UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Luiz Odilon de Lima Brites

PROPAGAÇÃO DE MENSAGENS COM PROTOCOLO COAP EM REDES LOWPAN COM SIMULADOR COOJA

Santa Maria, RS
2017

Luiz Odilon de Lima Brites

PROPAGAÇÃO DE MENSAGENS COM PROTOCOLO COAP EM REDES LOWPAN COM SIMULADOR COOJA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Engenheiro de Computação**.

ORIENTADOR: Prof. Dr. Carlos Henrique Barriquello

Santa Maria, RS
2017

DEDICATÓRIA

Aos colaboradores e amigos.

AGRADECIMENTOS

À minha mãe Lea e ao meu pai Francisco, pelo apoio e incentivo. À minha namorada Graciele pela determinação. À Universidade Federal de Santa Maria, pela oportunidade de fazer o curso. Ao Prof. Dr. Eng. Carlos Henrique Barriquello pelo auxílio e amizade. Aos demais professores por apontar solução aos problemas e pela dedicação. Ao Grupo de Desenvolvimento em Reatores Eletrônicos (GEDRE) e seus integrantes pela experiência compartilhada. Ao Projeto de Pesquisa e desenvolvimento de uma rede LoRaWAN na UFSM pelo suporte. Ao NUPEDEE pelo apoio nas tarefas iniciais realizadas.

Um galo sozinho não tece uma manhã: ele precisará sempre de outros galos. De um que apanhe esse grito que ele e o lance a outro; de um outro galo que apanhe o grito de um galo antes e o lance a outro; e de outros galos que com muitos outros galos se cruzem os fios de sol de seus gritos de galo, para que a manhã, desde uma teia tênue, se vá tecendo, entre todos os galos.

(João Cabral de Melo Neto)

RESUMO

PROPAGAÇÃO DE MENSAGENS COM PROTOCOLO COAP EM REDES LOWPAN COM SIMULADOR COOJA

AUTOR: Luiz Odilon de Lima Brites
ORIENTADOR: Dr. Carlos Henrique Barriquello

O padrão IEEE 802.15.4 tem baixo custo e forma redes de área pessoal sem fio que através da adaptação do protocolo IPv6 sobre essa camada física, essa adaptação é o 6LoWPAN, e assim há integração de locais e coisas monitoradas e conectadas à IoT por IP. A camada de aplicação mais próxima do usuário utiliza o protocolo CoAP neste trabalho e com ele é feita análise de latência da coleta de informação ou envio de comandos em pontos da rede. O encaminhamento é feito pelo protocolo RPL, ele necessita de um intermediário ao universo externo à rede, o roteador da borda. Os métodos interacionais de mensagens com o CoAP são descritos, assim como a pilha de protocolos necessária para ele entrar em ação nos dispositivos da rede LoWPAN. O Contiki OS tem *softwares* que permitem uso desses protocolos e tem o simulador Cooja, ambos são usados no trabalho para simular três topologias de rede: reta, triangular e quadrada. O Copper faz papel de agente externo à rede simulada no Cooja e assim o CoAP tem latência medida no ganho de informações de pontos da rede que resulta em crescimento exponencial de tempo com três recursos quanto mais se aumenta saltos do ponto solicitado. No caso de um único sensor ou de envio de comandos, a latência é linear, também conforme a distância em saltos do nó borda da rede. Outra conclusão é que as topologias devem favorecer menos solicitações simultâneas à borda da rede, isso registra menos erros de comunicação com esses protocolos. Uma aplicação é elaborada e nós da rede são posicionados em tempo de execução no simulador. Ela propõe solução para o consumo de energia na iluminação.

Palavras-chave: CoAP. Contiki. Copper. Iluminação. IoT. Latency. RPL.

ABSTRACT

MESSAGE PROPAGATION WITH COAP PROTOCOL IN LOWPAN NETWORKS WITH COOJA SIMULATOR

AUTHOR: Luiz Odilon de Lima Brites
ADVISOR: Dr. Carlos Henrique Barriquello

The IEEE 802.15.4 standard has low cost and forms wireless personal area networks which by adapting the IPv6 protocol over this physical layer, this adaptation is 6LoWPAN. Thus, there is information about assemblies and assemblies and connections to IoT. An application layer closer to the User used in the CoAudio protocol at work and latency analysis of information gathering or sending commands at network points. The routing is done by the RPL protocol and it requires an intermediary to the universe outside the network, the edge router. Interactive messaging methods with CoAP are described, as well as a protocol stack, to take action on LoWPAN network devices. The Contiki OS has softwares that have protocols and has the Cooja simulator, both are at work to simulate three network topologies: straight, triangular and square. Copper plays the role of agent external to the simulated network does not Cooja and thus the Coap has latency measured in the gain of information points of the network that results in exponential growth of time with three resources for more. In the case of a single sensor or sending commands, a latency is linear, also according to the hopping distance of the network edge node. Another conclusion is that as topologies should favor fewer requests simultaneously to the edge of the network, this is record of communication errors with these protocols. An application is built and network nodes are positioned at runtime on the simulator. It proposes solution for energy consumption in lighting.

Keywords: CoAP. Contiki. Copper. Lighting. IoT. Latency. RPL.

LISTA DE FIGURAS

Figura 2.1 – Crescimento da IoT. 17
Figura 2.2 – Arquitetura de uma Rede IoT. 19
Figura 2.3 – Pilha de Protocolos. 20
Figura 2.4 – Topologia RPL. 22
Figura 2.5 – Abstração do Protocolo CoAP. 26
Figura 2.6 – Transmissão Confiável de Mensagem. 28
Figura 2.7 – Transmissão Não-Confiável de Mensagem. 28
Figura 2.8 – *Get Request* com *Piggybacked Response*. 29
Figura 2.9 – *Get Request* com Respostas Separadas. 29
Figura 2.10 – Solicitação e Resposta de Uma Mensagem Sem Confirmação. 30
Figura 2.11 – Formato da Mensagem do Protocolo CoAP. 30
Figura 2.12 – Passos para Nova Simulação. 35
Figura 2.13 – Criando Nova Simulação. 37
Figura 2.14 – Área com os Controles de Simulação. 38
Figura 2.15 – Compilação do Código. 38
Figura 2.16 – Criação e Interação dos Nodos. 39
Figura 3.1 – Ferramenta do Simulador para Visualização dos Pacotes. 40
Figura 3.2 – Recursos Padrão Disponíveis da Aplicação er-example-server. 41
Figura 3.3 – Interface Disponível em Cada Ponto da Rede (o LED Vermelho Pode Ser Ligado pelo Copper). 41
Figura 3.4 – Copper: O Agente CoAP no Firefox. 42
Figura 3.5 – Reta em Um Sentido para Medir o Tempo de Propagação de Mensagens. 43
Figura 3.6 – Reta com Distribuição de *Motes* em Dois Sentidos. 43
Figura 3.7 – Acréscimo de Solicitações Concorrentes com a Topologia T. 44
Figura 3.8 – Acréscimo de Solicitações Concorrentes com Formato Quadrado. 44
Figura 4.1 – *Smart Lighting*. 53
Figura 4.2 – Liga os LEDs. 53
Figura 4.3 – Desliga os LEDs. 53
Figura 4.4 – Simulação de Rede LoWPAN para Iluminação com LEDs. 55
Figura 4.5 – Visão pelo Lado do Copper. 55

LISTA DE GRÁFICOS

Gráfico 4.1 – *Mote* x Tempo de Propagação do *POST* (Atuador - LED). 45
Gráfico 4.2 – Propagação do *POST* com Variação do Pacote (Com Confirmação). ... 46
Gráfico 4.3 – Propagação do *POST* com Variação do Pacote (Sem Confirmação). ... 46
Gráfico 4.4 – *GET* com 10 *motes* (Sem Confirmação). 47
Gráfico 4.5 – *GET* com 20 *motes* (Sem Confirmação). 47
Gráfico 4.6 – *GET* com 10 *motes* do *button-sensor* (Sem Confirmação). 48
Gráfico 4.7 – *GET* com 20 *motes* do *button-sensor* (Sem Confirmação). 48
Gráfico 4.8 – *GET* com 30 *motes* do *button-sensor* (Sem Confirmação). 49
Gráfico 4.9 – *GET* com 20 *motes* do *hello- world* (Sem Confirmação). 49
Gráfico A.1 – Descrição da Potência Consumida. 60
Gráfico B.1 – Potência Consumida (Z1). 61
Gráfico B.2 – Potência Consumida (Sky). 61
Gráfico C.1 – Gráfico de f(x) = x. 62
Gráfico C.2 – Gráfico da f(x) = e^x. 62

LISTA DE TABELAS

Tabela 2.1 – Sistemas Disponíveis no Cooja. 36
Tabela 2.2 – Capacidade de Memória dos Microcontroladores Presentes no Cooja. . . . 36
Tabela 4.1 – Topologia Linha Reta em Dois Sentidos. 51
Tabela 4.2 – Forma T ou Triangular. 51
Tabela 4.3 – Topologia Quadrada. 51
Tabela 4.4 – Tabela de Endereços CoAP. 54

LISTA DE QUADROS

Quadro 2.1 – Comparação entre Padrões Sem-Fio. 18
Quadro 2.2 – Terminologia Geral do CoAP. 25
Quadro 2.3 – Terminologia de Mensagens CoAP. 27
Quadro 2.4 – Uso dos Tipos de Mensagens. 31
Quadro 2.5 – Aplicações Utilizadas. 34
Quadro 2.6 – Recursos Utilizados. 37

LISTA DE ABREVIATURAS E SIGLAS

ACK	Confirmação de recebimento da mensagem *(Acknowledgement)*
$CoAP$	*Constrained Application Protocol*
$Cooja$	Contiki *Operating System* Java
$CSMA$-CA	Acesso Múltiplo com Verificação de Portadora e Anulação de Colisão
DAO	*Destination Advertisement Object*
DIO	DODAG *Information Object*
DIS	DODAG *Information Solicitation*
$DODAG$	*Destination Oriented Directed Acyclic Graph*
$DSSS$	*Direct Sequence Spread Spectrum*
$DTLS$	*Datagram Transport Layer Security*
ETX	*Expected Transmission Count*
$FHSS$	*Frequency-hopping spread spectrum*
$HTTP$	*Hypertext Transfer Protocol*
$IEEE$	Instituto de Engenheiros Eletricistas e Eletrônicos
JVM	*Java Virtual Machine*
LBR	LLN *Border Router*
LED	*Light Emitting Diode*
LLN	*Low-Power and Lossy Networks*
$LoWPAN$	*Low power Wireless Personal Area Networks*
$M2M$	*Machine-to-Machine*
MAC	*Media Access Control*
$MSPSim$	Simulador de instruções do microcontrolador MSP430
MTU	*Maximum Transmission Unit*
OF	*Objective Function*
OS	*Operating System*
$REST$	*Representational State Transfer*
$ROLL$	*Routing Over Low-Power and Lossy*

RPL	*Routing Protocol for Low power and Lossy Networks*
RTT	*Round-Trip Time* ou tempo de transmissão mais confirmação
$SLIP$	*Serial Line Internet Protocol*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
URI	Identificador Uniforme de Recursos
URL	*Uniform Resource Locator*
$WPAN$	*Wireless Personal Area Network*

LISTA DE SÍMBOLOS

bit	Menor informação processada por computador
$byte$	Equivale a 8 bits
G	Prefixo Giga que equivale a 10^9
$hops$	Saltos em uma rede
Hz	Unidade de frequência do S.I.
K	Prefixo Kilo que equivale a 10^3
M	Prefixo Mega que equivale a 10^6
m	Unidade metro
mW	milli-watt
m/s	Metros por segundo
ms	mili-segundo 10^{-3} s
N	Número de nós
T_{Atual}	Tempo gasto na transmissão de uma mensagem
T_R	Tempo de retransmissão da mensagem
W	watt

SUMÁRIO

1 INTRODUÇÃO 15
1.1 OBJETIVOS 16
1.2 JUSTIFICATIVA 16
2 REVISÃO BIBLIOGRÁFICA 17
2.1 IEEE 802.15.4 20
2.2 IPV6 E O 6LOWPAN 21
2.3 RPL 22
2.4 COAP: *CONSTRAINED APPLICATION PROTOCOL* 24
2.5 CONTIKI OS 33
2.5.1 Simulador Cooja 34
3 METODOLOGIA/DESENVOLVIMENTO 40
3.1 RELATO DO TRABALHO REALIZADO 42
4 RESULTADOS 45
4.1 TESTES E RESULTADOS 45
4.2 ILUMINAÇÃO INTELIGENTE 53
5 CONCLUSÃO 56
REFERÊNCIAS BIBLIOGRÁFICAS 57
APÊNDICE A – CONSUMO DO TRANSCEPTOR DO Z1 60
APÊNDICE B – POTÊNCIA CONSUMIDA DO Z1 E DO SKY EM MILLIWATT (MW) 61
APÊNDICE C – GRÁFICOS DAS FUNÇÕES EXPONENCIAL E LINEAR . 62
APÊNDICE D – APLICAÇÃO DESENVOLVIDA PARA ILUMINAÇÃO 63
ANEXO A – ARQUIVO DE CONFIGURAÇÃO DA PLATAFORMA Z1 PARA 25 NODOS 69
ANEXO B – APLICAÇÃO COAP (ER-EXAMPLE-SERVER) DO CONTIKI 74

1 INTRODUÇÃO

Este trabalho aborda uma visão geral das redes de baixa potência sem fio, até chegar ao cenário atual da IoT. O termo IoT ou Internet das Coisas parte da comunicação digital ao encontro de soluções para melhoria de muitos cenários, com monitoramento e atuação em coisas como eletrodomésticos, iluminação residencial e urbana. A IoT também permite aumento da qualidade de vida e um consumo consciente de recursos despertando à melhoria econômica.

Um exemplo de aplicação da IoT é o sensor de detecção de presença que quando acionado liga um LED e informa uma aplicação na Internet. São bilhões de dispositivos conectados nos mais diversos locais que não podem utilizar cabeamento de fibra ótica (COLINA et al., 2016). Porém de outro lado há limites principalmente de distância e sincronismo, para melhorar isso, um sistema operacional implementa formas comuns aos dispositivos para ocorrer a troca de informação.

Os protocolos IoT são diferentes do conhecido HTTP e além disso há diversos sistemas operacionais que formam essa grande conexão. O HTTP é um protocolo usado na *web* para ligação lógica de troca ou transferência de hipertextos[1]. Já o CoAP é voltado para dispositivos limitados em processamento, memória e potência consumida, com sensores embarcados e baixo *overhead*[2], ou seja, a mensagem tenta conter o máximo de informação útil possível.

O *hardware* dos dispositivos têm recursos limitados de CPU, memória, transmissão e energia. Os protocolos estão nos sistemas operacionais também para facilitar o desenvolvimento das aplicações. Um exemplo de sistema operacional é o Contiki[3], esse sistema tem o simulador de dispositivos Cooja[4] que possibilita testes, simulação e execução de uma aplicação.

O capítulo 2 contém: Revisão Bibliográfica (IEEE 802.15.4, IPv6/6LowPAN, RPL, CoAP, Contiki OS e Cooja), o capítulo 3: Metodologia/Desenvolvimento do trabalho, o capítulo 4: Resultados (Testes, Resultados e *Smart Lighting*) e o capítulo 5: Conclusão.

[1] Texto que agrega blocos de texto, palavras, imagens ou sons acessados por referências (hiperligação).
[2] Consumo excessivo de algum recurso computacional.
[3] http://www.contiki-os.org/
[4] *Contiki Operating System* Java.

1.1 OBJETIVOS

O objetivo é analisar o protocolo CoAP em termos de latência da comunicação utilizando o simulador Cooja, construindo uma solução para melhorar a iluminação em ambiente urbano. Para isso são definidas topologias de rede em linha reta, em forma de triângulo e em forma de quadrado. Através dos resultados encontrados, propõe-se uma solução adequada de rede de comunicação de baixa potência, aplicada ao problema da energia consumida em iluminação.

1.2 JUSTIFICATIVA

A *Internet of Things* é a uma evolução da Internet. As aplicações são revolucionárias, participam no modo como se trabalha, se educa, se aprende e se vive. A tecnologia de redes de baixa potência, tem dispositivos limitados em alcance, processamento e memória, econômicos energeticamente. Assim o simulador Cooja e o sistema operacional *Open Source*[5] Contiki, fornecem formas importantes para analisar o tempo de atuação do protocolo CoAP e desenvolver aplicações no universo revolucionário da IoT.

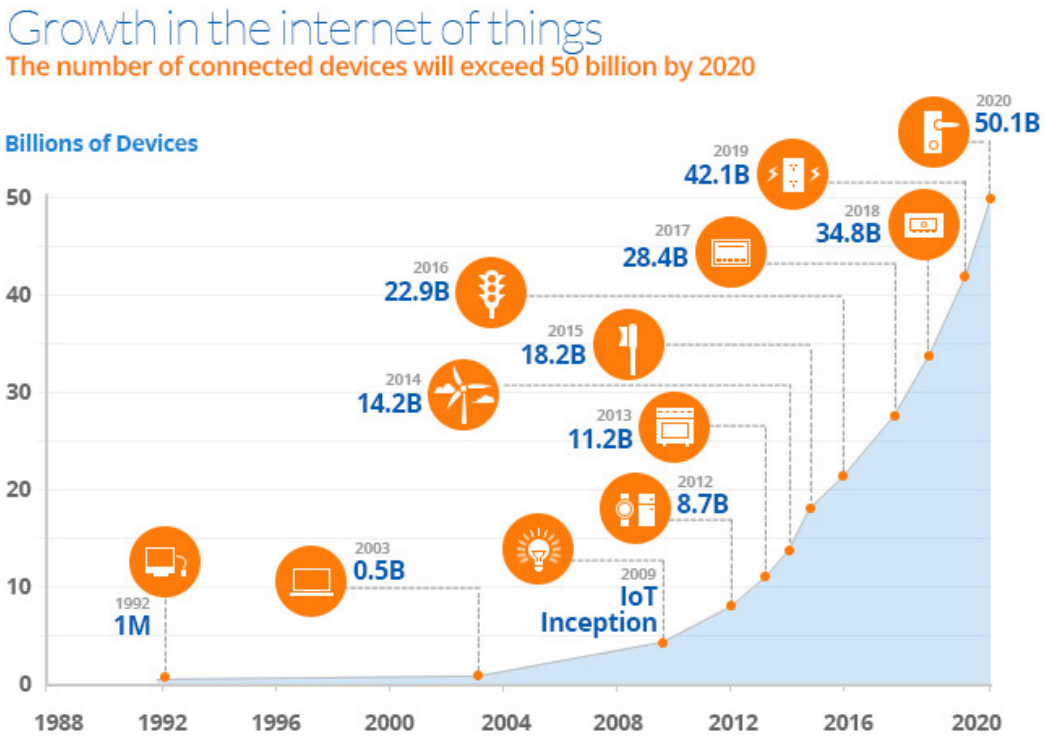
[5]Código fonte disponível sob licença de código aberto que fornece o direito de estudar, modificar e adequar à situações desejadas pelo usuário.

