UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Rafael de Lucena Valle

IMPLEMENTAÇÃO DO PROTOCOLO COAP 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

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CoAP	Constrained Aplication Protocol
EPOS	Embedded Parallel Operating System
HTTP	Hipertext Transfer protocol
IETF	Internet Engineering Task Force
M2M	Machine-to-Machine
REST	Representatational 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
RTT	Round-Trip Time

SUMÁRIO

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. Desenvolver a aplicação gateway GPRS/802.15.4.
3. Desenvolver uma WEP API para coleta da informação dos nós sensores.
4. 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 fotovoltáica. 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 mantem 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 et al., 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 comunicacam 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 realizads 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, industria, 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, 2004).

Este capítulo apresenta uma visão geral sobre redes de sensores sem fio, arquitetura orientada a serviços e os protocolos de aplicaçao 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 autonômo de operação e gerência de energia, auto-monitoramento, eficiência energética para longo tempo de operação e apta a incorporar diversos sensores (BULT et al., 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.

Desta forma permite disseminação da informação para processamento remoto, visualização, análise e armazenamento. A figura 2.1 dá uma visão geral sobre a comunicação destes nós.

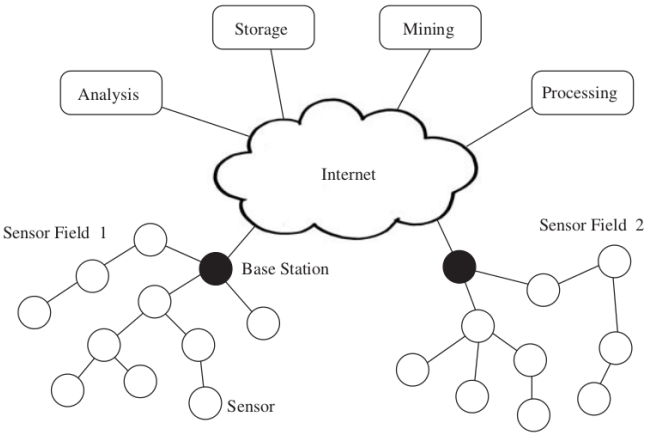


Figura 1 – Visão geral de uma RSSF (DARGIE; POELLABAUER, 2010).

Um nó pertencente a esta rede geralmente é um dispositivo especificamente desenvolvido para um próposito, 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 fi'sicos. Estas redes são chamadas de redes de sensores e atuadores sem fio, ou WSAN.

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 BISWANATH MUKHERJEE, 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 nó 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; STEENHAUT; CARO, 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 servicços oferecidos, suporte técnico e de negócio.

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

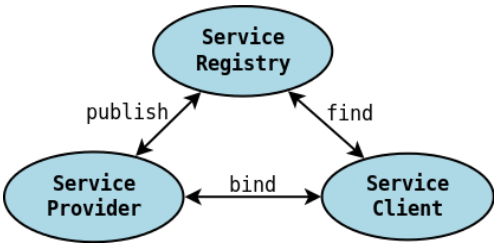


Figura 2 – Arquitetura Orientada a Serviços

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 (BOOTH et al., 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 restrição importante imposta pelo REST é o comportamento stateless do servidor, que não deve armazenar nenhuma informação dos clientes, requisições devem conter todas as informações necessárias para que sejam executadas.

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 .

Um aspecto importante é uma interface uniforme aonde recursos são gerenciados pela Web API e possuem as seguintes características: identificação de recursos, manipulação de recursos através de representações, mensagens auto descritivas e Hypermídia como mecanismo de estado da aplicação (FIELDING, 2000).

Um serviço RESTful sempre expõe a URI para cada pedaço de dado que o cliente quer operar sobre. 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; RUBY, 2008).

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.

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.

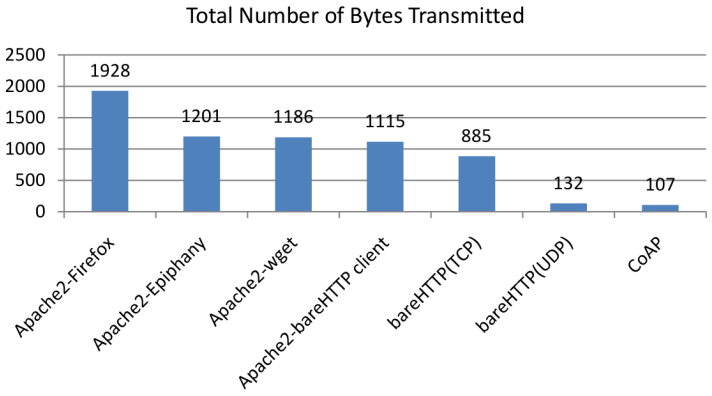


Figura 3 – Diferença de bytes transmitidos (KULADINITHI et al., 2011).

