PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

CENTRO DE CIÊNCIAS EXATAS, AMBIENTAIS E DE TECNOLOGIAS

RODOLFO FRANCISCO DE OLIVEIRA

PROPOSTA DE UM PROXY MANAGER PARA INTERNET DAS COISAS

CAMPINAS

2016

RODOLFO FRANCISCO DE OLIVEIRA

PROPOSTA DE UM PROXY MANAGER PARA INTERNET DAS COISAS

Dissertação apresentada ao Programa de Pós Graduação Stricto Sensu em Engenharia Elétrica do Centro de Ciências Exatas, Ambientais e de Tecnologias da Pontifícia Universidade Católica de Campinas como requisito para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Omar Carvalho Branquinho.

PUC-CAMPINAS

2016

Ficha Catalográfica
Elaborada pelo Sistema de Bibliotecas e
Informação - SBI - PUC-Campinas

t621.3851 Oliveira, Rodolfo Francisco de.
O48p Proposta de um Proxy Manager para Internet das coisas / Rodolfo Francisco de Oliveira. - Campinas: PUC-Campinas, 2016.
160p.

Orientador: Omar Carvalho Branquinho.
Dissertação (mestrado) – Pontifícia Universidade Católica de Campinas, Centro de Ciências Exatas, Ambientais e de Tecnologias, Pós-Graduação em Gestão de Redes de Telecomunicações.
Inclui anexo e bibliografia.

1. Redes de sensores sem fio. 2. Internet. 3. Redes de computação. 4. Programação (Computadores) - Gerência. I. Branquinho, Omar Carvalho. II. Pontifícia Universidade Católica de Campinas. Centro de Ciências Exatas, Ambientais e de Tecnologias. Pós-Graduação em Gestão de Redes de Telecomunicações. III. Título.

22.ed. CDD – t621.3851

RODOLFO FRANCISCO DE OLIVEIRA

PROPOSTA DE UM PROXY MANAGER PARA INTERNET DAS COISAS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro de Ciências Exatas, Ambientais e de Tecnologias da Pontifícia Universidade Católica de Campinas como requisito parcial para obtenção do título de Mestre em Gestão de Redes de Telecomunicações.
Área de Concentração: Engenharia Elétrica Redes e Serviços. Orientador: Prof. Dr. Omar Carvalho Branquinho

Dissertação defendida e aprovada em 12 de dezembro de 2016 pela Comissão Examinadora constituída dos seguintes professores:

Prof. Dr. Omar Carvalho Branquinho
Orientador da Dissertação e Presidente da Comissão Examinadora
Pontifícia Universidade Católica de Campinas

Prof. Dr. Frank Herman Behrens
Pontifícia Universidade Católica de Campinas

Prof. Dr. Edson Luiz Ursini
Universidade Estadual de Campinas

Dedico este trabalho a toda minha família, que sempre esteve presente em todos os momentos da minha vida.

AGRADECIMENTOS

Primeiramente a Deus,
Por ter me dado forças até agora.

A minha família,
Em especial minha esposa Carla, pelo apoio incondicional e pelo ânimo nas horas mais difíceis.

Ao Prof. Dr. Omar Carvalho Branquinho,
Incentivador, guia e amigo nesta parte da Jornada.

Ao colega e amigo Pedro Chaves,
Por todo apoio, conselhos e companheirismo nesta estrada.

À Pontifícia Universidade Católica de Campinas,
Pela conceção da bolsa no Programa de Mestrado, sem o qual este trabalho não poderia ter sido concretizado.

Ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo,
Em especial aos colegas de trabalho, ao qual de uma forma ou de outra contribuiram para a elaboração deste trabalho.

A todos os bolsistas e funcionários relacionados ao Laboratório de Sinais de Rádio da PUC Campinas,
Por toda ajuda e apoio logístico.

Aos colegas e companheiros da turma,
Pelo apoio e toda solidariedade

“Onde não falta vontade existe sempre um caminho”
(John Ronald Tolkien)

RESUMO

OLIVEIRA, Rodolfo Francisco. *Proposta De Um Proxy Manager para Internet Das Coisas.* 2016. 160 pgs. Dissertação (Mestrado Profissional em Gestão de Redes de Telecomunicações) - Pontifícia Universidade Católica de Campinas, Centro de Ciências Exatas, Ambientais e de Tecnologias, Campinas, 2016.

Uma nova revolução está acontecendo: é a Internet das Coisas, uma nova onda de tecnologias onde é possível conectar à Internet objetos do cotidiano de forma inteligente. Dentre estas, uma das mais críticas são as Redes de Sensores sem Fio. Uma Rede de Sensores sem Fio consiste em vários elementos denominados Nós Sensores, os quais coletam grandezas do meio ambiente, tais como luz, temperatura e qualidade do ar, disponibilizando as mesmas para aplicações na Internet, normalmente através de um elemento intermediário denominado *gateway*. Para o bom funcionamento de uma Rede de Sensores sem Fio, é imprescindível que os elementos que a compõem sejam gerenciados de maneira eficiente. No entanto, há uma escassez de soluções de gerência disponíveis, principalmente quando estas são voltadas para ambientes previamente conhecidos, como prédios comerciais, indústrias, residências e cidades. Esta escassez de ferramentas de gerência é um dos principais fatores que dificultam a implantação maciça de Redes de Sensores sem Fio e, consequentemente, da Internet das Coisas. Sendo assim, a proposta deste trabalho é apresentar uma plataforma de gerenciamento centralizado para Redes de Sensores sem Fio, instaladas em ambientes conhecidos. Conceitualmente, esta plataforma se denomina *Proxy Manager*, e ficará hospedada no *gateway* da rede. A mesma é capaz de se comunicar com um Gerente de Rede e outras aplicações localizadas na Internet através de *web services*, recebendo parâmetros de gerência e disponibilizando dados sumarizados. As funcionalidades do *Proxy Manager* são divididas em dois focos de gerência: da infraestrutura da rede e da aplicação, as quais são subdivididas nas áreas de gerência de Configuração, Desempenho e Falhas. No trabalho em questão, as funcionalidades implementadas referentes à gerência da infraestrutura da rede foram: configuração do *Proxy Manager* e dos Nós Sensores; estabelecimento de rotas para os mesmos tendo como base a RSSI (*Received Signal Strength Indicator*) e o número de saltos para os Nós Sensores; monitoramento de desempenho através da PER (*Packet Error Rate*); e recuperação de falhas, através do estabelecimento de rotas alternativas e geração de alertas para o administrador da rede. Em relação ao gerenciamento da aplicação, a função implementada foi a de configuração dos sensores, localizados nos Nós Sensores, os quais efetivamente irão coletar os dados. Testes relacionados as funcionalidades implementadas, envolvendo o estabelecimento de rotas, monitoramento do desempenho, recuperação de falhas, coleta de dados pelos sensores e desempenho da plataforma foram realizados, os quais demonstraram que a solução em questão é funcional. Por fim, espera-se que o presente trabalho possa contribuir para o estudo e implementação de novas funcionalidades de gerência e interconexão de Redes de Sensores sem Fio, tanto nesta quanto em outras plataformas, através de trabalhos vindouros.

Termos de indexação: Internet das Coisas; Rede de Sensores sem Fio; *Proxy Manager*; Gerência de Redes.

ABSTRACT

OLIVEIRA, Rodolfo Francisco. *Proposal Of A Proxy Manager For Internet Of Things.* 2016. 160 pgs. Master Thesis (Professional Masters in Telecommunications Network Management) - Pontifícia Universidade Católica de Campinas, Centro de Ciências Exatas, Ambientais e de Tecnologias, Campinas, 2016.

A new revolution is happening: the Internet of Things, a new wave of technologies where it is possible to connect everyday objects to the Internet. Among these technologies, one of the most critical is the Wireless Sensor Networks. A Wireless Sensor Network consists of several elements called Sensor Nodes, which collect environmental information such as light, temperature and air quality, providing data to applications on the Internet, usually through an intermediate element called the Gateway. For a Wireless Sensor Network to work properly, it is essential that the elements that make it are managed efficiently. However, there is a shortage of available management solutions, especially when these Wireless Sensor Networks are installed in known environments, such as office buildings, factories, homes and cities. The lack of management tools is one of the main factors that hinder the deployment of Wireless Sensor Networks and consequently the Internet of Things. The purpose of this work is to provide a centralized management platform for Wireless Sensor Networks installed in known environments. Conceptually, this platform is called the Proxy Manager, hosted by the network gateway. The platform is able to communicate with a Network Manager and other applications located on the Internet through web services, receiving management parameters and providing summarized data. The Proxy Manager functionalities are split into two management categories: network infrastructure and application, which in turn are divided into Configuration, Performance and Fault management. In this work, the implemented functionalities relating to the network infrastructure management are: Proxy Manager and Sensor Node configuration; route finding through RSSI (Received Signal Strength Indicator) and number of hops to the Sensor Node; performance monitoring via PER (Packet Error Rate); fault recovery using alternative routes and the generation of alerts to the network administrator. Regarding the application management, the functionality implemented is the sensor configuration, since this will effectively collect the data. Tests of the features implemented were done, involving the establishment of routes, performance monitoring, disaster recovery, data collection and platform performance. The tests showed that the proposed solution is functional. Finally, it is anticipated that this work will contribute to the study and implementation of new management functionalities and Wireless Sensor Networks interconnection.

Indexing Terms: Internet of Things; Wireless Sensor Network; Proxy Manager; Network Management

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

6LoWPAN	=	*IPv6 over Low power Wireless Personal Area Networks*
ABNT	=	*Associação Brasileira de Normas Técnicas*
BER	=	*Bit Error Rate*
BOSS	=	*Bridge Of SensorS*
CH	=	*Cluster Head*
CPU	=	*Central Processing Unit*
dBm	=	*Decibel Miliwatt*
Eb/N0	=	*Energy per Bit to Noise power spectral density ratio*
EEPROM	=	*Electrically Erasable Programmable Read Only Memory*
E-TORA	=	*Energy Temporally Ordered Routing Algorithm*
HTTP	=	*Hypertext Transfer Protocol*
GAF	=	*Geographic Adaptive Fidelity*
GEAR	=	*Geographic and Energy Aware Routing*
GNU	=	*GNU is Not Unix*
GOAF	=	*The Greedy Other Adaptive Face Routing*
GPS	=	*Global Position System*
IEEE	=	*Institute of Electrical and Electronics Engineers*
INDAMS	=	*Integrated Network and Data Management System for Heterougeneous WSN's*
ITU	=	*International Telecommunication Union*
LAN	=	*Local Area Network*
LEACH	=	*Low-energy Adaptive Clustering Hierarchy*
LEACH-C	=	*Low-energy Adaptive Clustering Hierarchy Cell*
LQI	=	*Link Quality Indicator*
MCFA	=	*Minimum Cost Forwarding Algorithm*
MER	=	Modelo Entidade Relacionamento
MVC	=	*Model-View-Control*

NCH	=	*Network Control Host*
NMA	=	*Network Management Application*
NME	=	*Network Management Entity*
NS	=	Nó Sensor
OSI	=	*Open Systems Interconnection*
RAM	=	*Random Access Memory*
RFID	=	*Radio Frequency Identification*
RR	=	*Routing Reflector*
RSSF	=	Redes de Sensores sem Fio
RSSI	=	*Received Signal Strength Indication*
SAR	=	*Sequential Assignment Routing*
SNMS	=	*Sensor Network Management System*
SOAP	=	*Simple Object Access Protocol*
PEGASIS	=	*Power-Efficient Gathering in Sensor Information Systems*
PER	=	*Packet Error Rate*
PHP	=	*PHP: Hypertext Preprocessor*
PM	=	*Proxy Manager*
TCP/IP	=	*Transmition Control Protocol / Internet Protocol*
TBRPF	=	*Topology Broadcast Dissemination Based on Reverse-Path Forwarding Protocol*
TEEN	=	*Thereshold Sensitive Energy Efficient Sensor Network*
TORA	=	*Temporally Ordered Routing Algorithm*
USB	=	*Universal Serial Bus*
UPnP	=	*Universal Plug and Play*
WSN	=	*Wireless Sensor Network*
WRT	=	*Wireless Routing Protocol*
ZRP	=	*Zone Routing Protocol*

SUMÁRIO

1 INTRODUÇÃO

A necessidade de coletar dados e atuar nos ambientes físicos fez surgir nos últimos anos diversas classes de tecnologia. Ambientes inteligentes, os quais possuem poder computacional integrado (sistemas embarcados), fazem cada vez mais parte do cotidiano, tais como casas com iluminação inteligente, monitoração de estufas, controle inteligente de emissão de gases tóxicos, industrias inteligentes, etc. Ao conjunto destas tecnologias e aplicações, dá-se o nome de Internet das Coisas [1].

Além do poder computacional, outro fator que deve ser observado neste cenário é a comunicação [2]. Neste aspecto, as Rede de Sensores Sem Fio (RSSF), ou WSN (*Wireless Sensor Network*), uma das tecnologias que implementam o conceito de Internet das Coisas, possuem um papel fundamental. Uma RSSF consiste em elementos denominados Nós Sensores (NS) [3], equipamentos de tamanho reduzido, com baixo poder computacional e energético [4], os quais tem como função interagir com o meio ambiente, coletando dados relacionados a grandezas (tais como: temperatura, luminosidade, umidade, pressão atmosférica e qualidade do ar) com sensores ou atuando sobre o mesmo, através de atuadores (tais como motores ou sistemas mecânicos).

RSSFs tem sido utilizada nas mais diversas áreas, com várias aplicações. Entre elas, pode-se destacar [4]:

- **Militares:** detecção de *snipers*, armas químicas, movimento de inimigos, etc;
- **Meio ambiente:** rastreamento de pequenos animais, detecção de incêndios em florestas, estudo de poluentes, etc;
- **Saúde:** monitoramento de pacientes, administração de drogas, rastreamento de pacientes e doutores em hospitais, etc;
- **Residenciais:** monitoramento do consumo de água, interligação de eletrodomésticos, etc;
- **Industriais:** diagnóstico de máquinas, rastreamento de veículos,

monitoramento de áreas de risco, localização de instrumentação, etc;

A interconexão de uma rede externa (ex: Internet) a RSSF é realizada normalmente através de um *gateway*, devido a impossibilidade de se utilizar as pilhas de protocolos tradicionais (como, por exemplo, TCP/IP (*Transmission Control Protocol / Internet Protocol*) nos NS, por conta do baixo poder computacional destes. Um *gateway* é um elemento com maior poder computacional e energético, o qual possui interfaces que atuam nas duas redes, possibilitando o intercâmbio de dados entre as mesmas [5]. O *gateway* a princípio atua como um simples repassador de pacotes entre as duas redes. No entanto, devido ao alto volume de dados coletados e o limite de banda para *upload* na rede externa [6], o envio de dados brutos coletados pela RSSF para a Internet é uma abordagem ineficiente.

Para que a RSSF e o *gateway* possam funcionar corretamente, é imprescindível que os mesmos sejam gerenciados. Dentre os principais aspectos de gerência que podem ser observados em uma RSSF, pode-se citar [3][7]: eficiência energética, configuração de parâmetros de inicialização da RSSF, temporização das atividades, comunicação entre os NS, roteamento eficiente de pacotes, agregação de dados, agendamento de tarefas, recuperação de falhas, etc.

Apesar de sua importância, atualmente existe uma escassez de ferramentas de gerência, sendo este um dos principais fatores que limitam a adoção em larga escala comercial e industrial de RSSF [7], principalmente voltadas a ambientes previamente conhecidos, como indústrias, cidades, residências e hospitais, onde a quantidade e localização dos NS são de conhecimento prévio. Em geral, tais informações não são consideradas nas propostas existentes.