2 REVISÃO BIBLIOGRÁFICA

Redes de sensores não se limitam somente à coleta de dados, há um ponto central que faz a transmissão das informações para outros locais com objetivo de processar e manipular os dados e observar tendência de eventos (variações ocorridas no meio industrial, na saúde individual, no meio ambiente, energia, alimentação, etc.). Meio bilhão de dispositivos foram lançados na indústria, com seu poder de processamento e memória e são base do desenvolvimento das redes de sensores sem fio.

Dispositivos de baixo consumo até radares de controle de tráfego aéreo (SOHRABY; MINOLI; ZNATI, 2007), atualmente provocam melhorias de menor custo na segurança, saúde e comércio. Na figura 2.1 nos anos seguintes à 2008, o começo da IoT faz crescer ainda mais a variedade de dispositivos conectados. O padrão IEEE 802.15.4 pertence às redes sem fios pessoais de baixa potência que tem também outros padrões consolidados. Atualmente ele colabora no número de dispositivos conectados.

Figura 2.1 – Crescimento da IoT.



Fonte: Estimativa da CISCO segundo iotonlinestore (2016)

A seguir, o quadro 2.1 apresenta protocolos sem fio utilizados e o padrão IEEE 802.15.4, e suas principais características.

Quadro 2.1 – Comparação entre Padrões Sem-Fio.

Padrão	Frequência	Taxa de dados	Acance
802.11n	2.4/5 GHz	248 Mbps[1]	250m
802.15.1	2.4 GHz	3 Mbps[2]	100m[3]
802.15.4	868/915 MHz 2.4 GHz	40 kbps 250 kbps	75m

[1] Dois canais (quatro antenas) (Wi-Fi).
[2] *Bluetooth 2.0*
[3] Dispositivo *Bluetooth* classe 1

Fonte: Adaptado de Tjensvold (2007).

O IEEE 802.15.4 foi padronizado em 2003 e contém um protocolo de acesso ao meio e uma camada física, utiliza comunicação via rádio-frequência e têm espalhamento espectral em sequência direta (DSSS) e modulação OQPSK, nas frequências apresentadas no quadro 2.1 (IEEE Computer Society, 2011). O DSSS é uma técnica de espalhamento espectral em que a largura de banda do sinal é maior do que somente a da informação, no receptor o sinal é demodulado oferecendo segurança nas transmissões. Essa técnica é usada no Wi-Fi, diferente do *bluetooth*, que é FHSS.

O 6LoWPAN é a adaptação do IPv6 sobre o IEEE 802.15.4, ele utiliza técnicas de encapsulamento (MONTENEGRO et al., 2007). O IPv6 têm um cabeçalho maior, de 128 bits e fornece 2^{128} endereços (COLINA et al., 2016) frente aos 2^{32} do IPv4, suficiente para a quantidade de dispositivos IoT estimados. O CoAP é um protocolo de aplicação M2M, para redes de baixa potência e altas perdas, redes LLNs.

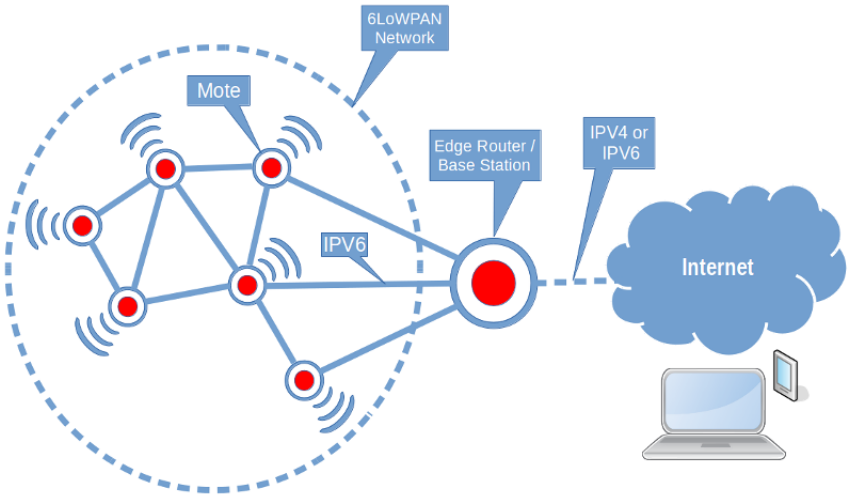
O protocolo IPv6 endereça a comunicação e ele é suportado pelo sistema operacional Contiki OS. Ele sustenta os protocolos IPv6, 6LoWPAN e CoAP, é sistema operacional embarcado e de tempo real e utiliza a linguagem C. O simulador Cooja é contido nele e através de um simulador de instruções possibilita a simulação de redes *wireless* com sensores e tem suporte para o microcontrolador MSP430 da Texas Instruments entre outros (DUNKELS; GRöNVALL; VOIGT, 2004). Se colhe resultados de simulação e se interage com os *motes*[1].

O protocolo CoAP se adapta à rede LoWPAN nos acessos remotos pelo navegador, essa comunicação forma um ambiente de IoT, a partir das redes sem fio como ilustrado na figura 2.2. Ela representa a arquitetura em dispositivos de baixo consumo e alta latência embora a IoT una outras tecnologias de maior alcance, maior consumo, menor latência e diferentes portanto, deste trabalho, também há outras tecnologias de simulação e aplicação disponíveis.

O presente trabalho se preocupa em trazer uma solução aplicável através de cenários diferentes. A tecnologia de comunicação é descrita mostrando seu funcionamento, é feita breve busca em revelar como a comunicação se comporta e atua no processo. Testes

[1] Derivado do termo *Re-mote*

Figura 2.2 – Arquitetura de uma Rede IoT.



Fonte: (Ron Segal, 2015)

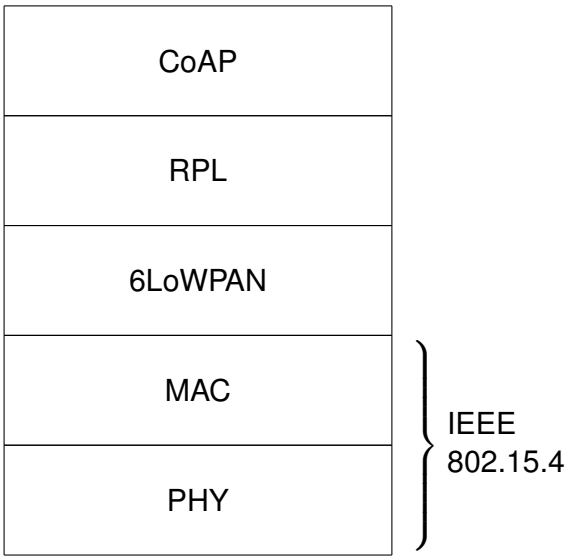
auxiliam a conceber a solução e a conclusão a partir dos tempos extraídos nas simulações com Cooja. Portanto é proposta a partir disso uma solução para iluminação inteligente em locais com tráfego de pedestres.

2.1 IEEE 802.15.4

O padrão IEEE 802.15.4 define as características da camada física (PHY) e da camada de enlace de dados (MAC) e opera na frequência de 2.4 GHz, com taxa de dados de 20 kbps à 250 kbps e usa o protocolo CSMA-CA (*Carrier Sense Multiple Access with Collision Avoidance*) para acesso ao meio (IEEE Computer Society, 2011).

A figura 2.3 mostra a pilha de protocolos utilizada com todas as camadas ou protocolos, com destaque para o padrão IEEE 802.15.4. O MAC permite o controlar e evitar colisões na comunicação. Antes de haver transmissão há detecção de sinal de comunicação próximo ao seu alcance, com o meio livre o transmissor envia a mensagem.

Figura 2.3 – Pilha de Protocolos.



Fonte: Shelby et al. (2014).

De forma resumida, a camada MAC, descrita pelo padrão IEEE 802.15.4, expressa controle de acesso ao canal aos nós da rede, a PHY realiza a modulação do sinal e o espalhamento espectral. A modulação altera a fase do sinal e gera uma nova versão dele (O-QPSK), o espalhamento espectral consiste em espalhar a energia do sinal pela largura de banda, evitando a interferência de estações oferecendo segurança. A cada bit de dado são gerados *n* bits usando um código de espalhamento (FOROUZAN, 2007). O padrão IEEE 802.15.4 lida com pacotes de no máximo 128 bytes.

2.2 IPV6 E O 6LOWPAN

O IPv4 é usado com o TCP e o UDP, formam redes com dispositivos de maior capacidade. O IP é o endereço ou identificação única de um computador ou impressora em uma rede. O IPv6 tem mais endereços que o IPv4 e é utilizado em sistemas menores como os contidos nas redes LoWPAN, de baixa utilização de potência. Já o 6LoWPAN é uma camada de adaptação do protocolo de endereçmento IPv6 sobre o padrão IEEE 802.15.4 da rede LoWPAN.

A qualidade de fragmentar um pacote ocorre quando ele ultrapassa a capacidade máxima da pilha mais próxima da camada física. O MTU do IP influencia a transmissão ou a retransmissão de maneira geral, portanto há uma adaptação devido ao esse fato. O seguinte acontecimento justifica o uso do 6LoWPAN:

- O cabeçalho IPv6 tem 40 octetos e o UDP 8 octetos.
- 802.15.4 carrega 25 octetos (segurança *null*) ou 25+21 = 46 octetos (AES-CCM-128).
- Com o tamanho do frame de 127 bytes há 54 octetos ou 33 octetos, com e sem segurança, respectivamente de espaço para dados.

O IPv6 exige da camada de enlace um MTU de 1280 octetos, é necessário fragmentação e agrupamento na camada de enlace (MONTENEGRO et al., 2007). A partir disso a necessidade de de resolver o problema da incompatibilidade dos tamanhos com o 6LoWPAN, uma solução ideal porque a *Internet of Things* têm como aspecto positivo a integração de muitos dispositivos e com eles também os de baixo consumo de potência.

A camada do 6LoWPAN permite o transporte de pacotes IPv6 sobre a camada de enlace do IEEE 802.15.4 através de fragmentação/agrupamento dos pacotes IPv6. Há compressão do cabeçalho IPv6 e do UDP e suporte à roteamento com baixo custo de processamento e memória (MONTENEGRO et al., 2007). O uso do 6LoWPAN integra a rede à IoT através do IP permitindo:

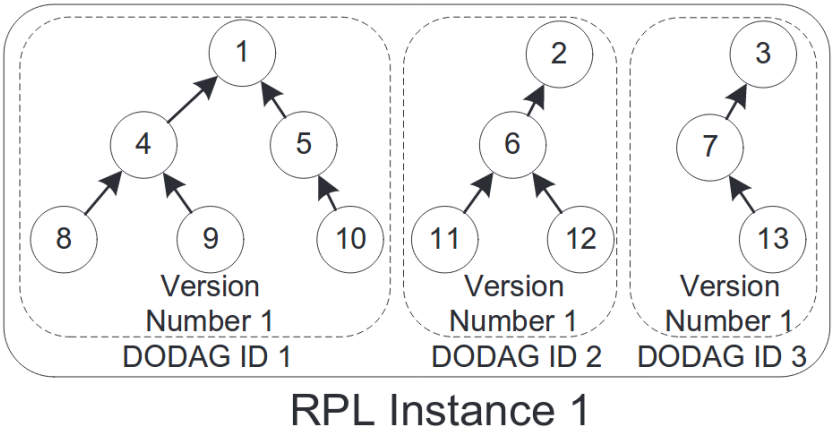
- Conexão de dispositivos com redes maiores usando o IP sem uso de *gateways* ou *proxies*.
- O IP permite o uso da infraestrutura de rede existente.

O protocolo IPv6 é ligado diretamente à IoT, a conectividade é um fator que torna o 6LoWPAN essencial para as redes LoWPAN, redes de baixo consumo de potência e de pequena área de abrangência, para consequentemente o uso do protocolo de aplicação CoAP. O Contiki possui suporte à esses protocolos.

2.3 RPL

A figura 2.4, mostra a troca de mensagens. O controle do pacote é feito por essas mensagens, elas provêm uma instância "Rede RPL" (ALI, 2012). A informação segue um caminho em direção ao extremo da rede, e ele possui características próprias. Há periodicamente uma atualização da rede e a seguir é detalhado esse mecanismo e feita uma abordagem geral de uma rede RPL. As aplicações ou programas dos sistemas podem ser configurados para se comportarem como nós folhas, intermediários ou raiz.

Figura 2.4 – Topologia RPL.



Fonte: (TSVETKOV, 2011)

O controle dos pacotes é feito por mensagens de controle e de forma geral há um tempo de descoberta da rede em que uma tabela é montada, a partir de uma função objetivo (OF), ela avalia a carga dos caminhos disponíveis no dispositivo, número de saltos até o LBR *(root)* (hops) e previne ciclos ou *loops* (ALI, 2012). Os dispositivos utilizam a camada de rede para a comunicação (WINTER et al., 2012) e o roteamento e o encaminhamento de informações pode ter perdas de dados devido aos limites do dispositivo, ao meio e à própria distância entre eles.

A carga tem relação com a taxa de transmissão, o Contiki usa o número de saltos e ETX, onde 2 nodos com 50% de sucesso na entrega de mensagens, o ETX vale 4 (ALI, 2012). Os propósitos: automação residencial, industrial e aplicações urbanas, determina (WINTER et al., 2012), seguem características do ROLL para estas redes. O protocolo AODV por exemplo é reativo, provê caminhos à medida que precisa, já o RPL, proativo porque provê rotas previamente à comunicação, retomando as rotas periodicamente (ALI, 2012).

Essa periodicidade é feita através de mensagens de controle, são quatro (TSVETKOV, 2011):

- DIO: Principal fonte de controle, armazena o *Rank* atual de um nodo, a instância atual e o endereço IPv6 do *root*, etc.
- DAO: Habilita suporte para tráfego de mensagens ponto à multi-ponto *(down traffic)* para propagar o destino ao longo do DODAG.
- DIS: Proporciona ao nodo solicitar mensagens DIO de um vizinho.
- DAO-ACK: Confirmação ou resposta à uma mensagem DAO.

2.4 COAP: *CONSTRAINED APPLICATION PROTOCOL*

O avanço de tecnologias de redes de sensores sem fio em dispositivos de baixa potência e o uso do protocolo IP, mudam o cenário da Internet (COLITTI et al., 2011). A habilidade de manter o transceptor de rádio-frequência não ativado (KOVATSCH; DUQUENNOY; DUNKELS, 2011), o *duty-cycle*, indica a transmissão e a recepção como maiores consumidores de energia. O Apêndice A, contempla aplicação de monitoramento de potência do Contiki OS com o simulador Cooja, o CoAP permite fazer solicitação que ativa o *transceiver* de determinado do *mote*.

Com propósito de apresentar o protocolo CoAP, a seguir se descreve suas funcionalidades padrão (SHELBY et al., 2014):

- Protocolo da *web* que cumpre requisitos M2M em ambientes.
- UDP de confiabilidade opcional suportando *uni* e *multicast*.
- Troca de mensagens assíncronas.
- Baixo *overhead* de cabeçalho e baixa complexidade de *parsing*[2].
- Integração à proxy e *caching*[3] simples.
- Mapeamento imparcial do protocolo HTTP, *proxys* dão acesso ao CoAP via HTTP uniformemente.
- Ligação segura por DTLS[4].

Alguns termos específicos são importantes no CoAP, segundo Fielding et al. (1999), presentes no quadro 2.2, que fornece a descrição deles. Como semelhança com o protocolo HTTP, ele tem independência nos dados transferidos do sistema desenvolvido, assim transforma o recurso solicitado em informação (REST[5]). Os termos são usuais do CoAP e auxiliam a compreensão e a caracterização dos diferentes nodos possíveis neste caso.

[2]Verificação ou análise, interpretação, de dados ou informação.
[3]Reserva de dados para futuro mais recente.
[4]Protocolo de segurança (encriptação) adicional
[5]Arquitetura de aplicação na rede que inclui um conjunto coordenado de limites aplicado aos componentes, enfatiza o fluxo de informações sem se preocupar com estado de conexão e o recurso.

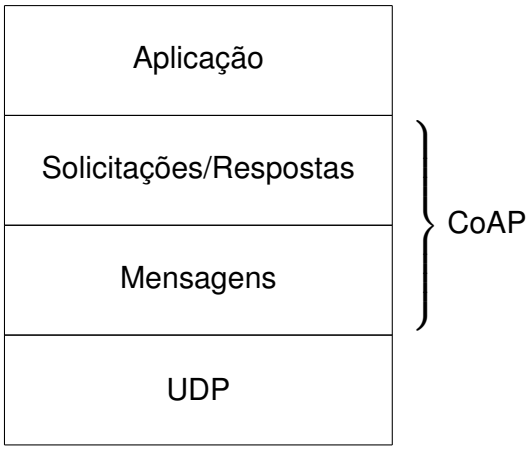
Quadro 2.2 – Terminologia Geral do CoAP.

Endpoint (Terminal)	Possui número de porta associado na rede.
Sender	Terminal fonte.
Recipient	Destinatário da mensagem.
Client	Origem de solicitações, destino da reação (respostas).
Server	Destino final das solicitações (*requests*), origem de reações.
Origin Server	Local onde é gerado o recurso.
Intermediary	Terminal Cliente e Servidor de captação de recursos de um Servidor Origem.
Proxy	Faz encaminhamento e retransmissões de solicitações, traduz nomes ou protocolos. Pode ser de encaminhamento ou reverso.
Forward-Proxy	Solicita em nome do Cliente, fazendo traduções necessárias.
Reverse-Proxy	Responde em nome do Servidor e faz as traduções
Proxy CoAP-CoAP	Usa o protocolo CoAP em ambos os lados.
Cross-Proxy	Proxy HTTP-CoAP ou CoAP-HTTP.

Fonte: Adaptado (SHELBY et al., 2014).

Pode ser feita ação no padrão cliente-servidor no CoAP, entretanto as interações M2M, resultam em implementações atuando como ambos (SHELBY et al., 2014). Solicitação consiste em *request* e requer ações do servidor, como no HTTP mas inclui a representação de recursos. Também diferentemente do HTTP o protocolo se estabelece em um transporte de datagramas orientado (UDP), realizado pela camada de mensagens, essas confiáveis ou não (SHELBY et al., 2014).

Figura 2.5 – Abstração do Protocolo CoAP.



Fonte: (SHELBY et al., 2014).

Naturalmente o CoAP concentra-se em ser um protocolo de trocas para nodos e redes restritos ou limitados tanto em funcionalidade quanto em consumo de energia (BORMANN; CASTELLANI; SHELBY, 2012), a figura 2.5 revela a abstração e indica onde ele está situado. O modelo troca mensagens via UDP entre terminais (SHELBY et al., 2014) e o formato da mensagem apresenta cabeçalho de 4 bytes. O formato de mensagens do CoAP é descrito na figura 2.11. A seguir expressões terminológicas das mensagens, tipos específicos que descrevem as trocas de maneira simples.