O teste consistiu em dois laptops conectados via rede GPRS, um cliente e outro servidor, com RTT configurado à 800ms, similar a uma RSSF multi-saltos. Ambos laptops configurados para tratar os protocolos HTTP e CoAP.

2.4 COAP

O CoAP é um protocolo de aplicação com intuito de ser utilizado em dispositivos eletrônicos simplificados, permitindo comunicação interativa com a Internet e outros dispositivos. A especificação do protocolo é a RFC 7252. 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 meca-

nismo 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.

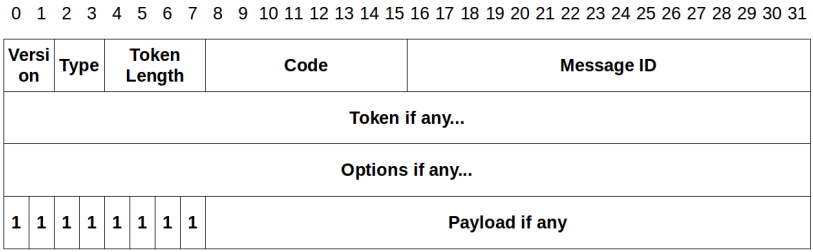


Figura 4 – Formato do pacote CoAP em bits (SHELBY KLAUS HARTKE, 2013).

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 implemetação 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 mensagems 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.

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.

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

Uma falha na transmissão ocorre quando atingir o número máximo de tentavivas 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 con-

firmadas, o cenário é ilustrado na Figura 2.4.2(SHELBY KLAUS HARTKE, 2013).

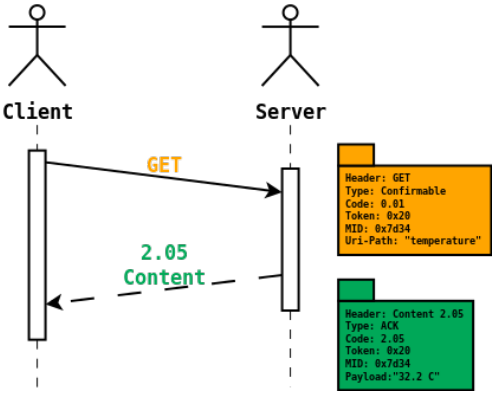


Figura 5 – Resposta na mensagem de confirmação, chamado de piggy-backed.

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.

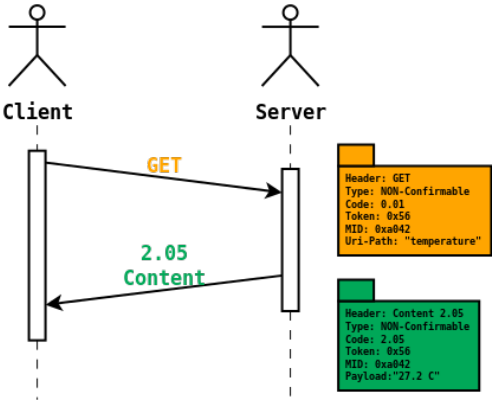


Figura 6 – Fluxo esperado de requisição e resposta sem confirmação

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

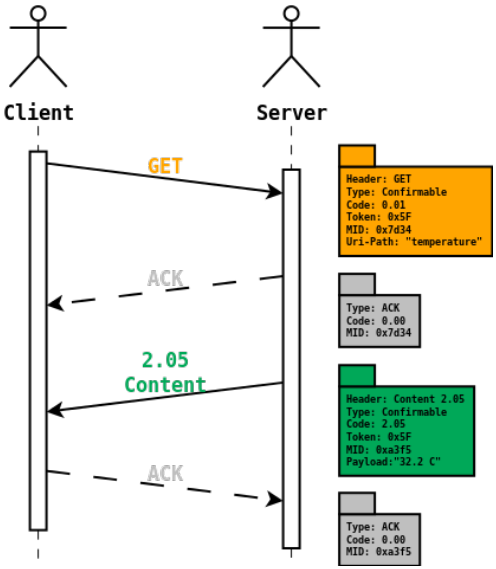


Figura 7 – Fluxo esperado de requisição e resposta com confirmação, com resposta separada

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, 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.