Tendo este cenário em vista, identifica-se a necessidade de uma plataforma de gerenciamento para a RSSF localizada no *gateway*, devido ao maior poder computacional deste último, voltada para cenários conhecidos. Esta plataforma terá como foco a gerência dos dados coletados (aplicação) e a infraestrutura da rede, através das áreas de gerência de Configuração, Desempenho e Falhas apresentados na Recomendação M.3400 da ITU (*International Telecommunication Union*) [8]. Conceitualmente, a plataforma se denomina *Proxy Manager (*Gerente Proxy). O termo é apresentado em [9], e designa um elemento intermediador entre

duas redes que não compartilham da mesma pilha de protocolos (a rede a ser gerenciada - RSSF - e a rede onde o Gerente de Rede está localizado – TCP/IP).

1.1 Motivação

Como abordado no tópico anterior, é fundamental que a gerência seja implantada em uma RSSF. Contudo, a delegação desta diretamente aos NS implica em *hardware* e protocolos mais complexos, aumentando o custo final da rede.

Além disso, o simples repasse passivo de dados, sejam eles relacionados aos sensores ou da infraestrutura da rede, através de um *gateway,* para uma aplicação ou Gerente de Rede hospedados em uma rede TCP/IP, pode sobrecarregar a infraestrutura desta última [6][10][11]. Isto se deve a enorme quantidade de dados brutos gerados por RSSF's e outras tecnologias relacionadas a Internet das Coisas. Sendo assim, é imprescindivel que a gerência dos dados coletados (aplicação) e da infraestrutura da rede seja realizada próxima da RSSF, como no *gateway*.

Por fim, atualmente existe uma escassez de ferramentas e estratégias que realizem o gerenciamento de uma RSSF, principalmente relacionada a ambientes previamente conhecidos, como estufas, industrias e hospitais, onde a localização física dos NS é de conhecimento prévio.

1.2 Objetivos

O objetivo principal deste trabalho é apresentar um *Proxy Manager* (PM), capaz de gerenciar uma RSSF para ambientes conhecidos, tanto do ponto de vista da infraestrutura de rede quanto da aplicação, através de parâmetros informados por uma aplicação Gerente externa a RSSF. Dentre as cinco áreas clássicas de gerência [8] – Configuração, Desempenho, Falhas, Contabilização e Segurança, a plataforma trabalhará com as três primeiras, visto que estão relacionadas à inicialização e manutenção da rede, sem a qual esta não pode operar de maneira estável.

Os seguintes aspectos de cada área serão tratados pela plataforma:

- **Configuração:** Inicialização da RSSF e do PM, estabelecimento de rotas entre o *Proxy Manager* e os NS, configuração dos sensores e criação de pacotes para solicitação de coletas de dados;

- **Desempenho:** Monitoramento da PER (*Packet Error Rate* – Taxa de Erro de Pacote), relacionada a cada NS;

- **Falhas:** Detecção de falhas envolvendo a PER, quando esta ultrapassar um limite tolerável;

A estratégia de gerenciamento utilizada será a centralizada, ou seja, todas as ações de gerência partirão do PM. Este deverá ser responsável por iniciar todas as ações de coleta de dados e realizar a análise dos mesmos. O objetivo desta medida é simplificar o *hardware* e *software* embarcado dos NS e, consequentemente, diminuir o custo e complexidade da RSSF. Além disso, em uma abordagem centralizada, o PM poderá ter uma visão total e abrangente da RSSF, controlando o *status* de cada elemento (NS e sensores) que a compõe.

No que tange a interface com a rede TCP/IP, realizada através de parametrizações e disponibilização de dados, o *Proxy Manager* deverá atender o maior número de aplicações possíveis, sejam elas gerentes ou que requisitem dados coletados pelos sensores, independente do *hardware ou software* em que os mesmos foram implementados. Para atingir tal objetivo, será utilizada a tecnologia de w*eb services*, a qual já demonstrou ser uma alternativa viável para interface de RSSFs com uma rede externa [12].

1.3 Organização do Trabalho

O restante deste trabalho é organizado da seguinte forma:

- **Capítulo 2:** Apresenta uma breve introdução ao conceito de Internet das Coisas, suas aplicabilidades e os principais desafios no âmbito de infraestrutura de redes;

- **Capítulo 3:** Trata dos conceitos relacionados a uma RSSF e os elementos que formam a mesma. Um aprofundamento maior em relação a roteamento em RSSF é abordado, devido a sua importância no âmbito deste trabalho;

- **Capítulo 4:** Apresenta conceitos relacionados a gerência de redes no âmbito geral e de RSSF, inerentes a este trabalho;

- **Capítulo 5**: Apresenta trabalhos semelhantes encontrados na literatura;
- **Capítulo 6:** Apresenta a proposta deste trabalho, ou seja, um *Proxy Manager* voltado para a Internet das Coisas;
- **Capítulo 7:** Apresenta a implementação da proposta deste trabalho;
- **Capítulo 8**: Apresenta os testes e resultados obtidos;
- **Capítulo 9:** Apresenta as considerações finais, através de uma conclusão e perspectivas de trabalhos futuros;
- **Referências Bibliográficas:** Apresenta as referências bibliográficas utilizadas neste trabalho;
- **Anexos:** Apresenta os anexos deste trabalho (o código fonte utilizado para implementação da proposta);

2 INTERNET DAS COISAS

Após a *World Wide Web* e a explosão de dispositivos *mobile*, uma nova revolução tecnológica está acontecendo na Internet, tendo como foco a conexão inteligente de objetos do cotidiano. Tecnologias como Rede de Sensores sem Fio (RSSF), *Radio Frequency Identification* (RFID) e localizadores em tempo real são exemplos dos pilares do conceito em questão: Internet das Coisas (*Internet of Things).* Estima-se que, até 2020, de 50 a 100 bilhões de dispositivos[1], tais como comidas, roupas, móveis, papéis, monumentos, veículos automotivos, obras de arte, etc; estarão conectados à Internet através de tecnologias pervasivas, se comunicando e colaborando entre si. Em termos econômicos, é esperado que a Internet das Coisas movimente $ 868 bilhões em 2016, aumentando para $ 1534 bilhões em 2020 [13].

Aplicações para diversas áreas envolvendo Internet das Coisas estão sendo desenvolvidas. Entre as principais, pode-se citar: automação residencial, aprendizagem reforçada, *e-hearth*, automação e manufaturação industrial, melhor logística, transporte inteligente de pessoas e bens, segurança inteligente, etc.[1]. Com isso, nota-se uma grande gama de aplicações tendo como foco indústrias e o consumidor final [14]. Como por exemplo, podem-se citar os prédios cognitivos [15], onde se espera levar um uso mais econômico de energia elétrica, água e outros recursos.

Com toda a sua potencialidade, as tecnologias envolvidas em Internet das Coisas trazem diversos desafios que necessitam ser pesquisados e desenvolvidos. Entre as principais estão a estrutura da rede, a segurança e o desenvolvimento de *softwares* apropriados para lidar com a diversidade de dados gerados [16].

Para atuar pervasivamente, as diferentes tecnologias que compõem a Internet das Coisas fazem uso de equipamentos equipados com sensores e atuadores, os quais possuem poder computacional e energético limitado. Para compensar tal limitação, os dados coletados por estes dispositivos são, idealmente, enviados a uma ou mais aplicações na Internet, onde plataformas com poder computacional e energético virtualmente ilimitados poderão processar os mesmos para diversos fins.

No entanto, a estrutura atual de telecomunicações não suporta um número

grande de dispositivos gerando e enviando uma massa de dados para serem processados na Internet [16][17]. Como constatação, nos últimos 20 anos o poder de processamento e armazenamento cresceu em um fator de 10^{18}, enquanto a largura de banda das redes apenas 10^{4}[17]. Sendo assim, estratégias relacionadas a tratamento e filtragem de dados, próximos dos dispositivos onde os mesmos são gerados, deve ser considerada. Além deste fator, aplicações relacionadas a Internet das Coisas exigem diversos requisitos de Qualidade de Serviço, tais como baixa latência, alta taxa de transferência, segurança eficiente e gerenciamento [14]. Soluções como *Fog Computing* [11] (termo que faz alusão a uma estrutura de computação em *cloud* próximo a rede onde os dados são gerados) e *Mobile Edge Computing* [18] levam estes fatores em consideração.

Além dos fatores apresentados, os protocolos tradicionais utilizados na Internet foram projetados para dispositivos com grande poder computacional e de armazenamento, [16], os quais vão na contra mão da necessidade dos dispositivos relacionados a Internet das Coisas. Por fim, diversos produtos disponíveis no mercado relacionados a Internet das Coisas utilizam padrões proprietários e fechados [16], os quais limitam a interoperabilidade e a inovação. Para contornar tal situação, é necessário o desenvolvimento de padrões abertos, nos mesmos moldes dos protocolos tradicionais, como o Ethernet, IP (*Internet Protocol*), TCP (*Transmission Control Protocol*), HTTP (*Hypertext Transfer Protocol*), etc.

O trabalho em questão apresenta uma proposta de solução para os problemas e desafios apresentados. O foco é uma plataforma de gerenciamento que opere próximo dos elementos da rede (uma RSSF – Rede de Sensores sem Fio), capaz de gerenciar tanto a infraestrutura de rede da mesma (garatindo aspectos de qualidade necessários), quanto os dados coletados pelos sensores, fornecendo uma estrutura para que os mesmos possam ser analisados e tratados antes de serem disponibilizados para uma aplicação localizada na Internet.

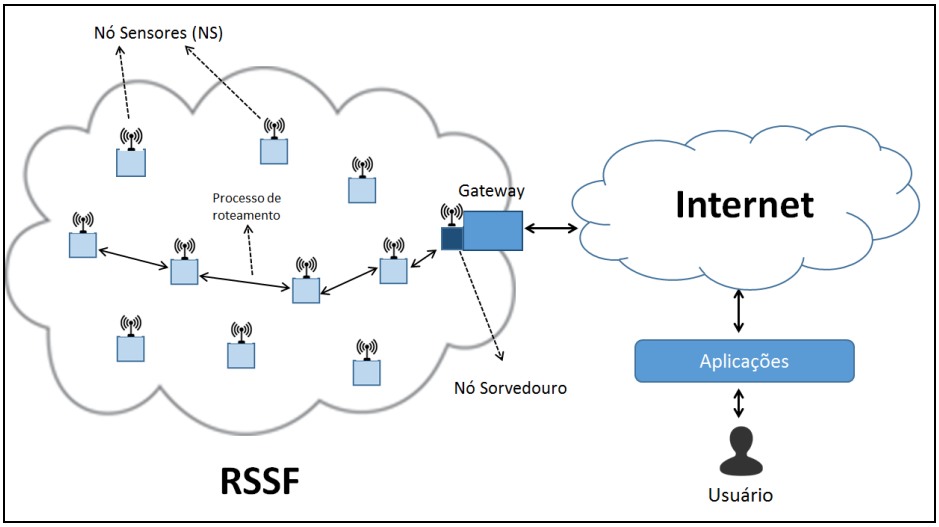
3 REDES DE SENSORES SEM FIO

Conforme mencionado na Introdução deste trabalho, uma RSSF (Rede de Sensores sem Fio) é uma rede onde seus elementos cooperativamente coletam e disponibilizam dados relacionados a grandezas do meio ambiente, tais como temperatura, pressão, radiação, luz, etc. Uma RSSF é um dos principais pilares de implementação do conceito de Internet das Coisas.

3.1 Elementos de uma RSSF

A Figura 1 ilustra os principais elementos que compõem uma RSSF, os quais são: Nó Sensor (NS), Nó Sorvedouro e *Gateway*. O objetivo final de uma RSSF é que os dados coletados pela mesma sejam consumidos por aplicações, localizadas em uma rede externa (normalmente a Internet). Para tal, os dados coletados por sensores, estes localizados nos NS, são encaminhados para um Nó Sorvedouro. Caso o NS não esteja diretamente ao alcance do Nó Sorvedouro, ele poderá encaminhar o pacote por intermédio de outros NS (roteamento), em um processo de múltiplos saltos. O Nó Sorvedouro, por sua vez, está conectado a um *gateway*, o qual será responsável por converter o pacote da pilha de protocolos da RSSF para a rede fim, como por exemplo, TCP/IP.

Figura 1 Elementos de uma RSSF



Fonte: Adaptado de [4]

Nos tópicos seguintes, um aprofundamento maior será dado ao NS e a

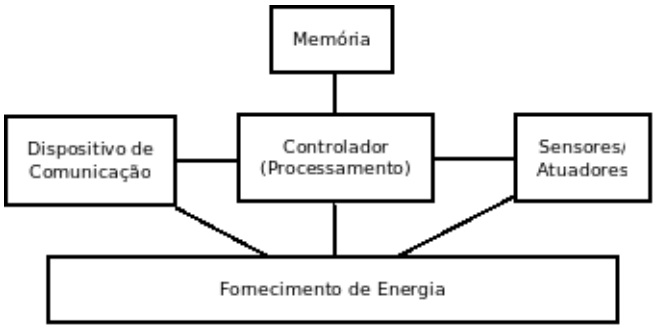
interconexão da RSSF com a Internet. Além disso, devido a importância para este trabalho, o roteamento em uma RSSF também será abordado.

3.1.1 Nó Sensor

O NS é o elemento responsável por coletar dados (grandezas) no ambiente e encaminhar os mesmos para o Nó Sorvedouro. Um NS pode desempenhar duas funções [4]: Coleta de dados e roteamento. A função de roteamento será abordada com detalhes no Tópico 3.2.

Um NS é composto pelos seguintes componentes: sensores e/ou atuadores, processamento (controle), memória, dispositivo de comunicação e fornecimento de energia [2]. O arranjo é mostrado na Figura 2.

Figura 2 Arquitetura de um NS



Fonte: [2]

A seguir, será dada uma explicação para cada um destes componentes.

- **Controlador:** O controlador (processador) é o componente que realiza o processamento de dados e atua sobre os sensores, atuadores e o dispositivo de comunicação. Ele também é responsável por escrever e recuperar dados na memória do NS. Para promover simplicidade e baixo consumo de energia, normalmente o poder computacional do controlador é baixo. Este tipo de processador é comumente denominado microcontrolador. Como exemplo de microcontroladores, pode-se citar: *Intel StrongARM, Texas Instruments MSP 430* e *Atmel Atmega*;

- **Memória:** A memória do sensor tem como função armazenar os dados coletados pelos sensores, pacotes a serem transmitidos (do próprio NS ou

de outros) e o *software* embarcado (*firmware*) que deverá ser executado. Comumente se utiliza memórias do tipo EEPROM (*Electrically Erasable Programmable Read Only Memory* – Memória de Somente Leitura Programável Apagável Eletricamente) ou *Flash,* para armazenamento não volátil de dados;

- **Dispositivo de Comunicação:** Responsável pela comunicação entre os NS e/ou o Nó Sorvedouro. Para tal, são utilizadas antenas em operação *half-duplex*. A escolha da antena apropriada é fundamental para a aplicação em questão, devendo ser observado aspectos como modulação, codificação, consumo de energia, ganho do sinal e sensibilidade do receptor. A comunicação sem fio se dá normalmente pela Rádio Frequência, pois a mesma é que mais se adapta aos requerimentos de aplicações para RSSF [2], devido ao maior alcance, altas taxas de transferência, níveis aceitáveis de erros e baixo consumo de energia;

- **Sensores:** Um sensor é um dispositivo físico capaz de ler uma grandeza do mundo físico (formato analógico) e convertê-la para o mundo virtual (formato digital), por intermédio de um conversor analógico-digital. Podem ser classificados em [2]: Sensores Passivos Omnidirecionais, onde ocorre a medição de um evento físico sem manipulação do ambiente em que se encontram (Exemplos: Microfones, umidade, termômetro, sensores de luz, etc); Sensores Passivos de Feixe Estreito, os quais também realizam a medição sem alteração do ambiente, mas a direção de onde está sendo feita a medida influencia diretamente os resultado (Exemplo: câmera de vídeo) e, por fim, Sensores Ativos, onde a medição ocorre por uma geração de eventos no ambiente em questão (Exemplos: Radar e Sonar, onde ondas de choque são criadas por pequenas explosões). Os tipos de sensores mais utilizados em uma RSSF são os sensores passivos omnidirecionais [2];

- **Atuadores:** Um atuador é um elemento que atua no meio ambiente, criando uma alteração física do mesmo. Normalmente, um NS será responsável por abrir ou fechar um *switch* ou relé, permitindo a passagem e/ou configuração de um valor, assim acionando um mecanismo de maior

potência, como um motor, sistema mecânico ou uma lâmpada [2];

- **Fornecimento de Energia:** A alimentação energética de um NS é um fator crucial, pois dela depende o funcionamento de todos os outros elementos que compõem o mesmo. Para tal, dois aspectos importantes devem ser levados em consideração: armazenamento de energia (normalmente através de baterias) e reposição da mesma (através de células fotovoltaicas, fluxo de ar ou líquidos, variações de pressão, mudanças de temperatura, etc) [2];

3.1.2 Interconexão com a Internet

A interconexão de uma RSSF com a *Internet* pode ser feita através de [19]: configuração direta da pilha de protocolos TCP/IP nos NS, *overlay* de endereçamento, utilização dos padrões IEEE 802.15.4 e *6LoWPAN (IPv6 over Low power Wireless Personal Area Networks)* ou através de um *Gateway* / *Proxy*.

