

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA**

Rafael de Lucena Valle

**IMPLEMENTAÇÃO DE UM GATEWAY COAP/GPRS PARA  
SERVIÇOS DE MONITORAMENTO EM REDES DE SENSORES  
SEM FIO**

Florianópolis

2014



## RESUMO

Redes de sensores e atuadores são utilizadas para a captura, processamento de informação e atuação sobre um ambiente, tornando-as importantes em aplicações de controle, telemetria e rastreamento.

Estas redes são compostas por nós sensores que trabalham em conjunto a fim de obter dados de um ambiente. Possuem processadores, transmissores e receptores de dados simplificados, restrições de memória e energia, geralmente sem alimentação constante de energia. Contudo possuem um custo baixo de equipamentos, tornando interessante a implantação destes sistemas.

O protocolo HTTP, um protocolo de aplicação muito utilizado na atualidade, foi desenvolvido para computadores de propósito geral, onde essas restrições não existem. Um protocolo leve como CoAP pode tornar viável o desenvolvimento de aplicações web em redes de sensores sem fio.

Este trabalho propõe uma infraestrutura de comunicação entre redes de sensores sem fio e a Internet, utilizando protocolos leves entre os nós sensores e um gateway GPRS para aproveitar a cobertura da tecnologia GPRS.

Com a implementação do CoAP é esperado uma redução de consumo de energia e memória, em relação a outros protocolos de aplicação existentes.

**Palavras-chave:** internetworking WSN IPv6 GPRS CoAP IoT



## LISTA DE FIGURAS

Figura 1	Visão geral de uma RSSF (DARGIE, POELLABAUER, 2010).	14
Figura 2	Arquitetura Orientada a Serviços .....	16
Figura 3	Diferença de bytes transmitidos (KULADINITHI K., BERGMANN O., PÖTSCH T., BECKER M., GORG C., 2011). ....	17
Figura 4	Formato do pacote CoAP em bits (SHELBY Z., HARTKE K., BORMANN C., 2013).....	18
Figura 5	Resposta na mensagem de confirmação, chamado de piggy-backed.....	20
Figura 6	Fluxo esperado de requisição e resposta sem confirmação....	21
Figura 7	Fluxo esperado de requisição e resposta com confirmação, com resposta separada .....	22
Figura 8	Overview do EPOS. ....	23
Figura 9	Visão geral sobre comunicação do sistema.....	29
Figura 10	Diagrama UML das entidades de software implementadas....	30
Figura 11	Diagrama de casos de uso.....	31



## LISTA DE ABREVIATURAS E SIGLAS

CoAP	Constrained Application Protocol .....
EPOS	Embedded Parallel Operating System .....
HTTP	Hipertext Transfer protocol.....
IETF	Internet Engineering Task Force .....
M2M	Machine-to-Machine .....
REST	Representational State Transfer .....
UDP	User Datagram Protocol .....
IoT	Internet of Things .....
JSON	JavaScript Object Notation .....
XML	Extensible Markup Language.....
ADESD	Application Driven Embedded System Design.....
SOC	Service-Oriented Computing .....
WSN	Wireless Sensor Networks.....
RSSF	Redes de Sensores sem fio .....





## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	9
1.1	OBJETIVOS	9
1.1.1	Objetivos Específicos	9
1.2	JUSTIFICATIVA	10
1.3	METODOLOGIA	11
<b>2</b>	<b>REVISÃO BIBLIOGRAFICA</b>	13
2.1	REDES DE SENSORES SEM FIO	13
2.1.1	Nós sensores	14
2.1.2	Sink Nodes	15
2.2	ARQUITETURA ORIENTADA A SERVIÇOS	15
2.3	WEBSERVICES	16
2.3.1	RESTful	16
2.4	COAP	18
2.4.1	Formato das mensagens	18
2.4.2	Transmissão de Mensagens	19
2.4.3	Requisição e Resposta	21
2.4.4	Opções	22
2.4.5	Recursos	22
2.5	EPOS	23
2.6	TRABALHOS RELACIONADOS	24
2.6.1	OpenWSN	24
2.6.2	Contiki	24
2.6.3	LibCoap	25
2.6.4	CantCoap	26
<b>3</b>	<b>DESENVOLVIMENTO</b>	27
3.1	LEVANTAMENTO DE REQUISITOS	27
3.1.1	Requisitos Funcionais	27
3.1.2	Requisitos Não Funcionais	28
3.2	ESPECIFICAÇÃO	28
3.2.1	Arquitetura	28
3.2.2	Componentes	29
3.3	TESTES	32
<b>4</b>	<b>AVALIAÇÃO</b>	33
4.1	ANÁLISE FUNCIONAL	33
4.1.1	Limitações Funcionais	33
4.2	ANÁLISE ESTRUTURAL	33
4.3	ANÁLISE DE DESEMPENHO	35

<b>5</b>	<b>CONSIDERAÇÕES FINAIS .....</b>	<b>37</b>
<b>5.0.1</b>	<b>Conclusão .....</b>	<b>37</b>
<b>5.0.2</b>	<b>Trabalhos Futuros .....</b>	<b>37</b>
	<b>REFERÊNCIAS .....</b>	<b>39</b>

## 1 INTRODUÇÃO

Redes de sensores e atuadores são utilizadas para a captação, processamento de informação e atuação sobre um ambiente, tornando-as importantes em aplicações de controle, telemetria e rastreamento de sistemas.

Nós que participam destas redes geralmente são compostos por computadores e rádios simplificados, que possuem restrições de memória, processamento, energia e capacidade de comunicação, mas um custo relativamente baixo de equipamentos.

O maior consumo de energia neste tipo de aplicação é do rádio, portanto o desafio dos algoritmos de comunicação nesta área é manter os rádios ligados o mínimo de tempo possível sem comprometer a conectividade do nó.

### 1.1 OBJETIVOS

O objetivo geral desse trabalho é descrever e implementar webservices em uma rede sensores sem fio que farão a aquisição dos dados do ambiente e disponibilizarão as informações captadas na Internet.

#### 1.1.1 Objetivos Específicos

Este trabalho realizará o desenvolvimento de aplicações web e software de sistema para fazer a ponte entre a rede de sensores e a Internet. O Sistema operacional utilizado será o EPOS, que possui uma implementação de pilha UDP/IP. A aplicação integradora GPRS/802.15.4 irá executar na plataforma EposMoteII utilizando uma extensão GPRS.

A comunicação entre os nós da rede será feita através do protocolo de aplicação CoAP, um protocolo específico para redes de sensores sem fio. Será utilizado um porte de uma implementação livre do protocolo CoAP.

Sendo assim, os objetivos específicos são:

1. Portar o protocolo CoAP para o EPOS;
2. Implementar uma aplicação para redes de sensores sem fio;
3. Desenvolver a aplicação gateway GPRS/802.15.4.
4. Desenvolver uma aplicação web para visualização da informação;
5. Avaliar a solução desenvolvida.

## 1.2 JUSTIFICATIVA

Os mecanismos de confiabilidade na transmissão de dados, técnicas para se manter uma conexão do TCP e rearranjos que são feitos para garantir a ordem das mensagens recebidas não são adequados para um dispositivos com suprimento limitado de energia, como uma bateria ou uma placa fotovoltaica. Estas técnicas fazem que os transmissores fiquem ligados por mais tempo, para manter a conexão ou até mesmo para reenvio de mensagens.

