram analisadas as seguintes características, comparativamente, para determinação dos resultados e conclusões da pesquisa e desenvolvimento:

#### 3.2.2.1 Avaliação comparativa: Segurança

Para comparação de segurança, foram avaliados:

| Camada | Item | Justificativa |
|---|---|---|
| Hardware | Texas Instruments LAUNCHPAD CC2650 | Trata-se de um dispositivo de grão fino, ou classe 1 que é especificamente vendido como dispositivo para redes de sensores sem fio e conectividade. |
| OS | Contiki-NG | O Contiki-NG é um sistema operacional embarcado para dispositivos do tipo grão fino que já implementa diversas funcionalidades de rede e portanto é ideal para desenvolvimento rápido e maduro das necessidades propostas. |
| Física/Enlace | 802.15.4 @ 2.4GHz | O padrão 802.15.4 foi especificamente desenvolvido para aplicações de baixo consumo e enxutas, portanto é ideal para realização de um *testbed* próximo a uma aplicação real. |
| Roteamento | RPL | RPL é um protocolo de roteamento *mesh* padrão em soluções de redes de sensores sem fio e, portanto, seu funcionamento não deverá ser influenciado pela camada de segurança proposta. |
| Rede | UDP | Tratando-se de aplicações restritas e buscando ter menor consumo de memória, o UDP é um padrão cuja implementação é simples em relação a outra opção TCP e, portanto, foi escolhido para o *testbed*. |
| Segurança | CoEP e TinyDTLS | Visto que no caso do TLS, o padrão depende da camada de rede (UDP, acima), o TinyDTLS foi escolhido para ser protocolo de comparação por ter sido especialmente criado e desenvolvido para ser camada de segurança em dispositivos restritos, dessa forma, sendo ideal para efeitos de comparação. |
| Aplicação | CoAP | O CoAP é um protocolo de aplicação para dispositivos restritos que realiza atividades comuns em aplicações de redes de sensores sem fio, portanto caracterizando-se ideal para realização de testes e avaliações, como exposto na seção de estado da arte. |

Tabela 4 - Tabela de definação do *testbed*.

- A probabilidade de descriptografia de um pacote qualquer;
- A probabilidade de falsificação de um pacote qualquer;
- A probabilidade de intrusão de dispositivos mal-intencionados em uma rede.

#### 3.2.2.2 Avaliação comparativa: Complexidade e funcionalidades

No caso de comparação de complexidade - no sentido de dificuldade de implementação - do código e funcionalidades, isto é, de comparação de quantidade de elementos necessários para os objetos, onde elementos são outros algorítmos e funções que façam parte do objeto; a metodologia utilizada será de comparação direta de tamanho de código. Mesmo sendo uma comparação objetiva neste caso, também será feita comparação subjetiva, visto que o valor intrínseco das funcionalidades pode ser de maior importância que uma comparação objetiva.

#### 3.2.2.3 Tamanho em memória volátil e não-volátil

Utilizando a plataforma de *testbed* determinada na tabela 4, foi compilado código das aplicações com TinyDTLS e CoEP para comparação do consumo de memória volátil e não-volátil utilizando a ferramenta *fpvgcc*, que é distribuída através de repositório *git* em <https://github.com/chintal/fpv-gcc>, e realiza cálculo tanto do tamanho total de compilados gerados, quando dividindo-os ainda por módulos, o que foi importante para ressaltar as diferenças entre duas implementações TinyDTLS e CoEP. Para que isso seja possível, foi processado o arquivo *MAP* gerado pela compilação.

A ferramenta *fpvgcc* quando aplicada em um arquivo binário ou do tipo *MAP*, retorna informações relevantes como quando de cada seção de recursos cada módulo utiliza, como exemplificado na imagem .

No caso, *FLASH* refere-se a código de execução, como comandos para o processador, portanto ocupa apenas espaço em memória não-volátil. Já *SRAM* refere-se a dados estáticos ou constantes, portanto que tem inicialização na memória volátil. Outros setores expostos também fazem parte da memória não-volátil e existem por questões funcionais do fabricante, mas são imutáveis entre as comparações.

| FILE | VEC | FLASH | SRAM | FLASH_CCFG | *default* | TOTAL |
|---|---|---|---|---|---|---|
| dtls.o | | 10093 | | | | 10093 |
| sicslowpan.o | | 5319 | 1773 | | | 7092 |
| rijndael.o | | 6768 | | | | 6768 |
| ieee-mode.o | | 3316 | 976 | | | 4292 |
| ecc.o | | 3740 | | | | 3740 |
| uip6.o | | 1912 | 1577 | | | 3489 |
| coap.o | | 2966 | 7 | | | 2973 |
| strformat.o | | 2510 | | | | 2510 |
| rpl-dag.o | | 2022 | 352 | | | 2374 |
| v7-m\libc.a | | 1212 | 1068 | | | 2280 |
| dtls-crypto.o | | 1760 | 496 | | | 2256 |
| rpl-icmp6.o | | 2108 | 48 | | | 2156 |
| rf-core.o | | 2033 | 81 | | | 2114 |
| uip-ds6.o | | 1456 | 400 | | | 1856 |
| coap-engine.o | | 1401 | 353 | | | 1754 |
| queuebuf.o | | 212 | 1448 | | | 1660 |
| rpl-timers.o | | 1552 | 64 | | | 1616 |
| dtls-ccm.o | | 1536 | 4 | | | 1540 |
| netq.o | | 321 | 1140 | | | 1461 |

Figura 6 - Resultado do comando *fpvgcc <arquivo.map> –sar*

## 3.3 Aplicação da Metodologia de Avaliação

Conforme descrito na seção 3.2.2, a metodologia de avaliação será utilizada para análise comparativa entre a camada CoEP e a camada TinyDTLS. Seguindo ainda a metodologia, serão comparados os seguintes itens, seguindo agora, as seguintes implementações de cálculos para cada avaliação dos resultados:

### 3.3.1 Segurança

Serão comparados:

**A probabilidade de descriptografia de um pacote qualquer** Essa probabilidade deve ser a mesma de se descriptografar uma mensagem confidencial, isto é, que foi criptografada por algum algoritmo de chave simétrica. Para o caso de avaliação, a metodologia refere-se à probabilidade de se adivinhar a chave criptográfica.

**A probabilidade de falsificação de um pacote qualquer** Essa probabilidade depende da própria probabilidade de descriptografia acima acrescentada de outras dificuldades que atacantes possam ter para personificar um dispositivo legítimo. Deve-se utilizar a probabilidade de um acerto no universo de possibilidades de criptografias

válidas.

**A probabilidade de intrusão de dispositivos mal-intencionados em uma rede**
Essa probabilidade é calculada através de quantas tentativas um dispositivo não autorizado a participar de uma rede deveria exercer antes de conseguir ter acesso à mesma.

### 3.3.2 Complexidade e funcionalidades

Será feita comparação objetiva utilizando tamanho total de código entre as duas implementações em CoAP: com TinyDTLS e com CoEP utilizando ferramenta de contagem de *bytes* do próprio sistema operacional. Além disso, deve-se levar em conta as bibliotecas que são dependências de cada solução. Também será feita comparação subjetiva das funcionalidades disponíveis, no formato de discussão.

### 3.3.3 Tamanho em memória volátil e não-volátil

Serão comparados tamanhos dos arquivos compilados finais em memória volátil e não-volátil, bem como tamanho das dependências de cada programa utilizando ferramenta *fpvgcc* descrita.

## 3.4 Materiais para avaliação e desenvolvimento da proposta

Nessa seção apresenta-se qual o ambiente que foi utilizado no desenvolvimento da proposta, bem como os materiais utilizados para o mesmo.

### 3.4.1 Ambiente de Programação de Desenvolvimento

O ambiente de programação utilizado foi o sistema operacional *Windows 10*. Nesse sistema, foi instalado um conjunto de ferramentas de compilação para ARM distribuída pela própria empresa ARM sob nome de *gcc-arm-none-eabi-8-2018-q4-major*, que inclui as versões mais recentes dos compiladores para plataforma ARM e outras ferramentas que fazem parte do processo de compilação.

Além disso, para compilação do Contiki-NG, foi necessária a utilização do *Cygwin*, que dispõe de inúmeras ferramentas para simular um ambiente *POSIX*, como uma distribuição de *Linux*, o que permite execução de comandos de compilação ou de outras ferramentas como a chamada *sed*, que é utilizada no processo de compilação.

Foi escolhido ambiente *Windows* principalmente pela facilidade de compilação e depuração utilizando a ferramenta gráfica *Microsoft Visual Studio Community*, a versão gratuita do produto *Visual Studio* que foi de grande utilidade para testes durante o desenvolvimento, pois seguiu metodologia de desenvolvimento orientado a testes.

Para programação dos equipamentos foi utilizado o software *Texas instruments SmartRF Programmer 2*, que tem funcionamento apenas no SO *Windows* e tem funcionalidades de programação da memória do LAUNCHPAD CC2650 (parte do *testbed*), extração de informações importantes e até mesmo depuração em *hardware*.

### 3.4.2 Materiais utilizados no desenvolvimento da proposta

Para desenvolvimento e implementação da proposta descrita nas próximas seções, conforme metodologia proposta, serão necessários alguns materiais para esse desenvolvimento, destacados a seguir:

- 2 plataformas de desenvolvimento LAUNCHPAPD CC2650 com SoC *ultra low power* CC2650 especificadas na tabela 4;
- IDE de desenvolvimento de software *Microsoft Visual Studio Community*, bem como compiladores para Linux ou Windows descritos anteriormente;
- Outros programas como: *Texas instruments SmartRF Programmer 2* para gravação das plataformas; *git* para controle de versões e outros.

Mais detalhes das ferramentas e execução dos processos de avaliação estão descritos na seção 5, de resultados.

## 3.5 Especificação da Arquitetura

Baseado nos requisitos mínimos, no estado da arte, na metodologia proposta e em reuniões com parceiros de mercado e tecnologia, foi levantado a estruturação e funcionalidades da camada de segurança, descritos em tópicos a seguir.

### 3.5.1 Funcionalidades

#### 3.5.1.1 Definição

A camada de segurança atuará de forma unificada, como um provedor de mensagens seguras. Ou seja, atuará como um serviço de segurança. A figura 7, demonstra como funciona o mecanismo, descrito a seguir.

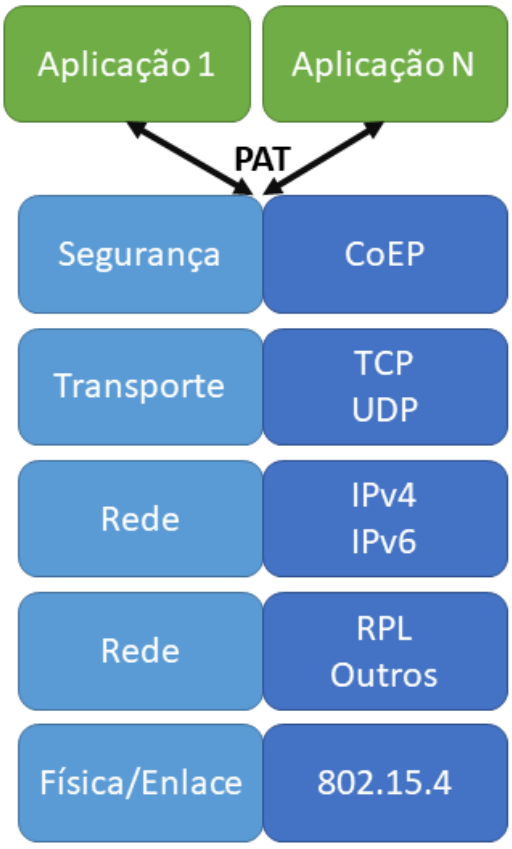



Figura 7 - Arquitetura da camada de segurança

A camada de segurança passa a atuar como um serviço, que funciona em porta específica, dada como padrão 49380 ou 0xC0E2. Para que a camada funcione de forma imperceptível, a mesma realizará um processo de PAT (*Port Address Transla-*

*tion*) para as portas das camadas de aplicação. Isto é, caso o CoAP que funciona em porta 80 deseje enviar um pacote para outro host contendo CoAP, o PAT do CoEP converterá essa porta para a porta padrão CoEP 49380, mas manterá a informação de qual porta utilizar no recebimento. Dessa forma, quando o CoEP de outro host receber a informação na porta 49380, está será retransmitida para a porta 80 internamente e de forma a respeitar a segurança.

Para realizar essa conversão, o CoEP utiliza o que chama de *Module*. Um *Module* faz a identificação de um ou mais *Module ID* (que podem ser portas de aplicação ou até mesmo identificadores quaisquer) para encaminhar a mensagem recebida para as funções nas camadas de aplicação que recebem os pacotes. Um *Module*, que no CoEP é definido como o tipo *coep_module_t* tem a seguinte estrutura, descrita em linguagem C:

```
typedef uint8_t coep_event_t;
typedef int(*message_processor_t)(COEP_IP_TYPE * from, uint8_t msgid,
                                  uint8_t * buffer, int maxlength);
typedef void(*event_processor_t)(coep_event_t event);

typedef struct coep_module_t {
    message_processor_t message_processor;
    event_processor_t event_processor;
    uint8_t range_min;
    uint8_t range_max;
} coep_module_t;
```

Ou seja, um *Module* tem um intervalo de IDs que pode processar; um processador de mensagens que é um ponteiro para função na camada de aplicação que deverá receber o pacote e retornar o número de *bytes* consumidos; e um ponteiro para uma função para processar os chamados eventos, que são mensagens do tipo ACK, NACK, *timeout* e desconexão.

Além da definição de *Module*, outra definição importante para a camada de segurança e comunicação é uma *conexão*. Uma *conexão* no CoEP é única entre dois endereços, porém qualquer um dos endereços desse par pode ser um endereço de

*unicast*, *multicast* ou *broadcast*. Dessa forma, o CoEP não apenas provê conexões seguras entre pontos, mas entre multipontos também. Pela complexidade da estrutura de dados de conexão do CoEP, apresenta-se abaixo apenas uma descrição dos seus elementos, já que o código em C não seria de entendimento trivial do leitor. Uma *conexão* no CoEP armazena as seguintes informações:

- *Socket* - endereçamento IP e porta - do outro par da conexão - dessa forma cada elemento da conexão tem o endereço de seu par;
- Chave simétrica de criptografia de confidencialidade;
- Chave de criptografia de autenticidade - criada aqui especificamente para a camada CoEP -, ou como será visto posteriormente, é equivalente aos estados para geradores de números pseudoaleatórios de autenticação;
- Lista de módulos que estão aguardando algum tipo de resposta;
- Detalhes da conexão, como status, número de falhas e tipo de conexão (ponto ou multiponto);
- Outros mecanismos internos para prover as funcionalidades descritas, como temporizadores.

#### 3.5.1.2 Justificativa

**Em relação à segurança** Essa arquitetura traz uma vantagem que é a centralização da segurança, o que torna o processo de desenvolvimento mais rápido e seguro. É vantagem, também em questão de segurança, pois garante que qualquer solução que utilize CoEP está segura, já que não é possível se comunicar com um dispositivo seguro com CoEP sem utilização do CoEP.

**Em relação à complexidade e ao tamanho em memória volátil e não-volátil** A escolha de trabalhar com o CoEP como um provedor de serviços de segurança que funciona em uma porta específica permite o uso de diversos protocolos de aplicação funcionando paralelamente sobre o mesmo esquema de segurança. Em questões de utilização de recursos, isso torna possível ter mais de um protocolo de aplicação em

um dispositivo de grão fino. Isso seria um grande desafio de ser implementado nesses dispositivos no caso outras soluções como com o TinyDTLS.

#### 3.5.1.3 Metodologia

Para o funcionamento dessa arquitetura, será necessário a criação de um PAT, como descrito anteriormente. Por convenção, o PAT encaminha mensagens a protocolos através de uma estrutura *Module*. Assim, um protocolo de aplicação que use CoEP pode ter um *Module* definido com as regras de PAT. Para o CoEP identificar qual o destino PAT de mensagens recebidas, basta utilizar um campo do seu *header* para a identificação chamada de *Module ID*, que é o identificador de *Module* ou porta a ser utilizada no PAT. Dessa forma, define-se o primeiro campo necessário no *header* do CoEP. Esse campo também é apresentado na figura 8.

### 3.5.2 Arquitetura de comunicação

Esta seção descreve o escopo da camada CoEP e como funciona o mecanismo geral de comunicação e transferência de informações no CoEP. Outros detalhes de cada processo - confidencialidade, autenticidade e integridade - são descritos separadamente, em seus respectivos tópicos.

**Definições e Restrições do CoEP**

- O CoEP suporta fragmentação. Qualquer fragmentação deve ser feita em camadas inferiores. Isso porque camadas, como 6LoWPAN, por exemplo, já realizam essa atividade e deseja-se evitar redundâncias de funcionalidades.
- O maior tamanho de um *payload* CoEP é de 128 *bytes*. Isso é padrão para garantir que até dispositivos com menos capacidade possam receber e processar pacotes CoEP, além de servir como desincentivo a utilização de algoritmos criptográficos ou protocolos de alto consumo, que não devem ser espaço em aplicações em dispositivos restritos.
- O pacote padrão de comunicação do CoEP é dado na figura 8. Seus campos são:

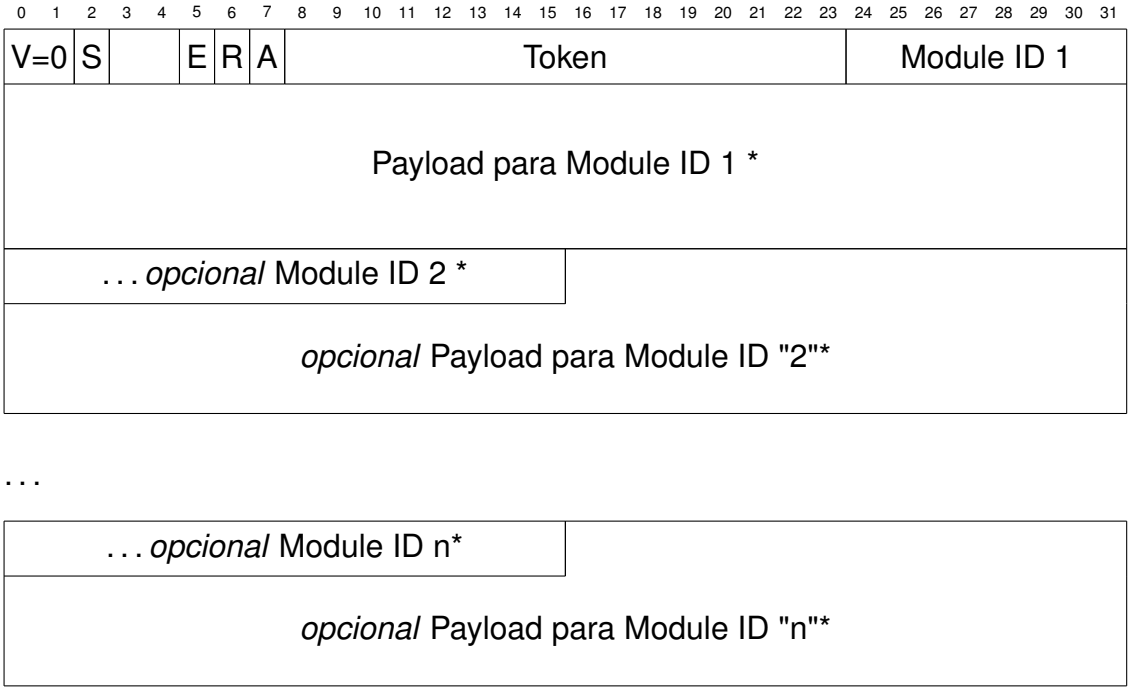



Figura 8 - Estruturação do pacote CoEP

- **V**: Versão do protocolo, atualmente 0x0, limitando-se a quatro versões do *header* da camada.
- **S**: Identificador de mensagem de sistema. Esse identificador serve para diferenciar as mensagens comuns de mensagens de sistema e *handshake* utilizando valor 1 nesse campo. Uma mensagem comum entre camadas de aplicação possui valor 0 nesse campo.
- **E**: Identificação se pacote está criptografado com chave de confidencialidade, o mecanismo de utilização de chaves está definido nas próximas seções.
- **R**: Identificador de requisição de resposta. Quando com valor 1 o CoEP espera alguma outra mensagem em resposta a essa dentro de um tempo de *timeout* predeterminado. Esse campo serve principalmente para identificar falha na entrega de mensagens que esperam respostas. A resposta aqui pode ser qualquer outro pacote vindo do *Socket* conectado.
- **A**: Identificador que mensagem necessita de resposta do tipo ACK quando com valor 1. A própria camada CoEP implementa *timeout* e resposta com ACK quando necessário, expandindo as funcionalidades do UDP. Sempre que esse campo for 1, o campo R também terá valor 1. Esse campo deter-

mina que a resposta (de R igual a 1) deve ser um pacote ACK.