Quadro 2.3 – Terminologia de Mensagens CoAP.

Mensagem Confiável	Retorna mensagem de ACK, caso não haja perda do pacote, senão retorna mensagem de *Reset*.
Mensagem Não-Confiável	Não exige confirmação: leitura de sensor.
Mensagem de ACK	Confirma a chegada de mensagem (ACK). Pode conter resposta *Piggybacked*.
Mensagem de *Reset*	Indica que a mensagem foi entregue, mas o contexto foi perdido. Ocorre quando o receptor perde sincronia.
Piggybacked Response	Confirma recebimento da solicitação, é enviado com o ACK.
Separate Response	Uma mensagem vazia, é enviada com o ACK e indica servidor sem respostas no momento.
Mensagem Vazia	Mensagem vazia, com código 0:00, não é uma solicitação nem uma resposta. Contém somente cabeçalho de 4 bytes.

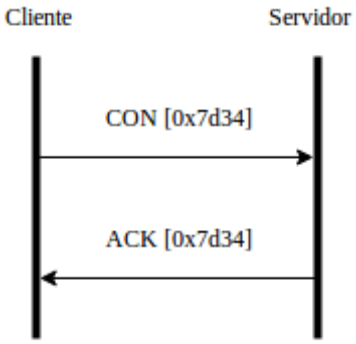
Fonte: Adaptado Shelby et al. (2014).

A mensagem é retransmitida utilizando *timeout* com recuo exponencial, ou seja, quando o tempo se esgota, cada mensagem seguinte, leva o dobro de tempo da anterior para ser transmitida (SHELBY et al., 2014). Há dessa forma controle de congestionamento no receptor, quando o destinatário não puder atender a solicitação, uma mensagem *reset* (RST) é enviada. Cada mensagem tem uma identificação. A equação 2.1 fornece o tempo que leva quando ocorre um *timeout*:

$$T_R = 2 \ X \ T_{Atual} \tag{2.1}$$

- *GET*: Recupera a representação correspondente ao recurso identificado pelo URI. Código 0.01.
- *POST*: Na solicitação, envia em anexo a representação que demanda processamento. É determinado pelo servidor de origem, portanto. Código 0.02.
- *PUT*: Solicita atualização/criação de recurso identificado pelo URI. Código 0.03.
- *DELETE*: Solicita a remoção de um recurso identificado pela URI. Código 0.04.

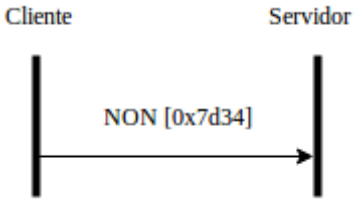
Figura 2.6 – Transmissão Confiável de Mensagem.



Fonte: (SHELBY et al., 2014)

Uma mensagem que não requer uma confirmação (CON) como oposto à figura 2.6 em uma rede de sensores por exemplo (NON) (figura 2.7), possui um identificador (ID), evitando mensagens duplicadas. Na impossibilidade de atender, o destinatário retorna mensagem de *reset* (RST). A mensagem confiável, retorna confirmação (ACK), a não-confiável não retorna. O RST indica perda de contexto, ocorre na mensagem quando confirmável. A não confirmação poupa recursos de energia (COLITTI et al., 2011).

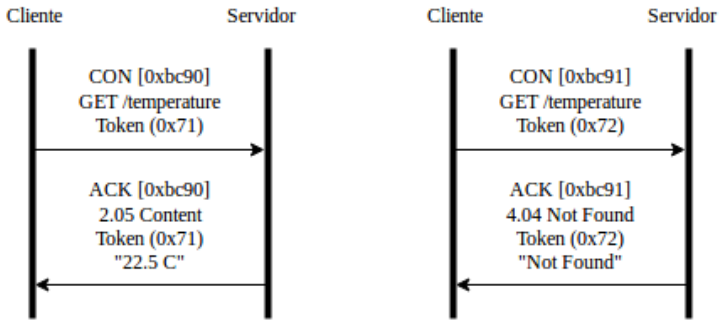
Figura 2.7 – Transmissão Não-Confiável de Mensagem.



Fonte: (SHELBY et al., 2014)

Na possibilidade de atender a solicitação imediatamente, o *piggybacked response* é portado juntamente ao ACK (a confirmação). O cliente retransmite a solicitação caso o ACK e o *piggybacked response* não cheguem. Segue dois exemplos, um de sucesso (figura 2.8) e um de não-encontrado (*Not Found*) (como na figura 2.9), indicando não recebimento do ACK (SHELBY et al., 2014).

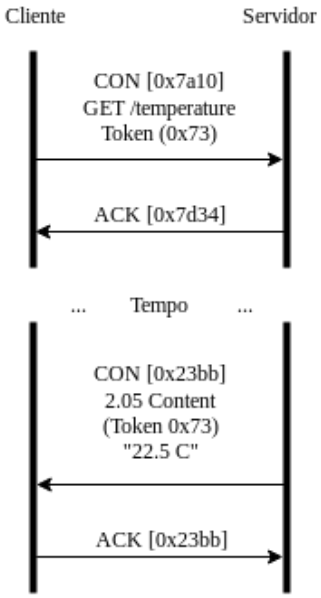
Figura 2.8 – *Get Request* com *Piggybacked Response*.



Fonte: (SHELBY et al., 2014)

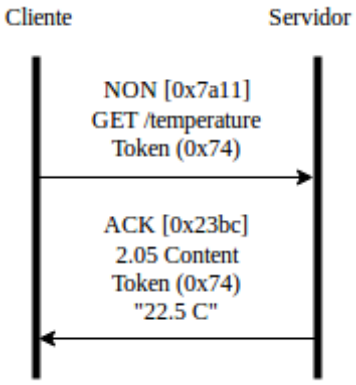
Se não houver respostas imediatas pelo servidor um ACK vazio é enviado ao cliente e ele interrompe as retransmissões. Quando o servidor apresentar resposta ele envia ela ao cliente (retornando um ACK) como na figura 2.9. Se a mensagem solicitada for sem confirmação a resposta também é sem confirmação alguma, exemplo ilustrado na figura 2.10. O CoAP usa para retransmissão a técnica de *stop-and-wait* com recuo exponencial para mensagens confirmáveis.

Figura 2.9 – *Get Request* com Respostas Separadas.



Fonte: (SHELBY et al., 2014)

Figura 2.10 – Solicitação e Resposta de Uma Mensagem Sem Confirmação.



Fonte: (SHELBY et al., 2014)

Existe detecção de confiabilidade ou não da mensagem com o protocolo CoAP (SHELBY et al., 2014). Na figura 2.11 o formato detalhado da mensagem segue o padrão RFC 7252, os campos são descritos e o formato é compreendido internamente à abstração do protocolo CoAP. O *payload* representa a informação em si.

Figura 2.11 – Formato da Mensagem do Protocolo CoAP.

0 1 2 3	4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Ver	T	TKL	Code	Mensagem ID
Símbolo (*Token*) (Se houver, bytes TKL) ...			
Opções (se houver) ...			
1 1 1 1 1 1 1 1	Payload (se houver) ...		

Fonte: (SHELBY et al., 2014)

- Ver: Indica versão do CoAP.
- Tipo (T): Com confirmação (0), Sem confirmação (1), ACK (2) ou *Reset* (3).
- Tamanho do Caractere (TKL): Indica o comprimento do *Token Length*, bits 9-15 são reservados.
- Code: Os 3 bits mais significativos e os 5 bits menos significativos, respectivamente, classe e sub-classe. Classe (0): solicitação, (2): Resposta de sucesso, (4): Falha na resposta do cliente e (5): Falha na resposta do servidor.
- Mensagem ID: Detecta duplicações.

O campo opções exclui algumas respostas pelo servidor em determinadas solicitações com redundância (BHATTACHARYYA et al., 2016). O controle de congestionamento é feito por recuo exponencial (equação 2.1). Para não ocasionar conexões simultâneas, elas devem ser limitadas, isso é considerável em tarefas ou solicitações importantes com necessidade de coleta ou envio de informação (atuadores), e certamente deve ter conformidade com a aplicação do protocolo CoAP pelo sistema operacional neste caso.

A transmissão da mensagem é assíncrona e como ele se limita a uma camada de transporte não confiável (UDP), pode haver desordenamento, transmissão duplicada ou perda de mensagens (RESCORLA et al., 2012). A retransmissão é feita com *stop-and-wait* com recuo exponencial para mensagem confirmável.

Os parâmetros da transmissão interferem em casos de demora e assim encaminha-se a retransmissão (sem uma camada de segurança). Pontos são identificados por um endereço IP e pelo número da porta UDP (SHELBY et al., 2014). O tipo da mensagem é identificado pelo cabeçalho e os quatro tipos de mensagens e as combinações possíveis no quadro 2.4. Já a equação 2.2 se refere ao tempo máximo de transmissão.

$$MAX\_TRANSMIT\_WAIT = ACK\_TIMEOUT \\ X(2^{MAX\_RETRANSMIT+1} - 1) \\ XACK\_RANDOM\_FACTOR \tag{2.2}$$

MAX_TRANSMIT_WAIT é o tempo máximo de transmissão de uma mensagem confirmável até o recebimento do ACK. O *ACK_TIMEOUT* é 2, *MAX_RETRANSMIT* é 4, e o *ACK_RANDOM_FACTOR* é 1.5, valores padrão segundo (SHELBY et al., 2014) em segundos. Esses valores são utilizados como padrão ao utilizar e criar redes com mensagens com protocolo CoAP, pode ocorrer retransmissões quando a mensagem não consegue ser transmitida com sucesso.

Quadro 2.4 – Uso dos Tipos de Mensagens.

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*[1]	-	X	X

[1] Normalmente a combinação não é usada.

Fonte: (SHELBY et al., 2014).

O protocolo de aplicação CoAP tem atuação na IoT, e basicamente tem a possibilidade de interagir remotamente devido às mensagens e métodos para realizar comunicação entre dispositivos. A arquitetura cliente-servidor é utilizada como parte importante nas respostas aguardadas por usuários. A seguir a seção 2.5 apresenta o sistema operacional Contiki OS, que possui aplicação do protocolo CoAP utilizado nos *motes*.

2.5 CONTIKI OS

O sistema operacional Contiki aplica a arquitetura REST, ela reduz a percepção do usuário (FIELDING, 2000) e fornece movimento aos dados que são orientados do local onde estão armazenados para onde são realmente úteis (solicitados). A aplicação CoAP no Contiki OS a utiliza. As conexões não são estáticas a ponto de cada *request* conter todas informações vitais ao conector, há encaminhamentos e cada solicitação é livre (independente das outras, anteriores) (FIELDING, 2000). Portanto segue vantagens do REST:

- Remoção da retenção de estado na aplicação (bom para o consumo de recursos físicos);
- Ele assente o processamento paralelo sem a necessidade de processamento da semântica envolvida da aplicação (utiliza bem a tecnologia, principalmente pelo UDP);
- Possibilita ao intermediário avistar e entender *requests* isoladamente (pode haver arranjo dinâmico principalmente), e
- Oferece possibilidade de reuso de informações, conferindo um fator importante para cada *request*.

O túnel, uma das aplicações do Contiki é um conector REST e simula uma interface de comunicação de rede (um SOCK[6]) (FIELDING, 2000) no caso de uma conexão HTTP CONNECT por exemplo. Também pode ser um SSL[7]. A seguir uma simulação com o protocolo CoAP com o agente CoAP Copper disponivel como extensão para o navegador Mozilla Firefox para sistemas Linux, o qual inclui características de tunelamento para efetivá-la. Portanto o simulador é essencial no desenvolvimento e nos testes dos dispositivos (SEHGAL, 2013).

Segundo Stallings (2012), o sistema operacional embarcado requer operações em tempo real reativas e tem que ser suficientemente configurável, sendo que o Contiki OS conquista estes quesitos. Além disso tem porte para o MSP430 da Texas Instruments, é desenvolvido em C e possui porte à outros dispositivos de *hardware* (DUNKELS; GRöNVALL; VOIGT, 2004). É um sistema operacional de código aberto para *Internet of Things* efetivo em pequenos microcontroladores, de baixo custo e baixa potência, portanto é focado em sistemas embarcados LoWPAN, redes pessoais de pequeno alcance.

[6] Protocolo de encaminhamento entre cliente-servidor.
[7] Protocolo padrão de segurança para encriptação de enlaces.

Ele suporta o IPv6 e o IPv4, o padrão 6LoWPAN, redes RPL e o protocolo CoAP, assim como HTTP. Conforme Colina et al. (2016), o sistema visa utilizar baterias na totalidade. Na figura A.1, no apêndice A e a B.1 no B, mostra-se o exemplo de uma das aplicações para monitoramento do consumo. Se percebe ênfase em controlar o consumo, com isso há flexibilidade na proteção do fluxo de informações e uso de interrupções (STALLINGS, 2012). Portanto o Contiki OS é adequado aos microcontroladores de 8 bits e desenvolvido para aplicações com sensores resumidamente.

A definição de aplicação é que pode ser um programa, serviço, pilha de protocolo de comunicação, controlador ou *driver*[8] de um sensor ou manipuladores de dados obtidos de sensores por exemplo (DUNKELS; GRöNVALL; VOIGT, 2004). O *kernel* do Contiki OS, controla a demanda de serviços feita por uma aplicação, orientando o sistema operacional à eventos ou à *multithreading* preemptivo[9] (DUNKELS; GRöNVALL; VOIGT, 2004).

Quadro 2.5 – Aplicações Utilizadas.

border-router.c	Estabelece comunicação externa em uma rede RPL.
er-example-server.c	É capaz de se comunicar via protocolo CoAP.
udp-sender.c	Tem comportamento de intermediar e enviar informação ao nó Sink.
udp-sink.c	Chama comunicação de um ou mais do tipo Sender e contém suas informações de roteamento.

Fonte: Adaptado do simulador Cooja.

2.5.1 Simulador Cooja

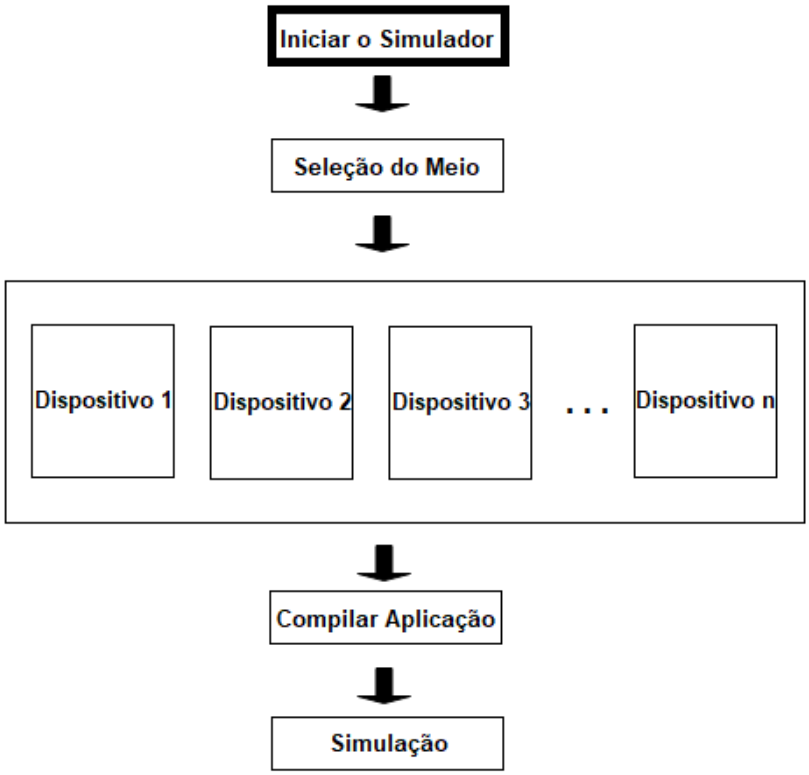
O simulador Cooja (Contiki OS Java) faz parte ao Contiki, sendo constituído de arquivos .java e .class basicamente, portanto um *software* Java. O MSPSim é um simulador de instruções em cada nó da rede (*mote*). O código das aplicação do sistema operacional é compilado pelo MSP430-gcc do MSPSim (também há compatibilidade com avr-gcc da Atmel©). A partir da compilação é gerado código objeto ao sistema correspondente. O MSPSim entre outros é parte *standalone* ou distribuído separadamente.

[8] Faz a comunicação com o Sistema Operacional com o dispositivo.
[9] Tira de execução um processo em detrimento de outro.

Houaiss (2001) indica os seguintes conceitos que se aplicam ao projeto e uso das aplicações no simulador Cooja:

- Emulação: representar com semelhanças certos aspectos de; imitar ou fingir, esforço para imitar algo. No caso do Cooja, um programa de computador aplica os seus recursos para imitar outro programa ou dispositivo eletrônico que faz execução de programas como se fosse o original.
- Simulação: fazer parecer real (e não é), aparentar, representar com semelhança certos aspectos, reproduzir. Atividade que recebe parâmetros de entrada, auxilia na predição e na análise (simulação de redes por exemplo).
- Virtualização: algo possível de ocorrer desde que simulado por computador. Pode ser diversos ambientes de execução em um computador e contém portanto simulação por exemplo.

Figura 2.12 – Passos para Nova Simulação.



Fonte: Autor

Pela figura 2.12 dos dispositivos no simulador Cooja, ele não possui emulação total de todos dispositivos para realizar as simulações, como se observa pelos sistemas disponibilizados na prática, e as particularidades (compatibilidade) deles. O sistema operacional Contiki estabelece redes de sistemas embarcados de baixa potência (LoWPAN), também não há total portabilidade aos dispositivos. Assim dois dispositivos o Z1 e o Sky ganham destaque porque trazem emulação parcial do primeiro e total do segundo.

Tabela 2.1 – Sistemas Disponíveis no Cooja.

Mote	Microcontrolador
MicaZ	ATmega128
eth1120	MSP430
trxeb1120	MSP430F5438
trxeb2520	MSP430
Exp2420	MSP430F5438
Exp1101	MSP430F5438
CC430	MSP430
EXP430F5438	MSP430F5438
Wismote	MSP430F5437
Z1	MSP430x2xx
Sky	MSP430F1611
ESB	MSP430

Fonte: Retirado de (MEMSIC, s.d), Simulador Cooja, (TEXAS INSTRUMENTS, 2012), (TEXAS INSTRUMENTS, 2009a), (ZOLERTIA, 2010) e (MOTEIV, 2006).

As informações são referentes à versão 3.0 do Contiki. Os *motes* (tabela 2.1) podem ser adicionados a uma simulação existente gravada em xml ou uma nova simulação que pode ser gerada. Cada mote pode ser programado através do uso do Contiki OS. O sistema Sky Moteiv (2006), tem sensores e oferece recursos nativos à programação. O mote Z1 Zolertia (2010) não possui os mesmos sensores, mas têm mais memória disponível (tabela 2.2) que o Sky. Outros sistemas não são funcionais nesta versão.

Tabela 2.2 – Capacidade de Memória dos Microcontroladores Presentes no Cooja.