O maior consumo de energia de um nó sensor é no envio e recebimento de dados, quando mantém seu transmissor ligado. Além disso quem recebe a mensagem precisa montá-la e tratar as partes corrompidas, podendo gerar retransmissões.

Por sua vez o protocolo do UDP, não mantém conexão, dados são recebidos fora de ordem e o envio é feito de uma mensagem por vez. Isto implica também na redução do tamanho do cabeçalho do pacote.

Estas características demostram uma alternativa interessante para estes equipamentos limitados. Testes feitos em implementações de sistemas operacionais similares ao EPOS, como Contiki e TinyOS, utilizando o protocolo CoAP demonstram redução no consumo de energia e memória em relação ao HTTP (KULADINITHI K., BERGMANN O., PÖTSCH T., BECKER M., GORG C., 2011).

A falta de padronização dos protocolos afeta o desenvolvimento de uma rede pública ubíqua de uma cidade inteligente, por exemplo. Grande parte das soluções utiliza protocolos proprietários que se comunicam apenas com os produtos de um mesmo fabricante.

O protocolo HTTP foi desenvolvido para comunicação de computadores de propósito geral, onde as restrições citadas não são comuns. Em relação ao tamanho, o pacote HTTP é um problema para redes 802.15.4, já que estas redes possuem uma restrição de 128 bytes em sua PDU. O protocolo TCP precisa transmitir mensagens adicionais para manter uma conexão, outra característica que não é interessante para RSSF.

Um protocolo leve como CoAP pode tornar viável a criação de aplicações web em redes de sensores sem fio por um baixo custo. Neste trabalho é proposto uma infraestrutura de comunicação entre redes de sensores sem fio e a Internet, utilizando protocolos leves entre os nós sensores e um gateway GPRS para áreas sem acesso à WIFI, aproveitando a vasta abrangência da tecnologia de telefonia. Com a utilização do CoAP é esperado uma redução de consumo de energia e memória, em relação a outros protocolos de aplicação existentes.

Em lugares aonde não existe o acesso a rede cabeada ou sem fio, como lugares afastados, na área rural, por exemplo a distribuição da informação

para Internet será feita através de um gateway.

O gateway será composto por um EposMoteII e um módulo GPRS, responsável por fazer a ponte entre a rede de sensores e a Internet. Atualmente o padrão GPRS oferece a maior cobertura dentre as tecnologias de transmissão de telefonia no Brasil, atingindo cerca de 5477 municípios.(ANATEL, 2014)

### 1.3 METODOLOGIA

Será feito um levantamento dos componentes de software e hardware necessários para o desenvolvimento do gateway 802.15.4/GPRS. Neste caso utilizando o mote EPOSMote II e um módulo GPRS, que será desenvolvido em paralelo a implementação do protocolo de aplicação CoAP no sistema operacional EPOS.

Durante o desenvolvimento do protocolo, testes serão executados para verificar o correto comportamento e diminuir a depuração em hardware, que geralmente leva mais tempo.

Nos testes de integração do gateway, será utilizada uma placa de desenvolvimento em conjunto com um módulo M95 da Quectel disponibilizada pelo LISHA. Serão realizados testes de envio de mensagens em diversos protocolos, inclusive testes com comandos proprietários adicionais do modem.

Para testes de integração, as aplicações serão executadas na plataforma de sensores sem fio EPOS Mote II utilizando o EPOS com o CoAP desenvolvido.



## 2 REVISÃO BIBLIOGRAFICA

As tecnologias de ambientes inteligentes estão sendo desenvolvidas e serão um grande passo para as áreas como construção civil, indústria, transporte e automação residencial.

Para agirem de forma inteligente, estes ambientes necessitam de informação sobre o contexto aonde serão implantados. As redes de sensores sem fio são um componente chave para estes ambientes, adquirem este contexto captando e comunicando os dados do ambiente (LEWIS F., 2004).

Este capítulo apresenta uma visão geral sobre redes de sensores sem fio, arquitetura orientada a serviços e os protocolos de aplicação existentes.

### 2.1 REDES DE SENSORES SEM FIO

O desenvolvimento destas redes foi inicialmente motivada por aplicações militares e hoje são utilizados em diversas aplicações na indústria, no monitoramento e controle de processos industriais, supervisão de máquinas, monitoramento de ambientes, estruturas e tubulações, automação residencial, coleta de dados de pacientes, entre outros.

Avanços recentes nas tecnologias de sistemas eletrônicos, semicondutores, sensores, microcontroladores e rádios tornaram possível o desenvolvimento de redes de sensores de baixo custo e baixo consumo de energia uma realidade.

Os requisitos para uma rede de sensores distribuída são: reconfiguração com estação base, controle autônomo de operação e gerência de energia, auto-monitoramento, eficiência energética para longo tempo de operação e apta a incorporar diversos sensores (BULT, CHANG, DONG, FIELDING, 1996).

Geralmente tais redes possuem centenas ou milhares de nós sensores e possuem as seguintes características: pouca memória, pouco alcance do rádio, baixa capacidade de processamento e bateria, e custo reduzido. Comunicam-se entre si e com estações base utilizando seus rádios sem fio, permitindo disseminação da informação para processamento remoto, visualização, análise e armazenamento. A figura 2.1 dá uma visão geral sobre a comunicação destes nós.

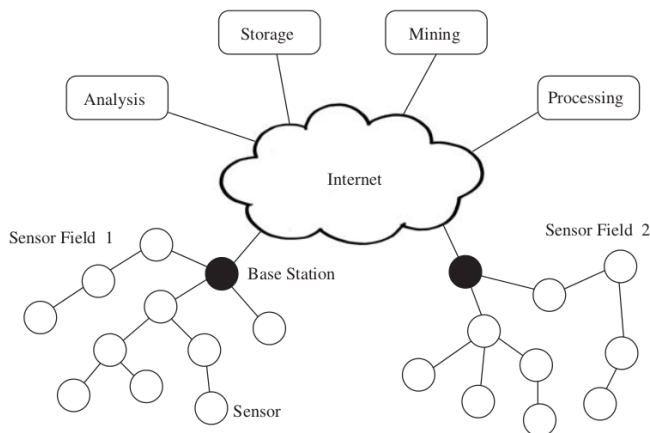


Figura 1 – Visão geral de uma RSSF (DARGIE, POELLABAUER, 2010).

Um nó pertencente a esta rede geralmente é um dispositivo especificamente desenvolvido para um propósito, que possui poucos recursos computacionais e energéticos e se comunicam entre seus semelhantes.

Muitas redes de sensores também possuem atuadores, que permitem controle direto ao mundo real. Uma rede de sensores e atuadores recebe comandos da unidade de processamento (microcontrolador) e transforma estes comandos em sinais de entrada para um atuador, que interage com processos físicos. Estas redes são chamadas de redes de sensores e atuadores sem fio, ou WSN.