- **Token**: Um *token* de 16 *bits* de autenticação, que se modifica para cada pacote enviado. A mensagem só é recebida caso o *token* seja válido. O funcionamento e utilização do *token* está descrito na seção 3.5.5.
- **Itens marcados com ***: Campos criptografados com chave de confidencialidade quando o campo E tiver valor 1.
- **Module ID**: Campo que carrega informação para realização do PAT de portas entre protocolos, através de estruturas *Module*, conforme descrito anteriormente. Esse campo tem utilização especial caso o campo "S"seja "1". Neste caso, a figura 9 demonstra a estrutura especial do *payload* e os possíveis valores de *Module ID* nesse caso.

- O *payload* de um pacote com S=1, segue a seguinte estrutura:

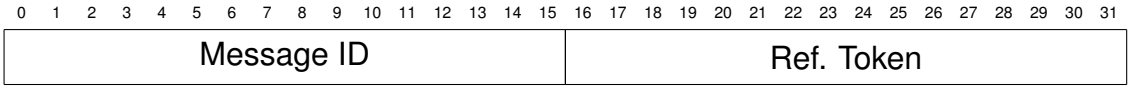



Figura 9 - Estruturação do payload CoEP quando S=1

- **Message ID**: Ao invés de referenciar um módulo, quando S=1, o campo *Module ID* torna-se uma referência ao tipo de mensagem, chamado de *Message ID*. Por definição as seguintes mensagens estão disponíveis: 0x0 para iniciar uma conexão através da primeira mensagem de um *handshake*; 0x1 à 0xEF para utilização interna do *handshake*; 0xFC para identificar uma desconexão; 0xFD para uso interno apenas, para aviso de *token* inválido (esse *Message ID* não é transmitido); 0xFE para NACK; e 0xFF para ACK. Os valores de 0xF0 à 0xFB são reservados para uso futuro.
- **Ref. Token**: Contém *token* da mensagem recebida que teve como resultado o *Message ID* identificado no campo anterior.

### 3.5.3 Arquitetura de Segurança: Visão Geral

A segurança é o ponto principal do funcionamento do CoEP. Em segurança, serão disponibilizadas as funcionalidades descritas a seguir:

- *Handshake* padrão com possibilidade de personalização;
- Autenticidade da mensagem;
- Confidencialidade da mensagem ponto-a-ponto e multipont -a multiponto bem como armazenamento de chaves;
- Integridade da mensagem.

A definição, justificativa e metodologia de cada um desses recursos bem como demais recursos provenientes desses estão descritos nas próximas respectivas seções.

### 3.5.4 Arquitetura de Segurança: *Handshake*

#### 3.5.4.1 Definição

Um dos pontos importantes no desenvolvimento de um protocolo mais enxuto é ter menor troca de mensagens. Isso é ainda mais importante considerando-se que a solução da literatura, o TinyDTLS, possui um processo de *handshake* complexo e delongado, derivado do DTLS.

Dessa forma, o processo de *handshake* padrão do CoEP foi arquitetado como na imagem 10 e explicado passo-a-passo em seguida. Para o *handshake* padrão, foi definido o algoritmo de chave pública/privada como sendo de Curvas Elípticas do Tipo *secp256r1*.

1. Inicialmente, deve-se enviar informações do esquema de criptografia a ser utilizado, seguindo estrutura específica descrita na figura 11. Na mesma mensagem, é enviada chave pública - do tamanho que for necessário - do cliente para uma nova conexão.
2. O servidor então entra no passo 2, onde valida a capacidade de trabalhar com o esquema de criptografia proposto. Caso não possa trabalhar com o esquema proposto, é enviada resposta de desconexão, no caso trata-se de um pacote CoEP com identificador de mensagem de sistema **S=1** e *Message ID*=0xFC, conforme descrito na figura 9. Caso possa trabalhar no esquema proposto, o servidor então passa pela etapa de geração, onde gera uma chave pública e

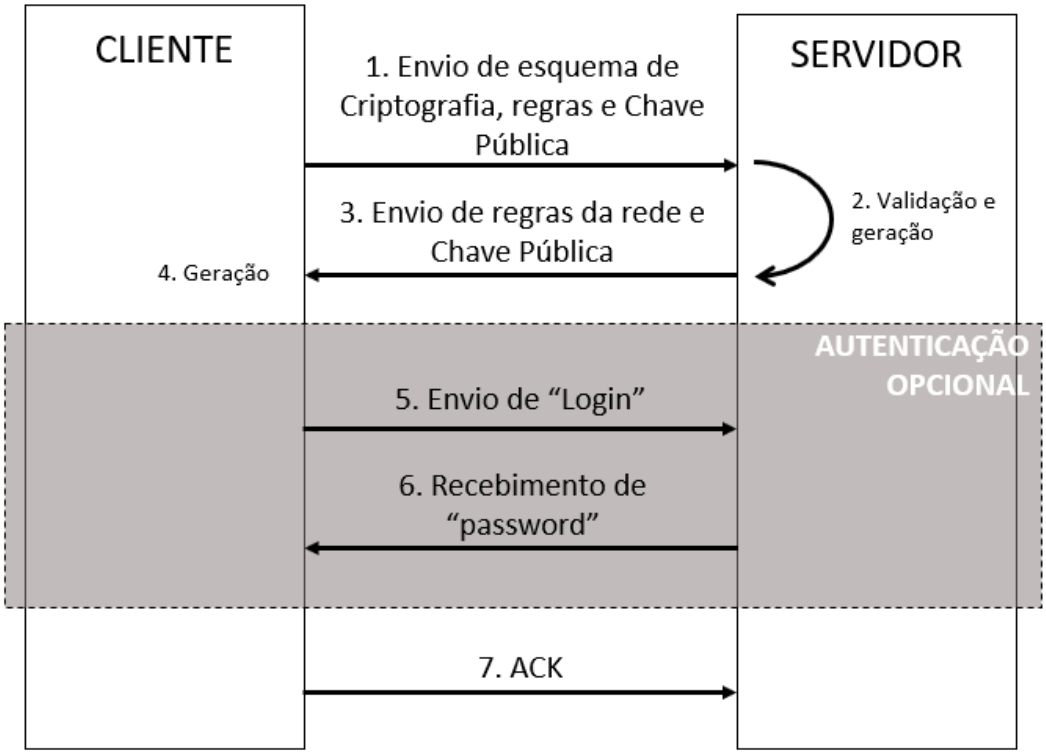



Figura 10 - Diagrama em ordem temporal do *handshake* da camada de segurança

uma chave privada. Ao final desse processo, com sua chave privada e a chave pública do cliente, o servidor já consegue gerar um segredo comum utilizando algoritmo do tipo Diffie–Hellman.

3. O servidor, no passo 3, então envia no mesmo pacote as regras da rede e sua própria chave pública, seguindo estrutura específica descrita na figura 11.

4. No passo 4, o cliente recebe as regras da rede e caso esteja de acordo com as regras ele utiliza a chave pública recebida e sua chave privada gerada para geração do mesmo segredo do servidor, assim como no servidor, utilizando algoritmo do tipo Diffie–Hellman. Neste passo, ambos, cliente e servidor, já tem um segredo comum que será utilizado para geração de duas chaves distintas: uma chave de 128 *bits* de confidencialidade e uma chave de 128 *bits* de autenticidade. A derivação dessas chaves se dá utilizando cada metade do segredo pela função de *hashing* SHA-256. A primeira metade gera a chave de criptografia e a segunda metade gera a chave de autenticidade. Em pseudocódigo:

```
segredo[tamanho_segredo];
chave de confidencialidade = SHA256(&segredo[0], tamanho_segredo/2);
```

```
chave de autenticidade = SHA256(&segredo[tamanho_segredo/2],
                                tamanho_segredo/2);
```

5. A partir de agora, todas transações de informação são criptografas usando a chave *chave de confidencialidade*, através de algoritmo padrão AES-128. Além disso, opcionalmente, dependendo do esquema de criptografia e regras da rede, será necessário ainda autenticar um cliente na rede. Para isso, como ambos, cliente e servidor, necessitam confirmar a autenticidade um do outro, o cliente inicia o processo de identificação enviando seu *login*, que é uma chave de 128 *bits* já pré-programada no cliente, seguindo a estrutura de pacote demonstrada na figura 12.

6. O servidor recebe informações de *login* do cliente e então pode buscar por uma segunda chave chamada de *password*. O servidor então envia, mais uma vez criptografa pela chave *chave de confidencialidade*, a chave *password* de 128 *bits* segundo a mesma estrutura de dados da figura 12.

7. Na sétima etapa, caso o cliente tenha realizado autenticação com *login* e *password*, ele deve comparar o *password* recebido com o valor pré-programado. Caso a autenticação seja bem sucedida ou não necessária, o cliente pode finalizar a troca de mensagens com uma mensagem do tipo ACK, confirmando que está de acordo para entrar na rede. Ao receber a mensagem, o servidor também entende que o cliente agora faz parte da rede. Nesse momento uma conexão, como definida na seção 3.5.2 é criada. Nesse processo, também é gerado o estado dos *tokens*, melhor descrito posteriormente, na seção 3.5.5.

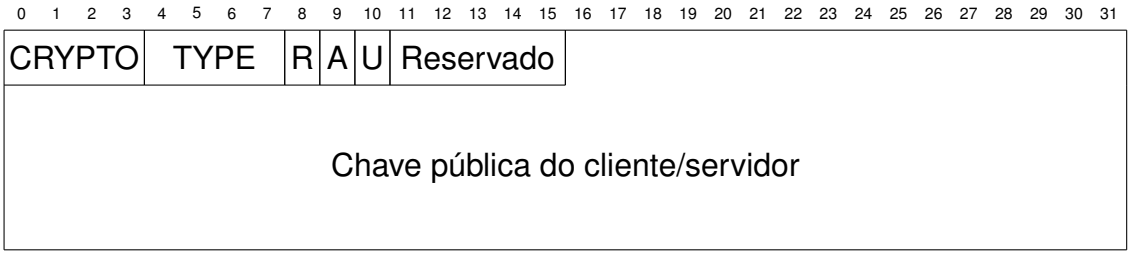



Figura 11 - Estruturação do pacote de *handshake* do CoEP, que deve ainda estar dentro de um pacote CoEP comum

**Mensagem de inicialização de *handshake*** A figura 11 descreve o pacote de troca de chaves e regras de rede, com seus campos:

- **CRYPTO**: Define algoritmo de criptografia. Atualmente suporta apenas 0x1 para Curvas Elípticas.
- **TYPE**: Identificador de tipo de criptografia no algoritmo escolhido. Atualmente suporta apenas 0x1 para curvas elípticas do tipo 256r1.
- **R**: 1 caso o ponto que envia a mensagem é um coordenador de rede (pode realizar autenticações).
- **A**: 1 caso o ponto apenas aceite entrar em rede com autenticação obrigatória.
- **U**: 1 caso o ponto aceite participar de redes que podem transmitir mensagens não criptografadas com chave de confidencialidade.
- **Reservado**: 5 *bits* reservados para determinação de regras em próximas versões do protocolo.

**Mensagem de *handshake* de autenticação** A figura 12 descreve o pacote de autenticação, com seus campos:


Figura 12 - Pacote de autenticação dentro do *handshake*. Total de 16 *bytes*.

- **Login/Password***: Identificação de 128 *bits*, criptografada.

#### 3.5.4.2 Justificativa

**Em relação à segurança** utilizou-se a mesma metodologia que no protocolo DTLS e da sua implementação TinyDTLS para esquema de chave pública privada, de forma a garantir mesma ou maior segurança. Ainda existe a utilização combinada de chave pública/privada com autenticação, o que em questão de segurança torna-se ainda melhor que o que é proposto no padrão DTLS.

**Em relação à complexidade e ao tamanho em memória volátil e não-volátil** Apesar de existir um processo a mais no *handshake*, chamado de autenticação, ele é equivalente a implementação do algoritmo PSK que existe no *handshake* do DTLS e do TinyDTLS. Dessa forma, mesmo com autenticação, ainda se manteve a complexidade equivalente.

**Em relação a aplicações -** A utilização em aplicações reais foi o principal determinante da necessidade de um processo de autenticação. A utilização do processo opcional faz garantia que o cliente e servidor se conhecem, uma vez que o servidor deve ter informações de *login* e *password* do cliente. Com a troca dessas informações incompletas entre as partes, a única forma de uma autenticação ocorrer é se ambas partes tiverem as duas informações, dessa forma, evita que qualquer dispositivo possa se comunicar com qualquer dispositivo não autorizado previamente. Isso cria, a nível de camada de segurança, uma rede protegida, como ocorre em uma rede WiFi sem perda da segurança que as chaves públicas/privadas trazem.

**Em relação à simplificação da comunicação -** O *handshake* possui pequena quantidade de troca de mensagens, sendo igual ou melhor ao protocolo TinyDTLS, dependendo das regras utilizadas. Além disso, as mensagens em si têm menor tamanho, dessa forma, tornando o CoEP um pouco mais econômico em tamanho de mensagens, quantidade de mensagens e consequentemente em recursos e energia.

#### 3.5.4.3 Metodologia

O *handshake* do CoEP foi desenvolvido de forma a ser mais compacto em memória e tamanho de pacote para o mesmo nível de segurança que outros protocolos, como o próprio TinyDTLS. O *handshake* do CoEP foi baseado no TinyDTLS, porém reduzido e adaptado para melhor atender as necessidades de dispositivos de baixa capacidade. Além disso, como supracitado, foi adicionada funcionalidade de autenticação devido a necessidade exposta pelas aplicações reais. Em conversas com necessidades de segurança junto a empresa que trabalha na disponibilização de conectividade desse tipo, um problema ressaltado é a falta de controle de quem pode

ou não participar da rede e proteção contra dispositivos maliciosos que tentem personificar dispositivos legítimos, o que foi sanado com a implementação da autenticação sem perda da segurança provida por chaves assimétricas.

Apesar de padrão no CoEP, a camada permite a personalização de *handshake*, seja por extensão dos padrões definidos ou por total substituição do padrão. Para fazer isso, basta definir duas funções:

```
bool coep_connect_to(COEP_IP_TYPE * ip, uint8_t flags);
bool coep_disconnect_from(COEP_IP_TYPE * ip);
```

E criar um *Module* para processar as mensagens de *handshake* a ser declarado como:

```
extern coep_module_t connector;
```

### 3.5.5 Arquitetura de Segurança: Autenticidade

#### 3.5.5.1 Definição

A autenticidade de uma mensagem é validada pelo chamado *token*, exibido no pacote CoEP na figura 8. O *token* é um número de 16 *bits* gerado para cada pacote transmitido através de um gerador de números pseudoaleatórios, especificamente o TinyMT. Esse número de *token* pseudoaleatório será gerado tanto no cliente quando envia mensagems, quanto no servidor que as recebe, dessa forma, uma mensagem é apenas aceita caso o *token* recebido pela rede e o *token* gerado localmente sejam iguais, visto que ambos terão a mesma semente de geração, que é derivada da chave de autenticação gerada na etapa de *handshake*. São utilizados *token* distintos de envio e recebimento, de forma a evitar falsificações de pacotes. A figura 13 demonstra o funcionamento do sistema de *tokens*, descrito abaixo.

1. Durante o *handshake*, a primeira mensagem que é de troca de chaves, o *token* é obrigatoriamente 0.
2. Na resposta no servidor, enquanto ambos não possuem segredos definidos, o *token* é obrigatoriamente 0.

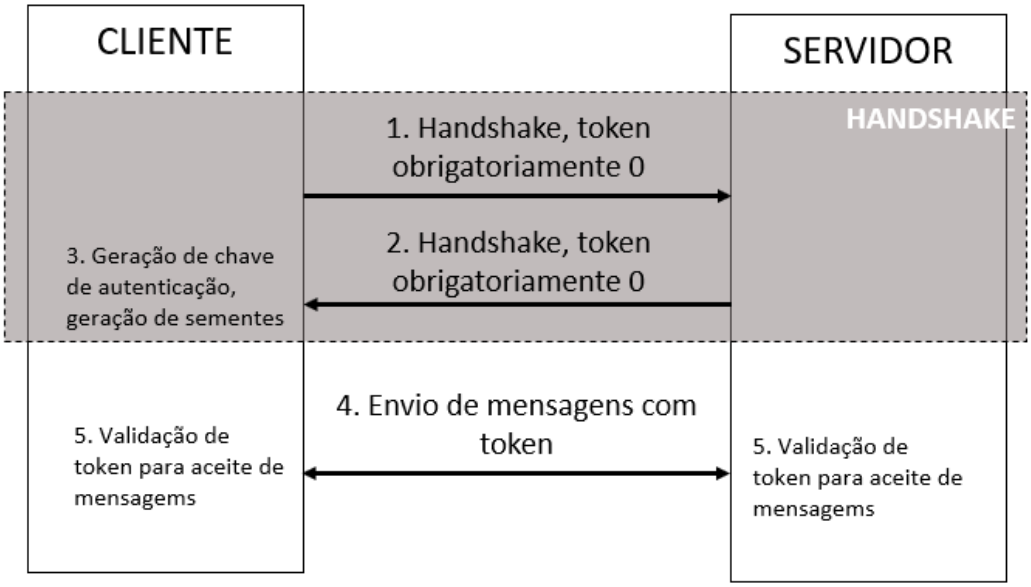



Figura 13 - Arquitetura de funcionamento do sistema *token*

3. Logo após a geração de segredos, como descrito na seção 3.5.4, ocorre a geração de *chave de confidencialidade* e *chave de autenticidade*. A *chave de autenticidade* gerada é então utilizada para servir como semente para o gerador de números pseudoaleatórios. Ocorre a geração de duas sementes de 128 *bits* de estado, da seguinte forma, seguindo pseudocódigo, para o ponto que requisitou a conexão (cliente):

```
uint32_t chave_de_autenticidade[8];
tinymt32_init_by_array(&receive_token, &chave_de_autenticidade[0], 4);
tinymt32_init_by_array(&send_token, &chave_de_autenticidade[4], 4);
```

E invertendo-se *receive_token* e *send_token* para o caso de quem recebe o pedido de conexão (servidor):

```
uint32_t chave_de_autenticidade[8];
tinymt32_init_by_array(&receive_token, &chave_de_autenticidade[4], 4);
tinymt32_init_by_array(&send_token, &chave_de_autenticidade[0], 4);
```

São dois geradores de números aleatórios porque os *tokens* de recebimento e envio deverão ter ordem distinta de geração, para evitar, por exemplo, utilizar um

*token* gerado num envio de cliente para falsificar um envio posterior pelo servidor (caso a sequência fosse a mesma).