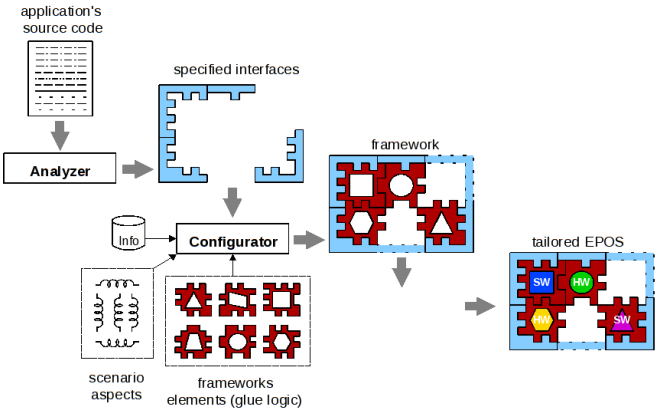


Figura 8 – Overview do EPOS.

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 (FRÃ¶HLICH, 1999).

A infraestrutura de comunicacação do EPOS para redes de sensores sem-fio é implementada pelo protocolo C-MAC, Configurable MAC, que provê suporte a comunicacação de baixo nível (MAC - Medium Access Control).

Possui uma pilha de protocolos basedo na arquitetura OSI e outra abordagem Cross-layer. A implementação IPv4 utiliza um controle de fluxo aonde os nós sinalizam a disponibilidade de buffer para mensagem única em tempos ajustando o tempo entre a troca de mensagens (FROHLICH et al., 2013).

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 possuí 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 et al., 2012).

A pilha de protocolos é totalmente implementada em C e pode ser compilada em qualquer toolchain que suporte uma plataforma alvo. Possui porte 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 (BERKELEY, 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 μA 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 informações de tempo de execução do processo e uma referência para uma protothread, na qual o cógido é armazenado na ROM.

Geralmente programas neste tipo de aplicação são escritos utilizando um modelo dirigido a eventos, que consome pouca memória. Estes programas 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 Dunkels, a pilha uIP e do Contiki (DUNKELS; SCHMIDT, 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. Foi desenvolvida para diminuir a complexidade de código.

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, 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, 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 licensa da biblioteca é GPL (2 ou maior) ou licensa 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 confirmação, mensagem de reset e erros.

É possível selecionar a camada de transporte é necessário selecionar utilizando flags de preprocessamento. 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. Esta biblioteca foca na simplicidade e oferece um conjuto 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 como base 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 à tarefa e o trabalho necessário para acoplar o protoloco. 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 EPOSMoteII. Utilizando testes para validar o funcionamento entre diferentes arquiteturas e compiladores. A execução dos testes foi feita no Qemu.

Foram implementados os mecanismos de transmissão para mensagens confirmáveis e não-confimáveis, 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

É um requisito geral deste trabalho definir uma 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 na Internet aonde é possível enviar requisições para a rede de sensores de teste e listar os serviços oferecidos e exibir as repostas.

3.1.1 Requisitos Funcionais

São requisitos funcionais da solução coletar informação do ambiente através de sensores e transmití-las através da Internet e 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 da da aplicação do gateway são:

1. Configuração, envio e recebimento de SMS.
2. Configuração de contexto PDP, Configuração GPRS.
3. Configuração UDP/IP no enlace GPRS.
4. Recebimento de requisições CoAP.
5. Envio de requisições CoAP por UDP/IP e SMS (POETSCH et al., 2013).

As funções da aplicação cliente serão:

1. Listar recursos CoAP web de uma RSFF.
2. Enviar Requisições CoAP para a RSFF.
3. Receber Repostas CoAP e exibir HTML.

3.1.2 Requisitos Não Funcionais

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

Além disso os serviços serão listados utilizando o padrão (SHELBY, 2012) disponibilizados na forma de webservices CoAP. A padronização na comunicação visa facilitar a interconexão dos sistemas de diversas plataformas.

Características herdadas dos nós simplificados são:

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

Energia: consumir pouca energia para longa durabilidade.

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

3.2 ESPECIFICAÇÃO

3.2.1 Arquitetura

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

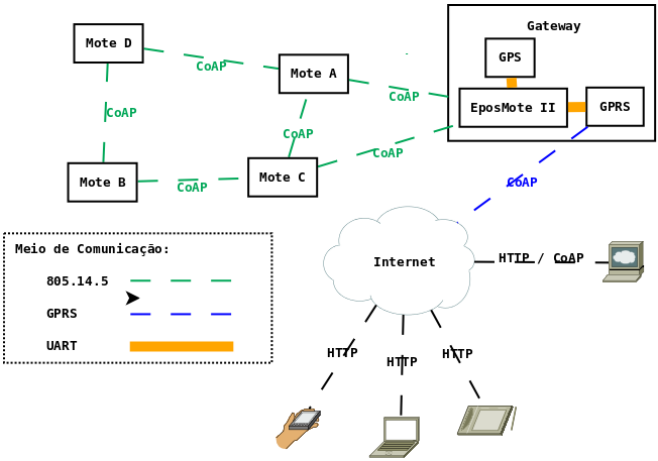


Figura 9 – Visão geral sobre comunicação do sistema.