A conservação de energia é um dos objetivos das redes de sensores sem fio, pois não estão ligados diretamente a fonte de energia. Deve-se minimizar o consumo em todos os níveis do sistema, da aplicação até o meio físico, iniciando com o projeto de rádio. (YICK, MUKHERJEE, GHOSAL, 2008)

### 2.1.1 Nós sensores

Sensores ligam o físico com o mundo digital, capturando e revelando fenômenos do mundo real e convertendo em uma forma que pode ser processada, armazenada e utilizada para tomada de decisão. A escolha entre qual sensor a ser utilizado depende muito da aplicação e da propriedade física a ser monitorada, alguns exemplos: temperatura, pressão, humidade, entre outros.

Geralmente os nós sensores são compostos por um microcontrolador,



transceiver, memória externa, fonte de energia e um ou mais sensores. Os nós são responsáveis por coletar dados, fazer uma certa análise da rede, correlação entre dados com outros nós e comunicação com uma estação base afim de centralizar a informação, para um processamento externo (DARGIE, POELLABAUER, 2010).

### 2.1.2 Sink Nodes

Como estas redes utilizam um espectro diferente de comunicação, é necessário uma integração com a rede já conhecida. Uma forma de integrar os nós sensores é utilizar nós específicos para executarem este trabalho.

Um exemplo é utilizar um nó como estação de sincronização, porém surge um problema, pois o nó sincronizador deve ter conhecimento de todos os recursos disponíveis na rede.

Quando os recursos são expostos diretamente pelo próprio dispositivo como num protocolo de aplicação tipo o CoAP a complexidade do nó gateway é bem reduzida, já que o papel é simplesmente repassar os pacotes de descoberta de recursos para Internet (COLITTI W., STEENHAUT K., DE CARO N., 2011).

## 2.2 ARQUITETURA ORIENTADA A SERVIÇOS

Arquitetura orientada a serviços é uma forma de organizar infraestrutura e aplicações de software em um conjunto de serviços. Estes são oferecidos por prestadores de serviço, servidores, organizações que implementam os serviços, fornecem descrição dos serviços oferecidos, suporte técnico e de negócio.

O modelo de computação utilizando este paradigma é conhecido como Computação Orientada a serviços (SOC) (PAPAZOGLU M., 2003).

Clientes destes serviços podem ser outras soluções, aplicações, processos ou usuários. Para satisfazer estes requisitos serviços devem:

**Tecnologicamente neutros:** utilizar-se de padrões reconhecidos e bem aceitos para comunicação, descrição e mecanismos de descoberta;

**Baixo acoplamento:** detalhes desnecessários (o quão desnecessário precisa ser discutido) devem ser escondidos do cliente, que não precisa ter conhecimento sobre o funcionamento interno para utilizar o serviço;

**Localidade transparente:** clientes devem ser atendidos independentemente da localidade do serviço disponível.

SOA não é apenas uma arquitetura sobre serviços, mas um relacionamento entre três entidades: o provedor de serviço (service provider), descoberta de serviço (service discovery agency) e o client (service requestor). Abaixo a figura 2.2 demonstra este relacionamento e suas interações: publicar (publish), encontrar (find) e vincular (bind).

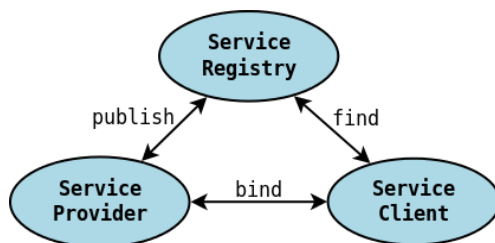


Figura 2 – Arquitetura Orientada a Serviços

## 2.3 WEBSERVICES

Um serviço web é um sistema de software projetado para suportar interoperabilidade em interações máquina-a-máquina na rede. Possui uma interface descritiva das funcionalidades num formato padronizado (especificamente WSDL). Esta notação abstrata que deve ser implementada por um agente (W3C Working Group, 2004).

O agente é algo concreto, pode ser um pedaço de hardware ou software que recebe e envia mensagens. Um exemplo é implementar o mesmo serviço web, utilizando agentes em diferentes linguagens. Embora o agente seja diferente, o serviço Web continua o mesmo.

Serviços Web provêm um padrão para a interoperabilidade entre diferentes aplicações de software, que executam em diferentes plataformas de software e hardware.

### 2.3.1 RESTful

RESTful é uma maneira de aplicar os princípios de design REST para serviços web. Uma requisição a um webservice RESTful, utiliza a informação do método como um verbo HTTP e a informação do escopo ao qual o verbo será utilizado na URI. Ao contrário um estilo RPC tende a ignorar o método HTTP, procurando pelo método a ser utilizado e o escopo na própria URI.

As operações suportas são métodos HTTP explícitos que não salvam estado das aplicações clientes e são idempotentes, são eles:

**GET:** solicita ao webserver a representação de uma informação de um determinado recurso.

**POST:** cria um recurso no webserver.

**PUT:** muda o estado de um recurso do webserver.

**DELETE:** remove o recurso ou alterar para um estado vazio.

Os Recursos usam um identificador único e persistente, as URIs. A URIs possuem estruturas de diretórios, uma URI é uma árvore com ramos subordinados e superordinados conectando os nós (FIELDING R., 2000).

Para transfeência de dados utiliza-se formatos genéricos que enfatizam simplicidade e usabilidade pela internet, como XML e JSON. Resource Oriented Architecture é uma arquitetura boa para RESTful webservices (RICHARDSON L., RUBY S., 2008).

Uma abordagem utilizando HTTP não é tão interessante para uma aplicação de RSSF, já que a quantidade de informação a ser transmitida é consideravelmente maior. A figura 2.3.1 faz um comparativo entre o número de bytes transmitidos de diversos servidores web e seus protocolos.

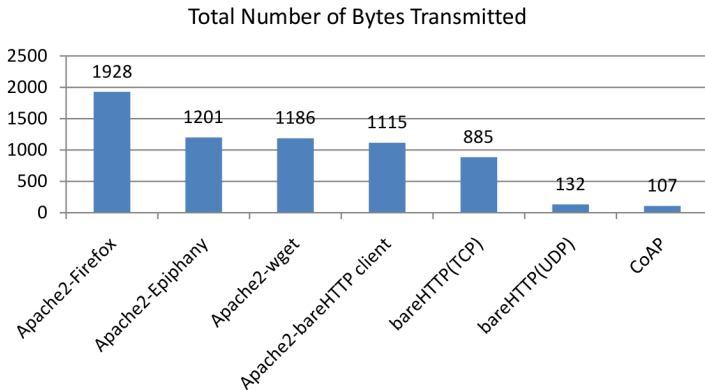


Figura 3 – Diferença de bytes transmitidos (KULADINITHI K., BERGMANN O., PÖTSCH T., BECKER M., GORG C., 2011).

Todos os webservices utilizam o conceito de URI, porém de maneiras diferentes. Mas um serviço RESTful sempre expõe a URI para cada pedaço

de dado que o cliente quer operar sobre. Os principais competidores aos serviços web RESTful são os RPC-styles.

2.4 COAP

Um dos principais objetivos do CoAP é ser uma alternativa protocolo web genérico para redes com dispositivos com restrição de energia e memória.

As vantagens de utilizar um protocolo compatível com o HTTP são a facilidade de integração e o reuso de aplicações. CoAP é um conjunto REST otimizado para M2M, com suporte a descoberta de recursos, multicast e troca de mensagens assíncronas com simplicidade e baixo overhead.