A vantagem das duas primeiras abordagens é a comunicação direta entre uma aplicação localizada na rede TCP/IP e os NS. No entanto, as mesmas exigem um grande poder computacional e energético, devido a utilização de protocolos mais complexos [19][20]. Já os padrões IEEE 802.15.4 e *6LoWPAN* permitem a conversão da pilha IPv6 através de métodos de compressão voltados para dispositivos mais simples, através de um *gateway.* No entanto, estas tecnologias estão em estado preliminares de desenvolvimento, e devem passar por mais mudanças até se tornarem disponíveis [19].

A última abordagem – utilização de *Gateways / Proxy*, apesar de representar um possível gargalo para o fluxo de dados entre as duas redes, quando é implementado em uma plataforma com baixo poder computacional, permite que sejam utilizados abordagens e protocolos mais eficientes e apropriados para uma RSSF. Dada a sua vantagem em relação aos outros métodos, a abordagem deste trabalho é utilizar um Gateway / *Proxy* para a interconexão da RSSF com a rede TCP/IP.

O *Gateway* ou *Proxy* pode atuar de duas formas [19]: como *relay* ou *front-end*. Como um r*elay,* o *Gateway / Proxy* irá repassar os dados coletados de um *host* IP

para a RSSF e vice-versa. A desvantagem deste método, como já discutido na introdução deste trabalho, é o “inundamento de dados” e consequente sobrecarregamento da infraestrutura de telecomunicações da Internet, quando utilizado em larga escala. Quando o *Gateway / Proxy* atua como *front-end*, os dados a serem enviados para as duas redes ficam armazenados em um banco de dados local, o qual pode ser acessado tanto a partir da RSSF quanto da Internet. Esta abordagem é mais eficiente do ponto de vista de utilização dos recursos de telecomunicações, pois um tratamento e agregação dos dados podem ser realizados antes do envio dos mesmos, descartando dados duplicados e desnecessários para as aplicações. Este trabalho utiliza a abordagem *front-end*.

3.2 Roteamento em RSSF

O roteamento em uma RSSF é necessário, pois nem sempre um NS está próximo o suficiente do Nó Sorvedouro para comunicação direta. Neste caso, a comunicação é realizada por NS intermediários, formando uma rede de múltiplos saltos.

Dentre os principais desafios [21][22] para o desenvolvimento de protocolos de roteamento, pode-se citar: eficiência energética, distribuição espacial entre os NS, modelo de entrega de dados, funções desempenhadas pelos NS, escalabilidade da RSSF, robustez do protocolo e topologia da RSSF.

Em [22] é apresentada uma classificação para protocolos de roteamento em RSSF, baseada na estrutura da rede, a qual pode ser do tipo *flat*, hierárquico ou baseado na localização. Para o entendimento e classificação do protocolo deste trabalho, a seguir será apresentada uma descrição de cada uma das possibilidades:

- **Flat:** Todos os NS desempenham a mesma função, seja ela de sensoriamento ou roteamento. Protocolos *flat* podem ainda ser subdivididos em [23]: pró-ativos ou reativos. Em protocolos *flat* pró-ativos, a descoberta de rotas é realizada antes do processo de envio dos dados, ou seja, todas as rotas são descobertas antes de serem utilizadas. Este processo depende de um momento exclusivo de inicialização e configuração da rede, porém aumenta a taxa de troca de pacotes no momento de envio dos dados. Exemplo de protocolos do

tipo *flat* são: WRP (*Wireless Routing Protocol*) e TBRPF (*Topology Broadcast Dissemination Based on Reverse-Path Forwarding*). Em protocolos flat reativos, a descoberta de rotas é realizada somente quando necessário, sob demanda. Esta abordagem diminui o tempo necessário para inicialização e configuração da RSSF, mas pode causar atrasos na transferência dos dados. Exemplo: TORA (*Temporally Ordered Routing Algorithm*), E-TORA (*Energy Temporally Ordered Routing Algorithm*), *Gossiping*, *Flooding*, RR (*Routing Reflector*);

- **Hierárquico:** Nesta classificação, os NS desempenham funções especificas (roteamento ou sensoriamento). Para tal, os mesmos são divididos em *clusters*. Um dado NS pertence somente a um *cluster*. Cada *cluster* possui um NS *Cluster Head* (CH), o qual é responsável pela comunicação do cluster com um CH de nível superior. Um determinado NS só pode se comunicar com seu CH. Exemplos de protocolos hierárquicos são: LEACH (*Low-energy Adaptive Clustering Hierarchy)*, LEACH-C (*Low-energy Adaptive Clustering Hierarchy Cell*), PEGASIS (*Power-Efficient Gathering in Sensor Information Systems*) e TEEN (*Thereshold Sensitive Energy Efficient Sensor Network*));

- **Baseado em localização:** Os NS são endereçados tendo como parâmetros sua localização, o qual é estimada pela coordenada geográfica dos mesmos, normalmente através de um GPS (*Global Position System*). Exemplos de protocolos baseados em localização são: GAF (*Geographic Adaptive Fidelity*), GEAR (*Geographic and Energy Aware Routing*) e GOAF (*The Greedy Other Adaptive Face Routing*);

Protocolos de roteamento baseado em localização encarecem o custo da RSSF, por exigir *hardware* adicional (o módulo GPS). Além disso, tal abordagem é desnecessária para ambientes conhecidos, onde a localização dos NS já é de conhecimento prévio. Protocolos hierárquicos, apesar de promover escabilidade para uma RSSF, também aumentam o custo da mesma, visto a necessidade de NS específicos para atuar como CH. Já os protocolos do tipo *flat* possuem a vantagem

de desempenharem as tarefas de roteamento e sensoriamento de forma simultânea, diminuindo o custo final da RSSF. Sendo assim, o protocolo de roteamento desenvolvido deste trabalho utiliza a abordagem *flat*. A subclassificação do mesmo é pró ativo, a fim de não diminuir a taxa de transferência dos dados durante a fase de coleta dos mesmos.

Diversos Protocolos de roteamento para RSSF com abordagem *flat* pró ativo foram propostos na literatura [21][22]. A seguir, serão listados alguns exemplos encontrados.

Topology Dissemination Based on Reverse-Path Forwarding Protocol (TBRPF) [24] utiliza a direção reversa (no sentido NS para Nó Sorvedouro) para encaminhar as atualizações de roteamento, usadas para calcular caminhos com o mínimo número de saltos. Em ambos os protocolos, porém, é necessário que os NS possuam alto poder de processamento e armazenamento, aumentando a complexidade e consequentemente, o custo final da RSSF.

Sequential Assignment Routing (SAR) [4] introduz o conceito de QoS (Quality of Service – Qualidade de Serviço) avaliando os recursos energéticos dos NS, o QoS dos enlaces e prioridades de entrega dos pacotes, criando uma árvore de roteamento entre os NS para garantir múltiplas rotas. No entanto, os dados referente a tabela de roteamento deve ser armazenado nos NS, aumentando o custo de memória dos mesmos.

Minimum Cost Forwarding Algorithm (MCFA) [25] não utiliza tabelas para armazenamento de rotas nos NS, mas sim um custo estimado dele mesmo para o *Gateway*. Um NS transmite um pacote através do processo de *broadcast*. Quando um NS recebe o pacote, o mesmo verifica se está no caminho de menor custo. Em caso positivo, ele transmite o pacote, até que se atinja o G*ateway*. A grande desvantagem deste protocolo é a grande quantidade de pacotes *broadcast* gerados, exaurindo os recursos dos NS.

4 GERENCIAMENTO DE REDES

4.1 Aspectos de Gerência

A medida que uma rede de computadores cresce e se torna mais complexa, se faz necessário criar um mecanismo para gerir a mesma. Sendo assim, a ITU (*International Telecommunication Union*) através da Recomendação M.3400 [8], definiu 5 áreas funcionais de gerenciamento para redes de computadores, apresentadas na Tabela 1.

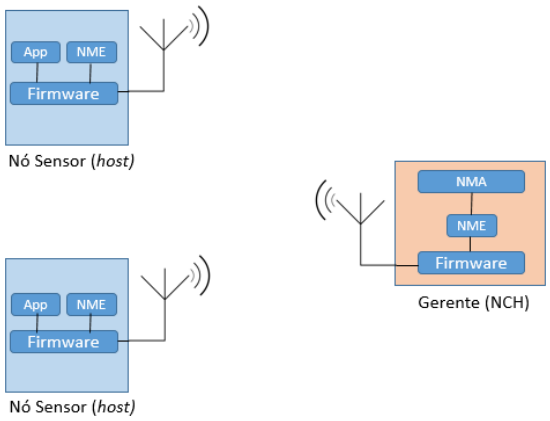
Tabela 1 Áreas Funcionais de Gerenciamento de Redes (ISO)

Área	Definição
Desempenho	Monitoração do desempenho dos elementos da rede e da qualidade das atividades de comunicação.
Falha	Detecção e correção de operações anormais na rede.
Configuração	Gerenciamento dos dados de inicialização e desligamento da rede, bem como a configurações pertinentes a operação da mesma.
Contabilidade	Bilhetagem e controle de custos a utilização de elementos da rede.
Segurança	Controle e proteção dos recursos de rede.

Fonte: Elaboração própria

Existem basicamente dois tipos de entidades que compõem um sistema de gerência: um Gerente (*Network Control Host* - NCH) e um ou mais Agentes [9]. Os Agentes ficam localizados nos *hosts* de rede a serem gerenciados. O NCH, por sua vez, está hospedado em um *host* dotado de uma interface para permitir que o usuário final interaja com o gerenciamento da rede. Tanto o Gerente quantos os *hosts* gerenciados possuem uma Entidade de Gerenciamento de Rede (*Network Management Entity* - NME), responsável por enviar dados referente a gerência para uma Aplicação de Gerenciamento de Rede (*Network Management Application* - NMA). A Figura 3 ilustra estes elementos, tendo como exemplo uma RSSF hipotética.

Figura 3 Ilustração hipotética dos elementos de gerência em uma RSSF



Fonte: Adaptado de [9]

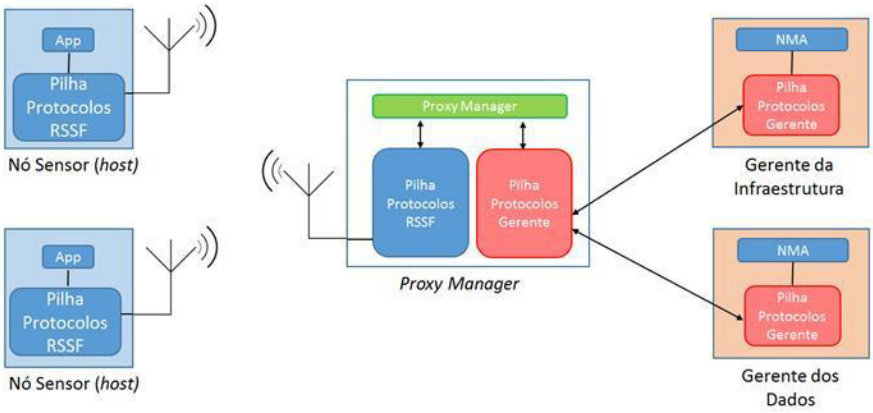
O esquema de gerenciamento ilustrado na Figura 3, no entanto, não é passível de ser implementado em alguns casos, devido a impossibilidade de implantação dos protocolos de gerenciamento nos *hosts* gerenciáveis. Os motivos principais para tal limitação são: falta de capacidade computacional e de armazenamento dos *hosts* gerenciáveis; sistemas antigos que não suportam os padrões de gerenciamento; componentes que não suportam *softwares* adicionais, tais como modems e multiplexadores [9].

Para resolver tal impasse, um Gerente pode se comunicar com os *hosts* gerenciáveis através de um *Proxy Manager* (PM) (Gerente Proxy, em uma tradução literal). O PM atua tanto na pilha de protocolos do Gerente quanto na pilha de protocolos da rede onde os *hosts* gerenciados estão localizados. Caberá a ele traduzir os parâmetros informados pelo Gerente para dados passiveis de serem lidos pelos *hosts* gerenciáveis, e reportar o *status* da rede para o gerente.

Devido a sua limitação em termos de capacidade computacional e a consequente utilização de protocolos especificos, o gerenciamento de uma RSSF pode ser intermediado através de um PM. A Figura 4 atualiza a Figura 3 com a presença do PM, ilustrando as duas diferentes pilhas de protocolo: a da rede a ser gerenciada (no exemplo, uma RSSF), e da rede onde o Gerente está localizado. Além da inserção do PM, é possível observar a presença de dois gerentes: o da Infraestrutura e o dos Dados. No contexto de uma RSSF, o Gerente de Infraesturutra

é responsável por manter a infraestrutura da rede operacional. Entre suas responsabilidades, estão a de endereçamento dos NS, descoberta de rotas, análise do desempenho, tratamento de falhas, etc. Já o Gerente dos Dados é responsável por administrar os dados referente a aplicação fim da RSSF, ou seja, as grandezas coletadas pelos NS, como temperatura, umidade, pressão atmosférica, etc. É importante salientar que os dois gerentes são indepententes, atendendo cada um suas respectivas peculiaridades. Neste caso, o PM pode intermediar o gerenciamento de ambos.

Figura 4 Introdução de um *Proxy Manager* para o gerenciamento de uma RSSF



Fonte: adaptado de [9]

Os Gerentes e o PM não necessitam estar condicionados no mesmo *hardware*, nem na mesma rede local. A comunicação entre os mesmos pode ser realizada por intermédio da Internet, por exemplo.

4.1 Gerenciamento em RSSF

Devido a suas características (limitação de recursos computacionais e energéticos, meio físico de transmissão, localização dos NS, etc) o gerenciamento em uma RSSF é único, não se comparando ao de redes tradicionais. Uma determinada plataforma de gerenciamento pode, por exemplo, descobrir rotas para os NS, mudar o *status* de determinado NS (ligado / desligado), diminuir ou aumentar a taxa de transmissão de dados e realizar reconfigurações (como descoberta de rotas alternativas) para tratar eventuais falhas [26]. Dentre os principais fatores que devem ser observados no desenvolvimento de novas plataformas de gerenciamento,

pode-se destacar [27]:

- **Topologia da Rede:** novos NS podem ser inseridos na RSSF ou podem mudar de local periodicamente, alterando rotas previamente estabelecidas. A plataforma de gerenciamento deve ser capaz de detectar estas mudanças e / ou periodicamente se alto reconfigurar, sem a necessidade de intervenção humana;

- **Tolerância a falhas:** um dado NS pode apresentar falhas, como uma PER (*Packet Error Rate*) alta, dificultando a coleta de dados ou, se este estiver no meio de uma rota para outros NS, comprometer a comunicação com estes. É desejável que a plataforma de gerenciamento possa se recuperar das falhas através de uma reconfiguração, com mínimo esforço humano ou, em último caso, alertar um administrador do problema em questão;

- **Limitação de recursos dos NS**: Baixo poder de processamento, armazenamento e limite de fornecimento de energia dos NS são questões importantes que devem ser consideradas pela plataforma de gerência. Delegar tarefas e decisões de gerenciamento complexas diretamente aos NS tende a exaurir os recursos dos mesmos;

O controle em uma plataforma de gerenciamento para RSSF pode ser classificado de três formas [27]: centralizado, descentralizado e hierárquico, cada um apresentado suas respectivas vantagens e desvantagens.