4. Nessa etapa, como tanto *chave de confidencialidade* quanto *chave de autenticidade* já estão geradas e o gerador de *tokens* também está populado, mesmo que ainda esteja ocorrendo *handshake* de autenticação (processo de *login*), todas mensagens deverão ser criptografadas e populadas com os 16 *bits* menos significativos do *token* de 32 *bits* gerado pelo gerador de números pseudoaleatórios TinyMT.

5. Nesta etapa, todas as mensagens enviadas deverão ser populadas com os 16 *bits* menos significativos do *token* de 32 *bits* gerado pelo gerador de números pseudoaleatórios de envio. Em caso de retransmissão, o mesmo *token* não poderá ser utilizado e deverá ser gerado um próximo *token* de envio. Também não são permitidos *tokens* repetidos em sequências de tamanho três. Todas as mensagens recebidas deverão ter *tokens* validados com o gerador de números pseudoaleatórios de recebimento de *tokens*. A validação de recebimento, porém, deve ser feita com os três próximos números gerados no gerador de números pseudoaleatórios de recebimento, isso acontece para garantir que uma mensagem possa ser retransmitida até três vezes antes de considerar que a conexão foi interrompida de forma não esperada. No caso de falha de três *tokens* consecutivos ou três mensagens sem resposta (quando o obrigatório pelo campo R=1 no *header*), a conexão é considerada corrompida e deverá ser reiniciada.

#### 3.5.5.2 Justificativa

**Em relação à segurança -** Prover um sistema de autenticação dessa forma é inovador no cenário de segurança e adiciona ainda mais dificuldade a ataques de invasores. Ataques como *man-in-the-middle* tornam-se mais difíceis e ataques como DDoS também tem melhoria de resistência, já que pacotes com *tokens* inválidos não são processados.

**Em relação à complexidade e ao tamanho em memória volátil e não-volátil**

Quanto à complexidade, essa funcionalidade traz aumento de complexidade

e consumo de memória volátil e não-volátil em relação ao TinyDTLS, já que é um comportamento específico do CoEP. Apesar disso, todas as ferramentas utilizadas, como o próprio TinyMT, são as mais enxutas - em recursos - no estado da arte e com menor tamanho em memória volátil e não-volátil. Dessa forma, apesar de haver impacto, foi um impacto calculado que deverá ser compensado pelos ganhos nas demais funcionalidades.

**Em relação à simplificação da comunicação -** Outro impacto positivo desse tipo de funcionalidade é sobre a necessidade de se processar um pacote da rede, o que tem impacto direto no consumo de energia. Como supracitado, essa funcionalidade foi desenvolvida principalmente pensando em reduzir o consumo de recursos e energia que se daria ao processar pacotes desnecessários, que poderiam ser ataques, inválidos ou de outros sistemas. Como os *tokens* não são criptografados por serem mutantes (o segredo nesse caso fica na sequência a ser gerada), é possível descartar um pacote antes mesmo de processar seu conteúdo, o que leva a economia de memória volátil, processamento e energia.

#### 3.5.5.3 Metodologia

A utilização de *tokens* não mutantes em troca de mensagens é comum e foi inspiração para criação desse método; por exemplo, quando se faz *login* em um *website*, *tokens* são gerados para lembrar seu *login*, para aceitar conexões a APIs (do inglês *Application Programming Interface*) e outros. A diferença é que no caso do CoEP, os *tokens* não são fixos, seus geradores que são; dessa forma, *tokens* são sequências de números pseudoaleatórios utilizados como chaves mutantes para validar autenticidade de um pacote. A ideia desse comportamento não veio diretamente do estado da arte, mas nasceu para tornar possível a funcionalidade de descartar mensagens antes de tentar descriptografa-las, já que a criptografia é um processo custoso para energia, tempo e buffers, além de funcionar como mais um artifício de segurança para o CoEP. Como existem 16 *bits* de *tokens*, as chances de se acertar um *token*, isto é, de uma mensagem falsa ser apenas aceita é 3 em 65536. Além de acertar o *token*, para uma mensagem ser recebida com sucesso em um ambiente criptografado, ainda se-

ria necessário acertar a *chave de confidencialidade*, que tem aproximadamente 1 chance em $3.4 \times 10^{38}$.

### 3.5.6 Arquitetura de Segurança: Confidencialidade

#### 3.5.6.1 Definição

A confidencialidade das mensagens é garantida por criptografia do tipo AES-128, utilizando como chave a *chave de confidencialidade*, gerada durante o *handshake*. Uma vez que o processo de *handshake* determina as chaves, toda comunicação é criptografada utilizando esse algoritmo. Como destacado na figura 8, não é todo pacote que é criptografado, apenas o *payload* que incluí o *Module ID* e o restante de suas informações. O *header* e *token* permanecem acessíveis e servem justamente para validar a autenticidade da mensagem antes de executar qualquer processo de descriptografia no pacote, evitando desperdícios, como descrito na seção 3.5.5.

#### 3.5.6.2 Justificativa

**Em relação à segurança** A confidencialidade de mensagens é parte essencial da segurança, evitando que terceiros tenham acesso às informações transmitidas. Dessa forma foi escolhido um algoritmo presente no estado da arte para fazer a criptografia de mensagens, mantendo-se a mesma segurança como nas opções da literatura, como no TinyDTLS. Essa funcionalidade é custosa em termos de memória volátil e não-volátil, energia e comunicação. Em comunicação porque qualquer mensagem criptografada terá tamanho múltiplo do tamanho do bloco da criptografia, isto é, 16 *bytes* no caso de AES-128. Por exemplo, uma mensagem de 1 byte assume tamanho de 16 *bytes* quando criptografada. Apesar disso, é indispensável ter esse tipo de segurança e faz parte dos requisitos mínimos da camada CoEP.

#### 3.5.6.3 Metodologia

Para a definição da criptografia de confidencialidade, foi necessário escolher algoritmo que tenha segurança aceitável nos padrões atuais. Como citado na seção 3.5.5.3, as chances de se acertar uma chave de 128 *bits* são de 1 chance em

$3.4 \times 10^{38}$, o que é suficientemente seguro, já que mesmo realizando 1 trilhão de tentativas por nanosegundo, seriam necessários 10 trilhões de anos em processamento. Além disso, principalmente é o método utilizado pelo TinyDTLS, referência de segurança na literatura. A criptografia do tipo AES-128 também foi selecionada por ser de implementação enxuta em software e por também ter extenso suporte em *hardware* - isto é, algoritmo implementado em *hardware* -, onde o cálculo é feito muito mais rapidamente e de forma econômica, sem depender de memória volátil ou não-volátil. Esse é o caso na plataforma selecionada para implementação, que dispõe de cálculo de AES-128 por *hardware*. Apesar disso, o suporte em *hardware* não foi utilizado por motivos de comparação do CoEP com o TinyDTLS, já que o TinyDTLS já implementa o algoritmo em código. Por fim, o algoritmo AES tem versões de 256 *bits* e mais, o que possibilita que pouca mudança seja feita para tornar o sistema ainda mais seguro caso necessário no futuro.

### 3.5.7 Arquitetura de Segurança: Integridade

#### 3.5.7.1 Definição

A integridade da mensagem CoEP é feita utilizando mecanismos das camadas inferiores UDP ou TCP. Ambos protocolos de Rede TCP e UDP possuem rodapé que verifica a integridade da mensagem, tornando-se desnecessário uma revalidação da integridade dos dados. Além disso, o algoritmo das camadas UDP e TCP validam principalmente seu *payload*, visto que o *header* IP recebido é substituído por um *pseudo-header*, dando maior validade a integridade dos dados verificados.

#### 3.5.7.2 Justificativa

A integridade é de importância em sistemas de computação de grão fino, pois evita processamento de pacotes corrompidos ou informações não válidas, o que tem impacto direto ao consumo de energia. Além disso, utilizar mecanismos já implementados em camadas inferiores evita desperdício de memória volátil e não-volátil e energia, além de simplificar a camada de segurança.

#### 3.5.7.3 Metodologia

Devido ao estado da arte e própria definição dos protocolos TCP e UDP, foi determinado que a utilização do *checksum* das camadas inferiores seria suficiente e não traria prejuízo a camada de segurança, utilizou-se os conhecimentos da literatura para identificar essa oportunidade.

# 4 IMPLEMENTAÇÃO

Este capítulo dedica-se a explicação de como foi feita implementação da arquitetura CoEP descrita na seção anterior em *testbed* de dispositivo de grão fino descrito na tabela 4 e detalhado ainda nessa seção.

## 4.1 Plataforma

Como citado na seção 3.2, foi utilizada plataforma Texas Instruments LAUNCHPAD CC2650. A plataforma é especificamente desenvolvida para trabalho com o SoC CC2650. A figura 14 é uma imagem da plataforma. A plataforma expõe todos pinos do SoC CC2650 e possibilita um ambiente de programação, compilação, testes e verificação simplificada e direta, para desenvolvimento acelerado.

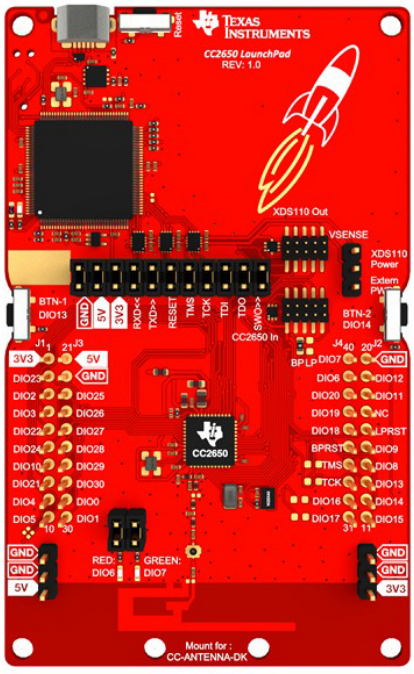



Figura 14 - Plataforma LAUNCHPAD CC2650

Como supracitado, a plataforma expõe o SoC CC2650, cujas características e funcionalidades foram consideradas adequadas para a implementação da camada de segurança CoEP. O SoC CC2650 tem as seguintes características relevantes para esse trabalho, extraídas de seu datasheet:

- Microcontrolador
  - ARM Cortex M3;
  - Até 48MHz de *Clock*;
  - 128KB de memória não-volátil interna;
  - 20KB de memória volátil interna, que pode ter informação conservada mesmo em modos de energia baixos;
  - 8KB de memória do tipo CACHE, que pode também ser utilizada como memória volátil;
  - Suporta gravação da própria memória não-volátil, o que possibilita atualizações auto instaláveis;
  - Arquitetura de 16 *bits*;
- Periféricos
  - Todos pinos configuráveis;
  - Quatro temporizadores de *hardware*;
  - Até 8 conversores analógico-digital com 12 *bits* de precisão a 200k/*samples*;
  - Portas específicas que suportam protocolos de comunicação UART, SSI, I2C, I2S;
  - Relógio de tempo real, do inglês *Real-Time Clock* (RTC)
  - Módulo AES-128
  - Gerador de números aleatórios de *hardware*, do inglês *True Random Number Generator* (TRNG)
- Baixo consumo
  - Opera entre 1.8 e 3.6V
  - Consumo na operação de recebimento de dados da rede: 5.9mA
  - Consumo na operação de envio de dados a 0dBm: 6.1mA
  - Consumo na operação de envio de dados a 5dBm: 9.1mA
  - Consumo do MCU ativo por MHz: 61 µA/MHz

- Consumo do MCU em modo de energia *StandBy*: 1 A
- Consumo do MCU em modo de energia *Shutdown*: 100 nA

Além dessas características, a figura 15 apresenta o diagrama oficial do produto com a arquitetura do MCU.

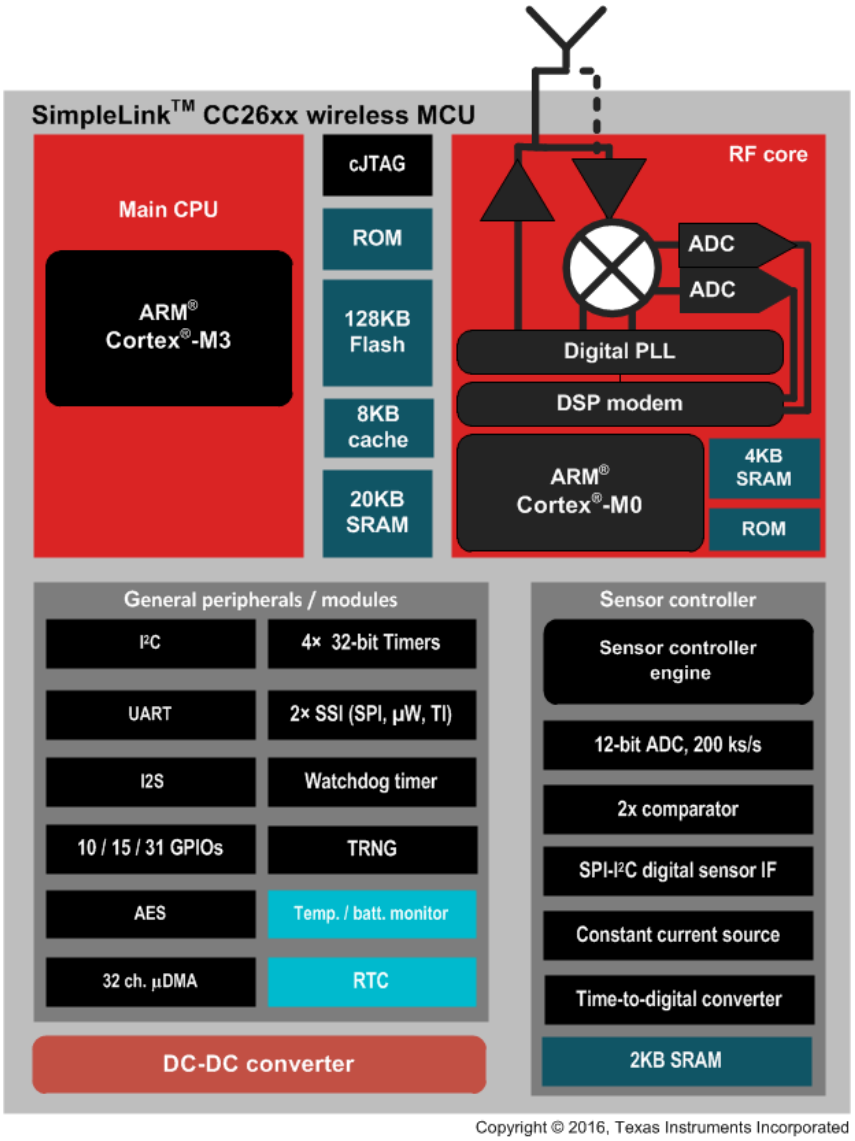



Figura 15 - Arquitetura do MCU CC2650, reprodução da imagem disponibilizada pela Texas Instruments

## 4.2 Sistema Operacional

Como citado na seção 3.2, o *testbed* utilizará o sistema operacional embarcado Contiki-NG. O principal determinante para utilização do Contiki-NG é que o mesmo

já tem suporte para o MCU CC2650 e também especificamente para a plataforma LAUNCHPAD CC2650. Dessa forma, funcionalidades básicas para operação do MCU, além de outras funcionalidades de *software* já estarem implementadas, testadas e validadas, o que limita o trabalho a desenvolver apenas algumas funcionalidades e implementar o CoEP. Além disso, o Contiki-NG já tem implementação oficial de CoAP com TinyDTLS.

## 4.3 Abstração e Implementação do CoEP

Dadas as plataformas e sistemas definidos, já é possível realizar a implementação do CoEP. Mesmo assim, é importante sempre manter abstração de *hardware* e sistema operacional, conforme definido como requisito na seção de requisitos mínimos. A implementação do CoEP está disponibilizada na linguagem C, isso porque que torna-se possível sua utilização direta na maioria dos MCUs disponíveis no mercado, além de ser facilmente utilizável por linguagem próximas como C++ e facilmente traduzível para outras linguagens de maior nível.

Ambas abstrações e implementações estão descritas nas próximas seções.

### 4.3.1 Abstração do CoEP

Para que a camada possa ser utilizada em diversas aplicações distintas, independentemente da arquitetura ou sistema operacional, foi definida a camada de abstração do CoEP. Sempre que possível, a abstração se utilizará de diretivas de compilação da linguagem de pré-processador C, como DEFINES. Isso porque, feito dessa forma, a abstração torna-se mais direta e econômica em código, visto que todas abstrações existem apenas antes da compilação, deixando o código compilado mais enxuto. Nos demais casos, um desenvolvedor que queira usar o CoEP, deverá implementar apenas algumas funções cujos protótipos já estão definidos no CoEP e que, muitas vezes, já existem implementadas nos sistemas operacionais. Detalhes desse processo estão descritos nas próximas seções.

#### 4.3.1.1 Abstração de tipo de endereço IP

Um vez que o CoEP não deve depender do endereço ou tipo de endereçamento (IPv4 ou IPv6, comprimidos ou não), mas precisa armazenar os endereços para realizar conexões, é necessário abstrair o tipo de estrutura que armazena o IP. Como também é necessário armazenar localmente o próprio IP (além de seu tipo), utiliza-se uma definição, da seguinte forma:

```
#ifndef COEP_CONF_IP_TYPE
#error "Define COEP_CONF_IP_TYPE with your IP type"
#else
#define COEP_IP_TYPE COEP_CONF_IP_TYPE
#endif
```

Dessa forma, para um desenvolvedor definir seu tipo de IP, basta incluir:

```
#define COEP_CONF_IP_TYPE <seu tipo>
```

Antes da importação do CoEP no projeto e o próprio CoEP irá utilizar esse tipo de dado para armazenamentos internos.

#### 4.3.1.2 Abstração de temporizadores

Uma vez que o CoEP faz a própria gestão de temporização de seus processos internos e *timeouts*, é necessária a implementação de um tipo de dado que armazene informações de temporização. Novamente, seguindo o esquema de definição em pré-processador, análogo ao IP:

```
#ifndef COEP_CONF_TIMER_TYPE
#error "Please define 'COEP_CONF_TIMER_TYPE' with your timer structure type"
#else
#define COEP_TIMER_TYPE COEP_CONF_TIMER_TYPE
#endif
```

Dessa forma, para um desenvolvedor definir seu tipo de temporizador, basta incluir:

```
#define COEP_CONF_TIMER_TYPE <seu tipo>
```

Antes de incluir o CoEP em seu projeto.

Além disso, para utilização dos temporizadores declarados, é necessária a implementação de três funções que façam controle dos temporizadores. Essas são:

```
typedef void(*timer_callback_t)(void * state);

void oe_timer_set(COEP_TIMER_TYPE * tmr, uint32_t interval_ms,
                        timer_callback_t callback, void * state);
void oe_timer_stop(COEP_TIMER_TYPE * tmr);
bool oe_timer_expired(COEP_TIMER_TYPE * tmr);
```

Onde COEP_TIMER_TYPE é o tipo de estrutura do temporizador, interval_ms é o tempo em milissegundos para expiração do temporizador; timer_callback_t é um tipo de ponteiro de função *void* que recebe um parâmetro de ponteiro de *void*; e state é o próprio ponteiro de *void* a ser propagado para a função *callback*.