A IETF estabelece as condições mínimas para o desenvolvimento de um protocolo de aplicação compatível com HTTP, mas focado em aplicações aonde energia e memória são escassas. O protocolo CoAP foi projetado levando em consideração as restrições energéticas e altas taxas de falha na transmissão dos pacotes em RSSF.

A comunicação entre os pontos no CoAP é assíncrona usando o UDP. A confiabilidade é um parâmetro opcional e funciona através de um mecanismo de retransmissão exponencial.

Possui 4 tipos de mensagem: Confirmável, Não-Confirmável, Confirmação (ACK) e Reset. A figura 2.4.1 mostra o formato do pacote.

2.4.1 Formato das mensagens

Uma mensagem CoAP deve caber num único pacote IP, para que seja transmitida numa camada de enlace limitada.

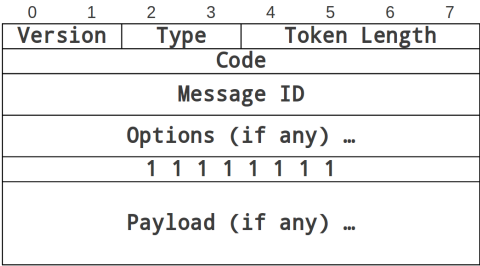


Figura 4 – Formato do pacote CoAP em bits (SHELBY Z., HARTKE K., BORMANN C., 2013).

Os campos do pacote CoAP são: a versão do CoAP, implementações devem utilizar este campo com o valor 1. O tipo: campo para definir o tipo da mensagem: Confirmável (0), Não-Confirmável (1), de Confirmação (2) ou Reset (3).

O tamanho do Token: utilizado para controle de requisições e repostas. O tamanho do Token pode variar entre 0 e 8 bytes. Tamanhos entre 9 a 15 são reservados e não devem ser usados. É um campo sempre gerado pelo cliente CoAP.

O Código: separados em 3-bit mais significativos para classes e 5-bits menos significativos para detalhe. As classes podem indicar uma requisição (0), uma resposta de sucesso (2), e uma resposta de erro do cliente (4), ou uma resposta de erro do servidor (5), as outras classes são reservadas. Em um caso especial o código 0.00 indica uma mensagem vazia.

O ID da mensagem: usada para deduplicação de mensagens e confirmação ou reset de mensagens. É gerado por quem envia a mensagem, no caso de uma mensagem confirmável ou reset, a resposta deve possuir o ID da mensagem enviada. A implementação da geração dos IDs está aberta, depende da aplicação que o CoAP será usado, porém é recomendado que o valor inicial seja randômico.

## 2.4.2 Transmissão de Mensagens

A transmissão de mensagens é controlada basicamente pelos parâmetros: ACK TIMEOUT, ACK RANDOM FACTOR, MAX RETRANSMIT, NSTART, Leisure e PROBING RATE.

Estes parâmetros são respectivamente: o tempo que uma mensagem confirmável aguarda o ACK; fator de randomicidade para gerar os ACK TIMEOUTs subsequentes; contador para o número máximo de tentativas de retransmissão; número limite de interações simultâneas mantidas por um servidor.

A Leisure é o tempo que o servidor aguarda para responder uma requisição multicast, é calculada:  $Leisure = S * G / R$ . Aonde S é o tamanho estimado da reposta, G é uma estimativa do tamanho do grupo e R é a taxa de transmissão. PROBING RATE: é a taxa média para transmissão de dados.

Estes parâmetros definem a temporização do sistema. Os valores padrões são mostrados na Tabela 2.4.2.

A retransmissão é controlada por um timeout e um contador. Inicialmente o contador é zero e atribuído um valor randômico entre o ACK TIMEOUT e (ACK RANDOM FACTOR \* ACK TIMEOUT) para o timeout. Quando este timeout é atingido o contador é incrementado timeout duplicado.

Nome	Valor padrão
ACK timeout	2 segundos
ACK random factor	1.5
NStart	1
Default Leisure	5 segundos
Probing rate	1 Byte/segundo
Max retransmit	4

Tabela 1 – Valores padrão do CoAP.

Uma falha na transmissão ocorre quando atingir o número máximo de tentativas ou receber uma mensagem de RESET. Quando receber um ACK a transmissão da mensagem confirmável é completa. O servidor irá ignorar mensagens que chegam por multicast quando não puder responder nada de útil.

Na situação aonde possuir uma informação suficientemente nova pode responder na própria mensagem de confirmação (ACK). Essa técnica é chamada de "Piggy-backed" um mecanismo de transmissão para mensagens confirmadas, o cenário é ilustrado na Figura 2.4.2(SHELBY Z., HARTKE K., BORMANN C., 2013).

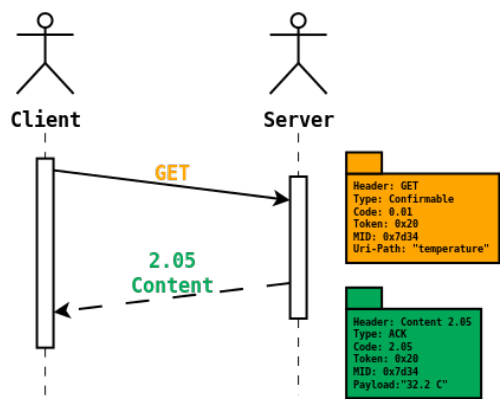


Figura 5 – Resposta na mensagem de confirmação, chamado de piggy-backed.

### 2.4.3 Requisição e Resposta

Uma requisição é inicializada ao preencher o campo código no cabeçalho do CoAP e gerar um token. Para finalizar o fluxo é necessário que a resposta chegue e o token seja o mesmo.

Fluxo esperado de requisição sem confirmação na figura 2.4.3.

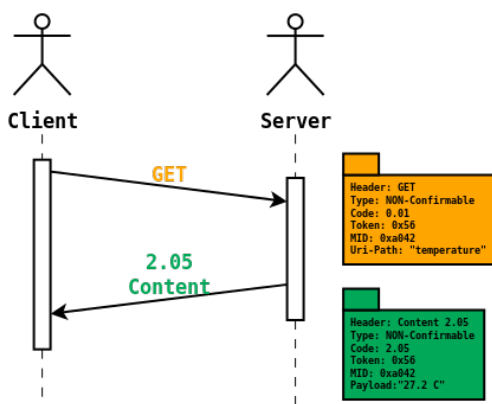


Figura 6 – Fluxo esperado de requisição e resposta sem confirmação

A especificação também prevê fluxo de requisição com confirmação, e resposta separada com confirmação. A figura 2.4.3 exemplifica:

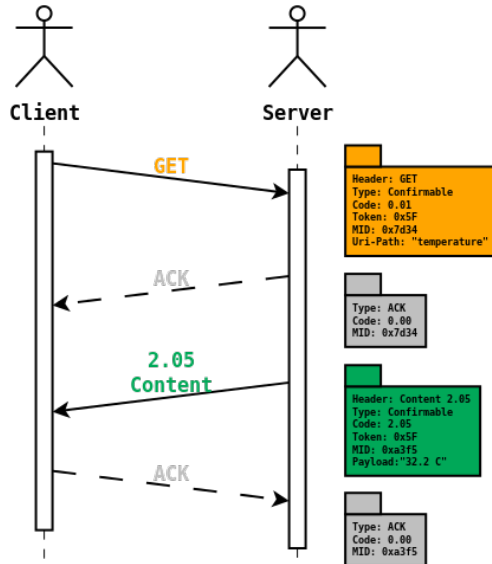


Figura 7 – Fluxo esperado de requisição e resposta com confirmação, com resposta separada

## 2.4.4 Opções

O CoAP possui diversas opções que são codificadas conforme a tabela abaixo. As opções são separadas entre eletivas, obrigatórias e ....