Em plataformas com foco centralizado, uma determinada estação base coleta dados de todos os NS e toma toda decisão relacionada ao gerenciamento da RSSF. Esta base deve possuir maior poder computacional e energético do que os NS. Assim, os NS podem ser elementos mais simples do ponto de vista de *hardware* e *software*, diminuindo os custos de implantação da RSSF. Caso a comunicação entre o NS e a base de gerenciamento não possa ser feita de forma direta, um NS intermediário poderá ser utilizado para rotear pacotes, sendo a rota informada pela base gerenciadora. A desvantagem deste método é que se a base de gerenciamento apresentar falhas, a RSSF ficará comprometida. Exemplos de plataformas com foco centralizado são: SNMS (*Sensor Network Management System*), Sympathy, MOTE-

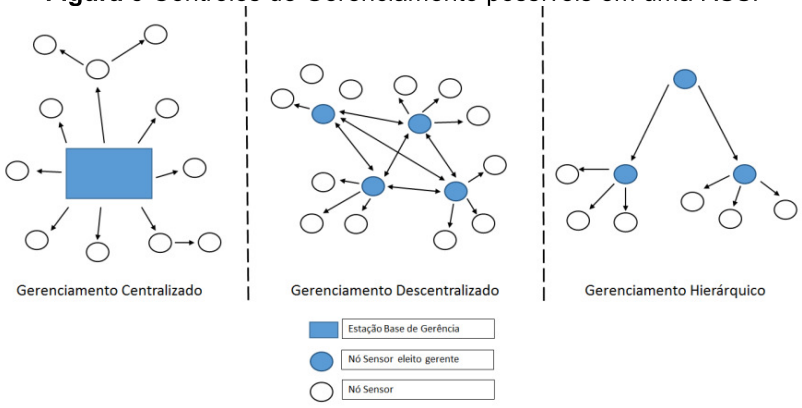
VIEW e BOSS (*Bridge Of SensorS*) [27].

Em plataformas com foco distribuído, diversos NS assumem a função de gerenciamento, cada um em uma porção da RSSF. Um determinado NS pode ou não estar desempenhando a função de gerenciamento em dado momento, dependendo de sua capacidade energética atual e computacional. O Gerenciamento distribuído é mais custoso de ser implementado devido ao aumento da complexidade dos algoritmos relacionados à gerência da rede que, por sua vez, devem ser executados nos NS. Com isso, existe uma maior necessidade de poder de processamento. Uma maior capacidade de armazenamento nos NS também deve ser levada em consideração, já que estes precisarão armazenar dados referente aos seus NS vizinhos, tais como nível de energia, *status* atual, rotas, etc. Exemplos de plataforma com foco distribuído são: Agila, *Sectoral Sweeper* e Siphon [27].

Por fim, em plataformas com foco hierárquico, alguns NS de dada região (ou *cluster*) assumem a função de gerência de forma semelhante ao gerenciamento distribuído, porém dispostos de forma hierarquica. Um NS assume a função de gerência após um processo de seleção, onde o nível de energia e poder de processamento pode ser levado em consideração. Cada um deles irá se reportar apenas a outro de nível maior. Exemplos de plataforma com foco hierárquico são: STREAM, AppSleep, TopDisc e SenOS [27].

A Figura 3 ilustra os três esquemas de controle de gerenciamento apresentados.

Figura 5 Controles de Gerenciamento possíveis em uma RSSF



Fonte: elaboração própria

Levando em consideração os paradigmas atuais de gerência para redes,

pode-se verificar um grande número de alternativas de gerência distribuída. No entanto, tal abordagem não é escalável para Internet das Coisas [28], devido principalmente a falta de visão global da rede. Devido aos fatores apresentados, a proposta deste trabalho terá como foco o gerenciamento centralizado.

5 TRABALHOS SEMELHANTES

Como neste trabalho é apresentada uma proposta para gerenciamento com arquitetura de gerenciamento centralizada, optou-se por pesquisar na literatura por alternativas que compartilhassem desta característica.

BOSS (*Bridge Of SensorS)* [29] é uma plataforma de gerenciamento para rede de sensores, sendo também possível de ser implementada em RSSF, ao qual implementa uma *bridge* (ponte) entre a rede gerenciada (RSSF, a qual utiliza um protocolo proprietário), e uma rede que trabalhe com o protocolo UPnP (*Universal Plug and Play)*, atuando como intermediário entre NS que não trabalham com o protocolo em questão e um Ponto de Controle UPnP. Todo processo de gerência acontece neste ultimo, sendo a plataforma BOSS encarregada da conversão dos pacotes. Os serviços de gerência implementados são: informações da rede, localização, sincronização e gerência de energia. Seu ponto fraco é a necessidade de um operador humano para tomada de decisões de gerência, sendo esta não realizada de maneira automática.

Sympathy [30] é um sistema que coleta métricas de desempenho em uma RSSF para identificar e localizar falhas na rede (denominada de eventos), em um contexto temporal. Falhas detectadas pelo mesmo podem ser, por exemplo, perda de pacote e flutuações no desempenho de roteamento. A RSSF pode ser dividida em dois tipos de NS: um *Sympathy-sink* e um *Sympathy-node*. Um *Sympathy-node* coleta as métricas de desempenho e os envia para um *Sympathy-sink*, para que estes possam processar os eventos e achar possíveis falhas que estejam ocorrendo. Um *Sympathy*-node pode decidir quando enviará os dados, podendo agregar os mesmos antes. No entanto, a detecção destas falhas ocorre apenas após as mesmas já terem ocorrido, já que o os dados são coletados pelo *Sympathy-node* não são necessariamente enviados em tempo real para análise.

SNMS (*Sensor Network Management System*) [31] tem como função monitorar a saúde da RSSF, através de duas funções: busca por dados relacionados ao desempenho e registro de eventos. Entre os valores utilizados para avaliação, estão o nível da bateria, temperatura e umidade nas proximidades dos NS. O tráfego pode ocorrer através da coleta dos dados e a disseminação de mensagens de

gerência para os NS. Para estabelecer rotas, cada NS mantém em sua memória o endereço do NS ao qual possui a RSSI (*Received Signal Strength Indicator*) maior. A troca de pacotes ocorre até se atingir uma estação base de gerência. A vantagem desta estratégia é que os NS não necessitam armazenar tabelas de roteamento, tornando a memória dos mesmos mais simples. Sua desvantagem é o fato de que o monitoramento da saúde da RSSF é passivo, necessitando de um operador humano para realizar intervenções e tratamento de falhas.

MOTE-VIEW [32] é uma ferramenta de gerência utilizada para monitoramento e gerenciamento de uma RSSF. Suas funcionalidades são a análise dos dados gerados pelos sensores e o monitoramento do desempenho dos NS de forma individual, bem como o da RSSF como um todo. Toda a análise dos dados acontece em uma estação base. O usuário final pode visualizar e fazer consultas de gerência através de uma interface. Sua arquitetura é composta por 4 camadas: Camada de Abstração de Acesso aos Dados (utilizada para recuperar os dados coletados pelos NS); Camada de Abstração do Nó (armazena metadados, como nome, configurações e calibrações dos NS), Camada de Abstração de Conversão (calibra e converte os dados coletados pelos NS) e a Camada de Abstração de Visualização, onde o usuário pode efetivamente consultar os dados através de gráficos, planilhas ou até mesmo a topologia da rede. Apesar do monitoramento, a plataforma não oferece reconfiguração dos NS no caso de ocorrência de falhas.

TinyDB [33] é um sistema que armazena meta-dados relacionados aos sensores e NS que compõem a rede, como tabelas de roteamento. Através destes dados, o mesmo pode prover uma topologia da RSSF. No entanto, o sistema não oferece um mecanismo para recuperação automática de falhas, necessitando de um operador humano para tal.

Em [34] é apresentado um Projeto para Gerenciamento Web para RSSFs utilizando a arquitetura MVC (*Model-View-Control).* A vantagem de se utilizar esta é a separação da interface do usuário (*View*) dos dados a serem manipulados (*Model*) e da lógica do sistema (*Control)*, facilitando o desenvolvimento de aplicações distintas. Os autores implementam o projeto utilizando 2 aplicações: uma para monitorar o consumo de energia em dispositivos eletrônicos e outra para monitorar a temperatura e umidade em uma sala. Os módulos da aplicação se dividiram em: tela

principal, dados estatísticos, logs e usuários. Apesar de lidar com os dados do usuário, a plataforma não leva em consideração possíveis falhas que possam ocorrer na RSSF, ficando esta a cargo de outra ferramenta.

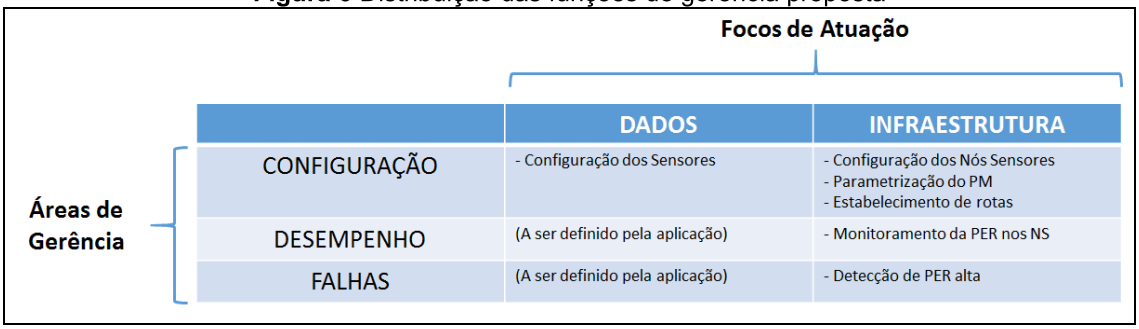
Em [35], é proposto um sistema de gerenciamento integrado para RSSFs heterogêneas (INDAMS – *Integrated Network and Data Management System for Heterougeneous WSN's)*, separando a função de gerenciamento da rede da aplicação e promovendo uma interface web acessível ao usuário final. Para suportar diferentes tipos de RSSFs, os autores propõem uma arquitetura baseada em 5 camadas (apresentação, aplicação, *gateway* unificado, agente e *mote*). A comunicação entre estas diferentes RSSFs é realizada através de um protocolo *agent-server*, onde a figura do Agente é responsável pela comunicação entre a tecnologia da RSSF e o *gateway*. O ponto fraco desta plataforma é também não lidar com possíveis falhas que venham a ocorrer na rede.

6 PROXY MANAGER PARA INTERNET DAS COISAS

A proposta deste trabalho é apresentar uma plataforma de gerenciamento centralizado, implementada no *Gateway* da RSSF, baseada em um *Proxy Manager* (PM), de baixo custo, com a finalidade de gerenciar uma RSSF com múltiplos saltos, instalada em ambientes previamente conhecidos. Nestes ambientes, o número de NS, bem como o endereço e localização geográfica dos mesmos já são de conhecimento prévio, informados por uma aplicação Gerente. Exemplos destes ambientes são: indústrias, residências, cidades, prédios comerciais, etc.

O PM atuará tanto na gerência da infraestrutura da rede quanto na gerência dos dados coletados, através das áreas de gerência de configuração, desempenho e falhas, propostas pela Recomendação M.3400 da ITU. A Figura 6 ilustra o modelo de distribuição das funções de gerência proposta neste trabalho, sendo composta de camadas horizontais (áreas de gerência) e verticais (foco da atuação).

Figura 6 Distribuição das funções de gerência proposta



Fonte: elaboração própria

As áreas de gerência escolhidas (Configuração, Desempenho e Falhas) foram escolhidas, pois dizem respeito as funcionalidades de parametrização, inicialização, monitoração e manutenção da RSSF, sem os quais esta não pode operar de maneira estável.

A interligação das camadas horizontais e verticais corresponde as funções desempenhadas pelo PM. Na Figura 6, são apresentadas as funções implementadas neste trabalho. A área de gerência para desempenho e falhas dos dados não foram implementadas, pois as mesmas podem variar de acordo com a aplicação, onde necessidades especificas devem ser analisadas de forma independente. No entanto, a plataforma deve oferecer o suporte para o desenvolvimento futuro de tais

funcionalidades, através da coleta e armazenamento dos dados coletados. É importante salientar que outras funcionalidades, independente da área de gerência e do foco de atuação, podem e devem ser desenvolvidas em trabalhos vindouros.

A seguir, será apresentada uma explicação para as funções de gerência propostas:

- **Configuração de Dados:** Tem como objetivo realizar a configuração de parâmetros relacionados a aplicação final, ou seja, referente aos dados de grandezas coletadas (temperatura, luminosidade, umidade, etc). Dentre possíveis funções, estão: inclusão e manutenção dos sensores para cada NS, priorizações de requisição de dados (escalonador de tarefas), etc;

- **Desempenho de Dados:** Permite que o PM faça análises estatísticas com os dados coletados, além de disponibilizar os resultados para outras aplicações através de *web services*. Esta função é de importância significativa para a diminuição do *upload* de dados brutos para a Internet;

- **Falhas de Dados:** Permite que o PM identifique um determinado padrão de leituras e faça algum tratamento emergencial em relação ao mesmo, como, por exemplo, um aumento subito na temperatura de determinado ambiente onde o NS se encontra. Neste caso, o PM poderia tratar a falha acionando um mecanismo externo, como um sistema de alarme sonoro;

- **Configuração de Infraestrutura:** Permite a configuração tanto do PM quanto dos NS, no que tange a aspectos de rede, como endereçamento, descoberta de rotas, limites toleráveis de PER nos NS, *status* dos NS, etc;

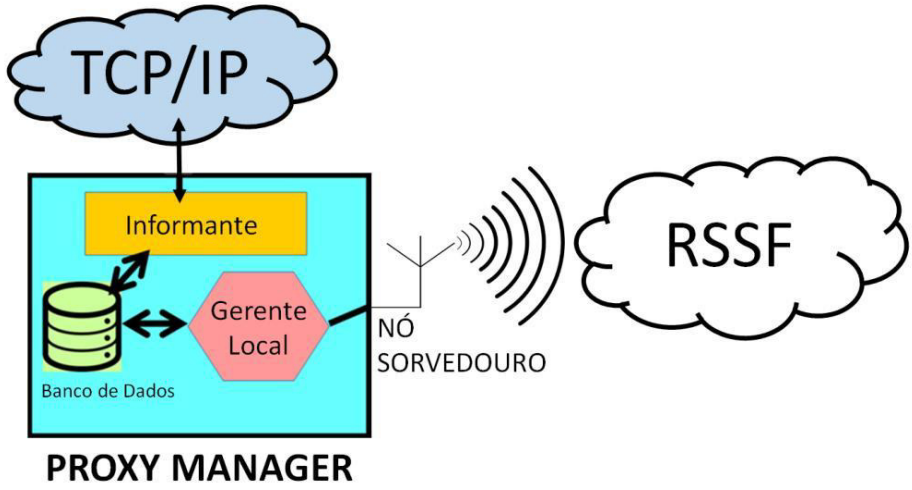
- **Desempenho de Infraestrutura:** Permite que o PM monitore constantemente o desempenho da infraestrutura da RSSF, como, por exemplo, a RSSI e a LQI dos enlaces e a PER dos NS;

- **Falha de Infraestrutura:** Permite que o PM identifique uma determinada falha envolvendo os parâmetros avaliados na função de Desempenho da Infraestrutura, como por exemplo, uma PER mais alta do que a tolerável em um determinado período de tempo. O PM pode tratar a falha realizando uma tentativa de encontrar rotas alternativas para dado NS ou acionando um mecanismo externo, como enviar um *e-mail* para um operador humano da RSSF;

A arquitetura proposta tem como abordagem o gerenciamento e coleta de dados de forma centralizada, sendo que toda e qualquer comunicação na RSSF inicia a partir do PM, através de um módulo denominado Gerente Local. É responsabilidade deste último implementar todas as atividades relacionadas ao funcionamento da RSSF, como por exemplo, descoberta de rotas, coleta de dados, monitoramento do desempenho da rede, acionamento de mecanismos de tratamento de falhas, etc. O módulo Gerente Local é dividido em dois processos: o *FluxoPrincipal*, responsável pela área de gerenciamento de configuração dos NS e do PM e coleta dos dados, e o *PERAvaliador*, responsável pelo gerenciamento de desempenho e falhas, através do monitoramento da PER para cada NS. Estes dois processos são executados em paralelo pelo PM.