O sistema operacional Contiki-NG já dispõe de um sistema de temporizadores, portanto, na implementação em questão basta importar o tipo *struct ctimer*. A implementação de toda abstração em Contiki-NG e no LAUNCHPAD CC2650 será tratada em detalhes nas próximas seções. Caso o desenvolvedor não disponha de uma biblioteca com temporizadores, ele terá que desenvolver uma para implementar o CoEP.

#### 4.3.1.3 Abstração de geração de números aleatórios

Parte essencial da segurança é prover um gerador de números aleatórios que é chamado de *True Random Number Generador* (TRGN), isto é, gerador de números verdadeiramente aleatórios. Isso porque esse tipo de gerador garante mais segurança ao evitar a utilização de algoritmos que gerem números pseudoaleatórios. Para isso, é necessário implementar duas funções cujos protótipos já estão definidos no CoEP que farão a geração de números aleatórios nos processos internos do CoEP:

```
void coep_random_seed(uint32_t seed);
uint8_t coep_random_byte();
```

#### 4.3.1.4 Abstração de Comunicação com demais Camadas da pilha de rede

Por fim, para abstrair a comunicação entre as camadas e o próprio *hardware* de envio de dados, basta fazer a chamada de uma função e a implementação de outra.

Primeiramente, a chamada da função

```
int coep_input(COEP_IP_TYPE * ip, uint8_t * buffer, uint8_t bufferlen);
```

realiza o recebimento de pacotes vindos de camadas inferiores deve ser feita a partir de camadas inferiores, como UDP. Nessa etapa, ocorre maior parte do processamento e da execução da criptografia, então é um processo bloqueante. Caso o tempo real seja importante, é necessário utilizar técnicas de paralelismo. A função retorna o número de *bytes* processados e um número negativo em caso de erro.

Além disso, é necessário implementar uma função que realiza o envio dos dados de aplicação para as camadas inferiores, uma vez que o CoEP tenha alguma informação para enviar. Para isso, basta implementar a função cujo protótipo já está definido que é:

```
void coep_output(COEP_IP_TYPE * ip, uint8_t * buffer, uint8_t bufferlen);
```

Com apenas essas abstrações já é possível ter o CoEP funcional em qualquer plataforma que compile a linguagem C. Apesar disso, o desenvolvedor pode utilizar a documentação para implementar melhorias, como abstrair para funções criptográficas que existam no *hardware* para evitar as implementações de software disponíveis no CoEP que ocupam espaço de memória não-volátil (mas tornam-se independentes da plataforma utilizada).

### 4.3.2 Implementação das Abstrações em CC2650 e Contiki, no *testbed*

Nesta seção, demonstra-se como é feita a implementação das abstrações detalhadas na seção , anterior, com utilização do sistema operacional Contiki-NG e o *hard-*

*ware* da plataforma LAUNCHPAD CC2650.

#### 4.3.2.1 Implementação de tipo de endereço IP

Como demonstrado em seção anterior, para essa implementação basta definir o tipo que armazena IP. Utilizando o Contiki-NG, esse é um trabalho simples, basta, antes de incluir o CoEP, definir o IP da seguinte forma:

```
#include "contiki-net.h"
#define COEP_CONF_IP_TYPE uip_ipaddr_t
```

#### 4.3.2.2 Implementação de temporizadores

Novamente, utilizando o Contiki-NG, esse tipo de funcionalidade já existe e tem conversão simples, bastando utilizar a estrutura *struct ctimer* e suas funções já existentes no sistema operacional da seguinte forma:

```
#include "contiki.h"
#define COEP_CONF_TIMER_TYPE struct ctimer

void oe_timer_set(OE_TIMER_TYPE * tmr, uint32_t interval_ms,
                  timer_callback_t callback, void * state){
    ctimer_set(tmr, (CLOCK_SECOND*interval_ms)/1000, callback, state);
}

void oe_timer_stop(OE_TIMER_TYPE * tmr){
    if(!ctimer_expired(tmr)) ctimer_stop(tmr);
}
bool oe_timer_expired(OE_TIMER_TYPE * tmr){
    return ctimer_expired(tmr);
}
```

O que finaliza a implementação de temporizadores do CoEP através do sistema operacional Contiki-NG.

#### 4.3.2.3 Implementação de números aleatórios

Neste caso, tanto Contiki-NG quanto o próprio SoC CC2650 possuem implementações que podem ser utilizadas no CoEP. Analisando os arquivos internos do Contiki-NG, o sistema operacional faz a utilização da funcionalidade TRNG do SoC CC2650. Por isso, foi escolhido utilizar a implementação do SoC, para comunicação direta com o mesmo, como o próprio Contiki-NG faz, evitando-se utilizar a implementação do Contiki-NG. Isso foi feito dessa forma para evitar saltos desnecessários no código compilado. Portanto, para implementar o gerador de números aleatórios no Contiki-NG com SoC CC2650, basta:

```
#include "contiki.h"
#include "ti-lib.h"

void coep_random_seed(uint32_t seed){
    clock_init();
}
uint8_t coep_random_byte(){
    return ti_lib_trng_number_get() % 0xFF;
}
```

#### 4.3.2.4 Implementação de Camadas

O processo de implementação de camadas é a abstração mais complexa no caso do recebimento de dados, pois depende de alterar código do próprio sistema operacional para possibilitar, de forma invisível ao desenvolvedor, a utilização do CoEP junto ao CoAP ou outros protocolos. Alternativamente, o desenvolvedor pode não alterar o sistema operacional e simplesmente criar uma aplicação CoEP com um *Module* para sua camada de aplicação, mas para isso será necessário criar um sistema adicional que cria as estruturas UDP e IP, externo ao SO. Essa escolha depende principalmente de se a camada de aplicação já está implementada ou não no Contiki-NG. A camada CoAP, por exemplo, já faz parte do Contiki-NG e, portanto, para utilizá-la com CoEP não há escolha além de alterar o SO. Já se o desenvolvedor tiver um protocolo próprio ou não utilizar nenhum que já é parte do Contiki-NG, a implementação torna-se

mais simples. Abaixo seguem as explicações para os dois tipos de implementações, porém vale ressaltar que a implementação utilizada nessa dissertação é a mais complexa, que envolve alterar o sistema operacional afim de se ter uma camada invisível ao desenvolvedor.

**Implementação alterando sistema operacional** Para esta implementação, o CoEP passa a fazer parte do Contiki-NG, o que o torna mais flexível e apto a ser utilizado com outros sistemas internos do Contiki-NG. Isso, porém, deve ser implementado para cada protocolo, visto que a arquitetura do Contiki não prevê uma camada de segurança unificada entre todos protocolos. Apesar do trabalho manual ser maior caso deseje-se utilizar o CoEP em vários protocolos, a utilização de memória volátil e não-volátil permanece a mesma em teoria (há eventuais adaptações mínimas nos protocolos, como no caso do CoAP), já que o CoEP vai funcionar como uma camada unificada, mesmo se integrado separadamente no código do sistema operacional.

Como essa dissertação tem como foco a comparação utilizando o protocolo de aplicação CoAP, as implementações descritas abaixo são feitas apenas para o caso CoAP. Dessa forma, para habilitar o CoEP no Contiki-NG com CoAP, o seguinte arquivo foi alterado: *contiki/os/net/app-layer/coap/coap-uip.c*. Por ser um código extenso, sua consulta para entendimento da implementação pode ser feita através do apêndice A.

Também se justifica que foi deixado como apêndice, porque é um caso complexo e necessita-se entender como o Contiki-NG funciona para melhor entendimento de como foi feita a implementação. Para identificar as implementações do CoEP nesse arquivo do sistema Contiki, basta buscar por códigos que estão entre as definições de pré-processador:

```
#ifdef WITH_COEP
 ... (código de integração CoEP) ...
#endif
```

**Implementação sem alterar sistema operacional** Para servir como exemplo de entendimento de como a implementação anterior foi feita, segue exemplo de imple-

mentação fora dos arquivos de sistema do Contiki-NG, que possui entendimento mais fácil. Nessa implementação, basta criar uma conexão UDP e chamar as funções Contiki-NG que realizam a transferência de dados entre as camadas inferiores:

```
#include "contiki.h"
#define UDP_PORT 0xCOE2

static struct uip_udp_conn * client_conn = NULL;

//criacao de conexao UDP
bool init_udp(){
    client_conn = udp_new(NULL, UIP_HTONS(UDP_PORT), NULL);
    if (client_conn == NULL) {
        return false;
    }

    udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));
    return true;
}

//funcao para envio para camadas inferiores
void coep_output(COEP_IP_TYPE * ip, uint8_t * buffer, uint8_t bufferlen){
    uip_udp_packet_sendto(client_conn, buffer, bufferlen, ip,
                          UIP_HTONS(UDP_PORT));
}

//processo que faz recebimento dos pacotes
PROCESS_THREAD(udp_client_process, ev, data)
{
    while (1) {
     PROCESS_WAIT_EVENT();
     if (ev == tcpip_event) {
         if (uip_newdata()) {
```

```
            uint8_t * buffer = (uint8_t*)uip_appdata;
            uint8_t bufferlen = uip_datalen();
            COEP_IP_TYPE * ip = &(UIP_IP_BUF->srcipaddr);
         coep_input(ip, buffer, bufferlen);
      }
     }
    }
}
```

Dessa forma, ao iniciar o processo *udp_client_process*, o CoEP já estará funcional, bastando apenas declarar *Modules* que façam a entrega dos pacotes ao protocolo de aplicação do desenvolvedor.

### 4.3.3 Implementação dos Algoritmos de Criptografia

Como descrito na seção 3, toda arquitetura de segurança depende de quatro algoritmos de criptografia: algoritmo de Chave Pública e Privada de Curvas Elípticas do tipo secp256r1; algoritmo de criptografia de chave simétrica AES-128; algoritmo de *hashing* SHA-256; e um gerador de números pseudoaleatórios para geração dos *tokens* das mensagens, o TinyMT.

Vale lembrar que o gerador de números pseudoaleatórios é distinto do TRNG, necessário na implementação do CoEP. Na verdade, o TRGN é utilizado internamente por alguns desses algoritmos. Dessa forma, nessa seção, demonstra-se como foram feitas essas implementações, para cada algoritmo. Nota-se que foram utilizadas principalmente bibliotecas autocontidas, já que reescrever tais algoritmos já disponíveis seria desnecessário.

#### 4.3.3.1 Curvas Elípticas secp256r1

Para o algoritmo de curvas elípticas, foi utilizado o recurso chamado de *Micro-ECC*. É uma biblioteca autocontida, isto é, sem dependências externas, que faz a implementação do algoritmo completo necessário e padrão. A biblioteca ainda oferece diversos tipos de curvas, não necessariamente apenas a utilizada secp256r1. Por fim,

a biblioteca foi feita de forma a proteger contra ataques comuns conhecidos e executa algumas operações diretamente em operações do tipo *arm-thumb*, que são operações padrões entre microprocessadores ARM descritas em baixo nível (*assembly*) para alta otimização e performance. A biblioteca suporta diversas arquiteturas da plataforma ARM e é distribuída sob licensa BSD cláusula 2 no repositório <https://github.com/kmackay/micro-ecc>.

#### 4.3.3.2 AES-128

Para o algoritmo AES-128, foi utilizada a própria implementação disponível no Contiki-NG. Além disso, para otimizar ainda mais o código é possível usar o *hardware* de cálculo AES-128 do próprio SoC CC2650. Porém, como o Contiki-NG já dispõe desse algoritmo e ele estará compilado junto ao sistema operacional, foi utilizado o mesmo por simplicidade e por não trazer diferença no tamanho final dos binários compilados. Por fim, a biblioteca AES-128 do Contiki-NG é aberta e autocontida, portanto pode ser extraída e utilizada separadamente caso necessário.

#### 4.3.3.3 SHA-256

O algoritmo SHA-256 é de domínio público e como o AES-128, tem implementações autocontidas. Após buscas, testes e validação do algoritmo, foi escolhido utilizar uma solução já desenvolvida, de código aberto e autocontida distribuída no repositório <https://github.com/amosnier/sha-2>.

#### 4.3.3.4 Gerador de Números Pseudoaleatórios

O gerador de números pseudoaleatórios foi escolhido segundo características de baixo consumo de memória volátil e não-volátil, sendo este o TinyMT. O algoritmo além de ser simples em implementação de um algoritmo do tipo *Mersenne Twister*, possuí um estado de apenas 128 *bits*, o que permite baixo consumo de memória volátil. Algoritmos comuns de geração de números pseudoaleatórios, como aqueles contidos em compiladores como o GNU utilizam diversas tabelas de inicialização e estados, ocupando muitas vezes milhares de *bytes* da memória volátil, o que seria infactível em um dispositivo de grão fino. Dessa forma, o TinyMT tornou-se ideal por

ter baixo consumo memória volátil e não-volátil e prover 128 *bits* de estado, o que garante grande quantidade de estados. O algoritmo TinyMT é distribuído pela licença BSD (do inglês *Berkeley Software Distribution*) clausula 3 e tem biblioteca autocontida, tornando-se ideal para utilização nesse esquema de criptografia. Sua distribuição também é feita em linguagem C, o que tornou sua implementação simples dentro do CoEP. O código da biblioteca está disponível no repositório <https://github.com/MersenneTwister-Lab/TinyMT>.

### 4.3.4 Implementação da Arquitetura Completa do CoEP

Uma vez implementadas arquiteturas e dependências de criptografia, basta implementar a arquitetura proposta do CoEP. A implementação foi feita em linguagem C, seguindo programação orientada a eventos. A implementação completa está mantida em repositório *git* para acesso público em <https://github.com/mangine/coep>.

# 5 RESULTADOS

Essa seção descreve o ambiente de extração de resultados, suas ferramentas e faz o levantamento dos resultados de forma direta e objetiva e, quando justificado, faz o levantamento subjetivo ou teórico segundo a metodologia de avaliação descrita na seção 3.2.1. As avaliações detalhadas, comparativas e conclusivas desses resultados, porém, encontram-se na seção 6 de conclusões.

## 5.1 Descrição do Ambiente de Avaliação e Extração de Resultados

Nesse seção apresenta-se qual o ambiente que foi utilizado nos testes, avaliação e extração de resultados.

Para simulação de uma aplicação, utilizou-se o exemplo de cliente CoAP disponível no Contiki-NG como protocolo de aplicação servindo dados de três sensores reais da plataforma LCUNCHPAD CC2650.

Para execução do código em ambiente de testes, foram utilizadas duas plataformas LAUNCHPAD CC2650 que serviram como cliente e servidor dos testes necessários. A avaliação do funcionamento nesse caso foi feita utilizando saída serial de informações. A saída serial por sua vez já é conectada a um módulo Texas Instruments chamado de *XDS110*, que além de depuração já faz a conversão de dados seriais para USB, o que torna possível acessar as informações por um computador comum com o sistema operacional *Windows 10*.

O código utilizado para testes de avaliação do *testbed* está disponível no apêndice B, e refere-se ao processo CoAP descrito acima. O código foi desenvolvido como exemplo de funcionamento de CoAP para o sistema operacional Contiki-NG.

Além disso, como citado anteriormente na seção 3.2 de metodologia e detalhado na seção 3.3, foi utilizada a ferramenta *fpvgcc* para extração dos tamanhos compilados de cada biblioteca baseada em suas dependências e o próprio sistema operacional para calcular o tamanho dos protocolos em *bytes*.

## 5.2 Resultados

Essa seção é dedicada a apresentação de resultados bem como cálculos e processos que levaram aos mesmos, seguindo as instruções da seção 3.3.

### 5.2.1 Segurança

Nesta seção são apresentados resultados referentes às análises de segurança. Essas análises se dão sempre de forma teórica e considerando apenas ataques contra de segurança de força-bruta, pois tem forma direta de se calcular e comparar. Ataques de força-bruta tentam adivinhar o resultado ou chave criptográfica por tentativa e erro, são um tipo simples de ataque que não envolve conhecimento do algoritmo de criptografia. Outros ataques envolvem literatura e processos mais complexos e podem ser base para estudos futuros.

#### 5.2.1.1 A probabilidade de descriptografia de um pacote qualquer

A implementação padrão do CoEP prevê chaves simétricas de confidencialidade de tamanho 128 *bits*, portanto, a probabilidade de descriptografia de um pacote qualquer através de força-bruta é 1 em $2^{128}$ ou em notação científica cerca de 1 em $3.4 \times 10^{38}$ possibilidades.

O TinyDTLS suporta diversos algoritmos de criptografia, porém em criptografia simétrica, sempre algoritmos de chave de tamanho de 128 *bits*, igualmente ao CoEP. Dessa forma, assim como no caso do CoEP, a probabilidade de descriptografia de um pacote qualquer através de força-bruta é 1 em $3.4 \times 10^{38}$ possibilidades.

Dessa forma, ambos CoEP e TinyDTLS têm o mesmo nível de segurança, como descrito na tabela 5.

| | Probabilidade de descriptografia de um pacote qualquer |
|---|---|
| TinyDTLS | 1 em $3.4 \times 10^{38}$ |
| CoEP | 1 em $3.4 \times 10^{38}$ |

Tabela 5 - Tabela de comparação do quesito probabilidade de descriptografia de um pacote qualquer por força-bruta

#### 5.2.1.2 A probabilidade de falsificação de um pacote qualquer

A probabilidade de falsificação de um pacote qualquer será a probabilidade de descriptografia vezes outras probabilidades necessárias para vencer as autenticações de segurança.

No esquema TinyDTLS, não existem proteções adicionais de autenticação, enquanto no esquema CoEP, existe a proteção via *token*, adicionando 3 em 65536 chances de acerto, já que há sempre três possíveis *tokens* a serem aceitos dentre os 65536 (quantidade possível com 16 *bits* de *token*) possíveis. As três chances existem porque o CoEP aceita perda de até três pacotes com *tokens*, então a todo momento existem três *tokens* válidos.

A tabela 6 apresenta então as chances de falsificação de um pacote TinyDTLS, que é a mesma de descriptografia; e as chances de falsificação no caso CoEP, é que a mesma de descriptografia vezes a segurança do *token*, que resulta em 1 chance em $7.4 \times 10^{42}$ possibilidades para tentativas de força-bruta.

| | Probabilidade de falsificação de um pacote qualquer |
|---|---|
| TinyDTLS | 1 em $3.4 \times 10^{38}$ |
| CoEP | 1 em $7.4 \times 10^{42}$ |

Tabela 6 - Tabela de comparação do quesito probabilidade de falsificação de um pacote qualquer por força-bruta

#### 5.2.1.3 A probabilidade de intrusão de dispositivos mal-intencionados em uma rede

Essa probabilidade se dá pelas chances que um dispositivo malicioso teria para conectar-se a uma rede protegida. Essa probabilidade depende do tipo de segurança sendo utilizada nas camadas de segurança.