URI, path, query observer **falta coisa....**

## 2.4.5 Recursos

A descoberta de recursos é feita quando um servidor recebe uma requisição GET para o recurso `/well-know/core`. O servidor CoAP deve responder no formato CORE link Format (SHELBY Z., 2012). A descoberta de serviços no protocolo CoAP é feita através de socket Multicast. Os recursos são identificados por uma URI, e os métodos são implementados de forma similar ao HTTP.



## 2.5 EPOS

O EPOS é um sistema operacional multithread com suporte a preempção, desenvolvido em C++ que faz uso intenso de programação orientada a aspectos utilizando templates.

Foi projetado utilizando ADESD, Application Driven Embedded System Design, um método para projeto de sistemas embarcados orientados à aplicação. Esta metodologia guia o desenvolvimento paralelo de hardware e software além de manter portabilidade. O EPOS possui porte para as seguintes arquiteturas: MIPS, IA32, PowerPC, H8, Sparc, AVR e ARM. (EPOS, 1999)

A portabilidade é atingida utilizando entidades chamadas de Mediadores de Hardware que fornecem interfaces simples para acesso as funções específicas de arquitetura. Estas interfaces são utilizadas por entidades abstratas como alarmes e threads periódicas. Abaixo uma a figura 2.5 ilustrando a visão geral do EPOS.

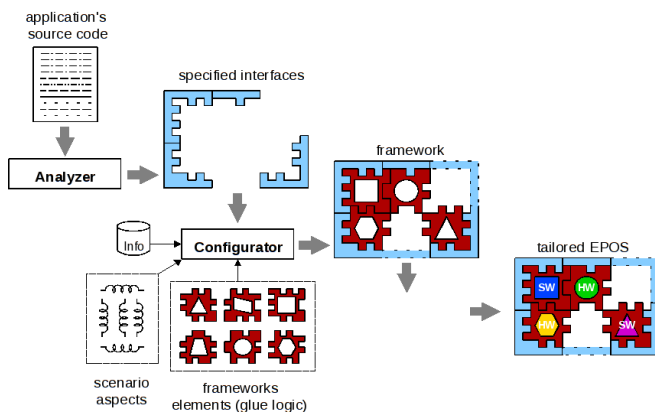


Figura 8 – Overview do EPOS.

O EPOS também possui uma interface de software/hardware que abstrai sensores de forma uniforme, definindo classes de dispositivos baseados numa finalidade. Possui abstrações para entidades temporais como relógio, alarme e cronômetro, biblioteca com estruturas de dados e sequenciadores. Permitindo o uso de ferramentas para geração automatizada de abstrações de sistemas (FRÖHLICH A., 1999).

A infraestrutura de comunicação do EPOS para redes de sensores sem-fio é implementada pelo protocolo C-MAC, Configurable MAC, que

provê suporte a comunicação de baixo nível (MAC - Medium Access Control).

**falar sobre UDP/IP do EPOS** A camada UDP/IP implementada no EPOS foi utilizada para a aplicação gateway.

## 2.6 TRABALHOS RELACIONADOS

### 2.6.1 OpenWSN

O projeto OpenWSN é um projeto de redes de sensores sem fio de código aberto que confia na comunidade para manter atualizado e encontrar erros. Possui uma implementação completa de código aberto, das pilhas de protocolos padrões pra Internet das Coisas. O OpenWSN também possui a primeira implementação aberta do padrão 802.15.4e, Time Synchronized Channel Hopping.

Por serem sincronizados os motes precisam acordar apenas para transmitir ou receber e periodicamente se comunicar para manter a rede sincronizada quando estiver inativo. Este overhead de temporização é pequeno, cerca de 0.02% de ciclo de trabalho do rádio (WATTEYNE, VILAJOSANA, KERKEZ, 2012).

A pilha de protocolos é totalmente implementada em C e pode ser compilada em qualquer toolchain que suporte uma plataforma alvo. Possui suporte para diversas plataformas de microcontroladores de 16 bits e para as arquiteturas mais novas de 32 bits dos Cortex-M. O projeto possui diversas ferramentas para depuração, simulação e ambiente necessário para integrar as aplicações a Internet (OpenWSN, 2014).

Resultados experimentais de uma rede de motes demonstram que os rádios operam num ciclo médio de trabalho de 0.1% e uma média de 0.68 microA em hardware comum. Este consumo baixo permite uma vasta gama de aplicações.

### 2.6.2 Contiki

O Contiki é um sistema operacional criado por Adam Dunkels em 2000, escrito em C, de código aberto para sistemas com restrição de recursos comunicarem numa rede. Foi desenvolvido para ser um sistema operacional para Internet das coisas. Possui uma camada de abstração RESTful para web services chamada Erbium, que implementa o protocolo CoAP.

Cada processo no Contiki possui bloco de controle, que contém informa-

ções de tempo de execução do processo e uma referência para uma protothread, na qual o código é armazenado na ROM.

Programas são escritos num modelo dirigido a eventos, geralmente são implementados como máquinas de estados explícitos, que com um grande número de estados o código começa a se tornar complexo, difícil de entender e depurar. Estas foram as principais motivações para que fosse desenvolvido um modelo de programação diferente, utilizado nos projetos desenvolvidos por Adam, a pilha uIP e do Contiki.(DUNKELS A., SCHMIDT O., 2005)

O conceito protothread foi desenvolvido por Adam Dunkels em 2005, uma combinação entre eventos e threads, possuem comportamentos de bloqueio e espera, que permite o intersequenciamento dos eventos, gerando um baixo overhead de memória por não necessitar de salvamento de contexto.

As limitações dessas técnicas são: variáveis automáticas não são preservadas Stack durante trocas de contexto entre as protothreads. Elas podem ser utilizadas durante uma protothread mas devem ser salvas antes da execução de do método WAITING. Além disso programas que utilizam protothreads não podem utilizar switch case, caso o fizer um erro de compilação é esperado.

Cada protothread consome 2 bytes de memória, que são utilizados para armazenar a continuidade local, uma referencia utilizada em um pulo condicional durante a execução da thread. É um método similar ao mecanismo de Duffy e Co-rotina em C. (Duffy T., 1988)

O Contiki propõe uma estratégia de ciclos de trabalho que consegue manter um nó comunicável em uma rede, porém com seus rádios desligados em aproximadamente 99% do tempo.(DUNKELS A., 2011)

### 2.6.3 LibCoap

LibCoap é uma biblioteca implementada em C do protocolo CoAP. Possui 292K de tamanho compilada estaticamente em sua versão 4.0.1. A licença da biblioteca é GPL (2 ou maior) ou licença BSD revisada.

Possui uma suíte de testes para regressão, utilizando o framework de testes CUnit (<http://cunit.sourceforge.net/>). A documentação pode ser encontrada em: <http://libcoap.sourceforge.net/>.