A parametrização e a disponibilização de dados coletados na RSSF para a Internet é realizada através de um módulo denominado Informante, o qual implementa uma série de *web services,* utilizando para isto o protocolo SOAP (*Single Object Access Protocol*). Todos os parâmetros e dados coletados pela RSSF são armazenados em um Banco de Dados Relacional local, onde os módulos Informante e Gerente Local estão conectados. A Figura 7 ilustra a interligação destes módulos.

Figura 7 Componentes do Proxy Manager



Fonte: elaboração própria

Conforme mencionado anteriormente, as funções desempenhadas pelo PM neste trabalho são elencadas na Figura 6. Estas são implementadas nos módulos Gerente Local (através dos processos *FluxoPrincipal* e *PERAvaliador*), e Informante.

Será abordado, nos tópicos seguintes, um detalhamento de cada uma destas funções, separadas pelas áreas de gerenciamento. Após a descrição dos mesmos, será apresentado a estrutura do Banco de Dados.

6.1 Gerenciamento de Configuração da Infraestrutura

O objetivo da área de gerenciamento de configuração da infraestrutura é realizar a inicialização, configuração e manutenção do PM e da RSSF. A configuração ocorre no início das atividades da RSSF, mas também pode acontecer por resposta a uma falha detectada ou por mudanças solicitadas por uma aplicação Gerente, através do módulo Informante. Conforme ilustrado na Figura 6, as funcionalidades referentes a configuração da infraestrutura implementadas neste trabalho são: Configuração dos NS, do PM e Estabelecimento de Rotas. Nos tópicos seguintes, são apresentadas cada uma das mesmas.

6.1.1 Configuração dos Nós Sensores e Proxy Manager

Toda inserção, alteração e exclusão de NS é realizado através de *web services*, no Módulo Informante. O nome e a descrição destes serviços são descritos na Tabela 2.

Tabela 2 *Web Services* para cadastro, alteração e exclusão de NS

Web Service	**Descrição**
insertNS	Cadastra os NS através de seu endereço, descrição, localização e PER Tolerável.
alterInformationNS	Altera a descrição e a localização do NS.
alterPERDataNS	Altera os dados relacionados a PER tolerável.
excludeNS	Exclui o cadastro de determinado NS
alterStatusNS	Modifica manualmente o *status* de determinado NS

Fonte: Elaboração própria

O PM associa a cada NS um *status*. Os *status* possíveis são listados na Tabela 3.

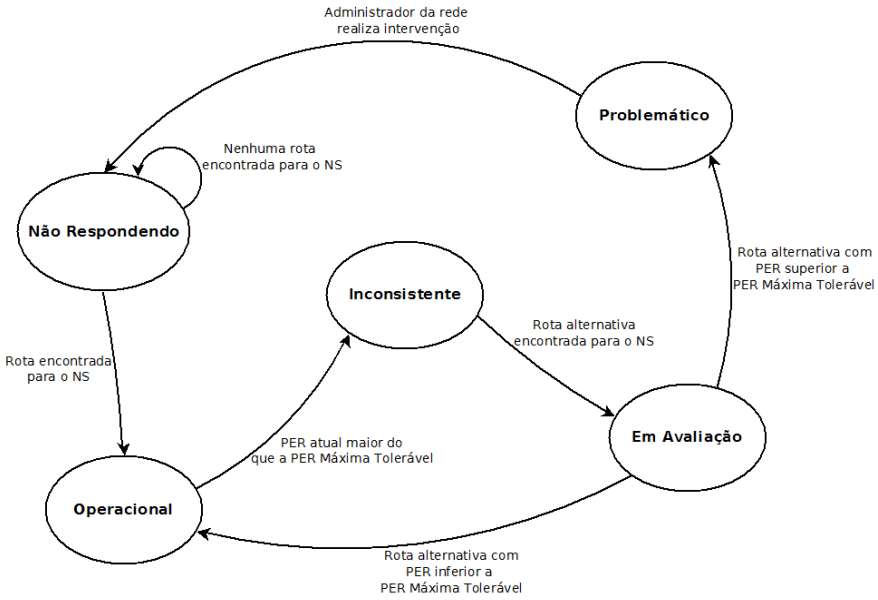
Tabela 3 *Status* possíveis de um NS

Estado	**Descrição**
NR	“Não Respondendo”. Indica que nenhuma rota para o NS foi encontrada.
OP	“Operacional”. NS com rota cadastrada e que está operando com uma PER menor do que a PER Tolerável.
IN	“Inconsistente”. NS está operando com uma PER maior do que a PER Tolerável.
EA	“Em Avaliação”. Uma rota alternativa está sendo avaliada.
PR	“Problemático”. NS com PER maior do que a PER Tolerável, e que não possui rota alternativa satisfatória.

Fonte: Elaboração própria

A Figura 8 apresenta o diagrama de transição dos estados apresentados para os NS, demonstrados na Tabela 3.

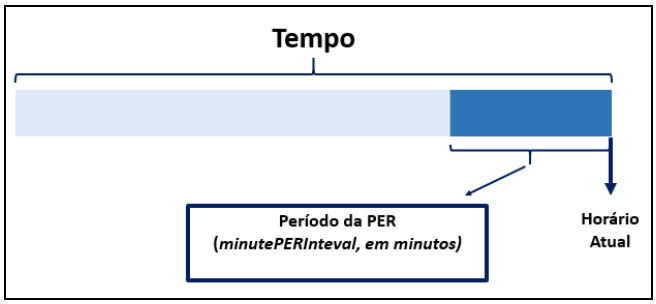
Figura 8 Diagrama de transição de estados dos NS



Fonte: Elaboração própria

Ao ser cadastrado, o NS irá receber o *status* de “NR” (Não Respondendo). Somente após a descoberta de uma rota (processo este detalhado no item 6.1.2) é que o *status* do mesmo será alterado para “OP” (Operacional). A área de configuração irá mudar o *status* para “IN” (Inconsistente) quando a PER atual for maior do que a PER Máxima Tolerável. A PER Máxima Tolerável é a maior PER que o PM irá tolerar para um determinado NS. Ela sempre estará associada a uma janela deslizante denominada Período da PER, onde o horário inicial é calculado subtraindo a variável *minutePERInterval* (informado no momento de cadastro do NS) do horário atual. A Figura 9 apresenta uma ilustração do Período da PER ao longo do tempo.

Figura 9 Período da PER - Ilustração



Fonte: Elaboração própria

Caso uma rota alternativa esteja sendo avaliada para um determinado NS, o

status do mesmo será alterado para EA (“Em Avaliação”). Qualquer mudança do *status* de um NS é registrada através de *logs* no Banco de Dados. Uma aplicação gerente pode alterar o *status* de determinado NS de forma manual, utilizando o w*eb service alterStatusNS*, quando julgar necessário.

De modo semelhante a configuração dos NS, toda alteração das configurações envolvendo o PM é realizada através de *web services,* os quais são descritos na Tabela 4.

Tabela 4 *Web Services* para alteração das configurações do PM

Web Service	**Descrição**
updateSinkAddress	Atualiza o endereço do Nó Sorvedouro.
updateSerialPort	Atualiza a porta serial utilizada para comunicação com a RSSF
updateLimitDaysPacket	Atualiza o limite de dias que os dados coletados referentes os pacotes podem ficar no sistema
forceInterrupt	Força processo de interrupção para descoberta de rotas para NS com *status* IN e NR

Fonte: Elaboração própria

Conforme já mencionado, o processo no módulo Gerente Local responsável pela área de configuração e coleta dos dados se denomina *FluxoPrincipal*. A primeira tarefa desempenhada por este processo é o teste da porta serial, onde o Nó Sorvedouro está conectado ao PM. Se a mesma não estiver funcionando, um *log* é criado reportando o fato, e um alerta é enviado ao operador humano para que o mesmo tome providências. O PM então interrompe qualquer atividade relacionada a RSSF.

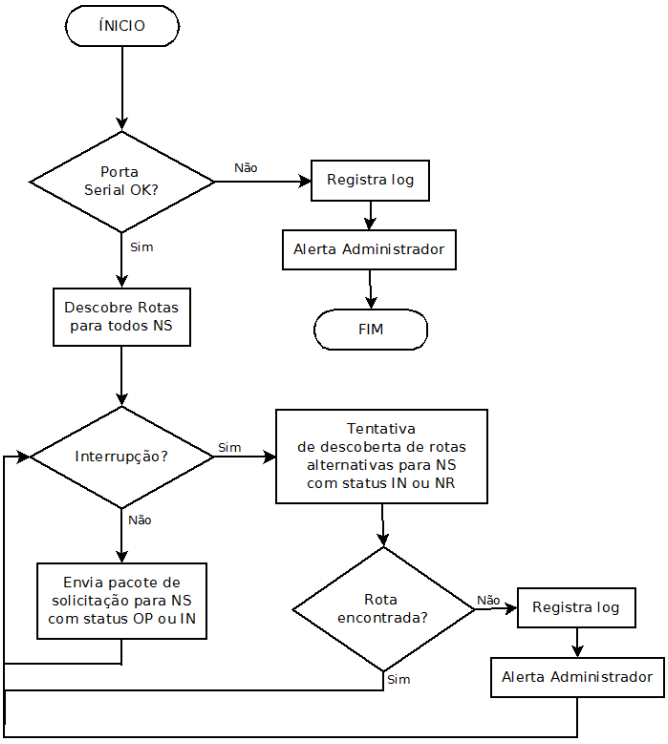
Se a porta serial estiver funcionando, o PM irá realizar uma descoberta de rotas para todos os NS (processo detalhado no item 6.1.2), independente do seu *status* cadastrado. Caso algum NS não seja encontrado neste processo, uma mensagem é enviada ao operador humano para que o mesmo tome providências.

Após a etapa descoberta de rotas, o PM começará a coletar dados dos sensores, enviando pacotes de solicitação para todos os NS com *status* “OP”. Neste processo, caso a PER de um ou mais NS seja maior do que PER Máxima Tolerável, o processo *PERAvaliator* gerará uma interrupção na coleta dos dados, para que a

área de configuração possa tentar descobrir rotas alternativas para o(s) NS em questão. Em caso positivo de descoberta, o *status* do(s) NS será alterado para EA, e ficará assim por um tempo igual ao Período da PER. No final, caso a PER for menor do que a PER Tolerável, o *status* do NS é alterado para OP. Caso contrário, é alterado para PR, e um alerta é enviado ao operador humano para tomada de providências.

A Figura 10 apresenta um fluxograma do processo *FluxoPrincipal*, onde as etapas da área de configuração estão presentes.

Figura 10 Fluxograma do Processo *FluxoPrincipal*



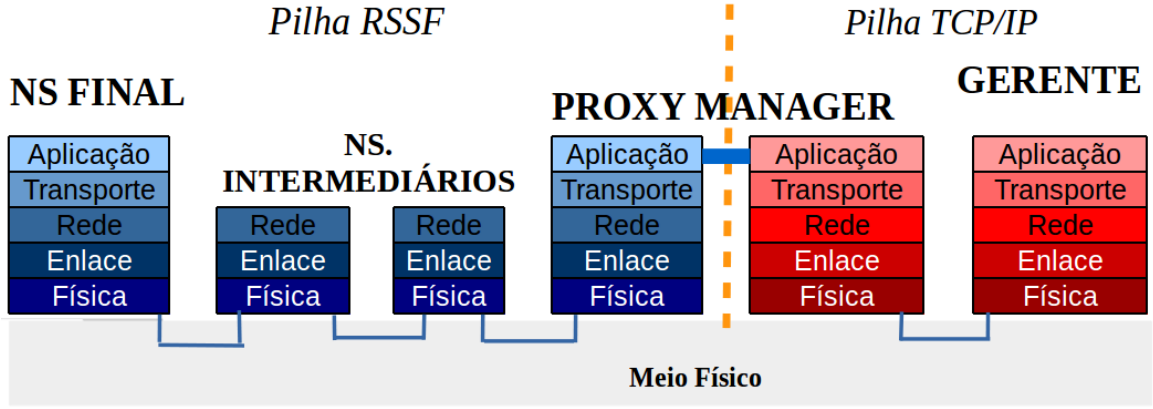
Fonte: elaboração própria

6.1.2 Estabelecimento de Rotas

A estratégia de roteamento utilizada pelo PM é de categoria *flat*, ou seja, os NS desempenham as funções de roteamento e coleta de dados, e pró ativa, onde as rotas são descobertas antes de serem utilizadas. Para estabelecer a comunicação com determinado NS, o PM cria um pacote de solicitação de dados com a rota escrita no mesmo. Um determinado NS sabe se desempenhará a função de coleta de dados ou roteamento tendo como base o endereço de destino final no pacote:

caso este seja igual ao seu, o pacote será processado até a camada de aplicação do NS a fim de realizar a coleta dos dados nos sensores; caso não seja, o processamento será realizado até a camada de rede, onde o processo de roteamento ocorre. A Figura 11 ilustra um exemplo, demonstrando quais camadas são implementadas em cada elemento. Neste exemplo, a comunicação do PM com o NS de destino (Final) é realizado por intermédio de dois NS intermediários.

Figura 11 Arquitetura de camadas propostas (Exemplo)



Fonte: elaboração própria

Cada rota a ser estabelecida pelo PM conterá um custo relacionado a mesma. O PM sempre dará preferência para rotas com custo menor. Para cálculo do custo, o PM levará em consideração dois fatores:

- Número de saltos entre os o Nó Sorvedouro e o NS de destino;
- RSSI (*Received Signal Strength Indicator*) dos enlaces entre o Nó Sorvedouro e o NS de destino;

A RSSI é um indicador de intensidade do sinal recebido em um receptor, sendo dada em dBm. A mesmo possui vantagens de não necessitar de *hardware* adicional para a medição [36], além de oferecer uma estimativa rápida sobre a qualidade do enlace [37][38]. Neste trabalho, a RSSI está sendo utilizada para se obter uma estimativa da qualidade dos enlaces, a fim de utilizar esta no processo de roteamento dos pacotes.