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 mecanimos de transmissão baseados em (SHELBY KLAUS HARTKE, 2013).

A biblioteca utilizada para montar o pacote CoAP foi a biblioteca CantCoap, na foram realizados algumas correções e testes para facilitar a verificação da execução correta dos algoritmos internos durante o desenvolvimento da aplicação. As alterações resultaram em contribuição para o referido projeto, aceita pelo mantenedor.

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

No mecanismo de requisiçõ e resposta, as requisições pendentes foram armezenadas 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:

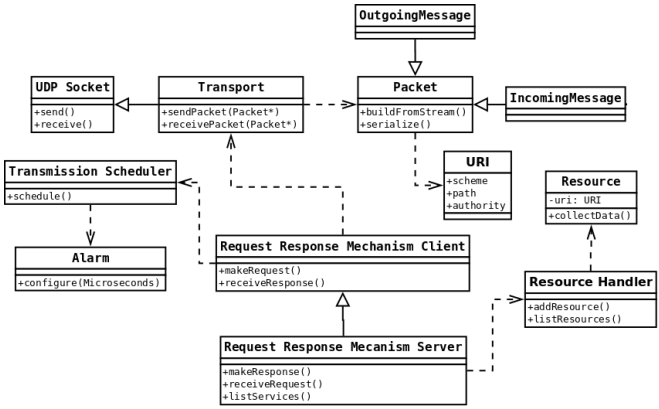


Figura 10 – Diagrama UML das entidades de software implementadas.

No desenvolvimento do mecanismo de retransmissão de mensagens não-confirmadas foi utilizado uma lista ordenada e alarme configurado para o próximo reenvio. A figura 3.2.2 ilustra a interação entre os módulos de software, incluindo os tipos dos dados e a forma de execução das funcços.

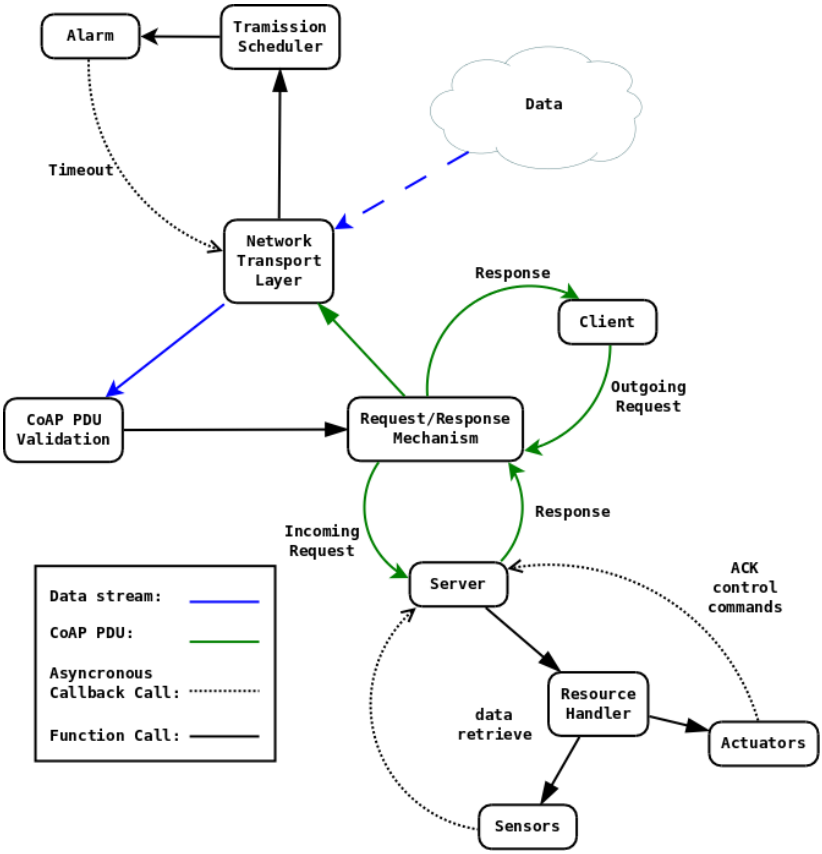


Figura 11 – Abstração dos módulos de software da aplicação gateway.