É uma biblioteca auto-condida, que possui parser do protocolo e funções básicas de rede. Depende de implementação de sockets tipo BSD e malloc. Possui implementação de Hash, String e URI utilizados para montar os pacotes CoAP.

Possui um módulo de rede, aonde é implementado as funções de envio/recebimento de requisições e respostas, com confirmação, sem confirma-

ção, mensagem de reset e erros.

É possível selecionar a camada de transporte é necessário selecionar utilizando flags de pré-processamento. O padrão é socket POSIX. A pilha uIP é selecionada com a flag `-DWITH_CONTIKI`, ou para selecionar a pilha lwIP `-DWITH_LWIP`.

## 2.6.4 CantCoap

É uma implementação em C++ desenvolvida por Ashley Mills em 2013. Utiliza uma licença similar a.

Esta biblioteca foca na simplicidade e oferece uma conjunto mínimo para montar pacotes CoAP. Também é possível montar os pacotes CoAP a partir de uma sequência de caracteres recebidos de uma placa de rede. Abaixo exemplos de uso da biblioteca retirados de <https://github.com/staropram/cantcoap>.

Abaixo um exemplo de uso para montar um pacote e enviar:

```
CoapPDU *pdu = new CoapPDU();
pdu->setType(CoapPDU::COAP_CONFIRMABLE);
pdu->setCode(CoapPDU::COAP_GET);
pdu->setToken((uint8_t*) "\3\2\1\0", 4);
pdu->setMessageID(0x0005);
pdu->setURI((char*) "test", 4);

/* send packet */
ret = send(sockfd, pdu->getPDUPointer(), pdu->getPDULength(), 0);
```

Quando receber a mensagem a forma de uso é mostrada abaixo:

```
// receive packet
ret = recvfrom(sockfd, &buffer, BUF_LEN, 0, &recvAddr, &recvAddrLen);
CoapPDU *recvPDU = new CoapPDU((uint8_t*)buffer, ret);
if(recvPDU->validate()) {
    recvPDU->getURI(uriBuffer, URI_BUF_LEN, &recvURILen);
    ...
}
```

Por ser uma biblioteca bem simplificada e não possuir dependências diretas com a implementação da camada de transporte e ser código livre, foi escolhida para a implementação teste do trabalho.

### 3 DESENVOLVIMENTO

Este trabalho implementa uma biblioteca que utiliza a camada UDP do EPOS para dar suporte ao protocolo CoAP, uma aplicação gateway GPRS / 802.14.5 utilizando o EPOS e um componente de hardware GPRS que será acoplado ao EposMoteII.

Durante o desenvolvimento foram realizados diversos estudos para escolher módulo GPRS adequado a tarefa e o trabalho necessário para acoplar o protocolo. Testes de validação dos sistemas de software e validação do módulo GPRS foram realizados. Foi realizado um levantamento de requisitos para porte de uma biblioteca CoAP para o EPOS (libCoap, libCantCoap, microCoap, entre outras) e EPOS MoteII. Utilizando testes para validar o funcionamento entre diferentes arquiteturas e compiladores. A execução dos testes foi feita no Qemu.

Implementação dos mecanismos de transmissão para mensagens confirmáveis e não-confirmáveis, requisição e resposta, e suporte a inserção de recursos, como sensores e atuadores como serviços CoAP.

A aplicação responsável pelo roteamento de mensagens para Internet utiliza a tecnologia GPRS, provida por um módulo GSM/GPRS da Quectel o M95.

#### 3.1 LEVANTAMENTO DE REQUISITOS

Infraestrutura flexível para a construção de aplicações embarcadas em modelo de webservices utilizando redes de sensores sem fio.

O usuário irá acessar a rede de sensores sem fio por uma aplicação html5, hospedada em na internet no endereço: <http://adefinir>. Nesta aplicação é possível enviar requisições para a rede de sensores de teste e listar os serviços oferecidos.

##### 3.1.1 Requisitos Funcionais

Coletar informação do ambiente através de sensores e transmití-las através da Internet. Fácil integração com a Internet mesmo em locais sem rede WIFI.

As principais funções deste gateway são receber os dados da rede de sensores e encaminhá-las para um servidor remoto que armazenará essas informações e exibirá de forma conveniente para o usuário final.

Será possível comunicar-se em tempo real com a rede de sensores, utilizando um módulo GPRS que irá repassar as requisições e respostas alimentadas pelo usuário.

As funções a da aplicação do gateway são:

1. Configuração, envio e recebimento de SMS;
2. Configuração contexto PDP, Configuração GPRS;
3. Configuração TCP/IP e manutenção de conexão TCP/IP.
4. Recebimento de requisições CoAP.

### 3.1.2 Requisitos Não Funcionais

Os webservices que vão executar nos motes devem herdar a característica de baixo consumo energético para que possam durar por anos, e serem extensíveis, podendo ser reutilizada em outras arquiteturas.

Além os serviços serão listados utilizando o padrão (SHELBY Z., 2012) disso os dados captados por sensores serão disponibilizados na forma de webservices CoAP, protocolo específico para este tipo de aplicação.

A padronização na comunicação visa facilitar a interconexão dos sistemas de diversas plataformas.

Características destes sistemas são eficiente em:

**Armazenamento:** deve ser suficientemente pequeno para ser utilizado em microcontroladores.

**Energia:** consumir pouca energia para longa durabilidade com bateria.

**Valor:** utilizar uma infraestrutura de hardware simples para realizar as tarefas.

## 3.2 ESPECIFICAÇÃO

### 3.2.1 Arquitetura

A aplicação é composta pelos nós webserver CoAP, um nó cliente CoAP que fará o roteamento para Internet utilizando um módulo GPRS. Os webserver informam a temperatura, através de respostas a requisições CoAP. A figura 3.2.1 ilustra a interconexão entre os nodos da rede.

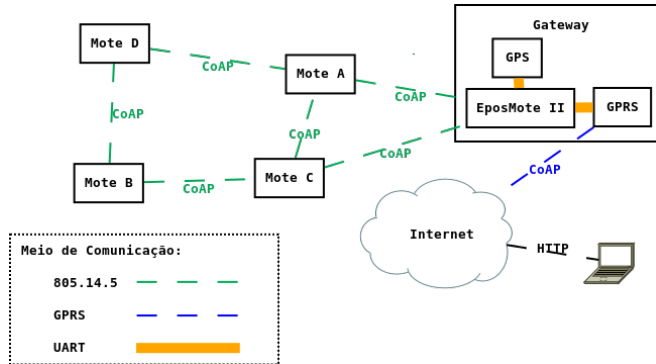


Figura 9 – Visão geral sobre comunicação do sistema.

### 3.2.2 Componentes

A aplicação do gateway é composta por: Mecanismos de temporização, camada UDP/IP, parser de pacote CoAP, conjunto de comandos AT, Estruturas de filas, Hash simples e Threads.

A implementação consiste num módulo que trata requisições, encapsula em pacotes e transmite por mecanismos de transmissão baseados em (SHELBY Z., HARTKE K., BORMANN C., 2013).

A biblioteca utilizada para montar o pacote CoAP foi: <https://github.com/staropram/cantcoap.git>. Na qual enviei algumas correções e testes para facilitar a verificação da execução correta dos algoritmos internos durante mudanças no código. As alterações podem ser visualizadas aqui: <https://github.com/staropram/cantcoap/commits?author=rafaeldelucena>.