Microcontrolador	RAM	ROM	EEPROM	Flash
ATmega128	4 KB	-	4KB	128KB
MSP430	128 bytes	1KB	-	-
MSP430F1611	128 bytes	2KB	-	48KB
MSP430x2xx	256 bytes	4KB	-	56KB
MSP430F5437	512 bytes	16KB	-	256KB
MSP430F5438	512 bytes	16KB	-	256KB

Fonte: Retirado de (ATMEGA, 2011), (TI MSP430, 2017), (TEXAS INSTRUMENTS, 2002), (MSP430G2x, 1995-2017) e (TEXAS INSTRUMENTS, 2009b).

No Apêndice B é apresentada a comparação do consumo dos dois tipos de *motes*. Os passos para simulação necessitam ter o Contiki OS e o MSPSim, as bibliotecas do ambiente de execução Java instalados em um sistema operacional com licença BSD ou a versão em forma de imagem, que pode ser executado na máquina virtual de sistema, também não inclui o MSPSim. Abaixo segue uma simulação de rede RPL válida com o Z1 com protocolo UDP da camada de transporte.

O simulador pode ser iniciado no diretório home/contiki/tools/cooja desde que possua os requisitos (quadro 2.6), seguido pelo comando ant run no terminal. Após isso, Cooja: *The Contiki Network Simulator* apresenta a interface mostrada na figura 2.13. No apêndice B o exemplo da aplicação collect-view, que fornece o consumo em watt dos dispositivos na rede e breve tutorial de uma forma para simulação.

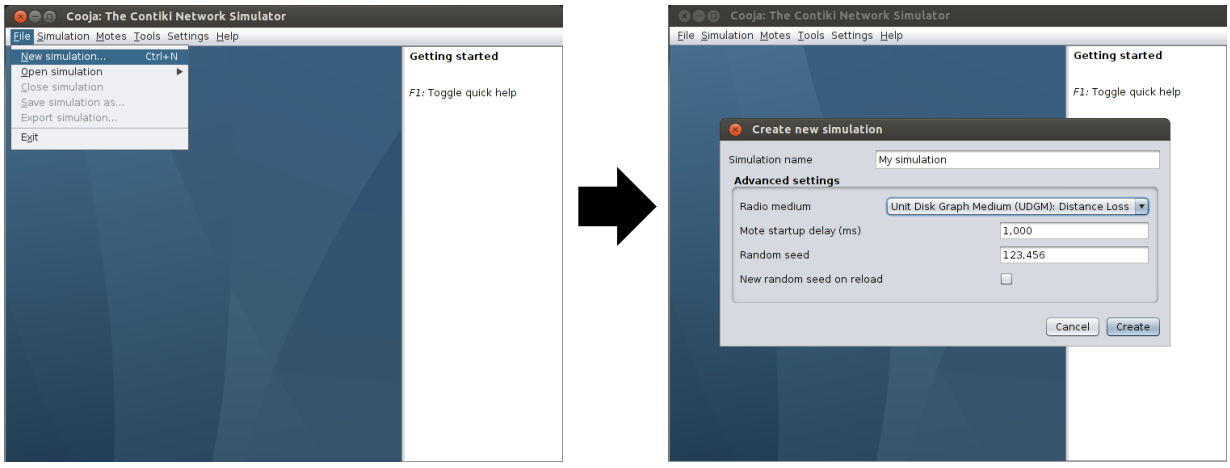
Quadro 2.6 – Recursos Utilizados.

Sistema Operacional	Ubuntu 14.04.5
Processador	Core i5-2450M 2.4 GHz
Memória	4 GB DDR3
Ambiente	Jre
Plataforma	Jdk
Versão	Contiki 3.0

Fonte: Autor.

Radium Medium é o meio no qual se insere o transceptor, atua no sucesso das transmissões, há possibilidade de haver muitos obstáculos ou simplesmente a perda de sinal proporcional à distância (parâmetro presente no trabalho). Também podem ser feitas alterações do atraso inicial e aleatoriedade na simulação, o padrão pré-disponível é o selecionado, portanto assim são encontrados parâmetros para simulação (figura 2.13).

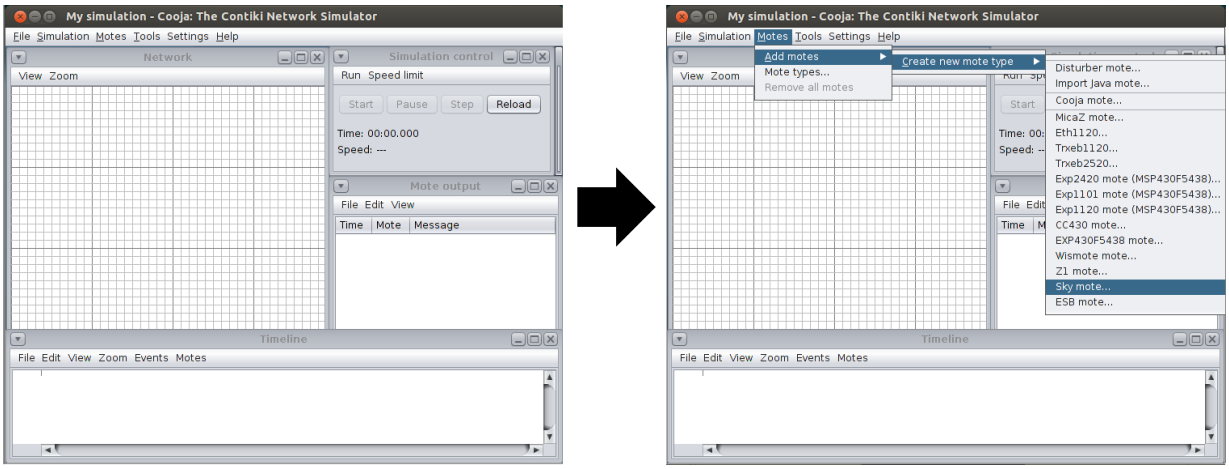
Figura 2.13 – Criando Nova Simulação.



Fonte: *Adaptado do Cooja Simulator*

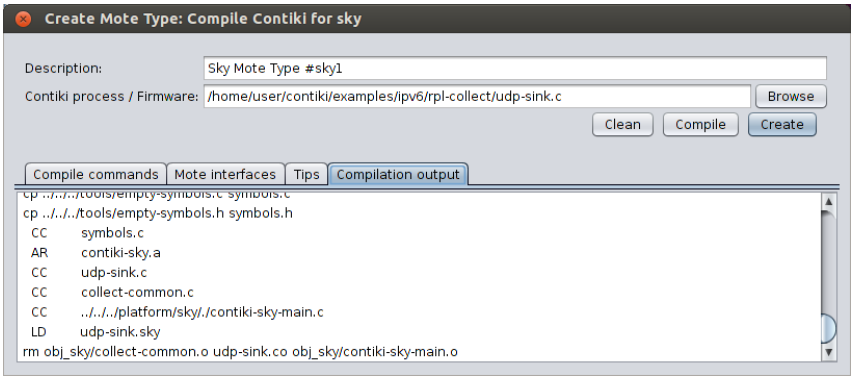
Primeiramente o Sky *mote* é designado para realizar a demonstração (figura 2.14) de uma compilação (geração do código em código objeto pelo compilador) e inserido no *mote* escolhido. Em caso de sucesso da compilação é criado o *mote* na rede com o *firmware*[10] (figura 2.15). O código objeto possui a extensão .sky especificamente.

Figura 2.14 – Área com os Controles de Simulação.



Fonte: *Adaptado do Cooja Simulator*

Figura 2.15 – Compilação do Código.

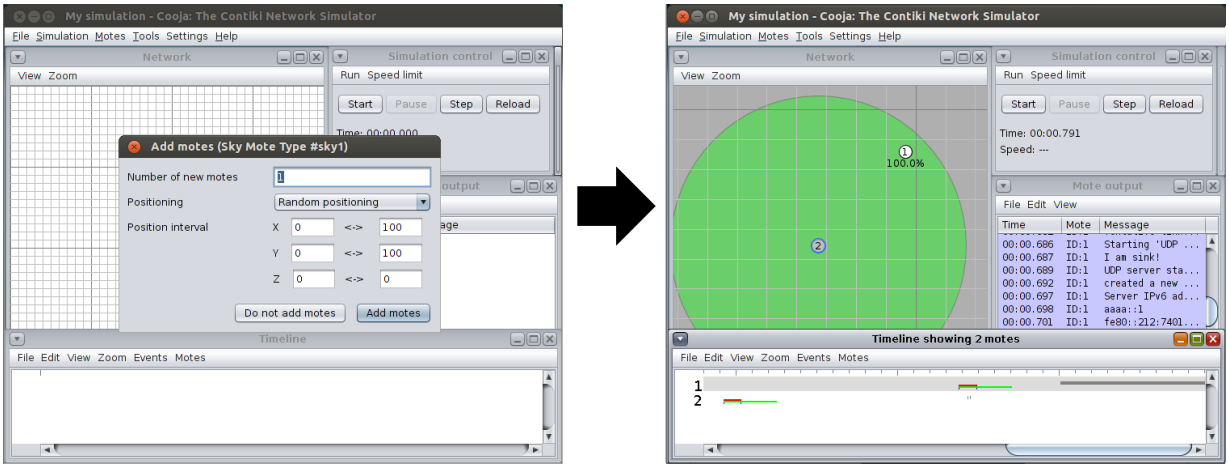


Fonte: *Adaptado do Cooja Simulator*

A figura 2.16 mostra portanto a comunicação entre um *nó* Sink e um Sender, a *timeline* mostra a comunicação e a saída informa a ação, o endereço IPv6 de cada ponto na rede. O *grid* mostra a abrangência omnidirecional de até 50 m sem perdas (diferindo do *mote* real de 10 m). O nodo Sink recebe solicitações agindo como *root* e o nodo Sender como folha (figura 2.16).

[10]Programa objeto que controla o *hardware*.

Figura 2.16 – Criação e Interação dos Nodos.



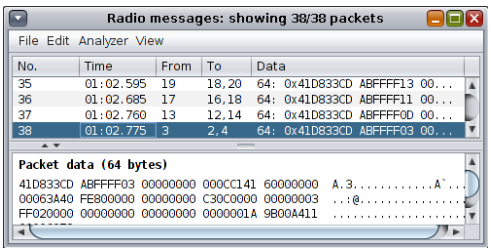
Fonte: *Cooja Simulator*

O objetivo de realizar uma comunicação atinge êxito como na figura 2.16, a troca de pacotes tem destaque na parte inferior da interface de simulação. IoT Networking Research Group (2016) expõe aplicações diferentes, já envolve protocolo que permite acesso externo à essas redes. O Z1 é selecionado justamente porque suporta melhor a aplicação CoAP (tabela 2.2).

3 METODOLOGIA/DESENVOLVIMENTO

Uma rede com número crescente de nodos é testada aumentando o número de *bytes* sem deixar ocorrer a fragmentação de *hardware* dos pacotes, que é a quebra em pacotes menores. Uma mensagem não excede os 128 bytes no total. O agente CoAP (Copper) proporciona a comunicação com o simulador e a interface da figura 3.1 exibe as transmissões e o rumo da informação na rede. O tempo gasto é portanto calculado de forma confiável dentro do simulador e se refere a partir deste momento ao *mote* Z1.

Figura 3.1 – Ferramenta do Simulador para Visualização dos Pacotes.



Fonte: Simulador Cooja

É aberto um *socket* no border-router da simulação, o comando para fazer o tunelamento é feito pelo terminal usando o comando make. Através de navegação ao endereço mostrado pelo *tunsplip*, que é a aplicação que realiza o tunelamento e revela o endereço do *mote* ([endereço IPv6 do border-router]). Então uma tabela da rede RPL do Contiki é mostrada e os endereços exibidos. Assim, cada elemento contido na rede pode ser contatado pelo agente Copper (extensão do Mozilla Firefox) pela URL coap://[endereço do mote].

- É medido o tempo gasto para atuar no LED na rede, nos nodos 5, 10, 15, 20, com confirmação e com cálculo sem ACK.
- É feito um *POST*, envia-se pacotes de tamanho variado em bytes sem fragmentá-los, com e sem confirmação sem exceder o máximo estabelecido pelo IEEE 802.15.4 e o cabeçalho da mensagem com CoAP da figura 2.11. O *payload* então sem ocorrência de fragmentação é de 76 bytes e aumentar o *payload* da mensagem leva a mais transmissões, com endereço, versão, etc. (*overhead*). Um caractere numérico mede um byte e assim é elevado o tamanho do pacote.
- O método *GET* é aplicado para descobrir os recursos. É necessário muitas vezes disponibilizar os recursos da aplicação no sistema ao CoAP, seja para atuação e/ou para obter valores de sensores. A confirmação não é solicitada, pois na equação 2.2 os valores de *ACK_TIME_OUT* afetam as retransmissões e ocorrem conforme aumenta-se o número de nodos. Portanto a não-confirmação é aplicada nos testes.

- Há simulações com diferentes topologias para analisar o comportamento do nó que se comunica com o agente diretamente, o *border-router*.
- Os posicionamento adotado dos *motes* é realizado antes de iniciar a execução da simulação e mais de 15 nodos requerem configurar o Contiki, assim define-se o padrão do desenvolvimento das simulações.

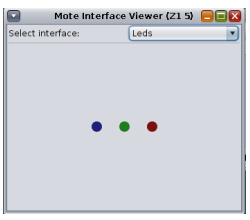
O agente CoAP obtêm informações pelo endereço coap://[aaaa::c30c:0:0:5] para o quinto elemento da rede por exemplo (figura 3.2). Os tempos são medidos no simulador desde o momento que partem do roteador da borda e encontram o destino ou quando o ACK e a *Piggybacked Response* simplesmente retornam à aplicação que aguarda a confirmação. Como o tempo limite de aguardo pode ser atingido, retransmissões ocorrem dependendo da situação do *mote* e da rede em si.

Figura 3.2 – Recursos Padrão Disponíveis da Aplicação er-example-server.

Fonte: Autor

Os recursos disponíveis são informados na figura 3.2 e para coletá-los a topologia empregada é uma linha reta crescente de *motes* na figura 3.1. O agente externo possui também limitações próprias do *hardware* e do sistema operacional que o hospeda basicamente. O cálculo do tempo de propagação é feito quando a mensagem encontra o roteador e se propaga. A figura 3.1 mostra o caminho quando ela atinge seu destino.

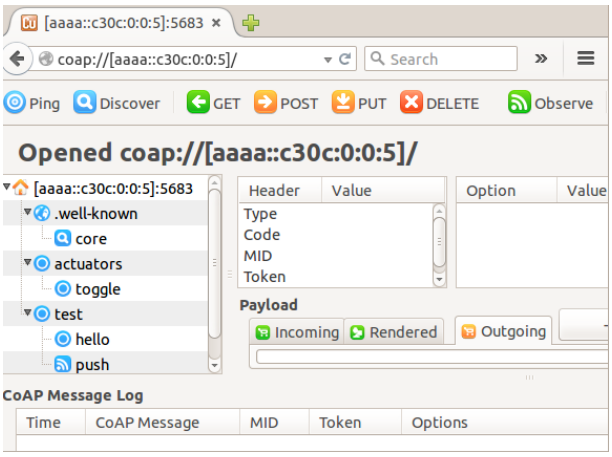
Figura 3.3 – Interface Disponível em Cada Ponto da Rede (o LED Vermelho Pode Ser Ligado pelo Copper).



Fonte: Simulador *Cooja*

Após o agente conhecer a existência dos recursos, é possível fazer o *POST* diretamente no atuador visto pelo Copper, a figura 3.4 anteriormente também mostra a interface do usuário e os comandos disponíveis. O *payload* pode ser preenchido por caracteres (método utilizado) para inserir informação na rede. Dessa forma são feitos os testes variando o tamanho da informação inserida e extraídos gráficos a seguir (seção 3.1).

Figura 3.4 – Copper: O Agente CoAP no Firefox.



Fonte: Adaptado pelo Autor de Copper no Firefox

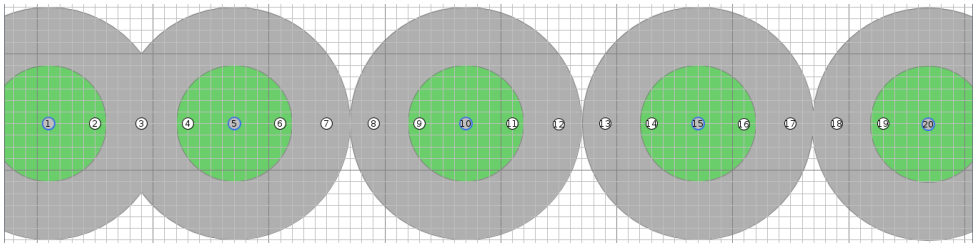
3.1 RELATO DO TRABALHO REALIZADO

A topologia em linha reta da figura 3.5, apresenta:

$$hops = N - 1 \tag{3.1}$$

Ela é ideal para encontrar o número de saltos na rede em uma topologia reta em um sentido, a seguir o tempo de propagação está disponível na seção 4. Com a topologia da figura 3.5 é variado o acesso aos *motes* (*N*), ela é utilizada na maioria dos testes. Outras topologias são apresentadas observando o esforço realizado por cada *mote*. Apresentam-se portanto 3 tipos de topologias para simulação, elas são construídas sem iniciar a execução das simulações, portanto iniciada após feito o posicionamento dos *motes*.

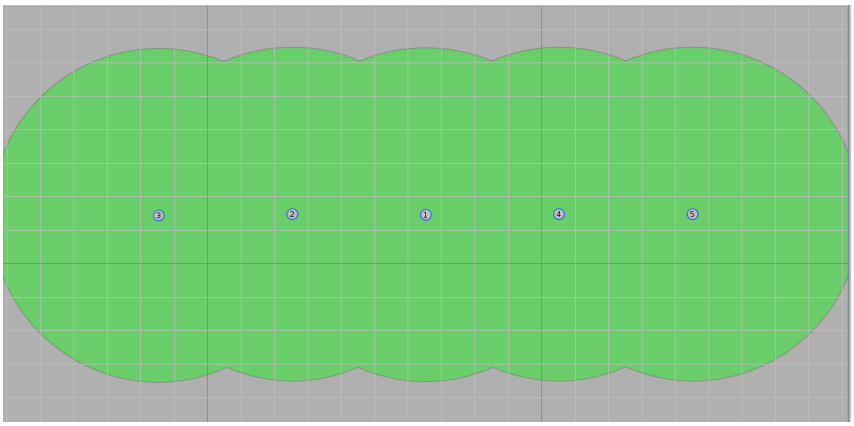
Figura 3.5 – Reta em Um Sentido para Medir o Tempo de Propagação de Mensagens.



Fonte: Adaptado do simulador Cooja pelo autor.

Uma rede de sensores sem-fio vai além de medição, coleta e entrega correta de dados medidos, como observado, também é possível atuar (MUSZNICKI; ZWIERZYKOWSKI, 2012). A topologia da figura 3.5 é utilizada para medir o tempo para o atuador entrar em funcionamento. A seguir o *border-router* frente à solicitações concorrentes, a topologia reta é utilizada novamente mas o nó fica central aos dois sentidos propostos (figura 3.6).