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

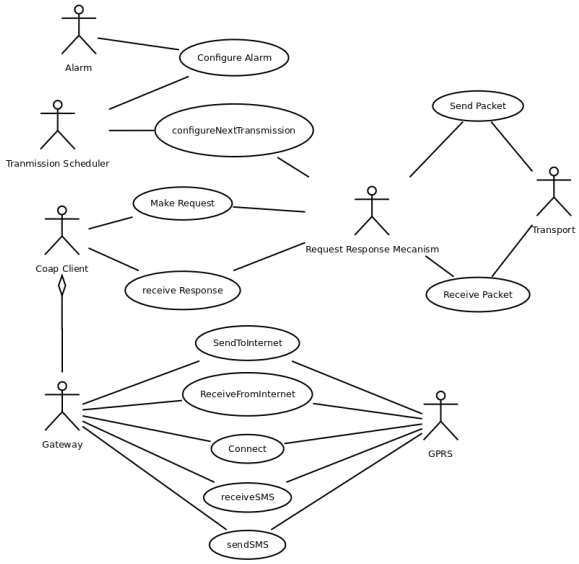


Figura 12 – Diagrama de casos de uso.

A aplicação cliente foi desenvolvida em um Linux utilizando tecnologias HTML5, JavaScript, JSON e HTML foram utilizados. O servidor HTTP utilizado foi HttpServer do NodeJS, que recebe requisições HTTP dos clientes para a rede de sensores sem fio utilizando a biblioteca 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. Foram utilizadas duas threads, uma produtora que ficará escutando o rádio 802.15.4 e outra consumidora que será responsável em utlizar estes dados na rede de sensores e encaminhá-los pra Internet usando a extensão GPRS do EPOSmote II. Para validar o comportamento foram utilizados testes da própria biblioteca CoAP portados para o EPOS.

3.3 IMPLEMENTAÇÃO

A modelagem da solução visa uma implementação modular, para facilitar a verificação do correto funcionamento e também para ser de fácil adaptaçã a diversas plataformas. O sistema proposto é composto pela aplicação gateway, por um servidor coap externo e um servidor http externo, este apenas para facilitar o acesso de dispositivos sem suporte ao protocolo CoAP.

Para funcionamento é necessário executar a aplicação coap-gateway-server, disponível em:
https://github.com/rafaeldelucena/coap-gateway-server.

Também é necessário iniciar o gateway e os dispositivos que irão se comunicar via 802.15.4 por protocolo CoAP.

No gateway o sistema desenvolvido possui como entradas mensagens sequências de caractdo transceiver 802.15.4 e do módulo GPRS, que são validados em pacotes CoAP.

Ao estabelecer o túnnel e obter um endereço IP e uma porta da rede móvel é enviada uma requisição CoAP para o recurso
coap://hostname/gateway/config/ip e para
coap://hostname/gateway/config/port.

Ao receber uma mensagem pelo módulo GPRS e validado o pacote CoAP é repassado para a rede de sensores conectada. No caso de receber uma mensagem por 802.15.4, esta mensagem é validada e repassada para o hostname anteriormente definido no sistema.

Com isto o sistema está apto para receber requisições externas e enviar para a rede de sensores, quando a resposta retornar da RSSF, o pacote também é validado e depois enviado para o endereço previamente configurado.

Para testar é necessário executar também o HTTP server, que receberá as requisições de clientes HTTP e irá repassar para qualquer servidor CoAP através de uma rota que o usuário informou.

Este servidor HTTP trata requisições para
http://http-server-hostname/coap-server-hostname/resources, que retorna os recursos do servidor coap em formato JSON e para
http://http-server-hostname/coap-server-hostname/resource, aonde resource é a rota disponível de um recurso CoAP.

3.4 TESTES

Durante o desenvolvimento foram realizados inúmeros testes para verificar e validar o correto comportamento dos componentes de software e hardware. A implementação do protocolo CoAP foi testada da seguinte maneira:

Testes unitários: Testes de construção de pacotes válidos e inválidos utilizando como entrada sequência de caracteres e tratamento de respostas válidas e inválidas.

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

Os procedimentos para testar o módulo GPRS foram: Enviar e recebimento de mensagens; criar socket UDP e TCP, enviar e receber mensagem via socket, fazer requisição e receber respostas HTTP.