Para a funcionalidade de autenticação na rede, o TinyDTLS implementa criptografia PSK, que é justamente criptografia de chaves pré-programadas, similar ao *login* CoEP. Quando CoEP utiliza o esquema de *login* e o TinyDTLS o esquema PSK, a probabilidade de intrusão é a mesma, de se adivinhar uma chave de 128 *bits*, que no TinyDTLS é a chave simétrica e no CoEP é ou a chave *login* ou a chave *password*, que como calculado anteriormente é de 1 em $3.4 \times 10^{38}$ possibilidades. Existe apenas a

diferença que como o CoEP obriga a utilização de chaves assimétricas sempre, o processo criptográfico de *login* do CoEP é mais seguro, pois as suas chaves simétricas sempre variam.

Já quando se utiliza curvas elípticas (ECC) na camada TinyDTLS ou não se utiliza *login* na camada CoEP, ambas não apresentam sistema de proteção contra dispositivos não autorizados.

Dessa forma, como exposto na tabela 7 ambos protocolos são equivalentes.

| Probabilidade de intrusão de dispositivos mal-intencionados em uma rede | |
|---|---|
| TinyDTLS ECC | Sem segurança |
| CoEP sem *login* | Sem segurança |
| TinyDTLS PSK | 1 em $3.4 \times 10^{38}$ |
| CoEP com *login* | 1 em $3.4 \times 10^{38}$ |

Tabela 7 - Tabela de comparação do quesito probabilidade de intrusão de dispositivos maliciosos em uma rede

### 5.2.2 Complexidade e Funcionalidades

Em relação a complexidade, isto é, extensão de código e quantidade de funcionalidades, a tabela 8 mostra o tamanho total das soluções em *bytes*, implementadas no sistema operacional Contiki-NG.

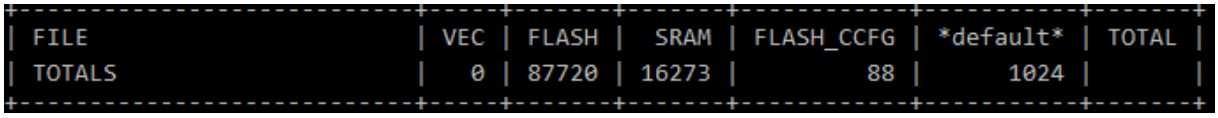
| Tamanho de código | |
|---|---|
| TinyDTLS | 405.414 *bytes* |
| CoEP | 151.161 *bytes* |

Tabela 8 - Tabela de comparação do quesito tamanho de código para funcionamento no sistema operacional Contiki-NG

Em termos de funcionalidades de criptografia, a implementação do TinyDTLS suporta *Pre-Shared-Key* (PSK) e ECC, ambos com criptografia de chave simétrica AES-128. Já o CoEP suporta apenas ECC com a combinação opcional similar ao PSK, de um sistema de *login*. Dessa forma, em questão de criptografia, o TinyDTLS e o CoEP têm o mesmo nível de complexidade, apenas arquitetados de forma distinta. Já em complexidade de operação, pelo TinyDTLS ser uma simplificação do DTLS que tem processos internos mais robustos, sua complexidade é maior, o que é comprovado por uma codificação mais significativamente mais extensa.

### 5.2.3 Tamanho em memória volátil e não-volátil

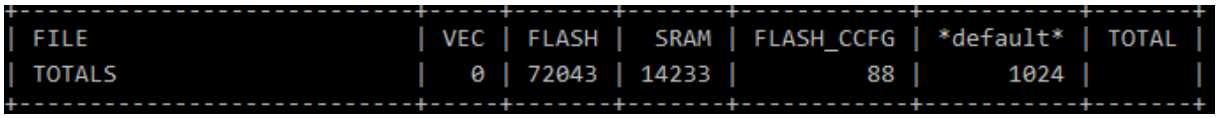
Ao executar a compilação do CoAP no Contiki-NG com TinyDTLS, resultou-se num arquivo *MAP*, que com a utilização da ferramenta *fpvgcc*, obteve-se os resultados da figura 16.



| FILE | VEC | FLASH | SRAM | FLASH_CCFG | *default* | TOTAL |
|---|---|---|---|---|---|---|
| TOTALS | 0 | 87720 | 16273 | 88 | 1024 | |

Figura 16 - Resultado do comando *fpvgcc* em arquivo *MAP* compilado para Contiki-NG, TinyDTLS e CoAP

Já ao executar a compilação do CoAP no Contiki-NG com CoEP, resultou-se num arquivo *MAP* que quando processado pela ferramenta *fpvgcc* retornou os resultados da figura 17.



| FILE | VEC | FLASH | SRAM | FLASH_CCFG | *default* | TOTAL |
|---|---|---|---|---|---|---|
| TOTALS | 0 | 72043 | 14233 | 88 | 1024 | |

Figura 17 - Resultado do comando *fpvgcc* em arquivo *MAP* compilado para Contiki-NG, CoEP e CoAP

Dessa forma, usando a metodologia descrita, pode-se levantar a tabela 9 com os resultados de consumo de memória volátil e não-volátil.

| Tamanho em memória volátil e não-volátil | | |
|---|---|---|
| Camada | Memória Volátil | Memória Não-Volátil |
| TinyDTLS | 16273 *bytes* | 88832 *bytes* |
| CoEP | 14233 *bytes* | 74145 *bytes* |

Tabela 9 - Tabela de comparação do quesito tamanho em memória volátil e não-volátil

Além da avaliação do impacto total em um programa compilado, é importante avaliar quanto CoEP e TinyDTLS ocupam em implementação de código individualmente. Para isso, utilizando a ferramenta *fpvgcc*, foi possível identificar as dependências de cada módulo e extrair o tamanho de implementação individual, obtendo-se os resultados das tabelas 10 e 11, e um total comparativo de cada camada de segurança na tabela 12.

O item **Outras Dependências** listado trata de dependências do próprio compilador, que varia pouco entre programas. Um exemplo de outra dependência que varia de um programa para outro é a função *printf* que, dependendo de quais tipos de *output* forem utilizados, precisa dispor de mais ou menos funcionalidades, que acabam impactando na utilização de memória. Essas dependências trazem pouca variação no resultado.

| Tamanho em memória volátil e não-volátil | | |
|---|---|---|
| Objeto | Memória Volátil | Memória Não-Volátil |
| dtls.o | 0 *bytes* | 10093 *bytes* |
| rijndael.o | 0 *bytes* | 6768 *bytes* |
| ecc.o | 0 *bytes* | 3740 *bytes* |
| dtls-crypto.o | 496 *bytes* | 1760 *bytes* |
| dtls-cmm.o | 4 *bytes* | 1536 *bytes* |
| netq.o | 1140 *bytes* | 321 *bytes* |
| coap-uip.o | 76 *bytes* | 1212 *bytes* |
| sha2.o | 0 *bytes* | 1234 *bytes* |
| dtls-hmac.o | 520 *bytes* | 374 *bytes* |
| dtls-support.o | 447 *bytes* | 196 *bytes* |
| dtls-peer.o | 56 *bytes* | 112 *bytes* |
| assert.o | 0 *bytes* | 53 *bytes* |
| Outras Dependências | 0 *bytes* | 236 *bytes* |
| Total | 2739 *bytes* | 27635 *bytes* |

Tabela 10 - Tabela de consumo de memória volátil e não-volátil por objeto compilado TinyDTLS

| Tamanho em memória volátil e não-volátil | | |
|---|---|---|
| Objeto | Memória Volátil | Memória Não-Volátil |
| uECC.o | 4 *bytes* | 5712 *bytes* |
| coep.o | 444 *bytes* | 2188 *bytes* |
| coap-uip.o | 72 *bytes* | 1710 *bytes* |
| aes-128.o | 176 *bytes* | 816 *bytes* |
| sha-256.o | 0 *bytes* | 840 *bytes* |
| tinymt32.o | 0 *bytes* | 616 *bytes* |
| coep_abstraction.o | 0 *bytes* | 76 *bytes* |
| Outras Dependências | 3 *bytes* | 0 *bytes* |
| Total | 699 *bytes* | 11958 *bytes* |

Tabela 11 - Tabela de consumo de memória volátil e não-volátil por objeto compilado CoEP

As tabelas completas do *outputs* da execução da ferramenta *fpvgcc* para CoAP com TinyDTLS está presente no apêndice D; e para CoAP com CoEP no apêndice C.

| Tamanho em memória volátil e não-volátil, apenas camadas de segurança | | |
|---|---|---|
| Camada | Memória Volátil | Memória Não-Volátil |
| TinyDTLS | 2739 *bytes* | 27635 *bytes* |
| CoEP | 699 *bytes* | 11958 *bytes* |

Tabela 12 - Tabela de consumo de memória volátil e não-volátil apenas das camadas de segurança

# 6 CONCLUSÃO

Essa seção é dedicada à análise e avaliação comparativa dos resultados do CoEP, em relação à camada de segurança utilizada atualmente TinyDTLS, segundo os tópicos organizados de forma a facilitar entendimento, como exposto a seguir.

## 6.1 Conclusões em relação à arquitetura

Em relação à arquitetura, a proposta do CoEP é inovadora em sistemas compostos por dispositivos de grão fino. Isso porque a camada de segurança nos padrões TLS atuais está sempre relacionada a um protocolo de aplicação, isto é, cada protocolo deve utilizar a camada de segurança intrinsecamente quando julgar necessário. Ou seja, protocolos distintos devem selecionar ou não o uso de TLS e cada protocolo deve manter seus parâmetros de conexão.

Já no CoEP, sua arquitetura visa oferecer segurança como um serviço - através de sua API -, como uma camada que já disponibiliza e controla a segurança sem necessidade de iteração com o usuário ou desenvolvedor. Há diversas vantagens nesse tipo de arquitetura, como em simplificação de códigos, segurança centralizada e enxuta e a obrigação da utilização de segurança em aplicação, que é uma tendência mais recente e necessária.

Apesar de diversas vantagens, utilizar uma arquitetura de segurança como serviço pode trazer desvantagens em algumas situações como ser mais difícil de se integrar, como, por exemplo, quando aplicações distintas quiserem utilizar esquemas criptográficos distintos ou regras de rede distintos. Mesmo assim, no CoEP isso significaria conexões adicionais com um mesmo ponto, apenas consumindo um pouco mais de memória volátil.

Mesmo assim, as vantagens dessa arquitetura são mais relevantes que suas desvantagens, principalmente considerando-se que o CoEP foi desenvolvido para ser utilizado especificamente em sistemas de redes de sensores sem fio de dispositivos de grão fino, onde demais vantagens - discutidas mais a frente - tem grande impacto.

Em questão de arquitetura, o CoEP também inovação na utilização de *tokens* que seguem uma sequência criptográfica pseudoaleatória, o que possibilita ao mesmo tempo identificação de mensagens, ordenação e segurança adicional de autenticação.

## 6.2 Conclusão Em relação aos resultados

Nesta seção, são feitas conclusões relacionadas aos resultados obtidos na seção 5.

### 6.2.1 Segurança

Em relação à segurança, como citado na seção 5 de resultados, por ser tópico de complexidade e diversos tipos de ataques e métodos possíveis, considerou-se apenas ataques do tipo força-bruta, que envolve acertar o segredo ou chave dentro do universo de opções possíveis para efeito de comparação. Além disso, é importante destacar as diferenças entre DTLS, TinyDTLS e CoEP. Com base nos resultados exibidos, comparativos entre TinyDTLS e CoEP nas condições determinadas, devido à implementação limitada do TinyDTLS, o CoEP mostrou-se um protocolo de segurança igual ou maior, conforme justificado a seguir.

Enquanto os algoritmos de criptografia de chave assimétrica e simétrica tem mesma força nas duas soluções CoEP e TinyDTLS, por serem os mesmos, o CoEP ainda pode oferecer a funcionalidade de *login* junto a funcionalidade de chaves assimétricas, o que permite que apenas dispositivos pré-cadastrados a participarem da rede e mantém criptografia com chaves que variam em cada conexão. No CoEP, sempre existe a necessidade de realizar-se a troca de chaves assimétricas, mesmo quando utilizar-se a funcionalidade de *login*. Já no DTLS e TinyDTLS, a aplicação pode escolher apenas entre chaves pré-cadastradas (algoritmo PSK) ou chaves assimétricas como forma de confidencialidade, o que torna o processo mais inseguro que no CoEP. Isso justifica-se porque em teoria, é possível adivinhar uma chave estática com mais facilidade do que uma chave mutável.

Outro ponto relevante é que o CoEP também utiliza de um esquema de segurança adicional de *tokens*. Esse sistema que gera uma sequência de números pseudoa-

leatórios utiliza uma verificação de 16 *bits* adicional em cada pacote, o que garante mais segurança aos algoritmos de criptografia de forma descorrelacionada e enxuta, o que aumenta dificuldade de falsificação de pacotes em mais de 20.000 vezes, como demonstrado na tabela 6.

Dessa forma, conclui-se que em relação ao TinyDTLS e a algumas funcionalidades do próprio DTLS, nas condições de ataques dadas e nas condições de implementação em dispositivos com recursos restritos, o CoEP é teoricamente mais seguro e melhor protegido contra os ataques levantados. Já em relação ao DTLS apenas, o mesmo já possui algoritmos de confidencialidade que são mais fortes e garantem mais segurança, porém a segurança de tais algoritmos dependem principalmente de chaves maiores, o que é impraticável em sistemas com dispositivos de grão fino. Portanto, apesar do DTLS poder ser mais seguro em ambientes com recursos relativamente infinitos, em ambientes com recursos limitados o CoEP pode ser considerado uma solução mais adequada.

### 6.2.2 Conclusão em relação à Complexidade e funcionalidades

Como exibido no estado da arte, a implementação TinyDTLS faz utilização de apenas dois métodos de criptografia: a troca de chaves públicas de curvas elípticas (ou ECDH); e utilização do algoritmo PSK para chaves pré-programadas. Equivalentemente, o CoEP oferece troca de chaves públicas obrigatoriamente e opcionalmente ainda oferece a autenticação por meio de chaves pré-programadas. Apesar de serem mecanismos distintos, os objetivos e resultados são parecidos, com a única diferença que o TinyDTLS possibilita o uso de apenas PSK sem chaves públicas, o que reduz o nível de segurança. Dessa forma, pode-se concluir que em disponibilização de funcionalidades, ambos protocolos são próximos e oferecem todas funcionalidades de segurança necessárias para dispositivos de grão fino.

Já em funcionalidades de rede, o CoEP oferece funcionalidades como *multicast* e *broadcast* criptográfico, o que é vantajoso para utilização com protocolos de aplicação - como o próprio CoAP - e, dessa forma, apresenta mais funcionalidade que o TinyDTLS.

Já em arquitetura e implementação, o TinyDTLS apresentou-se mais complexo.

Isso era esperado visto que o TinyDTLS é uma simplificação do padrão DTLS que possibilita um grande número de funcionalidades e algoritmos e, portanto, seria em sua natureza mais complexo. Isso, porém, é um fator negativo quando em ambientes com recursos restritos e por isso, conclui-se que o CoEP se demonstrou mais adequado nesse quesito.

Em suma, o CoEP apresentou mais funcionalidades que o TinyDTLS mantendo ainda complexidade menor, o que como uma conclusão geral de funcionalidade e complexidade é a consideração de que o CoEP é mais adequado para utilização em dispositivos de grão fino.

### 6.2.3 Conclusão em relação à utilização de memória volátil e não-volátil

Em relação ao tamanho em memória volátil e não-volátil, os resultados comparativos entre CoEP e TinyDTLS foram consideravelmente menores na utilização do CoEP. Isso era esperado, visto que, além da própria arquitetura mais compacta do CoEP, a camada foi desenvolvida especificamente para suprir as necessidades de redes de sensores sem fio de dispositivos de grão fino, utilizando mesmo ou melhor nível de segurança. Além disso, o impacto do CoEP é ainda maior caso sejam implementados mais de um protocolo de aplicação com segurança; isso porque o TinyDTLS precisará ser reimplementado nessas outras camadas de aplicação, enquanto o CoEP já estará servindo a pilha de comunicação independentemente de quantos protocolos o utilizam. Para implementar mais protocolos de aplicação junto do CoEP bastaria adicionar um *Module*, acrescentando poucos *bytes* de memória volátil e não-volátil.

Em relação aos resultados apresentados, as vantagens da utilização do CoEP são expressivas. A implementação do CoEP em C, no caso de estudo tiveram consumo de apenas 699 *bytes* em memória volátil e 11958 *bytes* em memória não-volátil, incluindo todos algoritmos criptográficos necessários contra um consumo de 2739 *bytes* em memória volátil e 27635 *bytes* em memória não-volátil na implementação do TinyDTLS. Dessa forma, o CoEP é pouco mais de 74% mais econômico em utilização de memória volátil, mesmo com todos *buffers* necessários para recebimento e processamento de pacotes; e mais de 56% mais econômico em consumo de memória não-volátil. Considerando-se plataformas como a do caso de estudo, que têm disponibilização

de apenas 28KB disponíveis de memória volátil e 128KB de memória não-volátil, a solução CoEP pode poupar cerca de 2KB de memória volátil e 5.5KB de memória não-volátil, ou cerca de 7% de toda memória volátil disponível e 4.3% de toda memória não-volátil disponível. Tal economia é ainda mais significativa para dispositivos que são ainda mais enxutos, aqueles de granularidade menor ou ainda de classe 0, segundo classificações abordadas nessa dissertação.

Em suma, os resultados de consumo de memória demonstraram significativa menor utilização de memória volátil e não-volátil na implementação do CoEP, o que o põe a frente do TinyDTLS também nesse quesito para utilização em dispositivos de grão fino.

## 6.3 Conclusão em relação ao desenvolvimento tecnológico

Em relação ao desenvolvimento tecnológico, é possível concluir que as inovações que o CoEP apresenta, como sua arquitetura e utilização de *tokens* de autenticação, tem resultados suficientemente positivos para consolidar a utilização do CoEP como uma camada de segurança e até mesmo de suas inovações para implementação de melhorias a outros protocolos, soluções e desafios. Espera-se que técnicas utilizadas no CoEP possam ser replicadas para propagar a utilização de segurança, principalmente quando tratando-se de dispositivos de grão fino, onde segurança ainda é difícil de se implementar.

## 6.4 Conclusão Em relação aos objetivos

Da própria seção 1.3 de objetivos, pode-se concluir para cada objetivo específico:

### 6.4.1 Objetivo específico 1

Levantar o estado da arte de protocolos de comunicação em redes de sensores e suas aplicações;

Este objetivo foi concluído durante o levantamento do estado da arte, que considerou esquemas de segurança e protocolos desenvolvidos para dispositivos enxuto.

Além disso, consideraram-se aplicações para levantamento do estado da arte com ajuda de empresa parceira. Toda camada de segurança CoEP foi desenvolvida levando em conta esses critérios.

### 6.4.2 Objetivo específico 2

Discutir e propor estratégias de segurança suficientes para garantir autenticidade, confidencialidade e integridade;

Novamente, dada a arquitetura proposta do CoEP, conclui-se que esse objetivo também foi atingido, pois o CoEP:

- pode proporcionar autenticidade tanto no processo de *login* quanto na própria implementação de autenticidade de pacotes utilizando o esquema de *tokens*;
- pode proporcionar confidencialidade através de suas *chaves de confidencialidade* geradas com segurança igual ou maior do TinyDTLS;
- pode proporcionar validação de integridade utilizando campos de camadas inferiores, evitando *overhead* desnecessário.

### 6.4.3 Objetivo específico 3

Propor uma arquitetura da camada;

Este objetivo também foi concluído, visto que a arquitetura foi proposta, implementada e testada, resultando em vantagens sobre o TinyDTLS e até sobre o DTLS em alguns casos específicos, como exibido nos resultados e discutido nas seções desta conclusão.