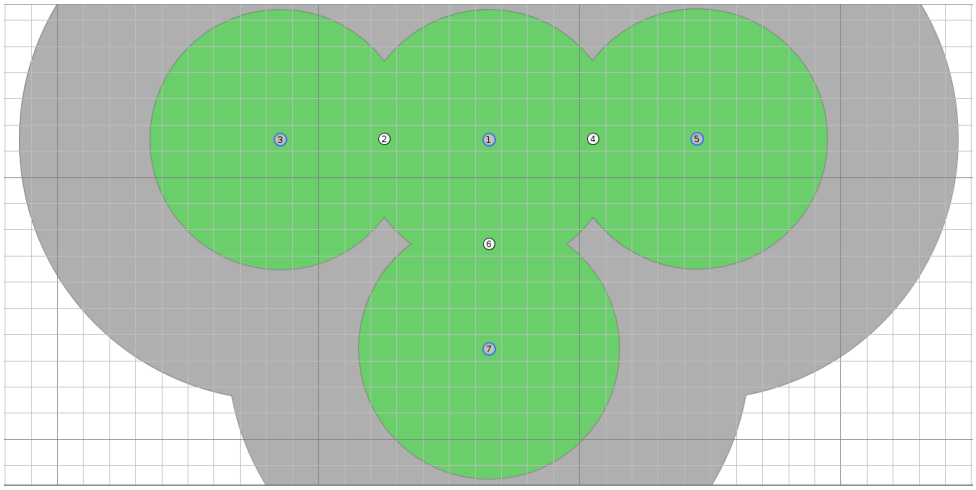
Figura 3.6 – Reta com Distribuição de *Motes* em Dois Sentidos.



Fonte: Adaptado do simulador Cooja pelo autor.

De fato a topologia da figura 3.7, aumenta o esforço do nó central da rede, que abastece o acesso externo. No máximo se estabelece aumento de dimensão de dois *hops* para posterior análise de resultados. A figura 3.8 se encarrega de também mostrar expansão (aumento de dimensão). Ela também visa observar o impacto da vizinhança diretamente no *border-router*.

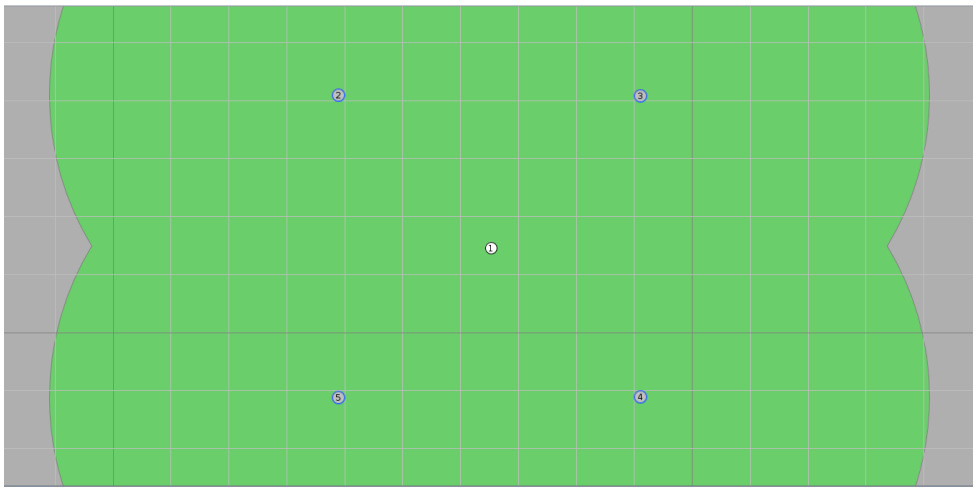
Figura 3.7 – Acréscimo de Solicitações Concorrentes com a Topologia T.



Fonte: Adaptado do simulador Cooja pelo autor.

A seguir um quadrado simula um ambiente estático, mas dependendo do referencial pode estar se movendo como um todo, quatro pontos são fixados (figura 3.8). Com as topologias determinadas, os resultados tem foco em apontar melhores alternativas e a numeração dos pontos é utilizada no endereçamento e cada quadrado no *grid* tem 10 metros, sendo que a comunicação têm sucesso em distâncias em torno de 40 metros.

Figura 3.8 – Acréscimo de Solicitações Concorrentes com Formato Quadrado.



Fonte: Adaptado do simulador Cooja pelo autor.

4 RESULTADOS

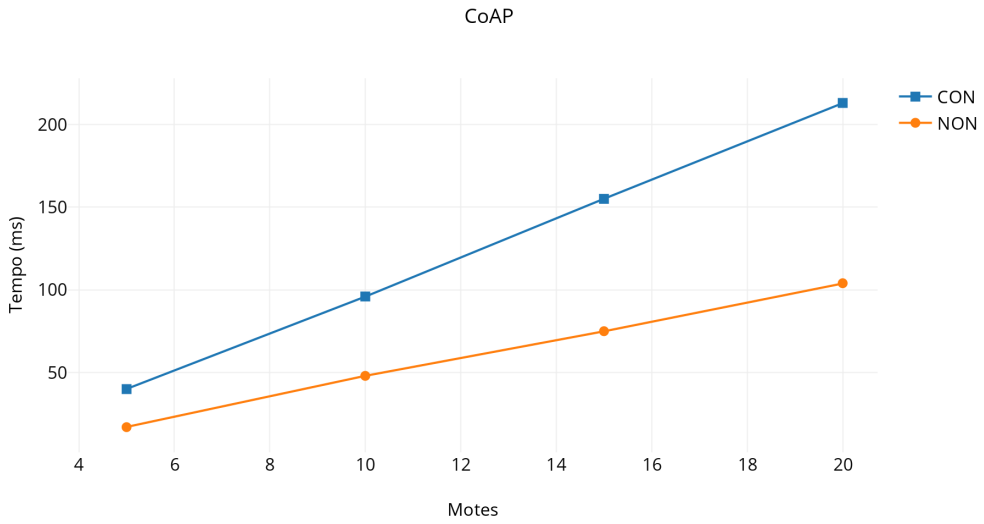
A comunicação é bem sucedida com um e dois sentidos ao nó central e os gráficos evidenciam crescimento de tempo exponencial na topologia de reta de um único sentido conforme se aumentam os saltos ou a dimensão da rede quando acionado o *GET* (coleta). Já no caso do atuador (*POST*) a forma do gráfico produzido é linear, os itens trazem as seguintes constatações:

- Na propagação do *POST* a confirmação é ou não aguardada pelo agente. A comunicação entre o agente e a borda não é visível e o simulador Cooja oferece as medidas de tempo embora exista RTT pelo *Copper* na rede simulada.
- Para o *GET* quanto mais recursos menos agilidade porque o tempo cresce exponencialmente.
- O número de saltos representa a dimensão da rede e aumentar rotas diversificadas piora a comunicação porque gera maior esforço.

4.1 TESTES E RESULTADOS

Os gráficos 4.1, 4.2 e 4.3 mostram *POST* na topologia reta em sentido único.

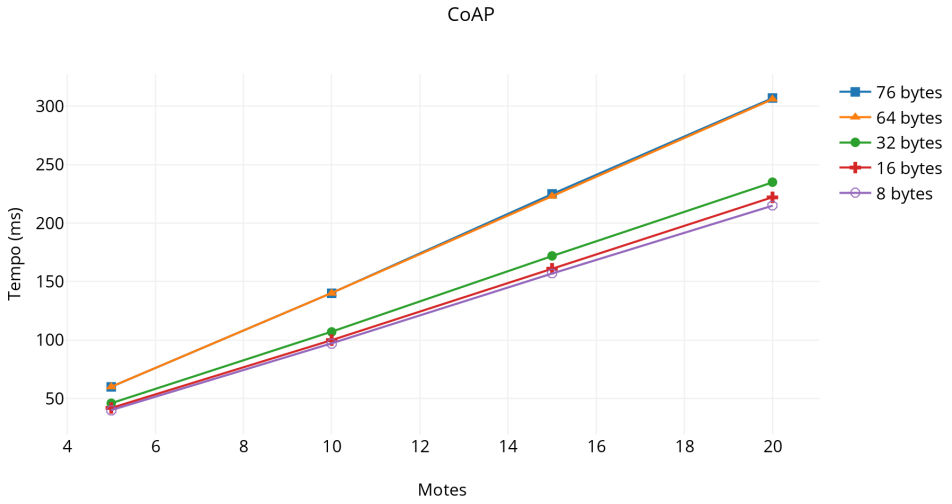
Gráfico 4.1 – *Mote* x Tempo de Propagação do *POST* (Atuador - LED).



Fonte: Autor.

O gráfico 4.1 confronta o fato da mensagem ser ou não ser confirmável e é possível perceber que confirmação acarreta atraso na recepção. A seguir, o gráfico 4.2 mostra o tempo da comunicação conforme se varia o tamanho do pacote.

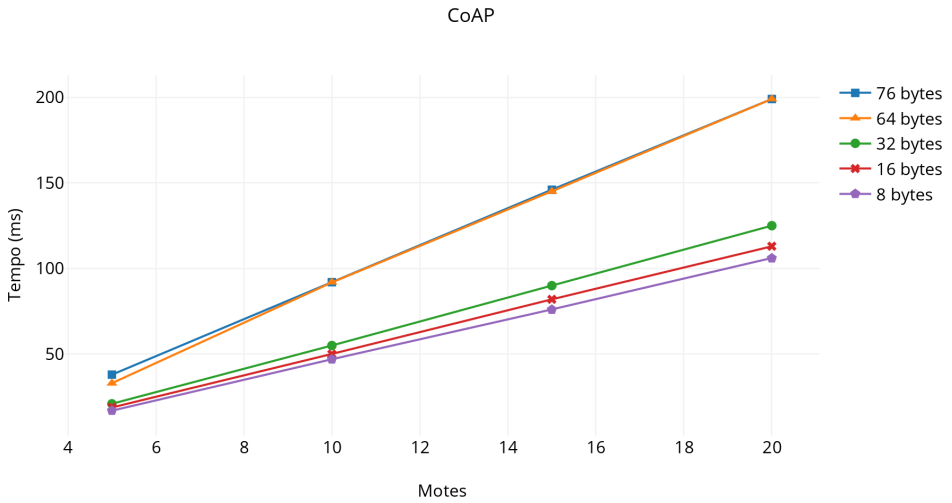
Gráfico 4.2 – Propagação do *POST* com Variação do Pacote (Com Confirmação).



Fonte: Autor.

O gráfico 4.3 apresenta rapidez da entrega quando não há confirmação. Subramanjan, Pasquale e Polyzos (2017) evidencia maior latência para o *POST*.

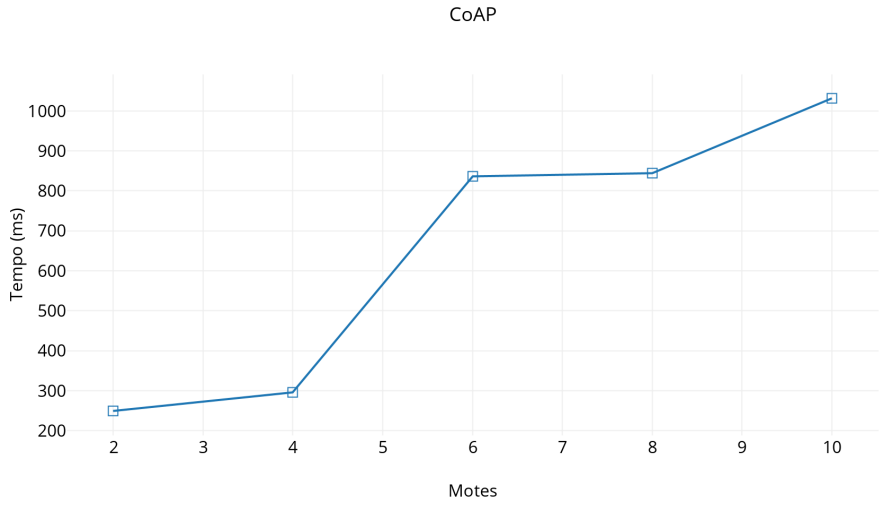
Gráfico 4.3 – Propagação do *POST* com Variação do Pacote (Sem Confirmação).



Fonte: Autor.

Constata-se o impacto também da quantidade de dados no tempo. A seguir, os gráficos continuam utilizando a topologia reta em um sentido, mas se aplica o método *GET* sem confirmação, o 4.4 adicionalmente coleta os recursos padrão dos nós 2, 4, 6, 8 e 10.

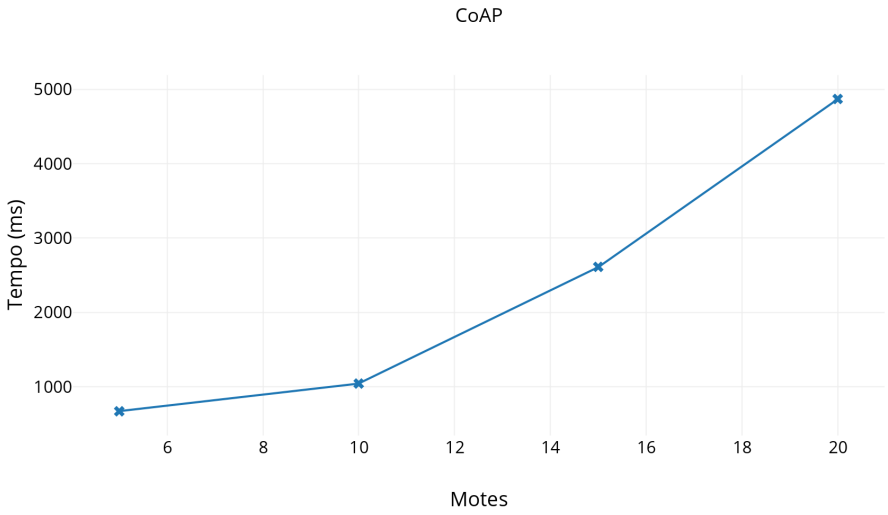
Gráfico 4.4 – *GET* com 10 *motes* (Sem Confirmação).



Fonte: Autor.

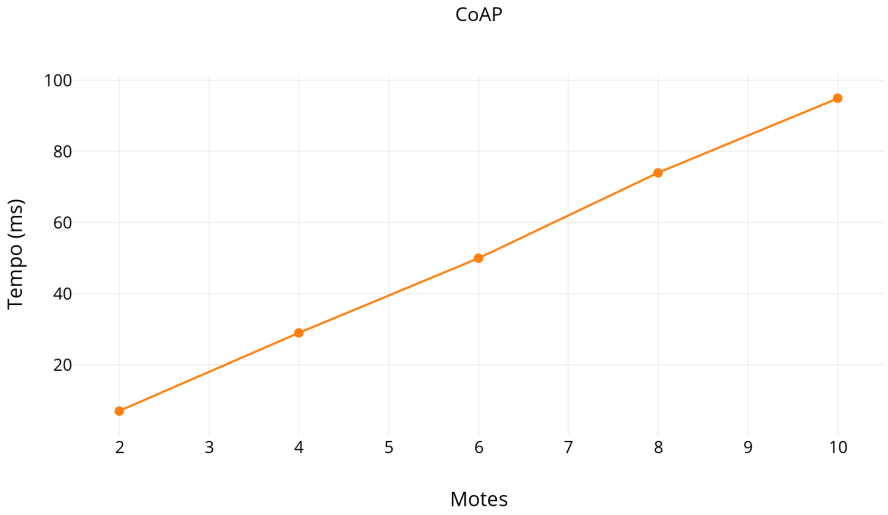
Todos os recursos padrão da aplicação er-example-server são extraídos com o *GET*. Os gráficos 4.4 e 4.5 são semelhantes na forma, mas o tempo aumenta com a dimensão da rede. Na sequência, os gráficos 4.6, 4.7 e 4.8, se referem a dados de um sensor simulado, sendo este botão associado à variável evento.

Gráfico 4.5 – *GET* com 20 *motes* (Sem Confirmação).



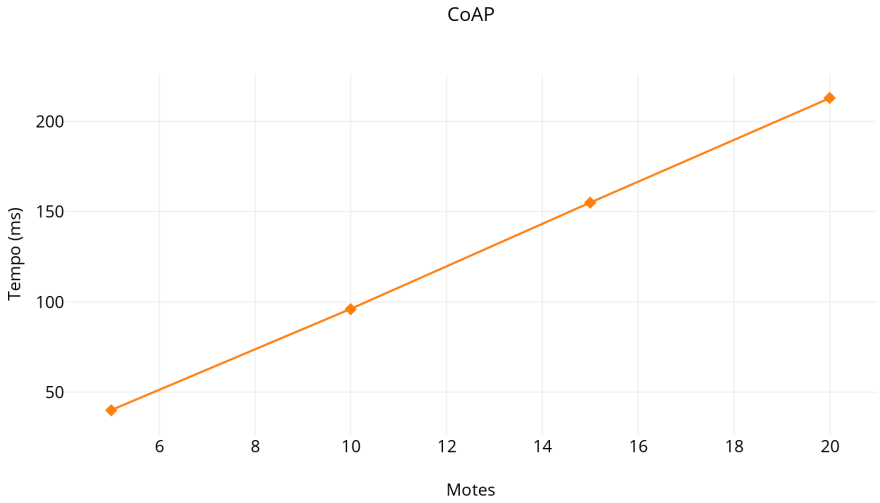
Fonte: Autor.

Gráfico 4.6 – *GET* com 10 *motes* do *button-sensor* (Sem Confirmação).



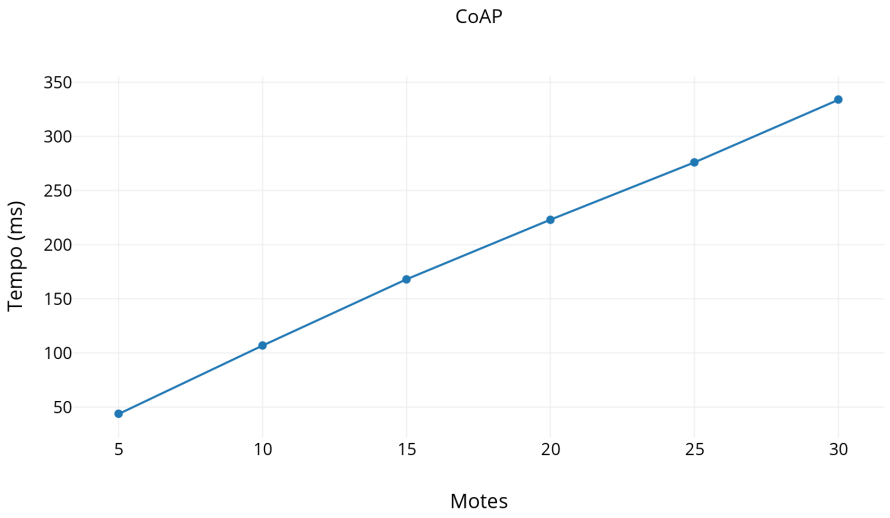
Fonte: Autor.