4 AVALIAÇÃO

4.1 ANÁLISE FUNCIONAL

A aplicação gateway é responsável por fazer uma ponte entre a rede de sensores e a Internet. É possível descobrir serviços e recursos, iniciar requisições, fazer descoberta de recursos, tudo isto é transparente para os pontos comunicantes.

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

4.2 ANÁLISE ESTRUTURAL

Um Fator muito importante para verificar a implemencação de protocolo é a interoperabilidade entre diferentes implementações. É essencial que a implementação cliente trate os mecanismos de requisição e resposta tanto para mensagens confirmadas e não confirmadas. Uma maneira de validar a interoperabilidade da implementação do protocolo é usar um padrão de testes entre diferentes sistemas.

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. Para validar a interoperabilidade os teste são: especificação básica do CoAP, transferência em bloco, observação de recursos CoAP, formato CORE link. Os teste são executados entre diferentes dispositivos e implementações CoAP. O cenário inicial de testes possui os seguintes requisitos:

- 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.

A tabela abaixo 4.2 mostra o resultado dos testes, os símbolos "✗"e "✓"para a solução desenvolvida neste trabalho.

CENÁRIO	RESULTADO
Testes para especificação básica CoAP	
Tratar GET, confirmável.	✓
Tratar POST, confirmável.	✓
Tratar PUT, confirmável.	✓
Tratar DELETE, confirmável.	✓
Tratar GET, sem confirmação.	✓
Tratar POST, sem confirmação.	✓
Tratar PUT, sem confirmação.	✓
Tratar DELETE, sem confirmação.	✓
Tratar GET com resposta separada.	✓
Tratar requisição com Token.	✓
Tratar requisição sem Token.	✓
Tratar requisição opções URI-Path.	✓
Tratar requisição opções URI-Query.	✓
Interoperablidade em contexto de perda (CON mode, piggybacked response)	✓
Interoperablidade em contexto de perda (CON mode, delayed response)	✓
Tratar GET com resposta separada, sem confirmação.	✓
Testes para validar o formato de dados CORE link Format	
Descoberta de recursos well-known.	✗
Utilização de consulta para filtrar resultados.	✗
Testes para validar a transferência de blocos	
Transferência de blocos grandes utilizando GET (negociação antecipada).	✗
Transferência de blocos grandes utilizando GET (negociação atrasada).	✗
Transferência de blocos grandes utilizando o PUT.	✗
Transferência de blocos grandes utilizando o POST.	✗
Testes para observação de recursos	
Tratar observação de recursos.	✗
Parar a observação de recursos.	✗
Detecção de deregistro do cliente (Max-Age).	✗
Detecção de deregistro do servidor (client OFF).	✗
Detecção de deregistro do servidor (RESET explícito).	✗

Tabela 2 – Resultados dos testes de interoperabilidade IOT Plugtest.

O teste consistiu em executar localmente os diferentes servidores e o cliente desenvolvido. A implementação irá utilizar apenas pacotes com no máximo de 128 bytes, portanto o suporte a transferência de grandes blocos fragmentados não será necessário no momento.

O gateway utiliza apenas uma implementação cliente CoAP, não sendo necessário possuir a implementação de descoberta de recursos e observação, pois estas funcionalidade estão implementada nos nodos sensores.

O gateway irá repassar a requisição para os webservers e a cada resposta encaminhará para a Internet, simplificando bastante a implementação. A idéia é o gateway ser totalmente transparente tanto para clientes e servidores. Para a aplicação proposta o conjunto de funcionalidades mínimas passou no teste de interoperabilidade.

4.3 ANÁLISE DE DESEMPENHO

A fim de mensurar algumas informações importantes no contexto da aplicação foi utilizada a versão 4.4.5 para o GCC x86 e o GCC 4.3.2 (Sourcery G++ Lite 2008q3-66) para ARM no Contiki encontrada (ALVIRA, 2014). A aplicação no EPOS utiliza a versão 4.4 do compilador GCC disponível em (EPOS, 1999).

As flags de compilação utilizadas foram as padrões de cada sistema de construção dos projetos para a arquitetura x86 e para para ARM7 do MC1224V utilizado no EPOSMoteII e no Econotag, plataforma presente no Contiki e utilizado para comparativos. Abaixo a tabela 4.3 mostra um comparativo do espaço utilizado das diferentes implementações CoAP.