### 6.4.4 Objetivo específico 4

Desenvolver camada junto às necessidades e recursos disponíveis em padrão IEEE 802.15.4 - detalhado na seção 2;

conclui-se que esse objetivo foi completo com sucesso, visto que que as limitações do CoEP são, segundo a própria definição de sua arquitetura, as mesmas que do pa-

drão 802.15.4 ou de outras camadas inferiores. Além disso, o CoEP foi desenvolvido em sua integridade para garantir níveis de segurança iguais ou melhores que a literatura equivalente, no caso o TinyDTLS, mas sempre respeitando as necessidades de economia de memória volátil e não-volátil e consumo de energia.

### 6.4.5 Objetivo específico 5

Testar e validar protocolo empiricamente bem como descrevê-lo comparativamente seguindo as premissas da seção 3.2.2;

Esse objetivo foi concluído na apresentação dos resultados onde foi demonstrado teoricamente que o CoEP, dadas as condições discutidas, é uma camada com segurança e compatibilidade mais adequada do que o TinyDTLS para dispositivos de grão fino; e, além disso, possibilitou as mesmas funcionalidades que o TinyDTLS na aplicação de teste, avaliação e validação. Por fim, o CoEP ainda possibilitou novas funcionalidades de *multicast* e *broadcast*, de grande importância para camadas de aplicação.

### 6.4.6 Objetivo principal

Propor, Desenvolver e avaliar uma camada de segurança para protocolos de comunicação de dispositivos de grão fino com amplo espectro de utilização.

Com base nos resultados apresentados e na conclusão dos demais objetivos, pode-se concluir que o objetivo principal dessa dissertação foi concluído com sucesso em sua integridade.

O CoEP foi desenvolvido como uma camada de segurança para protocolos de aplicação de dispositivos com recursos restritos, aqui caracterizados e chamados de dispositivos de grão fino. Além disso, dos resultados, o CoEP teve menor utilização de recursos como memória volátil e não-volátil nesses dispositivos em relação à solução mais enxuta abrangentemente adotada da literatura, o TinyDTLS. Ademais, o CoEP não trouxe mais limitações aos protocolos ou recursos de rede, possibilitando até mesmo a utilização de *multicast* e *broadcast* - como descrito em sua arquitetura -, promovendo possível maior espectro de utilização que o próprio TinyDTLS. Por fim,

por ter arquitetura que permite personalização de *handshake* e protocolos de camadas superiores, permite, em teoria, utilização abrangente.

**Dessa forma, corrobora-se que a dissertação atingiu todos seus objetivos.**

## 6.5 Discussões sobre resultados e conclusões

Esse seção é dedicada a discussões sobre alguns pontos do funcionamento da camada CoEP e de comentários do seu autor.

O CoEP foi desenvolvido especificamente para auxiliar no desenvolvimento de aplicações seguras em redes de sensores sem fio compostas de dispositivos de grão fino, cuja arquitetura tem se tornado tendência graças aos progressos econômicos e científicos de se desenvolverem novos negócios e novas aplicações. Por conta disso, reafirma-se que o CoEP foi desenvolvido para ser uma opção mais simples, fácil de se implementar, enxuta e, em suma, acessível a tais soluções no lugar de outras implementações que tiveram dificuldade de se adaptar a ambiente tão restrito de recursos, como o caso do DTLS e TinyDTLS.

Apesar disso, o CoEP não foi desenvolvido para substituir o DTLS seja em parte ou seja em integralidade nas aplicações em ambientes que não se configuram como aqueles abordados nessa dissertação. Isto é, o DTLS é um padrão largamente utilizado em soluções que não precisam ser enxutas, tem diversos recursos disponíveis e, nessas situações, sua utilização tem sido eficaz e insubstituível; ambientes como este não são adequados para a utilização do CoEP, pois suas vantagens se tornam poucas.

Outra motivação do uso do CoEP é a obrigação em utilização de segurança imposta pelo mesmo. Com o impacto esperado de WSNs, esses ambientes restritos necessitam estar naturalmente protegidos para suprir as necessidades de segurança das informações que nele trafegarão.

## 6.6 Trabalhos Futuros

O CoEP é um protocolo que foi desenvolvido por diversas razões, não apenas pela inovação tecnológica, mas também pela necessidade expressa que aplicações têm de esquemas de segurança compactos. Por isso, apesar de finalizado em sua primeira versão, o protocolo ainda passará por melhorias, otimizações e até mesmo implementações que sempre sigam sua filosofia de ser enxuto e garantir mesma ou maior segurança que protocolos similares - como TinyDTLS -, afim de se atender melhor as necessidades de aplicações e da comunidade de desenvolvedores.

Além disso, pretende-se realizar testes e ataques específicos no protocolo para avaliar potenciais melhorias e avanços de segurança para que seu tipo de arquitetura se consolide como referência em segurança e comunicação segura.

Por fim, dada oportunidade, tais trabalhos futuros serão realizados parcial ou integramente em uma progressão acadêmica do autor através de uma defesa de tese de doutorado.

## 6.7 Publicações do CoEP

- Durante o desenvolvimento do CoEP, seus resultados parciais e funcionais no momento da publicação foram exibidos durante o congresso IEEE INTERCON 2018 no Perú, através da publicação "CoEP: A secure lightweight application protocol for the Internet of Things"(MANINI et al., 2018). Nesta, o CoEP ainda não tinha toda arquitetura estabelecida como a exibida nesse trabalho completo, diferenciando-se principalmente de como a camada era implementada sob as demais e de como era feita a geração das chaves simétricas e de *token* após a troca de chaves assimétricas.

- Além dessa publicação acadêmica, o protocolo foi implementado em uma solução de Internet das Coisas em empresa parceira, o que resultou na publicação do artigo "SPIRI: Low Power IoT Solution for Monitoring Indoor Air Quality"(ESQUIAGOLA et al., 2018) no congresso IoTTBS 2018 em Portugal.

# REFERÊNCIAS


ALGHAMDI, T. A.; LASEBAE, A.; AIASH, M. Security analysis of the constrained application protocol in the internet of things. In: IEEE. **Future Generation Communication Technology (FGCT), 2013 second international conference on**. [S.l.], 2013. p. 163–168.

ALLIANCE, Z. Zigbee 2007 specification. **Online: http://www. zigbee. org/Specifications/ZigBee/Overview. aspx**, v. 45, p. 120, 2007.

BACCELLI, E. et al. **RIOT: One OS to rule them all in the IoT**. Tese (Doutorado) — INRIA, 2012.

BARKER, E. et al. Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. **NIST special publication**, Citeseer, v. 800, p. 56A, 2013.

BARKER, E.; DANG, Q. **NIST Special Publication 800-57 Part 3 Revision 1**. [S.l.]: NIST, 2016. 12 p.

BARKER, E.; ROGINSKY, A. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. **NIST Special Publication**, v. 800, p. 131A, 2011.

BARRY, R. **FreeRTOS, a FREE open source RTOS for small embedded real time systems**. 2003.

BEATON, W. **Eclipse tinydtls**. 2016. Disponível em: <https://projects.eclipse.org/projects/iot.tinydtls>.

BLAKE-WILSON, S. et al. Rfc4492: Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls). **RFC4492, Internet Engineering Task Force**, 2006.

BORMANN, C.; ERSUE, M.; KERANEN, A. **RFC 7228: Terminology for Constrained-Node Networks**. [S.l.], 2014.

BUSINESSWIRE. **Advanced Sensors for Smart Buildings Will Reach Nearly $3.7 Billion in Annual Revenue by 2020, Forecasts Navigant Research**. 2014. Disponível em: <http://www.businesswire.com/news/home/20140319005479/en/Advanced-Sensors-Smart-Buildings-Reach-3.7-Billion>.

CAPOSSELE, A. et al. Security as a coap resource: an optimized dtls implementation for the iot. In: IEEE. **Communications (ICC), 2015 IEEE International Conference on**. [S.l.], 2015. p. 549–554.

CHAN, C. K. et al. High-performance lithium battery anodes using silicon nanowires. **Nature nanotechnology**, Nature Publishing Group, v. 3, n. 1, p. 31–35, 2008.

CHANDRAKASAN, A. P.; SHENG, S.; BRODERSEN, R. W. Low-power cmos digital design. **IEICE Transactions on Electronics**, The Institute of Electronics, Information and Communication Engineers, v. 75, n. 4, p. 371–382, 1992.

COLLINA, M. et al. Internet of things application layer protocol analysis over error and delay prone links. In: IEEE. **Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC), 2014 7th**. [S.l.], 2014. p. 398–404.

DAEMEN, J.; RIJMEN, V. Aes proposal: Rijndael. 1999.

DIERKS, E. R. T. **The Transport Layer Security (TLS) Protocol Version 1.2**. [S.l.]: IETF, 2008.

DIFFIE, W.; HELLMAN, M. New directions in cryptography. **IEEE transactions on Information Theory**, IEEE, v. 22, n. 6, p. 644–654, 1976.

DIGICERT. **Behind the Scenes of SSL Cryptography**. 2018. Disponível em: <https://www.digicert.com/ssl-cryptography.htm>.

DUNKELS, A. The contikimac radio duty cycling protocol. Swedish Institute of Computer Science, 2011.

ESQUIAGOLA, J. et al. Spiri: Low power iot solution for monitoring indoor air quality. In: **IoTBDS**. [S.l.: s.n.], 2018. p. 285–290.

GNU. 2019. Disponível em: <http://gmplib.org/manual/Random-Number-Algorithms.html>.

GOMEZ, C.; OLLER, J.; PARADELLS, J. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. **Sensors**, Molecular Diversity Preservation International, v. 12, n. 9, p. 11734–11753, 2012.

GROUP, I. . W. et al. Ieee standard for local and metropolitan area networks—part 15.4: Low-rate wireless personal area networks (lr-wpans). **IEEE Std**, v. 802, p. 4–2011, 2011.

GUBBI, J. et al. Internet of things (iot): A vision, architectural elements, and future directions. **Future generation computer systems**, Elsevier, v. 29, n. 7, p. 1645–1660, 2013.

HAHM, O. et al. Operating systems for low-end devices in the internet of things: a survey. **IEEE Internet of Things Journal**, IEEE, v. 3, n. 5, p. 720–734, 2016.

HE, T. et al. Speed: A stateless protocol for real-time communication in sensor networks. In: IEEE. **Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on**. [S.l.], 2003. p. 46–55.

ISO. ISO, **Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1**.

JONSSON, J. et al. Pkcs# 1: Rsa cryptography specifications version 2.2. 2016.

KARAGIANNIS, V. et al. A survey on application layer protocols for the internet of things. **Transaction on IoT and Cloud Computing**, v. 3, n. 1, p. 11–17, 2015.

KAYAL, P. et al. A comparison of iot application layer protocols through a smart parking implementation. 2016.

LEE, E. A. et al. The swarm at the edge of the cloud. **IEEE Design & Test**, IEEE, v. 31, n. 3, p. 8–20, 2014.

LEVIS, P. Experiences from a decade of tinyos development. In: **OSDI**. [S.l.: s.n.], 2012. p. 207–220.

LORENZ, L. **The Market for Advanced Sensors in Intelligent Buildings Is Expected to Exceed $1 Billion in 2016**. 2016. Disponível em: <https://www.worldbuild365.com/news/z7tyeooz9/hvac/ascension-of-advanced-sensors-market-value-set-to-exceed-1billion>.

LU, C. et al. Rap: A real-time communication architecture for large-scale wireless sensor networks. In: IEEE. **Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE**. [S.l.], 2002. p. 55–66.

MANINI, M. et al. Coep: A secure & lightweight application protocol for the internet of things. In: IEEE. **2018 IEEE XXV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)**. [S.l.], 2018. p. 1–4.

MARETKSANDMARKETS. **Smart Building Market by Type (Building Automation Software, Services), Building Type (Intelligent Security System, Building Energy Management System, Infrastructure Management, and Network Management System), and Region - Global Forecast to 2022**. Disponível em: <http://www.marketsandmarkets.com/PressReleases/intelligent-building-automation-technologies-market.asp>.

MARKOVIC, D. et al. Ultralow-power design in near-threshold region. **Proceedings of the IEEE**, IEEE, v. 98, n. 2, p. 237–252, 2010.

MARTIN, R. **Wireless Control Systems for Smart Buildings are Expected to Reach $434 Million in Annual Revenue by 2023**. 2014. Disponível em: <https://www.ecmweb.com/lighting-control/wireless-control-systems-smart-buildings-are-expected-reach-434-million-annual-reve>.

MATLAB. 2019. Disponível em: <https://www.gnu.org/software/octave/doc/interpreter/Special-Utility-Matrices.html>.

MÉLARD, G. On the accuracy of statistical procedures in microsoft excel 2010. **Computational statistics**, Springer, v. 29, n. 5, p. 1095–1128, 2014.

MILLER, V. S. Use of elliptic curves in cryptography. In: SPRINGER. **Conference on the Theory and Application of Cryptographic Techniques**. [S.l.], 1985. p. 417–426.

NIEMINEN, J. et al. **RFC 7668-IPv6 over BLUETOOTH (R) Low Energy**. [S.l.]: IETF, 2015.

NIST. 197: Advanced encryption standard (aes). **Federal information processing standards publication**, v. 197, n. 441, p. 0311, 2001.

NIST, F. P. **180-1: Secure hash standard**. [S.l.]: April, 1995.

OCTAVE. 2019. Disponível em: <https://www.gnu.org/software/octave/doc/interpreter/Special-Utility-Matrices.html>.

OH, S.; KIM, J.-H.; FOX, G. Real-time performance analysis for publish/subscribe systems. **Future Generation Computer Systems**, Elsevier, v. 26, n. 3, p. 318–323, 2010.

PAPADOPOULOS, G. Z.; MONTAVONT, N. Multi-path selection in rpl based on replication and elimination. **Ad Hoc Mobile Wireless Networks: Protocols and Systems**, Springer, v. 11104, p. 15, 2002.

PETER, S. Rfc 6122: Extensible messaging and presence protocol (xmpp): Address format. **URL: http://tools.ietf.org/html/rfc6122**, 2011.

____. Rfc7622: Extensible messaging and presence protocol (xmpp): Address format. 2015.

PIEDRA, A. de la et al. Wireless sensor networks for environmental research: A survey on limitations and challenges. In: IEEE. **EUROCON, 2013 IEEE**. [S.l.], 2013. p. 267–274.

PISCARI. 2018. Disponível em: <http://www.omni-electronica.com.br/p/piscari>.

PLOENNIGS, J.; RYSSEL, U.; KABITZSCH, K. Performance analysis of the enocean wireless sensor network protocol. In: IEEE. **Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on**. [S.l.], 2010. p. 1–9.

PRIYA, S.; INMAN, D. J. **Energy harvesting technologies**. [S.l.]: Springer, 2009. v. 21.

R., M. **Commercial Building Automation Systems Revenue is Expected to Total More than $713 Billion from 2015-2023**. 2015. Disponível em: <https://www.businesswire.com/news/home/20150305005094/en/Commercial-Building-Automation-Systems-Revenue-Expected-Total>.

RAZA, S. et al. Lithe: Lightweight secure coap for the internet of things. **IEEE Sensors Journal**, IEEE, v. 13, n. 10, p. 3711–3720, 2013.

RESCORLA, E.; MODADUGU, N. Rfc 6347, datagram transport layer security version 1.2. **Internet Engineering Task Force**, 2012.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. **Communications of the ACM**, ACM, v. 21, n. 2, p. 120–126, 1978.

ROHAN. **Smart Building Market worth 31.74 Billion USD by 2022**. Disponível em: <http://www.marketsandmarkets.com/PressReleases/smart-building.asp>.

SAHI, M. **Robotics Technology Breakthroughs**. 2016. Disponível em: <https://www.tractica.com/automation-robotics/robotics-technology-breakthroughs-part-1/>.

SAITO, M.; MATSUMOTO, M. M. Tiny mersenne twister (tinymt): a small-sized variant of mersenne twister. p. 44, 2011.

SASAKI, K. R. et al. Ground plane influence on enhanced dynamic threshold utbb soi nmosfets. In: IEEE. **Devices, Circuits and Systems (ICCDCS), 2014 International Caribbean Conference on**. [S.l.], 2014. p. 1–4.

SCHANDY, J.; STEINFELD, L.; SILVEIRA, F. Average power consumption breakdown of wireless sensor network nodes using ipv6 over llns. In: IEEE. **Distributed Computing in Sensor Systems (DCOSS), 2015 International Conference on**. [S.l.], 2015. p. 242–247.

SHELBY, Z.; HARTKE, K.; BORMANN, C. Rfc 7252: The constrained application protocol, online. **IETF Std**, 2014.

STANKOVIC, J. A. Research challenges for wireless sensor networks. **ACM SIGBED Review**, ACM, v. 1, n. 2, p. 9–12, 2004.

STONE, H. **High Performance Computer Architecture**. [S.l.]: Addison-Wesley, 1993.

THUBERT, P.; HUI, J. W. Compression format for ipv6 datagrams over ieee 802.15. 4-based networks. 2011.

YASSEIN, M. B.; SHATNAWI, M. Q. et al. Application layer protocols for the internet of things: A survey. In: IEEE. **Engineering & MIS (ICEMIS), International Conference on**. [S.l.], 2016. p. 1–4.

Z. HARTKE K., B. C. S. Rfc 7252: The constrained application protocol, online. **IETF Std**, 2014.

ZIMMERMANN, H. Osi reference model–the iso model of architecture for open systems interconnection. **IEEE Transactions on communications**, IEEE, v. 28, n. 4, p. 425–432, 1980.