Gráfico 4.7 – *GET* com 20 *motes* do *button-sensor* (Sem Confirmação).



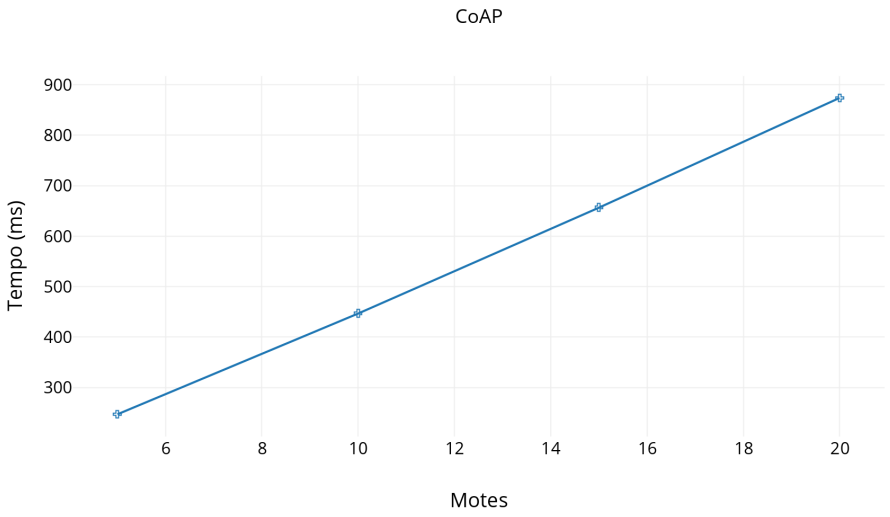
Fonte: Autor.

Gráfico 4.8 – *GET* com 30 *motes* do *button-sensor* (Sem Confirmação).



Fonte: Autor.

Gráfico 4.9 – *GET* com 20 *motes* do *hello- world* (Sem Confirmação).



Fonte: Autor.

A partir dos gráficos extraídos do método *GET* determina-se:

- Forma exponencial com os recursos existentes na aplicação CoAP no *mote* com Contiki, porque maior é o tempo gasto a cada salto (função 4.1).
- A figura 4.8 evidencia tempo pouco menor que 340 ms (0,34 s) porque somente um sensor ou recurso tem forma linear.
- O tempo máximo obtido com o *GET* com mais recursos foi de 5000 ms (5 s), pois ocorreram retransmissões.
- Os valores podem variar devido às combinações de recursos porém a forma gráfica se mantém (SUBRAMANJAN; PASQUALE; POLYZOS, 2017).

Como observação o recurso *Hello*, do gráfico 4.9, eleva o tempo da mensagem, mesmo sendo um único recurso da aplicação, mas ainda possui forma linear da função 4.2. Portanto semelhante na forma ao *POST* (gráfico 4.1, 4.2 e 4.3) e do *GET* do *button-sensor*.

$$f(x) = ax \tag{4.1}$$

$$f(x) = ae^{bx} \tag{4.2}$$

- *a* e *b* são parâmetros de ajuste da função. Aumentar *a* aumenta a inclinação da reta e faz crescer o número de motes crescer na exponencial.
- *b* aumenta o tempo diretamente nesse caso.

Anteriormente foi analisada a topologia em único sentido, até o gráfico 4.9, outras topologias resultam no que indicam as tabelas 4.1, 4.2 e 4.4 e se referem à coleta dos recursos padrão disponíveis na aplicação fornecida pelo Contiki. O *POST* têm grande latência conforme se aumenta as solicitações no centro da rede (SUBRAMANJAN; PASQUALE; POLYZOS, 2017). Os resultados produzidos por elas são:

- Da tabela 4.1 da topologia reta em dois sentidos, pode-se afirmar sucesso na comunicação em 2 *hops* da figura 3.6. Maior chance de sucesso mas perde para a de sentido único.
- A topologia em T, com três pontos da figura 3.7, resulta na tabela 4.2, com 66,67% de chance de sucesso.
- Já a topologia em forma de quadrado da figura 3.8, resulta na tabela 4.4, ela não encontra sucesso devido ao maior número de solicitações e controle simultâneos.

Os resultados se referem à uma simulação padrão, ou seja, os nós são localizados antes de iniciar o simulador. Subramanjan, Pasquale e Polyzos (2017) portanto apresenta também aumento de latência quando se aumentam os *hops*. A seguir as tabelas das simulações das topologias.

Tabela 4.1 – Topologia Linha Reta em Dois Sentidos.

Motes	Tempo (ms)
2	243
3	348
4	266
5	363

Fonte: Autor (extraído de Simulação com Cooja).

Tabela 4.2 – Forma T ou Triangular.

Motes	Tempo (ms)
2	-
3	350
4	343
5	340
6	-
7	342

Fonte: Autor (extraído de Simulação com Cooja).

O maior número de retransmissões acontece na topologia em quadrado, ocorre *timeout* e a comunicação não tem sucesso, portanto ocorre o maior número de perdas entre as topologias propostas nas condições padrão de simulação.

Tabela 4.3 – Topologia Quadrada.

Motes	Tempo (ms)
2	-
3	-
4	-
5	-

Fonte: Autor (extraído de Simulação com Cooja).

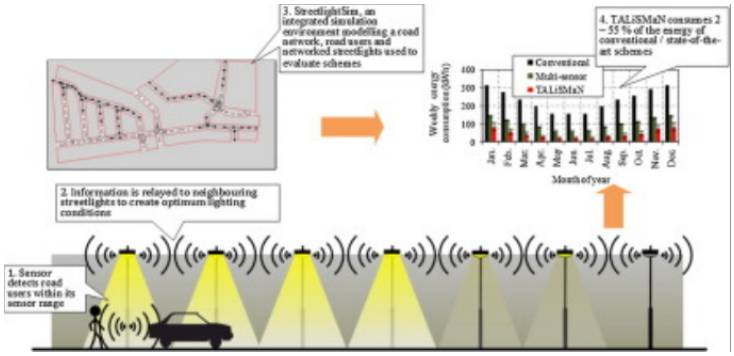
O número de solicitações simultâneas afeta a propagação de mensagens na simulação padrão ao posicionar os *motes* antes da execução propriamente dita. *Hops* e nodos solicitantes formam a topologia de uma rede de baixa potência com protocolo CoAP com 6LoWPAN para o endereçamento de pontos acessados pelo Copper. Menos carga no nó central da rede torna alta chance de obter os dados e a transmissão, sucesso.

A topologia em reta com dois sentidos apresenta a menor média de atraso entre as topologias propostas sem comparar com a reta em um sentido único de comunicação. A topologia com dois solicitantes não encontrou erros na simulação padrão do Cooja. A não-confirmação reduz o tempo nas transmissões e poupa energia, atuar tem resposta linear no gráfico tempo x *hops*, ao passo que obter 3 ou mais tipos de informação é exponencial.

4.2 ILUMINAÇÃO INTELIGENTE

A iluminação principalmente nas cidades tem alto custo ambiental, com isso reduzir o brilho traz melhora no consumo dos recursos naturais. Uma estratégia é encontrada na figura 4.1 que mostra detecção da presença (PINGLAU et al., 2015).

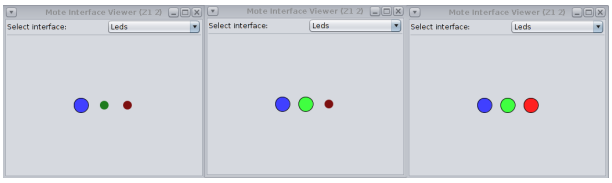
Figura 4.1 – *Smart Lighting*.



Fonte: Adaptado (PINGLAU et al., 2015).

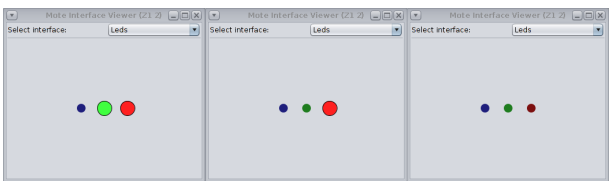
Um homem brasileiro com 70 anos ou mais se locomove à 1,09 m/s em média (NOVAES; MIRANDA; DOURADO, 2011). Caso aplicado uma distância de 1 m para cada sensor, aproximadamente 0,92 s de intervalo de tempo corresponde a um tempo esforçado em agrupar pessoas de diferentes idades. Com isso um LED é aceso simulando uma iluminação inteligente, um por vez é aceso e um botão faz o papel de sensor como na figura 4.2 e a figura 4.3. O tempo adaptado para simulação.

Figura 4.2 – Liga os LEDs.



Fonte: Adaptado da Interface dos LEDs do Cooja.

Figura 4.3 – Desliga os LEDs.



Fonte: Adaptado da Interface dos LEDs do Cooja.

Iluminação de estado sólido fornece inúmeros benefícios, entre eles controle, economia e cuidado ao meio ambiente (SCHUBERT; KIM, 2005). É proposta uma malha e com ela coletamos a seguinte tabela de endereços (tabela 4.4), remotamente aciona-se e monitora-se quantas vezes o sensor capta movimentos, a simulação é exibida na figura 4.4. Os motes são adicionados um a um no tempo de simulação e assim não se depara-se com erros na comunicação, diferentemente dos testes com posicionamento dos *motes* feitos no início, antes de começar a simulação.

- Ao pressionar o botão, simula-se objeto móvel passando pelo primeiro sensor.
- Ele tem 10 segundos (tempo no simulador) para passar ligando o LED seguinte *checkpoint*, senão É desligado o anterior e após o tempo esgotado o atual.
- Cada LED caso não alcançado o próximo *checkpoint* é desligado no tempo estabelecido.
- O código está contido no Apêndice D.

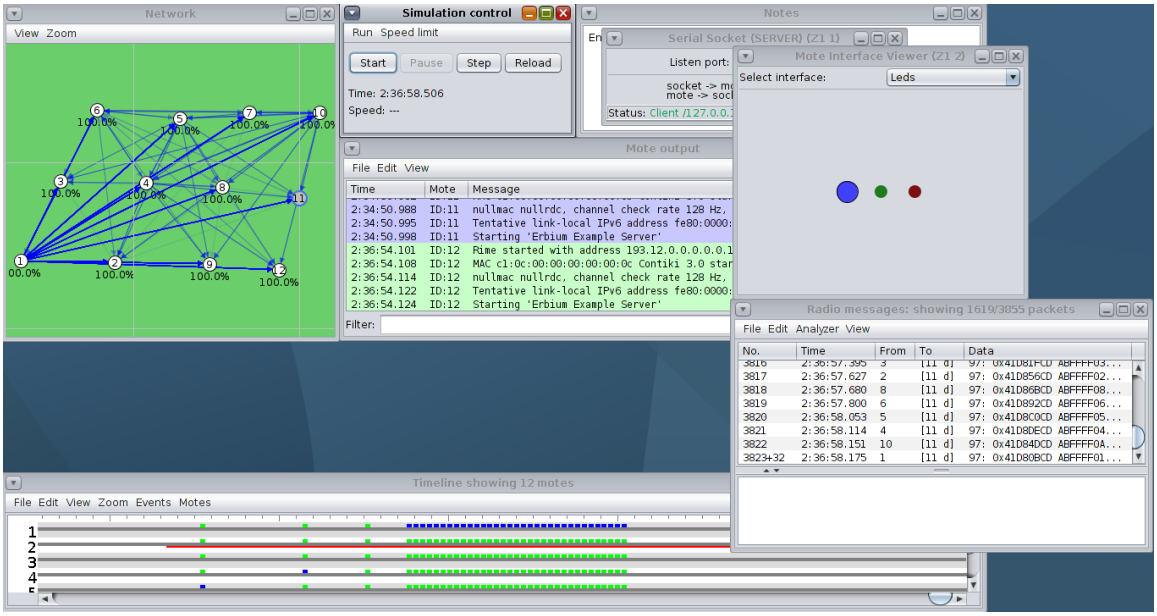
Tabela 4.4 – Tabela de Endereços CoAP.

Motes	Endereço
2	coap://[aaaa::c30c:0:0:2]
3	coap://[aaaa::c30c:0:0:3]
4	coap://[aaaa::c30c:0:0:4]
5	coap://[aaaa::c30c:0:0:5]
6	coap://[aaaa::c30c:0:0:6]
7	coap://[aaaa::c30c:0:0:7]
8	coap://[aaaa::c30c:0:0:8]
10	coap://[aaaa::c30c:0:0:a]
11	coap://[aaaa::c30c:0:0:b]
12	coap://[aaaa::c30c:0:0:c]

Fonte: Adaptado de [aaaa::c30c:0:0:1], Contiki RPL.

A figura 4.4 o monitoramento dos sensores a iluminação gerada pela interface dos LEDs, contendo as informações mostradas na simulação pelo lado do simulador Cooja.

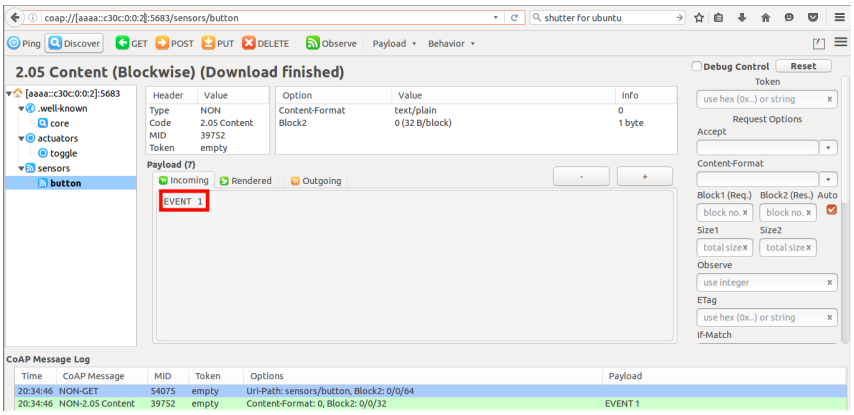
Figura 4.4 – Simulação de Rede LoWPAN para Iluminação com LEDs.



Fonte: Adaptado do Simulador.

Pela visão do Copper, a cada clique no botão que simula passar por cada sensor, o LED é ou não ligado incrementando "evento" para o Copper (figura 4.5).

Figura 4.5 – Visão pelo Lado do Copper.



Fonte: Adaptado do Add-on Copper no Firefox.

A tarefa de programar com o Contiki para gerar solução para iluminar áreas possui inúmeros desafios, a solução tenta ser escalável. A rede auxilia nesse ponto, mas a aplicação segue situações de mobilidade nesse caso. Uma mensagem leva o tempo de 5 ms na propagação do nodo 10 ao nodo 1 na rede, maior distância encontrada.

5 CONCLUSÃO

Concluindo este trabalho os resultados foram animadores. Podemos observar que a latência aumenta conforme a dimensão da rede (SUBRAMANJAN; PASQUALE; POLYZOS, 2017). Na iluminação o monitoramento pelo CoAP encontra junto à IoT controle de informações importantes ao consumo de energia. O protocolo CoAP estabelece comunicação na camada de aplicação interagindo por mensagens com os dispositivos da rede.

Uma aplicação é sugerida para atuar no melhor emprego de energia na iluminação, fazendo agir sob a demanda de presenças. A solução para o problema do consumo de recursos despendidos na iluminação inclui as redes de baixa potência e protocolos adequados a elas com o Contiki. Os resultados em gráficos contemplam situações de coleta de dados de um sensor inserido na rede.

Os melhores resultados são sem retransmissões nem perdas porque através das simulações a topologia interfere na comunicação, pois ocasiona sucesso ou não na transmissão. As solicitações também são fator determinante a eficiência na comunicação, alcançada de melhor modo reduzindo a carga do roteador de borda.

Portanto dessa forma se alcança o objetivo com sucesso através dos resultados das simulações com Cooja na análise da latência do CoAP utilizando uma solução de IoT inteligente para reduzir a energia gasta na iluminação.

REFERÊNCIAS BIBLIOGRÁFICAS

ALI, H. **A Performance Evaluation of RPL in Contiki: A Cooja Simulation based study**. 2012. 91 f. Dissertação (Master's Thesis in Computer Science) — Swedish Institute of Computer Science, Sweden, 2012.

ATMEGA128. ATMEL. **ATmega128 Datasheet**: 8-bit atmel microcontroller with 128kbytes in-system programmable flash. Rev. X. Atmel, 2011. 386 p. Acesso em 21 out. 2017. Disponível em: <http://www.atmel.com/pt/br/Images/doc2467.pdf>.

BHATTACHARYYA, A. et al. **RFC: 7967: "Constrained Application Protocol (CoAP) Option for No Server Response"**. United States, 2016. 18 p.

BORMANN, C.; CASTELLANI, A. P.; SHELBY, Z. Coap: An application protocol for billions of tiny internet nodes. **IEEE Internet Computing**, v. 16, n. 2, p. 62–67, 2012.

COLINA, A. L. et al. **IoT in 5 days**. Bogotá: Autoedición, 2016. 227 p.

COLITTI, W. et al. Evaluation of constrainedapplication protocol for wireless sensor networks. **Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on**, p. 6, 2011.

DUNKELS, A.; GRöNVALL, B.; VOIGT, T. Contiki - a lightweight and flexible operating system for tiny networked sensors. Tampa, United States, nov. 2004. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.2303&rep=rep1&type=pdf>.

FIELDING, R. et al. **RFC 2616:"Hypertext Transfer Protocol – HTTP/1.1"**. United States, 1999. 176 p.

FIELDING, T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. 180 p. Tese (Doutorado Ciência da Computação) — University of California, Irvine, 2000.

FOROUZAN, B. A. **Data Communications and Networking**. 4. ed. [s.n.], 2007. Disponível em: <http://library.aceondo.net/ebooks/Computer_Science/Data_Communication_and_Networking_by_Behrouz.A.Forouzan_4th.edition.pdf>.

HOUAISS, A. **Dicionário Houaiss da Língua Portuguesa**. Rio de Janeiro: Objetiva, 2001. 604 p.

IEEE Computer Society. **IEEE 802.15.4**. Rev. IEEE Std 802.15.4-2006. New York, 2011. 314 p.

IoT Networking Research Group. **Cooja Simulator Manual**. United Kingdom, 2016. 26 p.

iotonlinestore. **The Internet of Things: What it is and why you should care**: A us$ 300 billion industry by 2020. Dubai, 2016. Acesso em 02 dez. 2017. Disponível em: <http://www.iotonlinestore.com/>.

KOVATSCH, M.; DUQUENNOY, S.; DUNKELS, A. A Low-power CoAP for Contiki. In: **Proceedings of the IEEE Workshop on Internet of Things Technology and Architectures**. Valencia, Spain: [s.n.], 2011. Disponível em: <http://dunkels.com/adam/kovatsch11low-power.pdf>.

MEMSIC LEADER IN MEMS SENSOR TECHNOLOGY. **MICAZ**: Wireless measurement system. Rev. A. San Jose, s.d. 2 p. Acesso em 21 out. 2017. Disponível em: <http://www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf>.