Para estabelecer as rotas, o PM necessita ser parametrizado pelo Gerente de Rede, com os seguintes dados:

- **Limite máximo de saltos permitidos na RSSF (N):** O PM só estabelecerá comunicação com NS que estiverem a uma distância de até N saltos;

- **Quantidade de pacotes enviados para o NS no momento de descoberta de rota (M):** A fim de analisar a RSSI, serão enviados M pacotes para um determinado NS;

- **Custo de um único salto entre os NS (Y):** Rotas com vários saltos tenderão a ter um custo maior, por conta deste parâmetro. A justificativa para tal é que, para atingir um determinado NS com i saltos, serão consumidos os recursos de i-1 NS;

- **Custo limiar de um enlace (K):** Diminui o tempo necessário para descoberta das rotas, conforme seu valor aumenta. Caso uma rota para um NS contenha o menor número de saltos possíveis e enlaces com um custo menor ou igual a K, não serão mais realizadas tentativas de descoberta de rota para o NS, durante o processo de descoberta de rotas em questão;

- **Tabela de classificação dos Enlaces baseada na RSSI:** Irá indicar o custo de um determinado enlace, baseado na média das RSSIs nos pacotes enviados (M);

Os parâmetros informados são armazenados na tabela *severalTable,* e podem ser alterados a qualquer momento através dos *web services* listados na Tabela 5.

Tabela 5 *Web Services* para atualização de parâmetros relacionados a roteamento

Web Service	**Descrição**
updateRouteConfig	Atualiza o limite de saltos (N), o custo de um salto (Y), o número de pacotes a ser enviado para para o NS na descoberta (M)
updateTableLinkRSSI	Atualiza a tabela para classificação de enlaces baseada na RSSI

Fonte: Elaboração própria

O PM descobrirá as rotas para os NS entre 1 e N turnos, sendo N o valor máximo de saltos permitidos na RSSF, a partir do PM. Em um dado Turno t, somente

será realizado tentativas de descoberta de rotas para NS por intermédio de outros NS com rotas já descobertas com t-1 saltos. O PM enviará M pacotes de descoberta para um determinado NS. Dos pacotes recebidos como resposta, será calculada a média da RSSI informada nos mesmos. O PM irá cadastrar a rota para o NS na tabela *nsTable*, utilizando as seguintes informações:

- **Rota:** endereço dos nós intermediários e do NS em questão, sendo este cadastrado por último;
- **Número de Saltos:** número de saltos para o NS em questão. Deve ser sempre menor ou igual a N;
- **Custo:** Será calculado conforme a Equação 1:

$$C = (z - 1) * Y + Ci + Ce \quad (1)$$

Na Equação 1, C é o custo da rota em questão. Y é o custo referente a um único salto. O mesmo é multiplicado por (z – 1), ou seja, a quantidade de saltos (z) menos 1, a fim de encontrar o custo referente aos saltos para o NS. Se a comunicação entre o PM e o NS for direta, ou seja, o número de saltos é igual a 1, este custo não será contabilizado, pois não serão utilizados recursos de nenhum NS intermediário. O custo referente aos saltos é então somado com Ci, o custo da rota do NS intermediário conectado diretamente ao NS de destino, caso este exista (se a comunicação do PM e o NS de destino for direta, este parâmetro não é utilizado). Ao resultado, é somado o valor de Ce, ou seja, o custo do enlace entre o elemento que está conectado diretamente ao NS de destino (seja ele um NS intermediário ou o próprio PM) e o próprio NS de destino. Ce é obtido analisando a média das RSSI obtidas como leitura, comparando a mesma com a Tabela de Classificação de RSSI para Enlaces, a qual é baseada em [39]. Os custos em questão podem ser alterados via *web service*. A Tabela 6 apresenta um exemplo com os respectivos custos. Neste exemplo, é possível observar que enlaces com uma RSSI média maior possuirão um custo exponencialmente menor.

Tabela 6 Exemplo de Tabela para Classificação dos enlaces baseado na RSSI

RSSI	Legenda	Custo
> -50 dBm	Excelente	1
Entre -50 dBm e -59,99 dBm	Ótimo	2
Entre -60 dBm e -69,99 dBm	Bom	4
Entre -70 dBm e -79,99 dBm	Regular	8
Entre -80 dBm e -89,99 dBm	Ruim	16
< -90 dBm	Péssimo	32

Fonte: Adaptado de [39]

É importante salientar que a Equação 1 é empirica, ou seja, foi obtida através de analise de experimentos realizados anteriormente, onde se verificou que rotas com uma taxa maior de erro de pacote estavam associadas ao número maior de saltos e/ou enlaces com RSSI elevadas.

Rotas descobertas no primeiro turno (número de saltos atual igual a 1) terão custo igual a Ce, pois a comunicação entre o PM e o NS de destino é direta. Após o término do primeiro turno, quatro possibilidades podem ocorrer:

- **1º possibilidade:** nenhuma rota foi descoberta;
- **2º possibilidade:** rotas para todos os NS foram descobertas, e o custo das mesmas é igual ou inferior a Ca, obtido através da Equação 2.

$$Ca = (z - 1) * Y + z * K \quad (2)$$

Através da Equação 2, o PM verificará se a rota para o NS de destino é formada por enlaces com um custo menor ou igual a K (Custo Limiar de um Enlace). Conforme citado anteriormente, a utilização deste parâmetro (K) influencia no tempo descorrido para o estabelecimento das rotas. Sendo assim, Ca indicará qual será o peso de uma rota hipotética formada exclusivamente por enlaces com custo menor ou igual a K. Caso este o seja, não serão realizadas mais tentativas de descoberta de rota para o NS em questão. Na primeira parte da equação ((z – 1) * Y) é calculado o peso da rota referente a quantidade de saltos, tendo os mesmos princípios elucidados na

Equação 1. Já a segunda parte (z * K), informará o peso referente ao custo de todos os enlaces da rota. Assim como a Equação 1, a Equação 2 também é empírica.

- **3º possibilidade:** rotas para todos os NS foram descobertas, na qual ao menos uma possui um custo superior à Ca;

- **4º possibilidade:** rotas para um número parcial de NS foram descobertas.

No caso da 1º ou 2º possibilidade, o processo de descoberta de rotas encerrará suas atividades, registrando as ocorrências na tabela de *logs* e alertando o administrador (no caso de nenhum NS for encontrado). No caso da 3º ou 4º possibilidade, o número de saltos atual será incrementado em 1, fazendo o processo entrar no segundo turno. A partir deste, até o último turno, o PM irá realizar tentativas de descoberta de rota por intermédio de todos os NS com um número de saltos igual a z-1, na qual z é o número de saltos atual, para todos os NS que ainda não foram descobertos ou que não possuam rota com custo menor ou igual a Ca. Neste último caso, se uma rota alternativa for encontrada para um NS, esta será substituída caso aquela possua um custo inferior. Caso nenhuma rota seja descoberta em determinado turno, ou o número máximo de saltos for atingido (N), o processo de descoberta de rotas encerrará suas atividades.

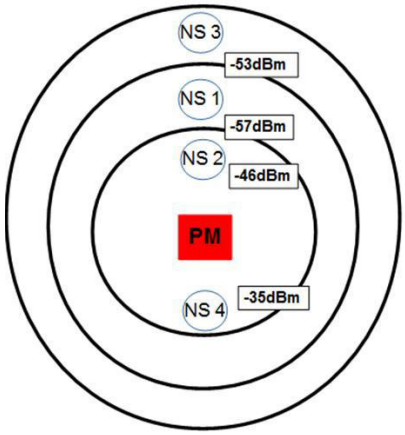
A Tabela 3 exemplifica os resultados obtidos em um cenário ilustrativo e fictício apresentado na Figura 12, onde Y (o peso de um único salto) é igual a 1, os dados para classificação do enlaces baseado na RSSI são os apresentados na Tabela 6, o custo limiar de um enlace é igual a 4 e o número máximo de saltos é igual a 3.

Tabela 7 Exemplo de Cadastro de Rotas

Endereço NS	Rota	Saltos	Custo
1	2,1	2	5
2	2	1	1
3	2,1,3	3	9
4	4	1	1

Fonte: Elaboração própria

Figura 12 Exemplo de RSSF com 4 NSs e a RSSI dos enlaces



Fonte: elaboração própria

O Algoritmo 1 apresenta a o algoritmo do protocolo de roteamento.

Algoritmo 1 Descoberta de rotas

Entrada: Endereço dos NS; Limite máximo de saltos permitidos (N); Quantidade de pacotes enviados para cada NS na descoberta de rota (M); Custo de um salto entre NS (Y); Custo limiar de um enlace (K), Tabela de classificação dos enlaces baseado na RSSI (Ce)
Saída: Rotas para os NS

```
    Inicio:
01:  num_salto_atual = 0
02:  while (num_salto_atual) <= N OR (Não descobriu todas as rotas) do
03:      num_salto_atual  =  num_salto_atual  + 1
04:     if  num_salto_atual  == 1 then
05:       for i = 1 to (Número total de NS) do
06:         enviar M pacotes para o iº NS
07:         if (iº NS respondeu) then
08:           calcular custo para rota (Custo = Ce)
09:           registrar rota
10:       end if
11:     end for
12:   else
13:     for i = 1 to (Número de NS sem rotas OU
        rotas com custo menor do que Ca = (actual_hop – 1) * Y + actual_hop * K)  do
14:       for x = 1 to (NS com rotas encontradas E número de saltos = num_salto_atual – 1)
15:         enviar M pacotes para o iº NS através do xº NS
16:         if (iº NS respondeu) then
17:           calcular custo para a rota (Custo = ( num_salto_atual – 1) * Y + Ci + Ce)
18:           registrar rota
19:  end while
```

6.2 Gerenciamento de Configuração de Dados

O objetivo da área de gerenciamento de configuração de dados é inicializar e manter os sensores, responsáveis pela coleta das grandezas do meio ambiente. A função implementada para o gerenciamento de configuração de dados é o cadastramento e manutenção dos sensores. Estas atividades são realizadas através

dos *web services* listados na Tabela 8.

Tabela 8 *Web Services* para cadastro e alteração dos Sensores

Web Service	**Descrição**
insertSensorNS	Cadastrada um sensor para um determinado NS
alterSensorNS	Altera os dados de um determinado sensor
excludeSensorNS	Realiza a exclusão de determinado sensor
alterStatusSensorNS	Altera o status de determinado sensor entre Ativado e Desativado.

Fonte: Elaboração própria

Um determinado NS pode conter um ou mais sensores. Quando o PM envia um pacote de solicitação de dados para um NS, este coleta dados de todos os seus sensores e os envia em pacote de resposta. Ao realizar o cadastro do sensor, é possível informar em que posição do pacote de resposta a parte inteira e fracionada do dado coletado está armazenada, sendo que estes podem variar de acordo com o NS.

Os dados coletados são armazenados no Banco de Dados, e são excluídos após um período específico. O objetivo disto é não exaurir toda a capacidade de armazenamento do PM, já que isto pode acarretar na paralização da execução de seus processos.

6.3 Gerenciamento de Desempenho da Infraestrutura

O desempenho da RSSF é realizado com a monitoração da PER para cada NS. O processo do módulo Gerente Local que realiza esta atividade, denominado *PERAvaliador*, é executado paralelamente ao processo *FluxoPrincipal*. Os NS avaliados pelo *PERAvaliador* possuem o *status* OP, IN ou EA. NS com *status* NR ou PR não são analisados pelo *PERAvaliator*, pois necessitam de intervenção do operador humano. Além disso, a PER sempre será referente ao Período da PER, a qual é único para cada NS. O motivo disso é que cada NS pode ter uma tolerância diferenciada para falhas, em comparação com os demais.

A PER calculada é também registrada no Banco de Dados em formato de *log*,

para disponibilização via *web service*, no módulo Informante.

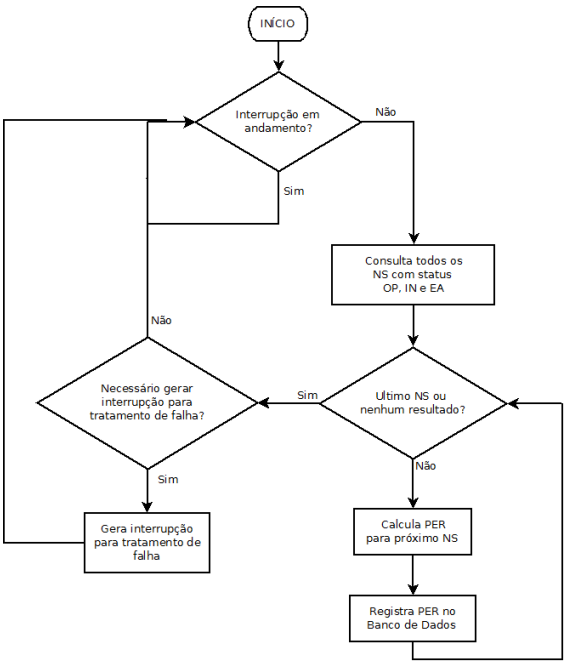
6.4 Gerenciamento de Falhas da Infraestrutura

Quando a PER atual de determinado NS ultrapassa a PER Tolerável no Período da PER, o processo *PERAvaliador* identifica uma falha. O *status* do NS será então alterado para IN. O tratamento da falha será realizado através de uma interrupção na coleta de dados. A área de configuração é então acionada a fim de tentar encontrar uma rota alternativa, ou, em caso negativo, alertar o operador humano do sistema.

Caso uma rota alternativa seja encontrada para o NS, o *status* do NS será alterado para EA, e ficará assim por um período igual ao Período da PER. No final deste, caso a sua PER atual seja igual ou inferior a PER Tolerável, seu *status* é alterado para OP. Caso contrário, o mesmo será alterado para PR e o administrador será alertado.

A Figura 13 apresenta um fluxograma com as etapas do processo *PERAvaliador,* responsável pela área de gerenciamento de desempenho e falhas da infraestrutura.

Figura 13 Fluxograma do processo *PERAvaliator*



Fonte: elaboração própria

Já o Algoritmo 2 apresenta o algoritmo do processo *PERAvaliador.*

Algoritmo 2 Processo *PERAvaliador*

Entrada: PER Máxima Tolerável (PMT) e Período da PER (PP) de todos os Nó Sensores com *status* OK, IN e EA

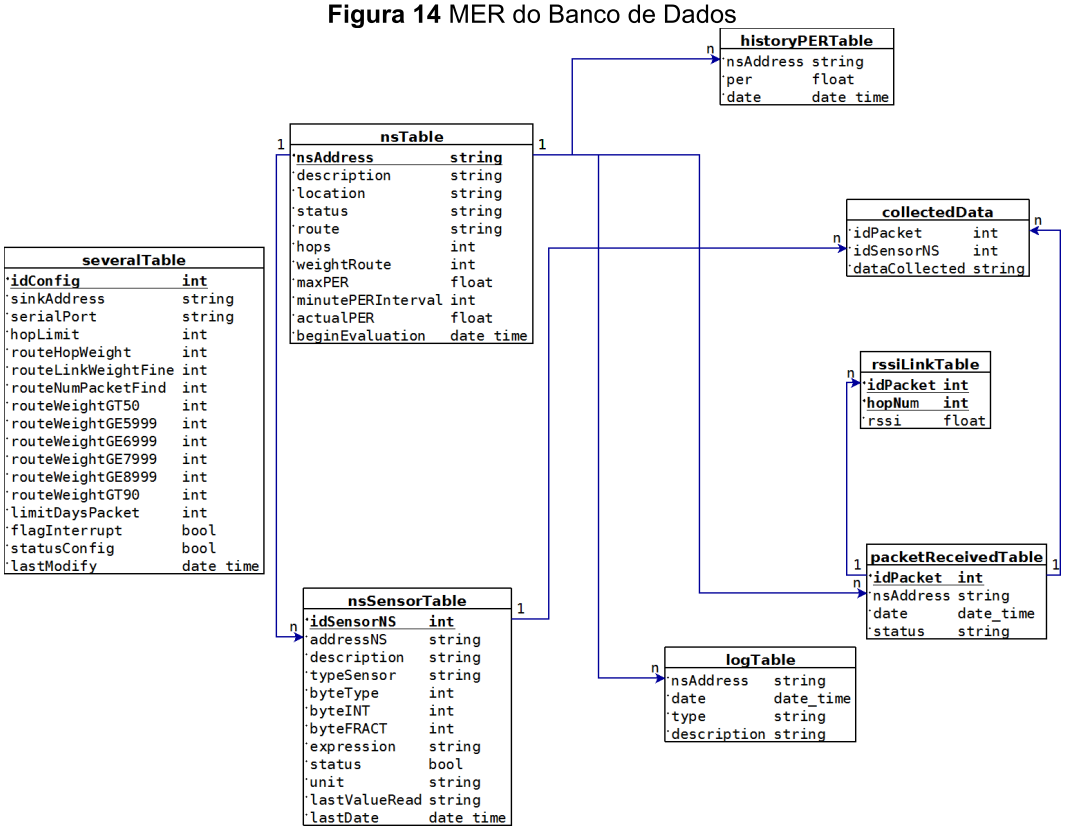
Inicio:

```
1:  for i = 0 to Número de NS com status OK, IN and EA do
2:    PER = (Calcular PER para o NS nº i)
3:    if PER > PMT then
4:      if (status do NS n º i == "EA") AND (Fim do Periodo da PER) then
5:        status do NS nº i = "PR"
6:        Alertar administrador
7:      else:
8:        Encontrar rota alternative para NS nº i
9:        if (rota alternative encontrada) then
10:         status do NS nº i = "EA"
11:       else
12:         status do NS nº i = "PR"
13:         alertar administrador
```

6.5 Estrutura do Banco de Dados

A Figura 14 apresenta o MER (Modelo Entidade Relacionamento ou *Entity Relationship Model*) do Banco de Dados. O MER tem como objetivo apresentar as entidades (tabelas) e seus atributos, bem como a relação entre os mesmos [40].