# APÊNDICE A – IMPLEMENTAÇÃO DO COEP NO ARQUIVO DE SISTEMA DO CONTIKI-NG CONTIKI/OS/NET/APP-LAYER/COAP/COAP-UIP.C

```
/*
 * Copyright (c) 2016, SICS, Swedish ICT AB.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the Institute nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ‘‘AS IS’’ AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
```

```
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

/**
 * \file
 *        CoAP transport implementation for uIPv6
 * \author
 *        Niclas Finne <nfi@sics.se>
 *        Joakim Eriksson <joakime@sics.se>
 *        Matheus Barros Manini <mangine@usp.br>
 */

/**
 * \addtogroup coap-transport
 * @{
 *
 * \defgroup coap-uip CoAP transport implementation for uIP
 * @{
 *
 * This is an implementation of CoAP transport and CoAP endpoint over uIP
 * with DTLS and CoEP support (implemented by Matheus Master's thesis).
 */

#include "contiki.h"
#include "net/ipv6/uip-udp-packet.h"
#include "net/ipv6/uiplib.h"
#include "net/routing/routing.h"
#include "coap.h"
#include "coap-engine.h"
#include "coap-endpoint.h"
#include "coap-transport.h"
```

```
#include "coap-transactions.h"
#include "coap-constants.h"
#include "coap-keystore.h"
#include "coap-keystore-simple.h"

/* Log configuration */
#include "coap-log.h"
#define LOG_MODULE "coap-uip"
#define LOG_LEVEL LOG_LEVEL_COAP

#ifdef WITH_DTLS
#include "tinydtls.h"
#include "dtls.h"
#endif /* WITH_DTLS */


#ifdef WITH_COEP
#include "net/coep/coep.h"
#include "net/coep/connectors/coep_ecdh.h"

int coep_coap_receive(COEP_IP_TYPE * from, uint8_t msgid, uint8_t * buffer, int
    maxlength);

//modulo de entrega de dados ao coap
coep_module_t coep_coap = { &coep_coap_receive, NULL, 80, 80 };
COEP_MODULES(&coep_coap);

#endif /* WITH_COEP */

/* sanity check for configured values */
#if COAP_MAX_PACKET_SIZE > (UIP_BUFSIZE - UIP_IPH_LEN - UIP_UDPH_LEN)
#error "UIP_CONF_BUFFER_SIZE too small for COAP_MAX_CHUNK_SIZE"
#endif

#define SERVER_LISTEN_PORT   UIP_HTONS(COAP_DEFAULT_PORT)
```

```
#define SERVER_LISTEN_SECURE_PORT UIP_HTONS(COAP_DEFAULT_SECURE_PORT)

#ifdef WITH_DTLS
static dtls_handler_t cb;
static dtls_context_t *dtls_context = NULL;

static const coap_keystore_t *dtls_keystore = NULL;
static struct uip_udp_conn *dtls_conn = NULL;
#endif /* WITH_DTLS */

PROCESS(coap_engine, "CoAP Engine");

static struct uip_udp_conn *udp_conn = NULL;

/*---------------------------------------------------------------------------*/
void
coap_endpoint_log(const coap_endpoint_t *ep)
{
  if(ep == NULL) {
    LOG_OUTPUT("(NULL EP)");
    return;
  }
  if(ep->secure) {
    LOG_OUTPUT("coaps://[");
  } else {
    LOG_OUTPUT("coap://[");
  }
  log_6addr(&ep->ipaddr);
  LOG_OUTPUT("]:%u", uip_ntohs(ep->port));
}
/*---------------------------------------------------------------------------*/
void
coap_endpoint_print(const coap_endpoint_t *ep)
{
  if(ep == NULL) {
```

```
    printf("(NULL EP)");
    return;
  }
  if(ep->secure) {
    printf("coaps://[");
  } else {
    printf("coap://[");
  }
  uiplib_ipaddr_print(&ep->ipaddr);
  printf("]:%u", uip_ntohs(ep->port));
}
/*---------------------------------------------------------------------------*/
int
coap_endpoint_snprint(char *buf, size_t size, const coap_endpoint_t *ep)
{
  int n;
  if(buf == NULL || size == 0) {
    return 0;
  }
  if(ep == NULL) {
    n = snprintf(buf, size - 1, "(NULL EP)");
  } else {
    if(ep->secure) {
      n = snprintf(buf, size - 1, "coaps://[");
    } else {
      n = snprintf(buf, size - 1, "coap://[");
    }
    if(n < size - 1) {
      n += uiplib_ipaddr_snprint(&buf[n], size - n - 1, &ep->ipaddr);
    }
    if(n < size - 1) {
      n += snprintf(&buf[n], size -n - 1, "]:%u", uip_ntohs(ep->port));
    }
  }
  if(n >= size - 1) {
```

```
    buf[size - 1] = '\0';
  }
  return n;
}
/*---------------------------------------------------------------------------*/
void
coap_endpoint_copy(coap_endpoint_t *destination,
                   const coap_endpoint_t *from)
{
  uip_ipaddr_copy(&destination->ipaddr, &from->ipaddr);
  destination->port = from->port;
  destination->secure = from->secure;
}
/*---------------------------------------------------------------------------*/
int
coap_endpoint_cmp(const coap_endpoint_t *e1, const coap_endpoint_t *e2)
{
  if(!uip_ipaddr_cmp(&e1->ipaddr, &e2->ipaddr)) {
    return 0;
  }
  return e1->port == e2->port && e1->secure == e2->secure;
}
/*---------------------------------------------------------------------------*/
static int
index_of(const char *data, int offset, int len, uint8_t c)
{
  if(offset < 0) {
    return offset;
  }
  for(; offset < len; offset++) {
    if(data[offset] == c) {
      return offset;
    }
  }
  return -1;
```

```
}
/*---------------------------------------------------------------------------*/
static int
get_port(const char *inbuf, size_t len, uint32_t *value)
{
  int i;
  *value = 0;
  for(i = 0; i < len; i++) {
    if(inbuf[i] >= '0' && inbuf[i] <= '9') {
      *value = *value * 10 + (inbuf[i] - '0');
    } else {
      break;
    }
  }
  return i;
}
/*---------------------------------------------------------------------------*/
int
coap_endpoint_parse(const char *text, size_t size, coap_endpoint_t *ep)
{
  /* Only IPv6 supported */
  int start = index_of(text, 0, size, '[');
  int end = index_of(text, start, size, ']');
  uint32_t port;

  ep->secure = strncmp(text, "coaps:", 6) == 0;
  if(start >= 0 && end > start &&
     uiplib_ipaddrconv(&text[start], &ep->ipaddr)) {
    if(text[end + 1] == ':' &&
       get_port(text + end + 2, size - end - 2, &port)) {
      ep->port = UIP_HTONS(port);
    } else if(ep->secure) {
      /* Use secure CoAP port by default for secure endpoints. */
      ep->port = SERVER_LISTEN_SECURE_PORT;
    } else {
```

```
      ep->port = SERVER_LISTEN_PORT;
    }
    return 1;
  } else if(size < UIPLIB_IPV6_MAX_STR_LEN) {
    char buf[UIPLIB_IPV6_MAX_STR_LEN];
    memcpy(buf, text, size);
    buf[size] = '\0';
    if(uiplib_ipaddrconv(buf, &ep->ipaddr)) {
      ep->port = SERVER_LISTEN_PORT;
      return 1;
    }
  }
  return 0;
}
/*---------------------------------------------------------------------------*/
static const coap_endpoint_t *
get_src_endpoint(uint8_t secure)
{
  static coap_endpoint_t src;
  uip_ipaddr_copy(&src.ipaddr, &UIP_IP_BUF->srcipaddr);
  src.port = UIP_UDP_BUF->srcport;
  src.secure = secure;
  return &src;
}
/*---------------------------------------------------------------------------*/
int
coap_endpoint_is_secure(const coap_endpoint_t *ep)
{
  return ep->secure;
}
/*---------------------------------------------------------------------------*/
int
coap_endpoint_is_connected(const coap_endpoint_t *ep)
{
#ifndef CONTIKI_TARGET_NATIVE
```

```
       if (!uip_is_addr_linklocal(&ep->ipaddr)
              && NETSTACK_ROUTING.node_is_reachable() == 0) {
              return 0;
       }
#endif

#ifdef WITH_COEP
       if (ep != NULL && ep->secure != 0) {
              coep_connection_t * c = coep_connection_get((uip_ipaddr_t *)&ep->
                  ipaddr);
              if (c != NULL && c->status.flags &
                  COEP_CONNECTION_STATUS_REACHABLE) {
                     return 1;
              }
       }
       return 0;
#endif

#ifdef WITH_DTLS
  if(ep != NULL && ep->secure != 0) {
    dtls_peer_t *peer;
    if(dtls_context == NULL) {
      return 0;
    }
    peer = dtls_get_peer(dtls_context, ep);
    if(peer != NULL) {
      /* only if handshake is done! */
      LOG_DBG("DTLS peer state for ");
      LOG_DBG_COAP_EP(ep);
      LOG_DBG_(" is %d (%sconnected)\n", peer->state,
             dtls_peer_is_connected(peer) ? "" : "not ");
      return dtls_peer_is_connected(peer);
    } else {
      LOG_DBG("DTLS did not find peer ");
      LOG_DBG_COAP_EP(ep);
```

```
      LOG_DBG_("\n");
      return 0;
    }
  }
#endif /* WITH_DTLS */

  /* Assume connected */
  return 1;
}
/*---------------------------------------------------------------------------*/
int
coap_endpoint_connect(coap_endpoint_t *ep)
{
  if(ep->secure == 0) {
    LOG_DBG("connect to ");
    LOG_DBG_COAP_EP(ep);
    LOG_DBG_("\n");
    return 1;
  }


#ifdef WITH_COEP
  LOG_DBG("COEP connect to ");
  LOG_DBG_COAP_EP(ep);
  LOG_DBG_("\n");
  return coep_connect_to(&ep->ipaddr, COEP_CONNECTION_STATUS_ISROOT);
#endif

#ifdef WITH_DTLS
  LOG_DBG("DTLS connect to ");
  LOG_DBG_COAP_EP(ep);
  LOG_DBG_("\n");

  /* setup all address info here... should be done to connect */
  if(dtls_context) {
```

```
    dtls_connect(dtls_context, ep);
    return 1;
  }
#endif /* WITH_DTLS */

  return 0;
}
/*---------------------------------------------------------------------------*/
void
coap_endpoint_disconnect(coap_endpoint_t *ep)
{
#ifdef WITH_DTLS
  if(ep && ep->secure && dtls_context) {
    dtls_close(dtls_context, ep);
  }
#endif /* WITH_DTLS */

#ifdef WITH_COEP
       if (ep && ep->secure) {
              coep_disconnect_from(&ep->ipaddr);
       }
#endif
}
/*---------------------------------------------------------------------------*/
uint8_t *
coap_databuf(void)
{
  return uip_appdata;
}
/*---------------------------------------------------------------------------*/
void
coap_transport_init(void)
{
  process_start(&coap_engine, NULL);
```

```
#ifdef WITH_COEP
  coep_init();
#endif

#ifdef WITH_DTLS
  dtls_init();

#if COAP_DTLS_KEYSTORE_CONF_WITH_SIMPLE
  coap_keystore_simple_init();
#endif /* COAP_DTLS_KEYSTORE_CONF_WITH_SIMPLE */

#endif /* WITH_DTLS */
}
/*---------------------------------------------------------------------------*/
#ifdef WITH_COEP
static void process_secure_data_coep(void) {
       LOG_INFO("CoEP receiving secure UDP datagram from [");
       LOG_INFO_6ADDR(&UIP_IP_BUF->srcipaddr);
       LOG_INFO_("]:%u\n", uip_ntohs(UIP_UDP_BUF->srcport));
       LOG_INFO(" Length: %u\n", uip_datalen());

       coap_endpoint_t * ep = (coap_endpoint_t *)get_src_endpoint(1);
       coep_input(&ep->ipaddr, uip_appdata, uip_datalen());
}
int coep_coap_receive(COEP_IP_TYPE * from, uint8_t msgid, uint8_t * buffer, int
     maxlength) {
       return coap_receive(get_src_endpoint(0), buffer, maxlength);
}
void coep_output(COEP_IP_TYPE * ip, uint8_t * buffer, uint8_t bufferlen) {
       uip_udp_packet_sendto(udp_conn, buffer, bufferlen, ip, 0xC0E2);
}
#endif
/*---------------------------------------------------------------------------*/
#ifdef WITH_DTLS
static void
```

```
process_secure_data(void)
{
  LOG_INFO("receiving secure UDP datagram from [");
  LOG_INFO_6ADDR(&UIP_IP_BUF->srcipaddr);
  LOG_INFO_("]:%u\n", uip_ntohs(UIP_UDP_BUF->srcport));
  LOG_INFO(" Length: %u\n", uip_datalen());

  if(dtls_context) {
    dtls_handle_message(dtls_context, (coap_endpoint_t *)get_src_endpoint(1),
                        uip_appdata, uip_datalen());
  }
}
#endif /* WITH_DTLS */
/*---------------------------------------------------------------------------*/
static void
process_data(void)
{
  LOG_INFO("receiving UDP datagram from [");
  LOG_INFO_6ADDR(&UIP_IP_BUF->srcipaddr);
  LOG_INFO_("]:%u\n", uip_ntohs(UIP_UDP_BUF->srcport));
  LOG_INFO(" Length: %u\n", uip_datalen());

  coap_receive(get_src_endpoint(0), uip_appdata, uip_datalen());
}
/*---------------------------------------------------------------------------*/
int
coap_sendto(const coap_endpoint_t *ep, const uint8_t *data, uint16_t length)
{
  if(ep == NULL) {
    LOG_WARN("failed to send - no endpoint\n");
    return -1;
  }

  if(!coap_endpoint_is_connected(ep)) {
    LOG_WARN("endpoint ");
```

```
    LOG_WARN_COAP_EP(ep);
    LOG_WARN_(" not connected - dropping packet\n");
    return -1;
  }


#ifdef WITH_COEP
       if (coap_endpoint_is_secure(ep)) {
              coep_connection_t * c = coep_connection_get((uip_ipaddr_t *)&ep->
                  ipaddr);
              if (c == NULL) return -1;
              int ret = coep_send(c, &coep_coap, 1, COEP_REPLY_NO, 80, (uint8_t
                   *)data, length);

              LOG_INFO("sent CoEP to ");
              LOG_INFO_COAP_EP(ep);
              if (ret < 0) {
                     LOG_INFO_(" - error %d\n", ret);
              }
              else {
                     LOG_INFO_(" %d/%u bytes\n", ret, length);
              }
              return ret;
       }

       LOG_INFO("sent to ");
       LOG_INFO_COAP_EP(ep);
       LOG_INFO_(" %u bytes\n", length);
       return length;
#endif


#ifdef WITH_DTLS
  if(coap_endpoint_is_secure(ep)) {
    if(dtls_context) {
```

```
      int ret;

      ret = dtls_write(dtls_context, (session_t *)ep, (uint8_t *)data, length);
      LOG_INFO("sent DTLS to ");
      LOG_INFO_COAP_EP(ep);
      if(ret < 0) {
        LOG_INFO_(" - error %d\n", ret);
      } else {
        LOG_INFO_(" %d/%u bytes\n", ret, length);
      }
      return ret;
    } else {
      LOG_WARN("no DTLS context\n");
      return -1;
    }
  }
#endif /* WITH_DTLS */


  uip_udp_packet_sendto(udp_conn, data, length, &ep->ipaddr, ep->port);
  LOG_INFO("sent to ");
  LOG_INFO_COAP_EP(ep);
  LOG_INFO_(" %u bytes\n", length);
  return length;
}
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(coap_engine, ev, data)
{
  PROCESS_BEGIN();

  /* new connection with remote host */
  udp_conn = udp_new(NULL, 0, NULL);
  udp_bind(udp_conn, SERVER_LISTEN_PORT);
  LOG_INFO("Listening on port %u\n", uip_ntohs(udp_conn->lport));
```

```
#ifdef WITH_DTLS
  /* create new context with app-data */
  dtls_conn = udp_new(NULL, 0, NULL);
  if(dtls_conn != NULL) {
    udp_bind(dtls_conn, SERVER_LISTEN_SECURE_PORT);
    LOG_INFO("DTLS listening on port %u\n", uip_ntohs(dtls_conn->lport));
    dtls_context = dtls_new_context(dtls_conn);
  }
  if(!dtls_context) {
    LOG_WARN("DTLS: cannot create context\n");
  } else {
    dtls_set_handler(dtls_context, &cb);
  }
#endif /* WITH_DTLS */

  while(1) {
    PROCESS_YIELD();

    if(ev == tcpip_event) {
      if(uip_newdata()) {
#ifdef WITH_DTLS
        if(uip_udp_conn == dtls_conn) {
          process_secure_data();
          continue;
        }
#endif /* WITH_DTLS */

#ifdef WITH_COEP
              process_secure_data_coep();
              continue;
#endif /* WITH_COEP */
        process_data();
      }
    }
```

```
  } /* while (1) */

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/

/* DTLS */
#ifdef WITH_DTLS

/* This is input coming from the DTLS code - e.g. de-crypted input from
   the other side - peer */
static int
input_from_peer(struct dtls_context_t *ctx,
              session_t *session, uint8_t *data, size_t len)
{
  size_t i;

  if(LOG_DBG_ENABLED) {
    LOG_DBG("received DTLS data:");
    for(i = 0; i < len; i++) {
      LOG_DBG_("%c", data[i]);
    }
    LOG_DBG_("\n");
    LOG_DBG("Hex:");
    for(i = 0; i < len; i++) {
      LOG_DBG_("%02x", data[i]);
    }
    LOG_DBG_("\n");
  }

  /* Ensure that the endpoint is tagged as secure */
  session->secure = 1;

  coap_receive(session, data, len);
```

```
  return 0;
}

/* This is output from the DTLS code to be sent to peer (encrypted) */
static int
output_to_peer(struct dtls_context_t *ctx,
             session_t *session, uint8_t *data, size_t len)
{
  struct uip_udp_conn *udp_connection = dtls_get_app_data(ctx);
  LOG_DBG("output_to DTLS peer [");
  LOG_DBG_6ADDR(&session->ipaddr);
  LOG_DBG_("]:%u %ld bytes\n", uip_ntohs(session->port), (long)len);
  uip_udp_packet_sendto(udp_connection, data, len,
                      &session->ipaddr, session->port);
  return len;
}

/* This defines the key-store set API since we hookup DTLS here */
void
coap_set_keystore(const coap_keystore_t *keystore)
{
  dtls_keystore = keystore;
}

/* This function is the "key store" for tinyDTLS. It is called to
 * retrieve a key for the given identity within this particular
 * session. */
static int
get_psk_info(struct dtls_context_t *ctx,
           const session_t *session,
           dtls_credentials_type_t type,
           const unsigned char *id, size_t id_len,
           unsigned char *result, size_t result_length)
{
  coap_keystore_psk_entry_t ks;
```

```
if(dtls_keystore == NULL) {
  LOG_DBG("--- No key store available ---\n");
  return 0;
}

memset(&ks, 0, sizeof(ks));
LOG_DBG("---===>>> Getting the Key or ID <<<===---\n");
switch(type) {
case DTLS_PSK_IDENTITY:
  if(id && id_len) {
    ks.identity_hint = id;
    ks.identity_hint_len = id_len;
    LOG_DBG("got psk_identity_hint: '");
    LOG_DBG_COAP_STRING((const char *)id, id_len);
    LOG_DBG_("'\n");
  }

  if(dtls_keystore->coap_get_psk_info) {
    /* we know that session is a coap endpoint */
    dtls_keystore->coap_get_psk_info((coap_endpoint_t *)session, &ks);
  }
  if(ks.identity == NULL || ks.identity_len == 0) {
    LOG_DBG("no psk_identity found\n");
    return 0;
  }

  if(result_length < ks.identity_len) {
    LOG_DBG("cannot return psk_identity -- buffer too small\n");
    return dtls_alert_fatal_create(DTLS_ALERT_INTERNAL_ERROR);
  }
  memcpy(result, ks.identity, ks.identity_len);
  LOG_DBG("psk_identity with %u bytes found\n", ks.identity_len);
  return ks.identity_len;
```

```
  case DTLS_PSK_KEY:
    if(dtls_keystore->coap_get_psk_info) {
      ks.identity = id;
      ks.identity_len = id_len;
      /* we know that session is a coap endpoint */
      dtls_keystore->coap_get_psk_info((coap_endpoint_t *)session, &ks);
    }
    if(ks.key == NULL || ks.key_len == 0) {
      LOG_DBG("PSK for unknown id requested, exiting\n");
      return dtls_alert_fatal_create(DTLS_ALERT_ILLEGAL_PARAMETER);
    }

    if(result_length < ks.key_len) {
      LOG_DBG("cannot return psk -- buffer too small\n");
      return dtls_alert_fatal_create(DTLS_ALERT_INTERNAL_ERROR);
    }
    memcpy(result, ks.key, ks.key_len);
    LOG_DBG("psk with %u bytes found\n", ks.key_len);
    return ks.key_len;

  default:
    LOG_WARN("unsupported key store request type: %d\n", type);
  }

  return dtls_alert_fatal_create(DTLS_ALERT_INTERNAL_ERROR);
}


static dtls_handler_t cb = {
  .write = output_to_peer,
  .read = input_from_peer,
  .event = NULL,
#ifdef DTLS_PSK
  .get_psk_info = get_psk_info,
#endif /* DTLS_PSK */
```

```
#ifdef DTLS_ECC
  /* .get_ecdsa_key = get_ecdsa_key, */
  /* .verify_ecdsa_key = verify_ecdsa_key */
#endif /* DTLS_ECC */
};

#endif /* WITH_DTLS */
/*-------------------------------------------------------------------------*/
/** @} */
/** @} */
```