Para o funcionamento desta biblioteca no EPOS, e para utilizar uma MTU limitada a 128 bytes utilizo um buffer com um valor máximo e armazeno os dados do pacote no buffer. Foi necessário alterar os tipos das variáveis para se adequarem ao EPOS.

O desenvolvimento de um mecanismo de retransmissão de mensagens não-confirmadas utilizando uma lista ordenada. Mecanismo de requisição e resposta, as requisições pendentes foram armazenadas num Hash com a chave sendo o token gerado pelo cliente.

O protocolo CoAP foi modelado conforme é mostrado na figura 3.2.2 abaixo:

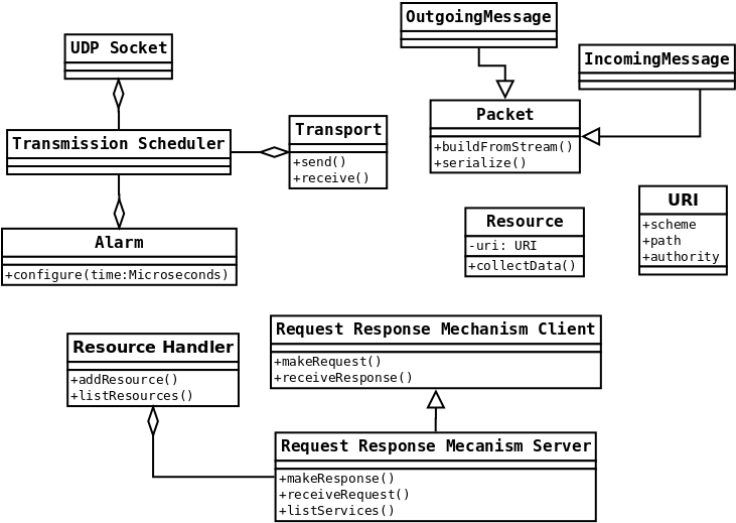


Figura 10 – Diagrama UML das entidades de software implementadas.

A figura 3.2.2 mostra o diagrama de caso de uso das principais funções desenvolvidas.



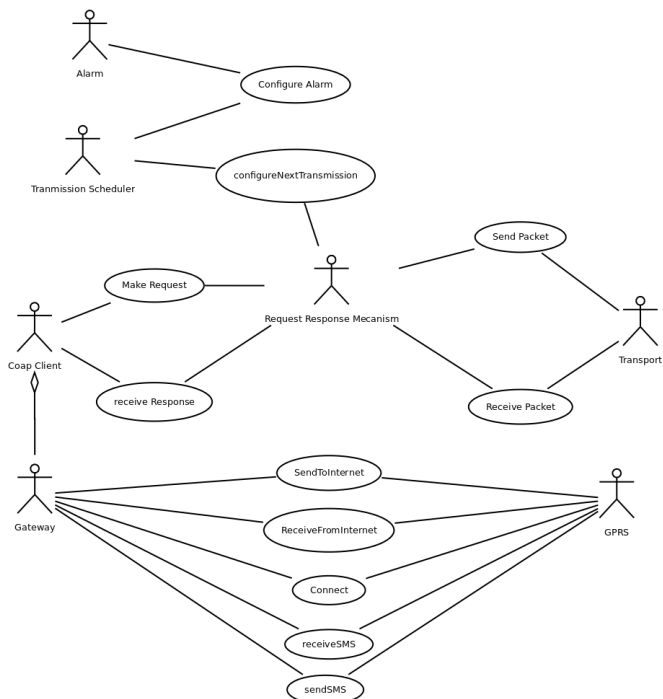


Figura 11 – Diagrama de casos de uso.

A aplicação cliente foi desenvolvida utilizando tecnologias HTML5. JavaScript, Json e HTML foram utilizados. A biblioteca CoAP foi: <https://github.com/mcollina/node-coap>

A aplicação desenvolvida no EPOS utiliza buffer para o recebimento de dados da rede 802.15.4 que será enviado para rede via GPRS. Duas threads, uma produtora que ficará escutando o rádio 802.15.4 e outra consumidora que será responsável em utilizar estes dados na rede de sensores e encaminhá-los pra Internet usando a extensão GPRS do EPOSmote II.

Para validar o comportamento utilizei alguns testes da própria biblioteca CoAP portados para o EPOS. Foi necessário implementar a função assert. Já que seria bem mais trabalhoso adicionar uma ferramenta de testes no sistema de build do EPOS.

### 3.3 TESTES

Foram realizados inúmeros testes durante o desenvolvimento para verificar e validar o correto comportamento dos componentes de software e hardware.

Para validar a implementação do protocolo CoAP foram efetuados os seguintes testes:

Testes de construção de pacotes válidos e inválidos utilizando como entrada sequência de caracteres.

Testes de interoperabilidade entre as implementações, utilizando cenários parecidos com o IOT Plugtest.

Fazendo uma requisição confirmáveis e não-confirmáveis do tipo: GET, POST, PUT, DELETE.

Recebendo respostas: válidas e inválidas.

Testes do Servidor: Recebendo e respondendo requisições: que possui recurso, que não possui, descoberta de recurso.

Os testes feitos foram: Enviar e recebimento de mensagens; criar socket TCP, enviar e receber mensagem via socket, fazer requisição HTTP, foi possível utilizando os comandos proprietários do modem.

## 4 AVALIAÇÃO

### 4.1 ANÁLISE FUNCIONAL

#### 4.1.1 Limitações Funcionais

##### em desenvolvimento

Atualmente a aplicação cliente não possui uma implementação segura do protocolo CoAP, que utiliza o DTLS. Para resolver este problema é necessário que o pacote seja criptografado.

### 4.2 ANÁLISE ESTRUTURAL

Fator muito importante para verificar a implementação de protocolo é a interoperabilidade entre diferentes implementações.

A ETSI em conjunto com a IPSO desenvolveram um conjunto de testes para validar o comportamento entre diversas implementações CoAP.

Este teste foi aplicado em 24, 25 de março de 2012 em Paris, conhecido como primeiro evento IOT CoAP plugtest. Os pontos principais objetivos do teste são:

- Especificação básica do CoAP.
- Transferência em bloco
- Observação de recursos CoAP.
- Formato CORE link.

O teste basicamente é executado entre diferentes dispositivos e implementações CoAP. O cenário inicial de testes é composto por:

- Cada equipamento deve estar configurado com um endereço unicast.
- A cache do cliente deve estar limpa.
- Utilização da opção ETag por padrão deve ser evitada a não ser que esteja explicitamente descrito no caso de teste.
- O uso de Tokens deve ser evitado a não ser que o caso de teste utilize, porém a implementação deve estar preparada para tratar o token.

- O uso de repostas por Piggybacked deve ser preferencial, a menos que a descrição do teste altere este padrão.

Os testes feitos para o protocolo CoAP são:

1. Cumprir transações GET, confirmável.
2. Cumprir transações POST, confirmável.
3. Cumprir transações PUT, confirmável.
4. Cumprir transações DELETE, confirmável.
5. Cumprir transações GET, sem confirmação.
6. Cumprir transações POST, sem confirmação.
7. Cumprir transações PUT, sem confirmação.
8. Cumprir transações DELETE, sem confirmação.
9. Cumprir transações GET com resposta separada.
10. Tratar requisição com opção de Token.
11. Tratar requisição sem opção de Token.
12. Tratar requisição contento várias opções URI-Path.
13. Tratar requisição contento várias opções URI-Query.
14. Interoperabilidade em contexto de perda de pacotes (CON mode, piggy-backed response)
15. Interoperabilidade em contexto de perda de pacotes (CON mode, delayed response)
16. Cumprir transações GET com resposta separada, sem confirmação.