Figura 14 MER do Banco de Dados



Fonte: elaboração própria

A seguir será dada uma descrição da função de todas as tabelas (entidades), que compõe o Banco de Dados:

- ***severalTable*:** Armazena configurações e parâmetros gerais, tais como o endereço do Nó Sorvedouro (*sinkAddress*), o limite de saltos permitido na RSSF (*hopLimit*), a porta serial utilizada para conexão com o Nó Sorvedouro (*serialPort*), parâmetros relacionados ao estabelecimento de rotas e o limite de tempo que os dados coletados ficarão armazenados no Banco de Dados;

- ***nsTable*:** Armazena dados relacionados aos NS, como endereço (*nsAddress*), descrição e localização (*description* e *location*) o estado atual do mesmo (*status*), a rota atual cadastrada (*route*), a quantidade de saltos (*hops*), o peso da rota (*weightRoute*), sua PER atual (*actualPER*), a PER Máxima Toleravel (*maxPER*) e a quantidade de minutos utilizado para calcular o Período da PER (*minutePERInterval*);

- ***nsSensorTable*:** Armazena os dados relacionados a um determinado sensor. Um NS pode possuir um ou mais sensores cadastrados. Entre os dados armazenados, estão uma descrição (*description*), um tipo (*typeSensor*), qual a posição que o dado coletado está no pacote enviado pelo NS (*byteType, byteINT* e *byteFRACT*), qual expressão utilizar para calcular o valor coletado (*expression*), o estado atual do sensor (*status*) e o último valor lido e seu respectivo horário (*lastValueRead* e *lastDate*);

- ***packetReceivedTable*:** Armazena dados relacionados ao pacotes solicitados. Os pacotes estão vinculados a um determinado NS. O campo *status* indica se o pacote foi ou não recebido com sucesso, já o campo *date* indica a data e horário de solicitação;

- ***collectedData*:** Armazena os dados coletados pelos sensores, caso o pacote tenha sido recebido com sucesso. Um pacote recebido pode conter dados de um ou mais sensores;

- ***rssiLinkTable*:** Armazena a RSSI dos enlaces entre o Nó Sorvedouro e

o NS de destino final, informados no pacote recebido. O campo *hopNum* informa qual é o número do salto, e o campo *rssi* indica o valor correspondente para o mesmo;

- ***historyPERTable*:** Armazena o histórico da PER, relacionada a determinado NS;

- ***logTable*:** Armazena os registros ocorridos no decorrer do tempo, relacionados tanto ao PM quanto aos NS. Exemplos de registros são: inicio e finalização de descoberta de rotas, porta serial não identificada, NS não responde, etc;

7 MATERIAL E MÉTODOS

Para implementar o PM e a RSSF para o teste da plataforma, foram utilizados *hardware* e *software* de códigos aberto (*open-hardware* e *open-software*) e de baixo custo, a fim de que a proposta fosse viável financeiramente e que novas pesquisas e funcionalidades pudessem ser desenvolvidas futuramente através de outros projetos.

Primeiramente, será detalhado tanto o *hardware* quanto os *softwares* utilizados para a implantação do PM. A seguir, será realizado o mesmo para a RSSF (Nó Sorvedouro e NS).

7.1 Proxy Manager

Para a escolha do *hardware* do PM, os seguintes aspectos e configurações foram levados em consideração:

- Tamanho físico;
- Capacidade de Processamento;
- Capacidade da Memória RAM;
- Capacidade de Armazenamento interno;
- Custo;
- Conectividade via USB e Ethernet;

Na época de desenvolvimento deste trabalho (Janeiro de 2016), uma pesquisa foi realizada para levantar as principais opções de plataformas computacionais com tamanho físico reduzido, voltados para atuar em aplicações de Internet das Coisas [41]. Os mesmos são: *Arduino UNO* [42]*, BeagleBone Black* [43] e *Raspberry Pi* [44]. A Tabela 9 apresenta um comparativo entre estes computadores, levando em consideração os aspectos listados a cima.

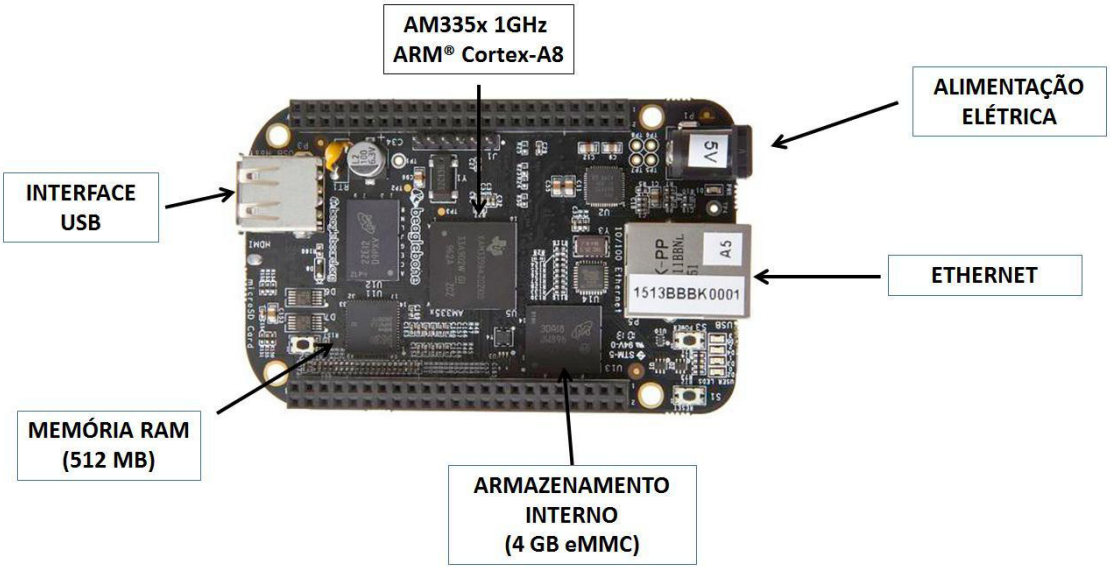
Tabela 9 Comparativo entre computadores de tamanho reduzido

Computador	Tamanho	Processador	Memória RAM	Armazenamento Interno	Custo	USB	Ethernet
Arduino UNO	68.6 x 53.4 mm	ATmega328P	2Kb	32KB	$24.95	1 *host*	Não possui
BeagleBone Black	86.3 x 53.3 mm	AM335x 1GHz ARM® Cortex-A8	512 MB	4 GB	$49	1 *host*	10/100 Mbps
Raspberry Pi	85.6 x 53.98 mm	ARM BCM2835	256 MB	Não possui	$35	2 *hosts*	10/100 Mbps

Fonte: Elaboração própria

Devido ao melhor poder computacional e capacidade de armazenamento interno, optou-se por utilizar a plataforma *Beaglebone Black* para hospedar os processos do PM. O preço do mesmo se justifica quando se é levado em consideração as melhores especificações de *hardware*. A Figura 15 apresenta uma foto do *Beaglebone Black*.

Figura 15 *Beaglebone Black*



Fonte: Adaptado de [43]

Em relação ao sistema operacional, optou-se pelo *GNU/Linux Debian* 8 (*jessie*), por ser a distribuição padrão do *Beaglebone Black,* o qual oferece um maior suporte e documentação.

Para a escolha do SGBD (Sistema Gerenciador de Banco de Dados), o qual irá gerir o Banco de Dados do sistema, a escolha se deu entre sistemas *open source* e que pudessem suportar requisições simultâneas (feitas através dos Módulos Gerente Local e Informante). A escolha ficou entre o *MySQL* e o *PostgreSQL*. O

banco de dados *sqlite*, apesar de sua enorme eficiência comprovada, não atende requisições simultâneas, conforme é mencionado em seu site oficial [45].

Em [46], é realizado uma comparação entre os dois SGBDs, ao qual é constatado que o *MySQL* apresentou melhores resultados na maioria dos testes. Por isso, optou-se pelo mesmo. O código fonte implementado para o Banco de Dados é apresentado por completo nos Anexos (Tópico 11.2).

Para o desenvolvimento do módulo Informante, optou-se novamente por linguagens de código aberto, e que tivesse o melhor desempenho em relação a *web services*. Em [47], uma comparação é realizada entre as *engines* de *web services* nas linguagens PHP, Java e C. A linguagem PHP apresentou um desempenho satisfatório, aliado a uma alta produtividade no desenvolvimento, em comparação com as outras alternativas. O desempenho de *scripts* PHP é similar ou em alguns casos maior do que *servlets* Java, além de possuir um menor número de linhas de código, conforme pode ser constatado em [48] e [49]. O código fonte implementado para o módulo Informante é apresentado por completo nos Anexos (Tópico 11.3).

Para o desenvolvimento do módulo Gerente Local, optou-se pela linguagem *Python* [50]. *Python* é uma linguagem *open-source,* interpretada*,* que oferece suporte a orientação a objetos e de tipagem dinâmica, com bibliotecas disponíveis para as mais diversas necessidades. Por ser uma linguagem interpretada, *Python* oferece um desempenho menor em termos de processamento do que linguagens compiladas, como C ou C++. No entanto, esta perda é balanceada pela maior produtividade dada ao desenvolvedor. O código fonte implementado para o módulo Gerente Local é apresentado por completo nos Anexos (Tópico 11.1).

A comunicação entre o PM e o Nó Sorvedouro será realizada através de uma interface USB (*Universal Serial Bus*), disponível na plataforma *Beagle Bone Black*.

7.2 Nó Sorvedouro e Nós Sensores

A implementação do Nó Sorvedouro e dos NS se deram de duas formas: nos primeiros testes, referente a funcionalidade do protocolo de roteamento, foram utilizados elementos de *hardware* e *software* reais. Para testar as funcionalidades

das áreas de gerenciamento de desempenho e falhas, no entanto, a estratégia foi modificada.

Para testar tais áreas, o planejamento inicial era dispor os NS em uma dada área, de forma que a distância e obstáculos físicos entre os mesmos refletissem atenuações de sinal desejadas, propiciando rotas com mais de um salto entre os NS. Para testar a funcionalidade de manutenção das rotas, mudanças no cenário também deveriam ser realizadas, de forma que a RSSI lida nos enlaces se deteriorasse a um ponto que refletisse em PER's maiores do que a PER Máxima Tolerável configurada para os NS. No entanto, foram encontradas dificuldades na configuração de cenários que refletissem tais objetivos. Isto se deve ao fato de que redes *wireless* são altamente probabilísticas, refletindo em um comportamento imprevisível. A fim de facilitar o estabelecimento de tais cenários para testar as funcionalidades das áreas de gerenciamento de desempenho e falhas da infraestrutura, ou seja, o estabelecimento e manutenção das rotas, optou-se pelo desenvolvimento de um algoritmo de emulação de RSSF na plataforma Arduino [42].

As duas implementações em questão (a implementação física e a emulada) são apresentadas nos próximos tópicos (7.2.1 e 7.2.2).

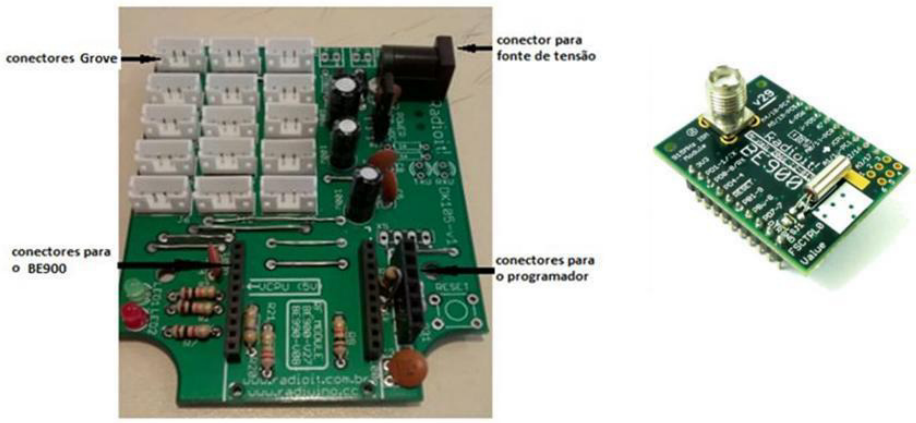
7.2.1 Implementação Física

A implantação do Nó Sorvedouro e dos NS, tanto a nível de *hardware* quanto de *software*, foi feita através da plataforma *Radiuino* [51]. O *Radiuino* é uma plataforma *open-source* para a criação de RSSF baseada na consolidada plataforma *Arduino* [42], o qual integra este último a um transceptor CC1101 da *Texas Instruments* e um microcontrolador *Atmega 328* da *Atmel.* O *firmware* utilizado pelo protocolo da plataforma *Radiuino* possui uma estrutura de 5 camadas (Aplicação, Transporte, Rede, Enlace e Física).

Para os NS, foram utilizadas as placas do kit DK-105 *Radiuino*, o qual está preparado para lidar com sensores da plataforma Grove [52]. A plataforma Grove leva em consideração a padronização de conectores para sensores, sendo indicada para sistemas de prototipagem. Diversos tipos de sensores estão disponíveis para a plataforma, tais como temperatura, luminosidade, gás, umidade de solo, etc.

O módulo de rádio utilizado tanto nos NS quanto no Nó Sorvedouro é o BE-900, que opera na faixa não licenciada ISM (*Industrial, Scientific and Medical*) de 915 Mhz, com modulação 2-FSK [51]. A Figura 16 apresenta o modelo DK-105, bem como o módulo de rádio BE-900.