MONTENEGRO, G. et al. **RFC 4944: "Transmission of IPv6 Packets over IEEE 802.15.4 Networks"**. United States, 2007. 30 p.

MOTEIV CORPORATION. **Tmote™Sky: Ultra low power IEEE 802.15.4 compliant wireless sensor module**: Humidity, light, and temperature sensors with usb. San Francisco, 2006. 28 p. Acesso em 21 out. 2017. Disponível em: <http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf>.

MSP430G2x. **Low-power MCU | MSP430G2x/i2x | Overview | Ultra-low power | TI.com**: 16-bit msp430g2x/i2x value line sensing microcontrollers. Texas Instruments, 1995–2017. Acessado em 21 out. 2017. Disponível em: <http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/msp430g2x-i2x/overview.html>.

MUSZNICKI, B.; ZWIERZYKOWSKI, P. Survey of simulators for wireless sensor networks. **International Journal of Grid and Distributed Computing**, v. 5, p. 28, 2012.

NOVAES, R. D.; MIRANDA, A. S.; DOURADO, V. Z. Velocidade usual da marcha em brasileiros de meia idade e idosos. **Rev Bras Fisioter**, v. 15, p. 6, 2011.

PINGLAU, S. et al. A traffic-aware street lighting scheme for smart cities using autonomous networked sensors. **Computers Electrical Engineering**, v. 45, p. 197–207, 2015.

RESCORLA, E. et al. **RFC 6347: "Datagram Transport Layer Security Version 1.2"**. United States, 2012. 32 p.

Ron Segal. **IoT WSN CoAP DTLS 6lbr**: Armote iot architecture. Moonshine Hill Road, 2015. Acesso em 02 dez. 2017. Disponível em: <http://wordpress.suretronic.com/tags/iot-wsn-coap-dtls-6lbr>.

SCHUBERT, E. F.; KIM, J. K. Solid-state light sources getting smart. **Science**, v. 308, n. 2, p. 1274–1278, 2005.

SEHGAL, A. **Using the Contiki Cooja Simulator**. Bremen, 2013. 7 p. Acesso em 22 out. 2017. Disponível em: <http://cnds.eecs.jacobs-university.de/courses/iotlab-2013/cooja.pdf>.

SHELBY, Z. et al. **RFC 7252: "The Constrained Application Protocol (CoAP)"**. United States, 2014. 112 p.

SOHRABY, K.; MINOLI, D.; ZNATI, T. **Wireless Sensor Networks**: Technology, protocols, and applications. 1. ed. Canada: John Wiley & Sons, 2007. 326 p.

STALLINGS, W. **Operating Systems: "Internals and Design Principles"**. New Jersey: Prentice Hall, 2012. 820 p.

SUBRAMANJAN, S. S.; PASQUALE, J.; POLYZOS, G. C. Coap for content-centric networks. In: CONSUMER COMMUNICATIONS NETWORKING CONFERENCE (CCNC), 2017 14TH IEEE ANNUAL, Las Vegas. [S.l.], 2017.

TEXAS INSTRUMENTS INCORPORATED. **MSP-EXP430F5438 Experimenter Board**: User's guide. Rev. dec. 2013. Dallas, 2009. 27 p. Acesso em 21 out. 2017. Disponível em: <http://www.ti.com/lit/ug/slau263i/slau263i.pdf>.

TEXAS INSTRUMENTS INCORPORATED. **MSP430F543x and MSP430F541x Mixed-Signal Microcontrollers**. Rev.(E). 2014. Dallas, 2009. 104 p. Acesso em 21 out. 2017. Disponível em: <http://www.ti.com/lit/ds/symlink/msp430f1611.pdf>.

TEXAS INSTRUMENTS INCORPORATED. **SmartRF Transceiver Evaluation Board "TrxEB"**: User's guide. Rev. A. Dallas, 2012. 69 p. Acesso em 21 out. 2017. Disponível em: <http://www.ti.com/lit/ug/swru294a/swru294a.pdf>.

TEXAS INSTRUMENTS INCORPORATED. TEXAS INSTRUMENTS. **MSP430F15x, MSP430F16x, MSP430F161x MIXED SIGNAL MICROCONTROLLER**. Rev. mar. 2011. Dallas, 2002. 79 p. Acesso em 21 out. 2017. Disponível em: <http://www.ti.com/lit/ds/symlink/msp430f1611.pdf>.

TI MSP430. **TI MSP430**. Wikimedia, 2017. Acessado em 21 out. 2017. Disponível em: <https://en.wikipedia.org/wiki/TI_MSP430>.

TJENSVOLD, J. M. Comparison of the ieee 802.11, 802.15.1, 802.15.4 and 802.15.6 wireless standards. In: . [S.l.: s.n.], 2007.

TSVETKOV, T. RPL: IPv6 Routing Protocol for Low Power and Lossy Networks. Munich, Germany, nov. 2011. Disponível em: <https://pdfs.semanticscholar.org/194b/6d79e3f05bc6dcf771c8d8233a0c9f3ed0be.pdf?_ga=2.140223159.504798312.1512342310-787973103.1512342310>.

WINTER, T. et al. **RFC 6550: "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks"**. United States, 2012. 157 p.

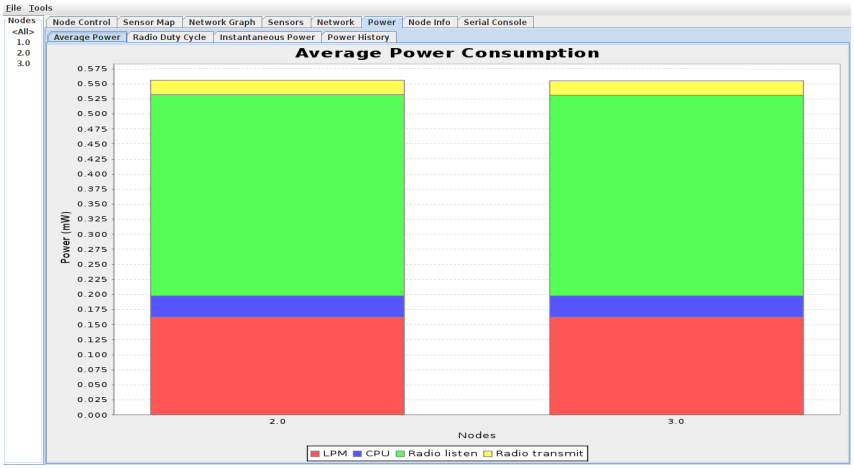
Z1, ZOLERTIA. **Zolertia Z1 Datasheet**. Rev. C. Zolertia, 2010. 20 p. Acesso em 21 out. 2017. Disponível em: <http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf>.

APÊNDICE A – CONSUMO DO TRANSCEPTOR DO Z1

O Contiki OS possui diversas aplicações, o *Collect View* é um serviço que utiliza aplicações e através delas é possível obter dados sobre a rede. É utilizado o Java para mostrar as informações da aplicação e ela é parte do Cooja. Uma interface dá acesso à programar os nodos diretamente através de edição de *script* de simulação, ele envia mensagens em 60000 ms (1 minuto).

O nodo programado como sink recebe mensagem do sender, ele ganha acesso à informações e a aplicação as dispõe. Ele faz um *log* que é mostrado em *javascript*. No *Collect View* a rede recebe um monitoramento interno no período de 60 segundos resumidamente, mostrado na figura A.1.

Gráfico A.1 – Descrição da Potência Consumida.



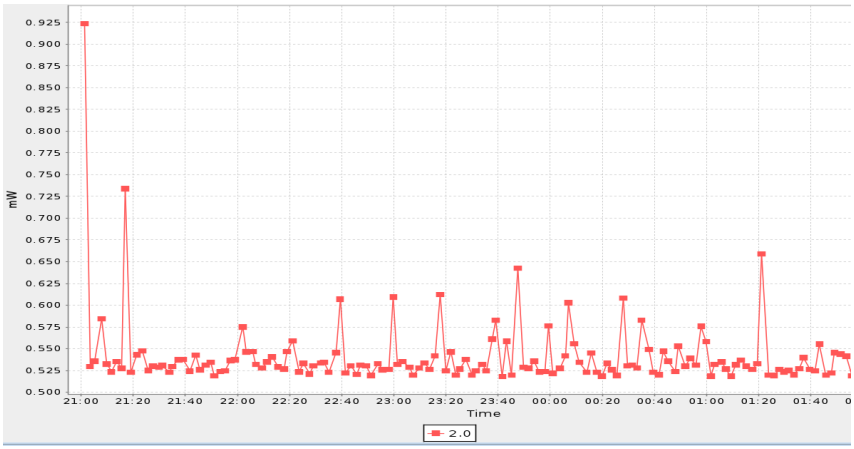
Fonte: Simulação feita pelo autor

Pela unidade utilizada, é a potência ativa e o módulo é o componente menos eficiente nesse caso. Ela foi realizada com somente três *motes*: 1 (um) mote originário de home/contiki/examples/ipv6/rpl-collect/udp-sink.c e os outros 2 (dois) home/contiki/examples/ipv6/rpl-collect/udp-sender.c. O consumo do módulo RF é grande, em azul o gasto da CPU por exemplo.

APÊNDICE B – POTÊNCIA CONSUMIDA DO Z1 E DO SKY EM MILLIWATT (MW)

No apêndice B, a aplicação exibe a potência dos nodos *senders*, neste caso um somente. Ele utiliza a mesma aplicação para visualizar o consumo no tempo, a cada 60 segundos os valores são atualizados. O sistema Z1 consome menos que o Sky, mas possui menos sensores (iluminação e temperatura), resultando em menor gasto, porém ele suporta maiores aplicações porque tem maior memória (ver tabela 2.2).

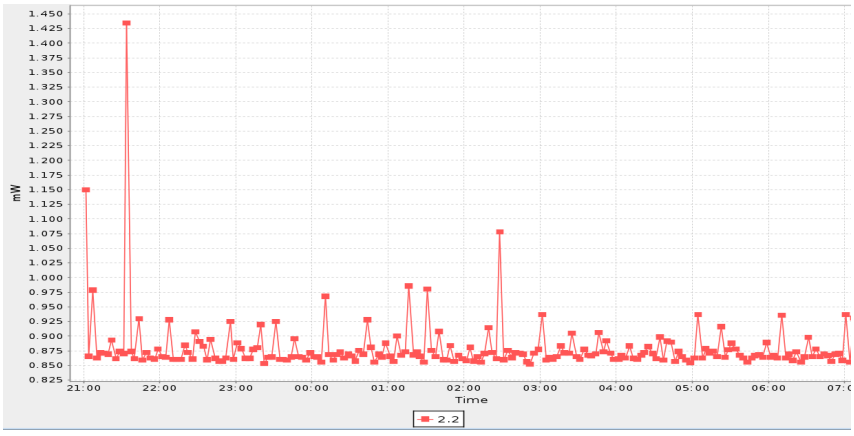
Gráfico B.1 – Potência Consumida (Z1).



Fonte: Simulação feita pelo autor

São simulações que mostram o histórico de consumo. É possível perceber a diferença entre os sistemas para LoWPAN disponíveis e fazer uma comparação.

Gráfico B.2 – Potência Consumida (Sky).

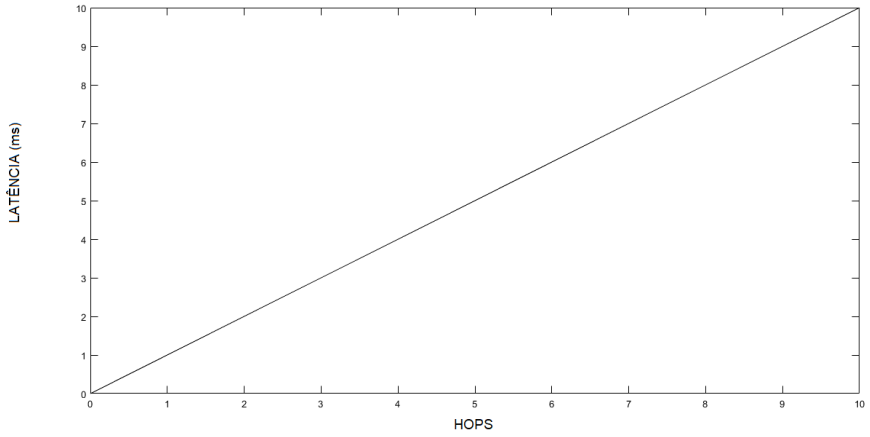


Fonte: Simulação feita pelo autor

APÊNDICE C – GRÁFICOS DAS FUNÇÕES EXPONENCIAL E LINEAR

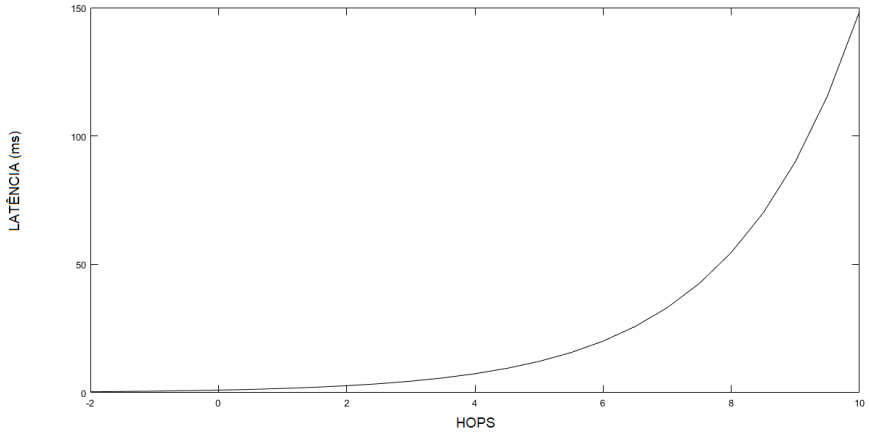
Dadas as funções na página 50 podem ser traçados os seguintes gráficos C.1 e C.2 abaixo. Eles representam o número de saltos e a latência das mensagens CoAP de forma ilustrativa, merecendo importância especialmente sua forma reta linear e curva exponencial.

Gráfico C.1 – Gráfico de f(x) = x.



Fonte: Autor

Gráfico C.2 – Gráfico da f(x) = e^{x}.



Fonte: Autor

APÊNDICE D – APLICAÇÃO DESENVOLVIDA PARA ILUMINAÇÃO

```
            /*

/*
 * 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) REST Engine example.
 * \author
 *      Matthias Kovatsch <kovatsch@inf.ethz.ch>
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
44 #include "contiki.h"
45 #include "contiki-net.h"
46 #include "rest-engine.h"
47 #include "dev/leds.h"
48 #include "sys/etimer.h"
49 //#if PLATFORM_HAS_BUTTON
50 #include "dev/button-sensor.h"
51 //#endif
52
53 #define DEBUG 0
54 #if DEBUG
55 #include <stdio.h>
56 #define PRINTF(...) printf(__VA_ARGS__)
57 #define PRINT6ADDR(addr) PRINTF("[%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x
    :%02x%02x:%02x%02x:%02x%02x]", ((uint8_t *)addr)[0], ((uint8_t *)addr)[1],
    ((uint8_t *)addr)[2], ((uint8_t *)addr)[3], ((uint8_t *)addr)[4], ((uint8_t
    *)addr)[5], ((uint8_t *)addr)[6], ((uint8_t *)addr)[7], ((uint8_t *)addr)
    [8], ((uint8_t *)addr)[9], ((uint8_t *)addr)[10], ((uint8_t *)addr)[11], ((
    uint8_t *)addr)[12], ((uint8_t *)addr)[13], ((uint8_t *)addr)[14], ((uint8_t
     *)addr)[15])
58 #define PRINTLLADDR(lladdr) PRINTF("[%02x:%02x:%02x:%02x:%02x:%02x]", (lladdr)
    ->addr[0], (lladdr)->addr[1], (lladdr)->addr[2], (lladdr)->addr[3], (lladdr)
    ->addr[4], (lladdr)->addr[5])
59 #else
60 #define PRINTF(...)
61 #define PRINT6ADDR(addr)
62 #define PRINTLLADDR(addr)
63 #endif
64
65 /*
66  * Resources to be activated need to be imported through the extern keyword.
67  * The build system automatically compiles the resources in the corresponding
     sub-directory.
68  */
69 extern resource_t
70   res_hello,
71   res_mirror,
72   res_chunks,
73   res_separate,
74   res_push,
75   res_event,
76   res_sub,
77   res_b1_sep_b2;
78 #if PLATFORM_HAS_LEDS
79 extern resource_t res_leds, res_toggle;
80 #endif
81 #if PLATFORM_HAS_LIGHT
```

```
82 #include "dev/light-sensor.h"
83 extern resource_t res_light;
84 #endif
85 /*
86 #if PLATFORM_HAS_BATTERY
87 #include "dev/battery-sensor.h"
88 extern resource_t res_battery;
89 #endif
90 #if PLATFORM_HAS_RADIO
91 #include "dev/radio-sensor.h"
92 extern resource_t res_radio;
93 #endif
94 #if PLATFORM_HAS_SHT11
95 #include "dev/sht11/sht11-sensor.h"
96 extern resource_t res_sht11;
97 #endif
98 */
99
100 int botao;
101
102 PROCESS(er_example_server, "Erbium Example Server");
103 AUTOSTART_PROCESSES(&er_example_server);
104
105 PROCESS_THREAD(er_example_server, ev, data)
106 {
107
108
109   PROCESS_BEGIN();
110
111   PROCESS_PAUSE();
112
113   SENSORS_ACTIVATE(button_sensor);
114
115   leds_off(LEDS_ALL);
116
117   PRINTF("Starting Erbium Example Server\n");
118
119
120
121 #ifdef RF_CHANNEL
122   PRINTF("RF channel: %u\n", RF_CHANNEL);
123 #endif
124 #ifdef IEEE802154_PANID
125   PRINTF("PAN ID: 0x%04X\n", IEEE802154_PANID);
126 #endif
127
128   PRINTF("uIP buffer: %u\n", UIP_BUFSIZE);
```

```
129   PRINTF("LL header: %u\n", UIP_LLH_LEN);
130   PRINTF("IP+UDP header: %u\n", UIP_IPUDPH_LEN);
131   PRINTF("REST max chunk: %u\n", REST_MAX_CHUNK_SIZE);
132 
133   /* Initialize the REST engine. */
134   rest_init_engine();
135 
136   /*
137    * Bind the resources to their Uri-Path.
138    * WARNING: Activating twice only means alternate path, not two instances!
139    * All static variables are the same for each URI path.
140    */
141 //  rest_activate_resource(&res_hello, "test/hello");
142 /*  rest_activate_resource(&res_mirror, "debug/mirror"); */
143 /*  rest_activate_resource(&res_chunks, "test/chunks"); */
144 /*  rest_activate_resource(&res_separate, "test/separate"); */
145 //  rest_activate_resource(&res_push, "test/push");
146   rest_activate_resource(&res_event, "sensors/button");
147 /*  rest_activate_resource(&res_sub, "test/sub"); */
148 /*  rest_activate_resource(&res_b1_sep_b2, "test/b1sepb2"); */
149 #if PLATFORM_HAS_LEDS
150 //  rest_activate_resource(&res_leds, "actuators/leds");
151   rest_activate_resource(&res_toggle, "actuators/toggle");
152 #endif
153 #if PLATFORM_HAS_LIGHT
154   rest_activate_resource(&res_light, "sensors/light");
155   SENSORS_ACTIVATE(light_sensor);
156 #endif
157 /*
158 #if PLATFORM_HAS_BATTERY
159   rest_activate_resource(&res_battery, "sensors/battery");
160   SENSORS_ACTIVATE(battery_sensor);
161 #endif
162 #if PLATFORM_HAS_RADIO
163   rest_activate_resource(&res_radio, "sensors/radio");
164   SENSORS_ACTIVATE(radio_sensor);
165 #endif
166 #if PLATFORM_HAS_SHT11
167   rest_activate_resource(&res_sht11, "sensors/sht11");
168   SENSORS_ACTIVATE(sht11_sensor);
169 #endif
170 */
171 
172   /* Define application-specific events here. */
173 
174   while(1) {
175       static struct etimer et;
```

```
176       PROCESS_WAIT_EVENT();
177 //#if PLATFORM_HAS_BUTTON
178     if(ev == sensors_event && data == &button_sensor) {
179       PRINTF("*******BUTTON*******\n");
180 
181       /* Call the event_handler for this application-specific event. */
182       res_event.trigger();
183       leds_on(LEDS_BLUE);
184       etimer_set(&et, CLOCK_SECOND*5);
185       PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et) || (ev == sensors_event &&
    data == &button_sensor));
186       if(ev == sensors_event && data == &button_sensor) {
187     res_event.trigger();
188           leds_on(LEDS_GREEN);
189     etimer_set(&et, CLOCK_SECOND*5);
190           PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et) || (ev == sensors_event
    && data == &button_sensor));
191     if(ev == sensors_event && data == &button_sensor) {
192             leds_on(LEDS_RED);
193       etimer_set(&et, CLOCK_SECOND*5);
194       PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
195       leds_off(LEDS_BLUE);
196     }
197     etimer_set(&et, CLOCK_SECOND*5);
198           PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
199           leds_off(LEDS_BLUE);
200     if(etimer_expired(&et)) {
201               etimer_reset(&et);
202           }
203     etimer_set(&et, CLOCK_SECOND*5);
204           PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
205           leds_off(LEDS_GREEN);
206     if(etimer_expired(&et)) {
207               etimer_reset(&et);
208           }
209       }
210       etimer_set(&et, CLOCK_SECOND*5);
211       PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
212       leds_off(LEDS_ALL);
213       if(etimer_expired(&et)) {
214           etimer_reset(&et);
215       }
216       /* Also call the separate response example handler. */
217       res_separate.resume();
218     }
219 //#endif /* PLATFORM_HAS_BUTTON */
220   }                             /* while (1) */
```

```
221
222     PROCESS_END();
223 }
224
225
226
```