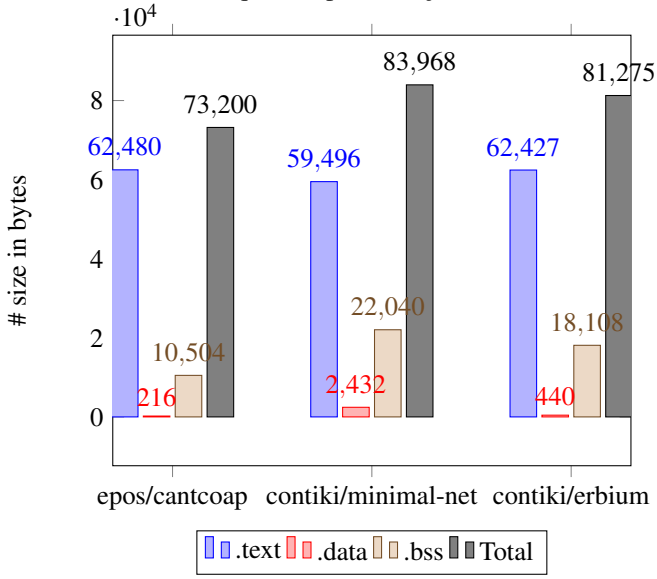
ISA	OS	CoAP	.text	.data	.bss	Total
x86 32	Contiki	Erbium	97995	1792	16864	116651
x86 32	EPOS	Cantcoap	52348	117	9652	62117
x86 32	GNU Linux	libcoap	45515	680	1952	48147
ARM7	Contiki	Erbium	62727	440	18108	81275
ARM7	Contiki	Minimal Net	59496	2432	22040	83968
ARM7	EPOS	Cantcoap	62480	216	10504	73200

Tabela 3 – Comparação das implementações em consumo de memória em bytes

Nesta tabela é possível verificar que para a arquitetura x86 a imple-

mentação do trabalho é 30% maior que a libcoap, porém em torno de 25% menor que as implementações do Contiki. Um dos fatores da diferença com a aplicação da libcoap é a utilização de bibliotecas dinâmicas (DEVELOPERS, 2014). Apesar de não ser a mais compacta é suficientemente pequena para ser utilizada na aplicação.

No gráfico 4.3 observa-se que a solução do EPOS é no total de memória utilizada é cerca de 12,82% menor que a implementação do Contiki Minimal Net e 9,93% menor que a implementação utilizando o Erbium.



Abaixo 4.1 mais informações do arquivo gerado após o sistema de construção. Para comparação os símbolos de depuração foram removidos do arquivo ELF. Para comparação

Listing 4.1 – Verificando o tipo de arquivo gerado no Linux utilizando a ferramenta file.

```
file epos/img/coap_client
img/coap_client: ELF 32-bit LSB executable, Intel 80386, version 1
      (SYSV), statically linked, stripped

file libcoap/examples/coap-client
examples/coap-client: ELF 32-bit LSB executable, Intel 80386,
     version 1 (SYSV), dynamically linked (uses shared libs), for
     GNU/Linux 2.6.18, stripped

file contiki/examples/er-rest-example/er-example-client.elf
er-example-client.elf: ELF 32-bit LSB executable, ARM, version 1 (
     SYSV), statically linked, stripped
```

```
file contiki/examples/rest-example/coap-client-example.elf
coap-client-example.elf: ELF 32-bit LSB executable, ARM, version 1
      (SYSV), statically linked, stripped

file epos/img/coap_client.img
coap_client.img: ELF 32-bit LSB  executable, ARM, version 1,
     statically linked, stripped
```

A grande quantidade de dados inicializados na implementação deve-se ao fato do uso de buffers de 128 bytes para cada pacote, afim de simplificar a gerência de memória do software desenvolvido.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresentou alguns conceitos sobre redes de sensores e atuadores que utilizam um protocolo de aplicação específico, para disponibilizar recursos diversos e facilitar a interoperabilidade de diversos sistemas.

As principal contribuições deste trabalho são a avaliação de diferentes implementações em diversas plataformas e a implementação de uma infraestrutura para o desenvolvimento de aplicações que utilizem redes de sensores sem fio.

O objetivo principal foi atigindo, o desenvolvimento de um gateway simplificado para redes de sensores que repassa as mensagens recebidas para um servidor CoAP na Internet. É possui acessar o sistema utilizando um cliente HTTP, facilitando a disponibilização de serviços de sensoriamente e atuação.

O tamanho do código respeitou o espaço de armazenamento restrito, comum desse tipo de aplicação e foi possível criar uma aplicação que possa ser utilizado em uma vasta gama de dispositivos. Entres os trabalhos futuros possíveis se destacam os seguintes:

A implementação da versão segura do protocolo CoAP, utilizando DLTS para comunicação segura entre os nós.