# APÊNDICE B – CÓDIGO DE TESTE E VALIDAÇÃO DE APLICAÇÃO COAP

```
/*
 * Copyright (c) 2013, Institute for Pervasive Computing, ETH Zurich
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the Institute nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ‘‘AS IS’’ AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
```

```
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * This file is part of the Contiki operating system.
 */

/**
 * \file
 *      Erbium (Er) CoAP client example.
 * \author
 *      Matthias Kovatsch <kovatsch@inf.ethz.ch>
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "contiki-net.h"
#include "coap-engine.h"
#include "coap-blocking-api.h"
#if PLATFORM_SUPPORTS_BUTTON_HAL
#include "dev/button-hal.h"
#else
#include "dev/button-sensor.h"
#endif

/* Log configuration */
#include "coap-log.h"
#define LOG_MODULE "App"
#define LOG_LEVEL LOG_LEVEL_APP

/* FIXME: This server address is hard-coded for Cooja and link-local for
    unconnected border router. */
#define SERVER_EP "coap://[fe80::212:7402:0002:0202]"
```

```
#define TOGGLE_INTERVAL 10

PROCESS(er_example_client, "Erbium Example Client");
AUTOSTART_PROCESSES(&er_example_client);

static struct etimer et;

/* Example URIs that can be queried. */
#define NUMBER_OF_URLS 4
/* leading and ending slashes only for demo purposes, get cropped automatically
    when setting the Uri-Path */
char *service_urls[NUMBER_OF_URLS] =
{ ".well-known/core", "/actuators/toggle", "battery/", "error/in//path" };
#if PLATFORM_HAS_BUTTON
static int uri_switch = 0;
#endif

/* This function is will be passed to COAP_BLOCKING_REQUEST() to handle
   responses. */
void
client_chunk_handler(coap_message_t *response)
{
  const uint8_t *chunk;

  int len = coap_get_payload(response, &chunk);

  printf("|%.*s", len, (char *)chunk);
}
PROCESS_THREAD(er_example_client, ev, data)
{
  static coap_endpoint_t server_ep;
  PROCESS_BEGIN();

  static coap_message_t request[1]; /* This way the packet can be treated as
      pointer as usual. */
```

```
  coap_endpoint_parse(SERVER_EP, strlen(SERVER_EP), &server_ep);

  etimer_set(&et, TOGGLE_INTERVAL * CLOCK_SECOND);

#if PLATFORM_HAS_BUTTON
#if !PLATFORM_SUPPORTS_BUTTON_HAL
  SENSORS_ACTIVATE(button_sensor);
#endif
  printf("Press a button to request %s\n", service_urls[uri_switch]);
#endif /* PLATFORM_HAS_BUTTON */

  while(1) {
    PROCESS_YIELD();

    if(etimer_expired(&et)) {
      printf("--Toggle timer--\n");

      /* prepare request, TID is set by COAP_BLOCKING_REQUEST() */
      coap_init_message(request, COAP_TYPE_CON, COAP_POST, 0);
      coap_set_header_uri_path(request, service_urls[1]);

      const char msg[] = "Toggle!";

      coap_set_payload(request, (uint8_t *)msg, sizeof(msg) - 1);

      LOG_INFO_COAP_EP(&server_ep);
      LOG_INFO_("\n");

      COAP_BLOCKING_REQUEST(&server_ep, request, client_chunk_handler);

      printf("\n--Done--\n");

      etimer_reset(&et);
```

```
#if PLATFORM_HAS_BUTTON
#if PLATFORM_SUPPORTS_BUTTON_HAL
   } else if(ev == button_hal_release_event) {
#else
   } else if(ev == sensors_event && data == &button_sensor) {
#endif

     /* send a request to notify the end of the process */

     coap_init_message(request, COAP_TYPE_CON, COAP_GET, 0);
     coap_set_header_uri_path(request, service_urls[uri_switch]);

     printf("--Requesting %s--\n", service_urls[uri_switch]);

     LOG_INFO_COAP_EP(&server_ep);
     LOG_INFO_("\n");

     COAP_BLOCKING_REQUEST(&server_ep, request,
                           client_chunk_handler);

     printf("\n--Done--\n");

     uri_switch = (uri_switch + 1) % NUMBER_OF_URLS;
#endif /* PLATFORM_HAS_BUTTON */
   }
  }

  PROCESS_END();
}
```

# APÊNDICE C – RESULTADO DA FERRAMENTA FPVGCC PARA COAP COM COEP

| FILE | VEC | FLASH | SRAM | FLASH_CCFG | *default* | TOTAL |
|---|---|---|---|---|---|---|
| sicslowpan.o | | 5319 | 1773 | | | 7092 |
| uECC.o | | 5712 | 4 | | | 5716 |
| ieee-mode.o | | 3316 | 976 | | | 4292 |
| uip6.o | | 1912 | 1577 | | | 3489 |
| coap.o | | 2966 | 7 | | | 2973 |
| coep.o | | 2188 | 444 | | | 2632 |
| strformat.o | | 2506 | | | | 2506 |
| rpl-dag.o | | 2022 | 352 | | | 2374 |
| rpl-icmp6.o | | 2108 | 48 | | | 2156 |
| rf-core.o | | 2033 | 81 | | | 2114 |
| v7-m\libc.a | | 980 | 1068 | | | 2048 |
| uip-ds6.o | | 1456 | 400 | | | 1856 |
| coap-uip.o | | 1710 | 72 | | | 1782 |
| coap-engine.o | | 1401 | 353 | | | 1754 |
| queuebuf.o | | 212 | 1448 | | | 1660 |
| rpl-timers.o | | 1552 | 64 | | | 1616 |
| nbr-table.o | | 1136 | 292 | | | 1428 |
| rpl-ext-header.o | | 1424 | | | | 1424 |
| csma-output.o | | 1016 | 366 | | | 1382 |
| setup_rom.o | | 1288 | | | | 1288 |
| coap-transactions.o | | 328 | 820 | | | 1148 |
| frame802154.o | | 1128 | 2 | | | 1130 |

| startup_gcc.o | | 96 | | | 1024 | 1120 |
|---|---|---|---|---|---|---|
| process.o | | 708 | 396 | | | 1104 |
| uip-sr.o | | 628 | 420 | | | 1048 |
| aes-128.o | | 816 | 176 | | | 992 |
| coap-example-client.o | | 658 | 332 | | | 990 |
| coap-observe.o | | 601 | 308 | | | 909 |
| sha-256.o | | 840 | | | | 840 |
| uip-icmp6.o | | 772 | 48 | | | 820 |
| link-stats.o | | 576 | 240 | | | 816 |
| spi-arch.o | | 784 | 16 | | | 800 |
| sys_ctrl.o | | 792 | 8 | | | 800 |
| packetbuf.o | | 620 | 177 | | | 797 |
| v7-m\libgcc.a | | 780 | | | | 780 |
| rpl-neighbor.o | | 612 | 148 | | | 760 |
| cc26xx-uart.o | | 724 | 24 | | | 748 |
| contiki-main.o | | 748 | | | | 748 |
| lpm.o | | 712 | 4 | | | 716 |
| rpl.o | | 689 | 20 | | | 709 |
| soc-trng.o | | 619 | 77 | | | 696 |
| tcpip.o | | 661 | 33 | | | 694 |
| uip-ds6-nbr.o | | 368 | 304 | | | 672 |
| uip-nd6.o | | 636 | 25 | | | 661 |
| tinymt32.o | | 616 | | | | 616 |
| button-hal.o | | 555 | 35 | | | 590 |
| etimer.o | | 556 | 24 | | | 580 |
| platform.o | | 579 | | | | 579 |
| framer-802154.o | | 536 | 2 | | | 538 |
| uiplib.o | | 527 | | | | 527 |
| uipbuf.o | | 484 | 24 | | | 508 |
| serial-line.o | | 214 | 285 | | | 499 |
| ctimer.o | | 471 | 21 | | | 492 |
| coap-res-well-known-core.o | | 422 | 36 | | | 458 |
| ext-flash.o | | 448 | | | | 448 |
| uip-ds6-route.o | | 364 | 84 | | | 448 |
| coap-timer.o | | 436 | 5 | | | 441 |

| | | | | | | |
|---|---|---|---|---|---|---|
| mac-sequence.o | | 184 | 256 | | | 440 |
| rpl-mrhof.o | | 396 | 36 | | | 432 |
| soc-rtc.o | | 416 | 4 | | | 420 |
| coap-observe-client.o | | 232 | 176 | | | 408 |
| setup.o | | 388 | | | | 388 |
| list.o | | 356 | | | | 356 |
| stack-check.o | | 317 | 32 | | | 349 |
| clock.o | | 324 | 8 | | | 332 |
| coap-timer-default.o | | 283 | 44 | | | 327 |
| chipinfo.o | | 284 | | | | 284 |
| rpl-dag-root.o | | 264 | 17 | | | 281 |
| board.o | | 244 | 20 | | | 264 |
| coap-blocking-api.o | | 260 | | | | 260 |
| aux-ctrl.o | | 252 | 4 | | | 256 |
| spi.o | | 252 | | | | 252 |
| oscillators.o | | 236 | | | | 236 |
| rpl-nbr-policy.o | | 216 | 10 | | | 226 |
| leds.o | | 216 | | | | 216 |
| memb.o | | 216 | | | | 216 |
| csma.o | | 201 | | | | 201 |
| board-buttons.o | | 21 | 172 | | | 193 |
| ddi.o | | 188 | | | | 188 |
| contiki-watchdog.o | | 160 | | | | 160 |
| gpio-hal-arch.o | | 156 | | | | 156 |
| log.o | | 144 | 4 | | | 148 |
| uip-udp-packet.o | | 132 | | | | 132 |
| osc.o | | 112 | | | | 112 |
| netstack.o | | 100 | 4 | | | 104 |
| snprintf.o | | 96 | | | | 96 |
| ccfg.o | | | | 88 | | 88 |
| coep_abstraction.o | | 76 | | | | 76 |
| timer.o | | 72 | | | | 72 |
| gpio-hal.o | | 64 | 4 | | | 68 |
| dbg.o | | 64 | | | | 64 |
| printf.o | | 56 | 8 | | | 64 |

| ieee-addr.o | | 60 | | | | 60 |
|---|---|---|---|---|---|---|
| trng.o | | 56 | | | | 56 |
| ringbuf.o | | 52 | | | | 52 |
| rtimer.o | | 44 | 4 | | | 48 |
| stimer.o | | 48 | | | | 48 |
| aon_batmon.o | | 48 | | | | 48 |
| pwr_ctrl.o | | 48 | | | | 48 |
| linkaddr.o | | 32 | 8 | | | 40 |
| adi.o | | 40 | | | | 40 |
| aon_rtc.o | | 36 | | | | 36 |
| csma-security.o | | 36 | | | | 36 |
| node-id.o | | 24 | 2 | | | 26 |
| fault-handlers.o | | 24 | | | | 24 |
| int-master.o | | 24 | | | | 24 |
| autostart.o | | 24 | | | | 24 |
| mac.o | | 24 | | | | 24 |
| rtimer-arch.o | | 16 | | | | 16 |
| gpio-interrupt.o | | 16 | | | | 16 |
| random.o | | 16 | | | | 16 |
| cpu.o | | 16 | | | | 16 |
| rf-ble.o | | 12 | 1 | | | 13 |
| energest.o | | 4 | | | | 4 |
| leds-arch.o | | 4 | | | | 4 |
| TOTALS | 0 | 72043 | 14233 | 88 | 1024 | |

# APÊNDICE D – RESULTADO DA FERRAMENTA FPVGCC PARA COAP COM TINYDTLS

| FILE | VEC | FLASH | SRAM | FLASH_CCFG | *default* | TOTAL |
|---|---|---|---|---|---|---|
| dtls.o | | 10093 | | | | 10093 |
| sicslowpan.o | | 5319 | 1773 | | | 7092 |
| rijndael.o | | 6768 | | | | 6768 |
| ieee-mode.o | | 3316 | 976 | | | 4292 |
| ecc.o | | 3740 | | | | 3740 |
| uip6.o | | 1912 | 1577 | | | 3489 |
| coap.o | | 2966 | 7 | | | 2973 |
| strformat.o | | 2510 | | | | 2510 |
| rpl-dag.o | | 2022 | 352 | | | 2374 |
| v7-m\libc.a | | 1212 | 1068 | | | 2280 |
| dtls-crypto.o | | 1760 | 496 | | | 2256 |
| rpl-icmp6.o | | 2108 | 48 | | | 2156 |
| rf-core.o | | 2033 | 81 | | | 2114 |
| uip-ds6.o | | 1456 | 400 | | | 1856 |
| coap-engine.o | | 1401 | 353 | | | 1754 |
| queuebuf.o | | 212 | 1448 | | | 1660 |
| rpl-timers.o | | 1552 | 64 | | | 1616 |
| dtls-ccm.o | | 1536 | 4 | | | 1540 |
| netq.o | | 321 | 1140 | | | 1461 |
| nbr-table.o | | 1136 | 292 | | | 1428 |
| rpl-ext-header.o | | 1424 | | | | 1424 |
| csma-output.o | | 1016 | 366 | | | 1382 |

| | | | | | | |
|---|---|---|---|---|---|---|
| coap-uip.o | | 1212 | 76 | | | 1288 |
| setup_rom.o | | 1288 | | | | 1288 |
| sha2.o | | 1234 | | | | 1234 |
| coap-transactions.o | | 328 | 820 | | | 1148 |
| frame802154.o | | 1128 | 2 | | | 1130 |
| startup_gcc.o | | 96 | | | 1024 | 1120 |
| process.o | | 708 | 396 | | | 1104 |
| uip-sr.o | | 628 | 420 | | | 1048 |
| coap-example-client.o | | 658 | 332 | | | 990 |
| coap-observe.o | | 601 | 308 | | | 909 |
| dtls-hmac.o | | 374 | 520 | | | 894 |
| uip-icmp6.o | | 772 | 48 | | | 820 |
| link-stats.o | | 576 | 240 | | | 816 |
| spi-arch.o | | 784 | 16 | | | 800 |
| sys_ctrl.o | | 792 | 8 | | | 800 |
| packetbuf.o | | 620 | 177 | | | 797 |
| v7-m\libgcc.a | | 780 | | | | 780 |
| rpl-neighbor.o | | 612 | 148 | | | 760 |
| cc26xx-uart.o | | 724 | 24 | | | 748 |
| contiki-main.o | | 748 | | | | 748 |
| lpm.o | | 712 | 4 | | | 716 |
| rpl.o | | 689 | 17 | | | 706 |
| soc-trng.o | | 619 | 77 | | | 696 |
| tcpip.o | | 661 | 33 | | | 694 |
| uip-ds6-nbr.o | | 368 | 304 | | | 672 |
| uip-nd6.o | | 636 | 25 | | | 661 |
| dtls-support.o | | 196 | 447 | | | 643 |
| button-hal.o | | 555 | 35 | | | 590 |
| etimer.o | | 556 | 24 | | | 580 |
| platform.o | | 579 | | | | 579 |
| framer-802154.o | | 536 | 2 | | | 538 |
| uiplib.o | | 527 | | | | 527 |
| uipbuf.o | | 484 | 24 | | | 508 |
| serial-line.o | | 214 | 285 | | | 499 |
| ctimer.o | | 471 | 21 | | | 492 |

| | | | | | | |
|---|---|---|---|---|---|---|
| coap-res-well-known-core.o | | 422 | 36 | | | 458 |
| ext-flash.o | | 448 | | | | 448 |
| uip-ds6-route.o | | 364 | 84 | | | 448 |
| coap-timer.o | | 436 | 5 | | | 441 |
| mac-sequence.o | | 184 | 256 | | | 440 |
| rpl-mrhof.o | | 396 | 36 | | | 432 |
| soc-rtc.o | | 416 | 4 | | | 420 |
| coap-observe-client.o | | 232 | 176 | | | 408 |
| setup.o | | 388 | | | | 388 |
| list.o | | 356 | | | | 356 |
| stack-check.o | | 317 | 32 | | | 349 |
| clock.o | | 324 | 8 | | | 332 |
| coap-timer-default.o | | 283 | 44 | | | 327 |
| chipinfo.o | | 284 | | | | 284 |
| rpl-dag-root.o | | 264 | 17 | | | 281 |
| board.o | | 244 | 20 | | | 264 |
| coap-blocking-api.o | | 260 | | | | 260 |
| aux-ctrl.o | | 252 | 4 | | | 256 |
| spi.o | | 252 | | | | 252 |
| oscillators.o | | 236 | | | | 236 |
| rpl-nbr-policy.o | | 216 | 10 | | | 226 |
| leds.o | | 216 | | | | 216 |
| memb.o | | 216 | | | | 216 |
| csma.o | | 201 | | | | 201 |
| board-buttons.o | | 21 | 172 | | | 193 |
| ddi.o | | 188 | | | | 188 |
| dtls-peer.o | | 112 | 56 | | | 168 |
| contiki-watchdog.o | | 160 | | | | 160 |
| gpio-hal-arch.o | | 156 | | | | 156 |
| log.o | | 144 | 4 | | | 148 |
| uip-udp-packet.o | | 132 | | | | 132 |
| osc.o | | 112 | | | | 112 |
| netstack.o | | 100 | 4 | | | 104 |
| snprintf.o | | 96 | | | | 96 |
| ccfg.o | | | | 88 | | 88 |

| timer.o | | 72 | | | | 72 |
|---|---|---|---|---|---|---|
| gpio-hal.o | | 64 | 4 | | | 68 |
| dbg.o | | 64 | | | | 64 |
| printf.o | | 56 | 8 | | | 64 |
| ieee-addr.o | | 60 | | | | 60 |
| trng.o | | 56 | | | | 56 |
| assert.o | | 53 | | | | 53 |
| ringbuf.o | | 52 | | | | 52 |
| rtimer.o | | 44 | 4 | | | 48 |
| stimer.o | | 48 | | | | 48 |
| aon_batmon.o | | 48 | | | | 48 |
| pwr_ctrl.o | | 48 | | | | 48 |
| linkaddr.o | | 32 | 8 | | | 40 |
| adi.o | | 40 | | | | 40 |
| aon_rtc.o | | 36 | | | | 36 |
| csma-security.o | | 36 | | | | 36 |
| node-id.o | | 24 | 2 | | | 26 |
| fault-handlers.o | | 24 | | | | 24 |
| int-master.o | | 24 | | | | 24 |
| autostart.o | | 24 | | | | 24 |
| mac.o | | 24 | | | | 24 |
| rtimer-arch.o | | 16 | | | | 16 |
| gpio-interrupt.o | | 16 | | | | 16 |
| random.o | | 16 | | | | 16 |
| cpu.o | | 16 | | | | 16 |
| rf-ble.o | | 12 | 1 | | | 13 |
| energest.o | | 4 | | | | 4 |
| leds-arch.o | | 4 | | | | 4 |
| TOTALS | 0 | 87720 | 16273 | 88 | 1024 | |