ANEXO A – ARQUIVO DE CONFIGURAÇÃO DA PLATAFORMA Z1 PARA 25 NODOS

O arquivo fica em: `contiki/platform/z1`, a linha 143 e 145, é modificado para 25 o número de vizinhos e rotas, o padrão é 15.

```
1            /*
2  * Copyright (c) 2010, Swedish Institute of Computer Science.
3  * All rights reserved.
4  *
5  * Redistribution and use in source and binary forms, with or without
6  * modification, are permitted provided that the following conditions
7  * are met:
8  * 1. Redistributions of source code must retain the above copyright
9  *    notice, this list of conditions and the following disclaimer.
10 * 2. Redistributions in binary form must reproduce the above copyright
11 *    notice, this list of conditions and the following disclaimer in the
12 *    documentation and/or other materials provided with the distribution.
13 * 3. Neither the name of the Institute nor the names of its contributors
14 *    may be used to endorse or promote products derived from this software
15 *    without specific prior written permission.
16 *
17 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND
18 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20 * ARE DISCLAIMED.  IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
21 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
22 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
23 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
24 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
25 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
26 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
27 * SUCH DAMAGE.
28 *
29 */
30
31 #ifndef CONTIKI_CONF_H
32 #define CONTIKI_CONF_H
33
34
35 #include "platform-conf.h"
36
37 #define XMAC_CONF_COMPOWER          1
38 #define CXMAC_CONF_COMPOWER         1
39
40 #if NETSTACK_CONF_WITH_IPV6
41
```

```
42 /* Network setup for IPv6 */
43 #define NETSTACK_CONF_NETWORK sicslowpan_driver
44 #define NETSTACK_CONF_MAC      csma_driver
45 #define NETSTACK_CONF_RDC      contikimac_driver
46 #define NETSTACK_CONF_RADIO    cc2420_driver
47 #define NETSTACK_CONF_FRAMER   framer_802154
48
49 /* Specify a minimum packet size for 6lowpan compression to be
50    enabled. This is needed for ContikiMAC, which needs packets to be
51    larger than a specified size, if no ContikiMAC header should be
52    used. */
53 #define SICSLOWPAN_CONF_COMPRESSION_THRESHOLD 63
54
55 #define CC2420_CONF_AUTOACK                1
56 #define NETSTACK_RDC_CHANNEL_CHECK_RATE    8
57 #define RIME_CONF_NO_POLITE_ANNOUCEMENTS   0
58 #define CXMAC_CONF_ANNOUNCEMENTS           0
59 #define XMAC_CONF_ANNOUNCEMENTS            0
60
61 #define QUEUEBUF_CONF_NUM                  4
62
63
64 #else /* NETSTACK_CONF_WITH_IPV6 */
65
66 /* Network setup for non-IPv6 (rime). */
67
68 #define NETSTACK_CONF_NETWORK rime_driver
69 #define NETSTACK_CONF_MAC      csma_driver
70 #define NETSTACK_CONF_RDC      contikimac_driver
71 #define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 8
72 #define NETSTACK_CONF_FRAMER   contikimac_framer
73
74 #define CC2420_CONF_AUTOACK                1
75
76 #define COLLECT_CONF_ANNOUNCEMENTS         1
77 #define RIME_CONF_NO_POLITE_ANNOUCEMENTS   0
78 #define CXMAC_CONF_ANNOUNCEMENTS           0
79 #define XMAC_CONF_ANNOUNCEMENTS            0
80 #define CONTIKIMAC_CONF_ANNOUNCEMENTS      0
81
82 #define CONTIKIMAC_CONF_COMPOWER           1
83 #define XMAC_CONF_COMPOWER                 1
84 #define CXMAC_CONF_COMPOWER                1
85
86 #define COLLECT_NBR_TABLE_CONF_MAX_NEIGHBORS       32
87
88 #define QUEUEBUF_CONF_NUM             8
```

```
89
90 #endif /* NETSTACK_CONF_WITH_IPV6 */
91
92 #define PACKETBUF_CONF_ATTRS_INLINE 1
93
94 #ifdef RF_CHANNEL
95 #define CC2420_CONF_CHANNEL RF_CHANNEL
96 #endif
97
98 #ifndef CC2420_CONF_CHANNEL
99 #define CC2420_CONF_CHANNEL                    26
100 #endif /* CC2420_CONF_CHANNEL */
101
102 #ifndef CC2420_CONF_CCA_THRESH
103 #define CC2420_CONF_CCA_THRESH                 -45
104 #endif /* CC2420_CONF_CCA_THRESH */
105
106 #define IEEE802154_CONF_PANID          0xABCD
107
108 #define SHELL_VARS_CONF_RAM_BEGIN 0x1100
109 #define SHELL_VARS_CONF_RAM_END 0x2000
110
111
112 #define CFS_CONF_OFFSET_TYPE   long
113
114 #define PROFILE_CONF_ON 0
115 #define ENERGEST_CONF_ON 1
116
117 #define ELFLOADER_CONF_TEXT_IN_ROM 0
118 #define ELFLOADER_CONF_DATAMEMORY_SIZE 0x400
119 #define ELFLOADER_CONF_TEXTMEMORY_SIZE 0x800
120
121 #define AODV_COMPLIANCE
122 #define AODV_NUM_RT_ENTRIES 32
123
124 #define WITH_ASCII 1
125
126 #define PROCESS_CONF_NUMEVENTS 8
127 #define PROCESS_CONF_STATS 1
128 /*#define PROCESS_CONF_FASTPOLL     4*/
129
130
131 #define UART0_CONF_TX_WITH_INTERRUPT 0 // So far, printfs without interrupt.
132
133 #ifdef NETSTACK_CONF_WITH_IPV6
134
135 #define LINKADDR_CONF_SIZE                 8
```

```
136
137 #define UIP_CONF_LL_802154                1
138 #define UIP_CONF_LLH_LEN                  0
139
140 #define UIP_CONF_ROUTER                   1
141
142 /* Handle 10 neighbors */
143 #define NBR_TABLE_CONF_MAX_NEIGHBORS      20
144 /* Handle 10 routes    */
145 #define UIP_CONF_MAX_ROUTES    20
146
147 #define UIP_CONF_ND6_SEND_RA      0
148 #define UIP_CONF_ND6_REACHABLE_TIME      600000
149 #define UIP_CONF_ND6_RETRANS_TIMER       10000
150
151 #define NETSTACK_CONF_WITH_IPV6                    1
152 #define UIP_CONF_IPV6_QUEUE_PKT           0
153 #define UIP_CONF_IPV6_CHECKS              1
154 #define UIP_CONF_IPV6_REASSEMBLY          0
155 #define UIP_CONF_NETIF_MAX_ADDRESSES      3
156 #define UIP_CONF_IP_FORWARD               0
157 #define UIP_CONF_BUFFER_SIZE      140
158
159 #define SICSLOWPAN_CONF_COMPRESSION               SICSLOWPAN_COMPRESSION_HC06
160 #ifndef SICSLOWPAN_CONF_FRAG
161 #define SICSLOWPAN_CONF_FRAG                      1
162 #define SICSLOWPAN_CONF_MAXAGE                    8
163 #endif /* SICSLOWPAN_CONF_FRAG */
164 #define SICSLOWPAN_CONF_MAX_ADDR_CONTEXTS         2
165 #else /* NETSTACK_CONF_WITH_IPV6 */
166 #define UIP_CONF_IP_FORWARD         1
167 #define UIP_CONF_BUFFER_SIZE        108
168 #endif /* NETSTACK_CONF_WITH_IPV6 */
169
170 #define UIP_CONF_ICMP_DEST_UNREACH 1
171
172 #define UIP_CONF_DHCP_LIGHT
173 #define UIP_CONF_LLH_LEN            0
174 #define UIP_CONF_RECEIVE_WINDOW    48
175 #define UIP_CONF_TCP_MSS           48
176 #define UIP_CONF_MAX_CONNECTIONS 4
177 #define UIP_CONF_MAX_LISTENPORTS 8
178 #define UIP_CONF_UDP_CONNS         12
179 #define UIP_CONF_FWCACHE_SIZE      30
180 #define UIP_CONF_BROADCAST         1
181 #define UIP_ARCH_IPCHKSUM          1
182 #define UIP_CONF_UDP               1
```

```
183 #define UIP_CONF_UDP_CHECKSUMS    1
184 #define UIP_CONF_PINGADDRCONF     0
185 #define UIP_CONF_LOGGING          0
186
187 #define UIP_CONF_TCP_SPLIT        0
188
189
190 #ifdef PROJECT_CONF_H
191 #include PROJECT_CONF_H
192 #endif /* PROJECT_CONF_H */
193
194
195
196 #endif /* CONTIKI_CONF_H */
197
```

ANEXO B – APLICAÇÃO COAP (ER-EXAMPLE-SERVER) DO CONTIKI

Para adicionar o botão aos recursos e interagir, comentar as linhas 46 e 48. Os recursos podem ser removidos, comentando as linhas 128, 132 e 138.

```
1   /*
2  * Copyright (c) 2013, Institute for Pervasive Computing, ETH Zurich
3  * All rights reserved.
4  *
5  * Redistribution and use in source and binary forms, with or without
6  * modification, are permitted provided that the following conditions
7  * are met:
8  * 1. Redistributions of source code must retain the above copyright
9  *    notice, this list of conditions and the following disclaimer.
10 * 2. Redistributions in binary form must reproduce the above copyright
11 *    notice, this list of conditions and the following disclaimer in the
12 *    documentation and/or other materials provided with the distribution.
13 * 3. Neither the name of the Institute nor the names of its contributors
14 *    may be used to endorse or promote products derived from this software
15 *    without specific prior written permission.
16 *
17 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND
18 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20 * ARE DISCLAIMED.  IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
21 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
22 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
23 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
24 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
25 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
26 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
27 * SUCH DAMAGE.
28 *
29 * This file is part of the Contiki operating system.
30 */
31
32 /**
33 * \file
34 *      Erbium (Er) REST Engine example.
35 * \author
36 *      Matthias Kovatsch <kovatsch@inf.ethz.ch>
37 */
38
39 #include <stdio.h>
40 #include <stdlib.h>
41 #include <string.h>
```

```
42 #include "contiki.h"
43 #include "contiki-net.h"
44 #include "rest-engine.h"
45
46 #if PLATFORM_HAS_BUTTON
47 #include "dev/button-sensor.h"
48 #endif
49
50 #define DEBUG 0
51 #if DEBUG
52 #include <stdio.h>
53 #define PRINTF(...) printf(__VA_ARGS__)
54 #define PRINT6ADDR(addr) PRINTF("[%02x%02x:%02x%02x:%02x%02x:%02x%02x
    :%02x%02x:%02x%02x:%02x%02x]", ((uint8_t *)addr)[0], ((uint8_t *)addr)[1],
    ((uint8_t *)addr)[2], ((uint8_t *)addr)[3], ((uint8_t *)addr)[4], ((uint8_t
    *)addr)[5], ((uint8_t *)addr)[6], ((uint8_t *)addr)[7], ((uint8_t *)addr)
    [8], ((uint8_t *)addr)[9], ((uint8_t *)addr)[10], ((uint8_t *)addr)[11], ((
    uint8_t *)addr)[12], ((uint8_t *)addr)[13], ((uint8_t *)addr)[14], ((uint8_t
     *)addr)[15])
55 #define PRINTLLADDR(lladdr) PRINTF("[%02x:%02x:%02x:%02x:%02x:%02x]", (lladdr)
    ->addr[0], (lladdr)->addr[1], (lladdr)->addr[2], (lladdr)->addr[3], (lladdr)
    ->addr[4], (lladdr)->addr[5])
56 #else
57 #define PRINTF(...)
58 #define PRINT6ADDR(addr)
59 #define PRINTLLADDR(addr)
60 #endif
61
62 /*
63  * Resources to be activated need to be imported through the extern keyword.
64  * The build system automatically compiles the resources in the corresponding
     sub-directory.
65  */
66 extern resource_t
67   res_hello,
68   res_mirror,
69   res_chunks,
70   res_separate,
71   res_push,
72   res_event,
73   res_sub,
74   res_b1_sep_b2;
75 #if PLATFORM_HAS_LEDS
76 extern resource_t res_leds, res_toggle;
77 #endif
78 #if PLATFORM_HAS_LIGHT
79 #include "dev/light-sensor.h"
```

```
80 extern resource_t res_light;
81 #endif
82 /*
83 #if PLATFORM_HAS_BATTERY
84 #include "dev/battery-sensor.h"
85 extern resource_t res_battery;
86 #endif
87 #if PLATFORM_HAS_RADIO
88 #include "dev/radio-sensor.h"
89 extern resource_t res_radio;
90 #endif
91 #if PLATFORM_HAS_SHT11
92 #include "dev/sht11/sht11-sensor.h"
93 extern resource_t res_sht11;
94 #endif
95 */
96
97 PROCESS(er_example_server, "Erbium Example Server");
98 AUTOSTART_PROCESSES(&er_example_server);
99
100 PROCESS_THREAD(er_example_server, ev, data)
101 {
102   PROCESS_BEGIN();
103
104   PROCESS_PAUSE();
105
106   PRINTF("Starting Erbium Example Server\n");
107
108 #ifdef RF_CHANNEL
109   PRINTF("RF channel: %u\n", RF_CHANNEL);
110 #endif
111 #ifdef IEEE802154_PANID
112   PRINTF("PAN ID: 0x%04X\n", IEEE802154_PANID);
113 #endif
114
115   PRINTF("uIP buffer: %u\n", UIP_BUFSIZE);
116   PRINTF("LL header: %u\n", UIP_LLH_LEN);
117   PRINTF("IP+UDP header: %u\n", UIP_IPUDPH_LEN);
118   PRINTF("REST max chunk: %u\n", REST_MAX_CHUNK_SIZE);
119
120   /* Initialize the REST engine. */
121   rest_init_engine();
122
123   /*
124    * Bind the resources to their Uri-Path.
125    * WARNING: Activating twice only means alternate path, not two instances!
126    * All static variables are the same for each URI path.
```

```
127     */
128   rest_activate_resource(&res_hello, "test/hello");
129 /*  rest_activate_resource(&res_mirror, "debug/mirror"); */
130 /*  rest_activate_resource(&res_chunks, "test/chunks"); */
131 /*  rest_activate_resource(&res_separate, "test/separate"); */
132   rest_activate_resource(&res_push, "test/push");
133 /*  rest_activate_resource(&res_event, "sensors/button"); */
134 /*  rest_activate_resource(&res_sub, "test/sub"); */
135 /*  rest_activate_resource(&res_b1_sep_b2, "test/b1sepb2"); */
136 #if PLATFORM_HAS_LEDS
137 /*  rest_activate_resource(&res_leds, "actuators/leds"); */
138   rest_activate_resource(&res_toggle, "actuators/toggle");
139 #endif
140 #if PLATFORM_HAS_LIGHT
141   rest_activate_resource(&res_light, "sensors/light");
142   SENSORS_ACTIVATE(light_sensor);
143 #endif
144 /*
145 #if PLATFORM_HAS_BATTERY
146   rest_activate_resource(&res_battery, "sensors/battery");
147   SENSORS_ACTIVATE(battery_sensor);
148 #endif
149 #if PLATFORM_HAS_RADIO
150   rest_activate_resource(&res_radio, "sensors/radio");
151   SENSORS_ACTIVATE(radio_sensor);
152 #endif
153 #if PLATFORM_HAS_SHT11
154   rest_activate_resource(&res_sht11, "sensors/sht11");
155   SENSORS_ACTIVATE(sht11_sensor);
156 #endif
157 */
158
159   /* Define application-specific events here. */
160   while(1) {
161     PROCESS_WAIT_EVENT();
162 #if PLATFORM_HAS_BUTTON
163     if(ev == sensors_event && data == &button_sensor) {
164       PRINTF("*******BUTTON*******\n");
165
166       /* Call the event_handler for this application-specific event. */
167       res_event.trigger();
168
169       /* Also call the separate response example handler. */
170       res_separate.resume();
171     }
172 #endif /* PLATFORM_HAS_BUTTON */
173   }                             /* while (1) */
```

```
174
175     PROCESS_END();
176 }
```