Uma implementação de servidor CoAP completa para executar utilizando a plataforma do EPOSMote II. Para isto seria necessário desenvolver um módulo genérico o suficiente para adicionar diversos recursos como sensores e atuadores, com suporte a diferentes tipos de funções, dados e ações como coletar dado de um ADC ou enviar comandos a um servo.

Uma aplicação que poderia reduzir ainda mais o custo para integrar as redes a Internet, é a implementação de um gateway que utilize Software Defined Transceiver, assim é possível integrar uma gama enorme de dispositivos e utilizar apenas um módulo transceiver para enviar os dados da RSSF para 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 CoAP/UDP/IP, arquiteturas e tipos de sensores.

REFERÊNCIAS

ALVIRA, M. **Library code for the Freescale MC13224v ARM7 SoC with 802.15.4 radio**. 2014. Disponível em: <https://github.com/malvira/libmc1322x>.

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

BERKELEY, U. **OpenWSN WIKI**. 2014. Disponível em: <https://openwsn.atlassian.net/wiki>.

BOOTH, D. et al. **Web Services Architecture**. [S.l.], fev. 2004. Disponível em: <http://www.w3.org/TR/ws-arch>.

BULT, K. et al. Low power systems for wireless microsensors. In: **Low Power Electronics and Design, 1996., International Symposium on**. [S.l.: s.n.], 1996. p. 17–21.

COLITTI, W.; STEENHAUT, K.; CARO, N. D. De integrating wireless sensor networks with the web. In: **In Proceedings of Workshop on Extending the Internet to Low power and Lossy Networks**. [S.l.: s.n.], 2011.

DARGIE, W.; POELLABAUER, C. **Fundamentals of wireless sensor networks: theory and practice**. [S.l.]: John Wiley & Sons, 2010.

DEVELOPERS, I. **Managing code size**. 2014. Disponível em: <http://publib.boulder.ibm.com/infocenter/comphelp/v101v121/index.jsp?topic=/com.ibm.xlf121.aix.doc/proguide/managingcodesize.html>.

DUFFY, T. **Duffy Mechanism**. 1988. Disponível em: <http://www.lysator.liu.se/c/duffs-device.html>.

DUNKELS, A. **The ContikiMAC Radio Duty Cycling Protocol**. 2011.

DUNKELS, A.; SCHMIDT, O. **Protothreads - Lightweight, Stackless Threads in C**. 2005.

EPOS. **EPOS Project**. 1999. Disponível em: <http://epos.lisha.ufsc.br/HomePage>.

FIELDING, R. T. **Representational State Transfer (REST)**. Tese (Doutorado), 2000. Disponível em: <http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm>.

FROHLICH, A. et al. A cross-layer approach to trustfulness in the internet of things. In: **9th Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS), Paderborn, Germany**. [S.l.: s.n.], 2013. p. 884–889.

FRÃ¶HLICH, W. S.-P. A. A. Epos: an object-oriented operating system. 1999.

KULADINITHI, K. et al. Implementation of coap and its application in transport logistics. **Proc. IP+ SN, Chicago, IL, USA**, 2011.

LEWIS, F. L. Wireless sensor networks. **Smart environments: technologies, protocols, and applications**, New York: Wiley, p. 11–46, 2004.

PAPAZOGLOU, M. P. Service-Oriented Computing: Concepts, Characteristics and Directions. In: **Web Information Systems Engineering**. [S.l.: s.n.], 2003. p. 3–12.

POETSCH, T. et al. Transport of coap over sms. **Transport**, 2013.

RICHARDSON, L.; RUBY, S. **RESTful web services**. [S.l.]: O'Reilly, 2008.

SHELBY KLAUS HARTKE, C. B. Z. **Constrained Application Protocol (CoAP)**. IETF, 2013. Internet-Drafts. (Internet Draft, 18). Disponível em: <http://www.ietf.org/id/draft-ietf-core-coap-18.txt>.

SHELBY, Z. **Constrained RESTful Environments (CoRE) Link Format**. IETF, 2012. RFC 5280 (Proposed Standard). (Request for Comments, 6690). Disponível em: <http://tools.ietf.org/rfc/rfc6690.txt>.

WATTEYNE, T. et al. Openwsn: a standards-based low-power wireless development environment. **Transactions on Emerging Telecommunications Technologies**, John Wiley & Sons, Ltd, v. 23, n. 5, p. 480–493, 2012. ISSN 2161-3915. Disponível em: <http://dx.doi.org/10.1002/ett.2558>.

YICK BISWANATH MUKHERJEE, D. G. J. **Wireless sensor network survey**. [S.l.]: Elsevier, 2008.