É previsto também testes para validar o formato de dados CORE link Format. Abaixo os testes:

1. Descoberta de recursos utilizando a interface de acesso well-known.
2. Utilização de consulta para filtrar resultados.

Testes para validar a transferência de blocos:

- 1. Tratar a transferência de blocos para recursos grandes utilizando o método GET (negociação antecipada).
- 2. Tratar a transferência de blocos para recursos grandes utilizando o método GET (negociação atrasada).
- 3. Tratar a transferência de blocos para recursos grandes utilizando o método PUT.
- 4. Tratar a transferência de blocos para recursos grandes utilizando o método POST.

Testes para observação de recursos:

- 1. Tratar observação de recursos.
- 2. Parar a observação de recursos.
- 3. Detecção de deregistro do cliente (Max-Age).
- 4. Detecção de deregistro do servidor (client OFF).
- 5. Detecção de deregistro do servidor (RESET explícito).

4.3 ANÁLISE DE DESEMPENHO

em desenvolvimento: coletando os dados das outras implementações

O tempo de execução das principais funções foi medido. Abaixo a tabela 4.3, que faz um comparativo com as diversas arquiteturas e operações entre as principais soluções livres.

OS	CoAP	App	Size
Contiki	Erbium	REST client	
OpenWSN	OpenCoAP	CoAP client	
EPOS	CantCoap	CoapClient	

Tabela 2 – Comparação das implementações

Abaixo um comparativo entre as principais funções utilizadas nos sistema, utilizando um pacote padrão e um pacote com tamanho máximo de 128 bytes, o valor máximo de MTU utilizado.

CoAP	Function	Time in miliseconds
Erbium	coap_parser	
OpenCoap	parse	
CantCoap	CoapPDU	

Tabela 3 – Comparação das implementações em tempo de execução

Tabela que demonstra a vazão do sistema para mensagens, com tamanho definido anteriormente.

OS	CoAP	App	Número de Mensagens
Contiki	Erbium	ER-Rest-Server	
Linux	libCoap	Coap Server	
OpenOS	OpenCoap		

Tabela 4 – Comparação da vazão entre as implementações

## 5 CONSIDERAÇÕES FINAIS

### 5.0.1 Conclusão

O objetivo principal foi atingido, o desenvolvimento de um gateway simplificado para redes de sensores que utilizem o protocolo CoAP para disponibilizar serviços de sensoriamento e atuação. Além disso o consumo de energia e o espaços de armazenamento são muito baixos, respeitando os requisitos de um nó sensor.

em desenvolvimento...

### 5.0.2 Trabalhos Futuros

Entre os trabalhos futuros possíveis se destacam os seguintes: A implementação do protocolo coaps, utilizando DLTS para comunicação segura entre os nós.

Uma implementação de servidor CoAP completa para executar utilizando a plataforma do EPOSMote II.

Implementação de um gateway que utilize Software Defined Radio, assim apenas um transceiver será necessário para fazer a integração com a Internet.

Um gerador de código que utiliza como entrada uma linguagem de especificação dos possíveis recursos e como saída código ANSI C mínimo de um servidor web utiliza CoAP e seus respectivos recursos. Este gerador deve ser genérico suficiente para ser fácil a adaptação de diferentes pilhas UDP/IP, arquiteturas e tipos de sensores.

em desenvolvimento...





## REFERÊNCIAS

- Richardson, Leonard and Ruby, Sam. *RESTful web services*. 2008.
- Jennifer Yick, Biswanath Mukherjee, Dipak Ghosal. *Wireless Sensor Network survey*. 2008.
- Antônio Augusto Fröhlich, Wolfgang Schröer-Preikschat. *Embedded Parallel Operating System*. 1999.
- Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. 2000.  
<<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>>.
- Z. Shelby, K. Hartke, C. Bormann *Constrained Application Protocol (CoAP)*. 2013. <<http://www.ietf.org/id/draft-ietf-core-coap-18.txt>>.
- Z. Shelby . *Constrained RESTful Environments (CoRE) Link Format*. 2012.  
<<http://tools.ietf.org/rfc/rfc6690.txt>>.
- Koojana Kuladinithi, Olaf Bergmann, Thomas Pötsch, Markus Becker, Carmelita Görg *Implementation of CoAP and its Application in Transport Logistics*. 2011.  
<<http://hinrg.cs.jhu.edu/joomla/images/stories/coap-ipsn.pdf>>
- Mike P. Papazoglou. *Service-Oriented Computing: Concepts, Characteristics and Directions*. Web Information Systems Engineering - WISE , pp. 3-12, 2003.
- Franck L. Lewis. *Wireless Sensor Networks*. Journal Smat environments: technologies, protocols, and applications, pp. 11-46, 2004.
- Alexandre Massayuki Okazaki, Antônio Augusto Fröhlich. *ADHOP: an Energy Aware Routing Algorithm for Mobile Wireless Sensor Networks*. 2012.
- Booth, David and Haas, Hugo and McCabe, Francis and Newcomer, Eric and Champion, Michael and Ferris, Chris and Orchard, David *Web Services Architecture*. <<http://www.w3.org/TR/ws-arch>>. 2004.
- Watteyne, Thomas and Vilajosana, Xavier and Kerkez, Branko and Chraim, Fabien and Weekly, Kevin and Wang, Qin and Glaser, Steven and Pister, Kris. *OpenWSN: a standards-based low-power wireless development environment*. 2012.

Dargie, Waltenegus and Poellabauer, Christian *Fundamentals of wireless sensor networks: theory and practice*. 2010. John Wiley & Sons.

Bult, K. and Burstein, A. and Chang, D. and Dong, M. and Fielding, M. and Kruglick, E. and Ho, J. and Lin, F. and Lin, T.-H. and Kaiser, W.J. and Marcy, H. and Mukai, R. and Nelson, P. and Newburg, F. L. and Pister, K.S.J. and Pottie, G. and Sanchez, H. and Sohrabi, K. and Stafsudd, O.M. and Tan, K. B. and Yung, G. and Xue, S. and Yao, J. *Low power systems for wireless microsensors*. 1996

Adam Dunkels. *The ContikiMAC Radio Duty Cycling Protocol*. 2011

Adam Dunkels and Oliver Schmidt. *Protothreads - Lightweight, Stackless Threads in C*. 2005

Walter Colitti and Kris Steenhaut and Niccolò De Caro. *De Integrating Wireless Sensor Networks with the Web*. 2011

Cheng, Peng and Chuah, Chen-Nee and Liu, Xin. *Energy-aware node placement in wireless sensor networks*. 2004

<<http://sistemas.anatel.gov.br/stel>>. 2014

<<https://openwsn.atlassian.net/wiki>>

Tom Duffy <<http://www.lysator.liu.se/c/duffs-device.html>>. 1988.

<<http://epos.lisha.ufsc.br/HomePage>>. 1999.