Figura 16 Kit *Radiuino* DK-101 e BE-900



Fonte: [51]

7.2.1 Implementação por Emulação

O algoritmo de emulação de RSSF desenvolvido na plataforma Arduino leva em consideração a probabilidade de perda de pacote em um dado enlace (composto por um NS de origem e outro de destino), tendo como parâmetro um determinado intervalo de RSSI esperado neste. Desta forma, caso seja necessário emular uma deterioração na qualidade do final (refletida posteriormente em um aumento da PER referente a um determinado NS), basta modificar via *software* o intervalo de RSSI esperada em um ou vários enlaces. Os intervalos de RSSI implementados no algoritmo em questão são os mesmos apresentados na Tabela 6. As PERs relacionadas as RSSIs em questão foram estimadas de acordo com o seguinte procedimento: primeiramente, deve-se derivar uma dada RSSI para Eb/N0 (*Energy per Bit to Noise power spectral density ratio* - relação entre a energia do bit e o ruído térmico AWGN (*Additive White Gaussian Noise*)) através da Equação 3 (procedimento este demonstrado em [53]):

$$E_b/N_0 = \frac{RSSI}{6} + 28,7 \quad (3)$$

Através de Eb/N0, é possível estimar a BER (*Bit Error Rate*) e, consequentemente, a PER. Para estimativa da PER através da BER, foi considerado um comprimento de pacote de 512 bits (N), tamanho este utilizado na plataforma Radiuino. A Equação 4 apresenta a expressão utilizada para obter a BER a partir de Eb/N0 [4].

$$BER = \frac{1}{2} e^{-(E_b/N_0)/2} \quad (4)$$

Já a Equação 5 [4] é utilizada para a PER a partir da BER.

$$PER = 1 - (1 - BER)^N \quad (5)$$

Sendo assim, a PER média para os intervalos de RSSI utilizados durante os testes são apresentados na Tabela 10.

Tabela 10 PER média dos intervalos de RSSI utilizados

RSSI	PER Média
Entre > -40 dBm e -49,99 dBm	0,6%
Entre -50 dBm e -59,99 dBm	1,4%
Entre -60 dBm e -69,99 dBm	3,3%
Entre -70 dBm e -79,99 dBm	7,5%
Entre -80 dBm e -89,99 dBm	16,4%
Entre -90 dBm e -99,99 dBm	33,3%

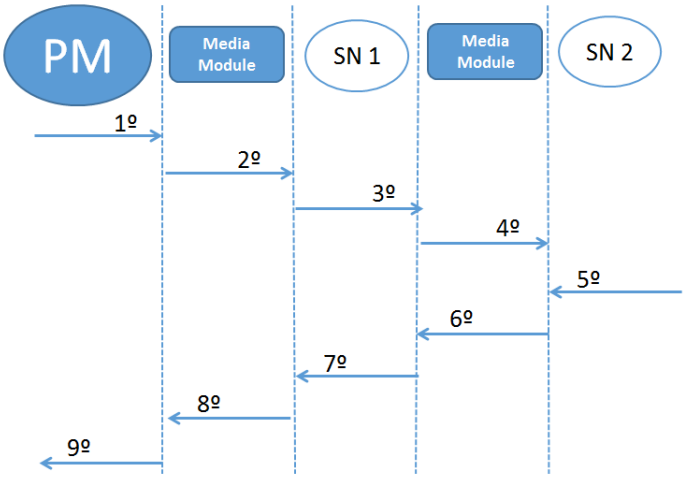
Fonte: Elaboração própria

O algoritmo de emulação de RSSF implementado possui dois tipos de módulo: o primeiro emula um NS, e o segundo o meio físico. O primeiro módulo, denominado Módulo NS, emula um determinado NS, o qual obedece a estrutura de cinco camadas da plataforma Radiuino. O segundo módulo, denominado Módulo de Meio Físico calcula a probabilidade (através da função *rand*) de perca de pacote em determinado enlace da RSSF.

A Figura 17 apresenta um diagrama temporal ilustrando a comunicação entre os módulos utilizados no algoritmo de emulação. Nela, o PM tenta estabelecer comunicação com o NS de endereço 2, sendo que a comunicação é intermediada pelo NS de endereço 1. É possível observar que o Módulo de Meio Físico sempre

intercepta o pacote a ser transmitido, a fim de calcular a probabilidade de perda de pacote nos enlaces utilizados. Caso a probabilidade de perda de pacote seja positiva, o módulo de meio físico descarta o mesmo.

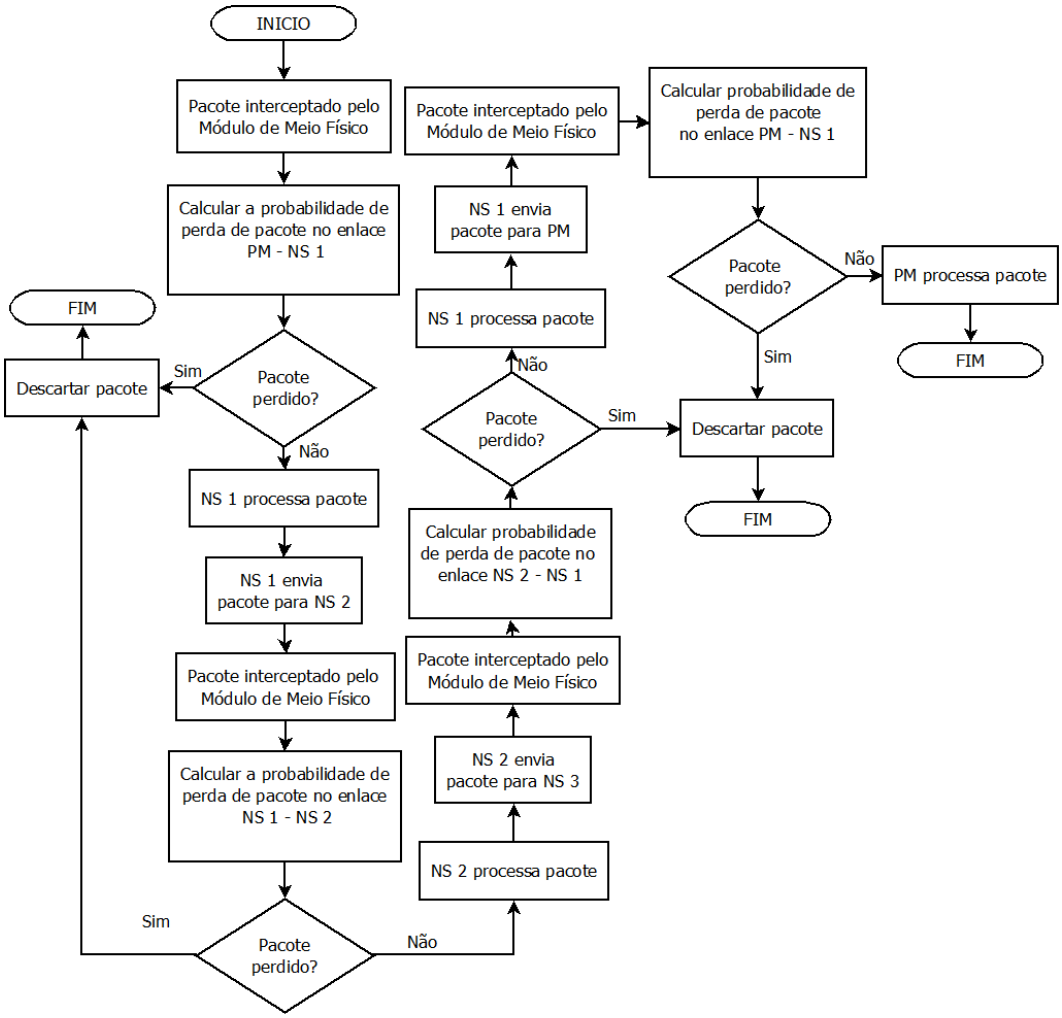
Figura 17 Diagrama Temporal de exemplo de comunicação dos módulos no algoritmo de emulação da RSSF



Fonte: Elaboração própria

A Figura 18 apresenta um fluxograma do algoritmo de emulação da RSSF implementado, utilizando como exemplo o mesmo caso apresentado na Figura 16. O código fonte desenvolvido para o algoritmo de emulação é apresentado por completo nos Anexos (Tópico 11.4).

Figura 18 Fluxograma do Algoritmo de Emulação da RSSF



Fonte: Elaboração própria

7.3 Pacote

A plataforma *Radiuino* define um pacote contendo 52 bytes, os quais são divididos em cinco camadas [51], da seguinte forma:

- Física (4 bytes);
- MAC (*Media Access Control*) (4 bytes);
- Rede (4 bytes);
- Transporte (4 bytes);
- Aplicação (36 bytes) (carga útil, contendo 6 AD (*Analogic Digital*) e 6 IO

(*Input / Output*);

O mapa do pacote original do *Radiuino* é ilustrado na Figura 19. Na mesma, é possível observar a quantidade de *bytes* disponíveis para cada camada.

Figura 19 Mapa de pacote original do *Radiuino*

FÍSICA				MAC				REDE				TRANSPORTE					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
RSSIDLink	LQIDLink	RSSIULink	LQIULink	TBD	TBD	TBD	TBD	End. Dest.	End. Nwk Dest.	End. Orig.	End. Nwk Orig.	Cont	TBD	TBD	TBD		
APLICAÇÃO																	
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
AD0_W	AD0_H	AD0_L	AD1_W	AD1_H	AD1_L	AD2_W	AD2_H	AD2_L	AD3_W	AD3_H	AD3_L	AD4_W	AD4_H	AD4_L	AD5_W	AD5_H	AD5_L
34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
IO0_W	IO0_H	IO0_L	IO1_W	IO1_H	IO1_L	IO2_W	IO2_H	IO2_L	IO3_W	IO3_H	IO3_L	IO4_W	IO4_H	IO4_L	IO5_W	IO5_H	IO5_L

Fonte: Adaptado de [51]

O mapa do pacote utilizado na proposta em questão apresenta algumas diferenças em relação ao mapa original. A Figura 20 apresenta o mapa de pacotes modificado.

Figura 20 Mapa de pacote modificado para a proposta

FÍSICA						MAC									
0	1	2	3	4	5	6	7	8	9						
RSSIDLink1	RSSIDLink2	RSSIDLink3	RSSIDLink4	RSSIDLink5	RSSIDLink6	TBD	TBD	TBD	TBD						
REDE															
10	11	12	13	14	15	16	17	18	19	20	21	22	23		
End. Dest.	TBD	End. Orig.	Tipo de Pacote	End. Descoberta	Máximo de saltos	Num. Saltos atuais	End. Nó Descobridor	Rota pt 1	Rota pt 2	Rota pt 3	Rota pt 4	Rota pt 5	Rota pt 6		
TRANSPORTE				APLICAÇÃO											
24	25	26	27	28	29	30	31	32	33	34	35	36	37		
Cont	TBD	TBD	TBD	AD0_W	AD0_H	AD0_L	AD1_W	AD1_H	AD1_L	AD2_W	AD2_H	AD2_L	AD3_W		
APLICAÇÃO															
38	39	40	41	42	43	44	45	46	47	48	49	50	51		
AD3_H	AD3_L	IO1_W	IO1_H	IO1_L	IO2_W	IO2_H	IO2_L	IO2_W	IO2_H	IO2_L	IO3_W	IO3_H	IO3_L		

Fonte: Elaboração própria

Na implementação deste projeto, o limite de saltos tolerado na RSSF é seis. Desta forma, os campos reservados para a Camada de Transporte e MAC poderam ser mantidos, para poderem ser utilizados em trabalhos futuros. A RSSI de todos os seis possíveis *links* por onde o pacote será ser roteado é registrado no cabeçalho da camada física.

No cabeçalho da camada de rede, o qual foi modificado para atender os requisitos de roteamento da plataforma, os seguintes campos foram criados:

- **Tipo de pacote**: O pacote pode indicar uma das quatro opções listados na Tabela 11.

Tabela 11 Tipos de pacotes possíveis

Valor	Descrição
1	Solicitação de descoberta de rota
2	Resposta positiva de descoberta de rota
3	Solicitação de coleta de dados
4	Resposta positiva de coleta de dados

Fonte: Elaboração própria

- **Endereço de descoberta:** Caso o pacote seja do tipo 1 (Solicitação de descoberta de rota), o endereço do NS a ser encontrado é informado neste campo;

- **Máximo de saltos**: Indica quantos saltos o pacote poderá fazer na RSSF, através de NS intermediários;

- **Número de saltos atuais:** A cada salto, o NS intermediário incrementará este campo. Quando o número de saltos atuais for igual ao número máximo de saltos informado no pacote, o NS sabe que necessita retornar o mesmo para o PM ou descartar o pacote;

- **Endereço do Nó Descobridor:** Endereço do NS que descobriu o NS apontado no campo Endereço de descoberta, caso o pacote seja de descoberta de rota;

- **Rota pt 1 – Rota pt 6:** Informa o endereço dos possíveis 6 NS que irão rotear o pacote até o NS final;

8 TESTES E RESULTADOS

Para testar a proposta implementada, optou-se por utilizar como norteador a norma NBR ISO/IEC 9126 [54], da ABNT (Associação Brasileira de Normas Técnicas), a qual estabelece um modelo de qualidade para produtos de *software.* A norma em questão lida com 6 características de qualidade, apresentados na Tabela 12.

Tabela 12 Características de qualidade da norma NBR ISO/IEC 9126

Característica	Descrição
Funcionalidade	Capacidade do software em fornecer as funcionalidades definidas, bem como resultados corretos e aspectos de segurança.
Confiabilidade	Capacidade do software em manter sua execução e um nível de performance em caso de falhas, bem como se recuperar das mesmas
Usabilidade	Capacidade do software em ser entendido e usado, bem como sua atratividade para o usuário.
Eficiência	Capacidade do software em manter um comportamento apropriado em relação a tempo e recursos de hardware.
Manutenibilidade	Capacidade de permitir manutenção em suas funcionalidades.
Portabilidade	Capacidade do software em ser instalado, adaptável a ambientes diferentes e coexistir com outros softwares no mesmo ambiente.

Fonte: Adaptado de [54]

Tendo as características da norma como base, optou-se pela realização de testes que demonstrassem a funcionalidade do PM (a capacidade de executar as funcionalidades propostas), bem como a eficiência da mesmo em relação ao *hardware*, já que este possui recursos mais modestos (CPU e Memória Principal), se comparado a *Desktops* ou Servidores.

Sendo assim, optou-se por utilizar as áreas de Funcionalidade e Eficiência, através dos seguintes testes:

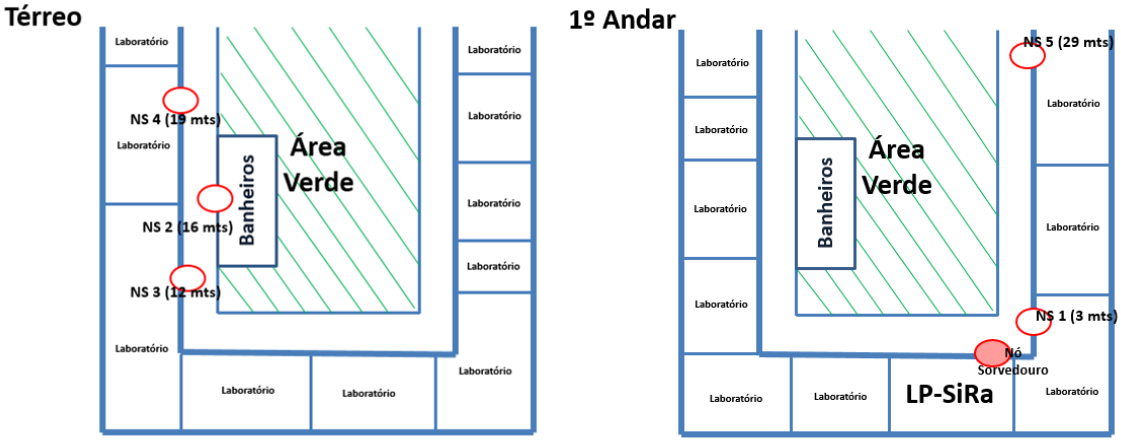
- **Funcionalidade:** Adequação do protocolo de roteamento; Adequação da funcionalidade de gerenciamento de desempenho e falhas; Adequação da funcionalidade de coleta de dados de sensores;

- **Eficiência:** Avaliação em relação ao tempo de resposta de requisições de dados via *web service,* Avaliação em relação a utilização de recursos de *hardware* (utilização da CPU e da Memória Principal);

8.1 Adequação do Protocolo de roteamento

Para testar o protocolo de roteamento desenvolvido para o PM, foi instalada uma RSSF composta por 5 NS utilizando a plataforma Radiuino (Modelo DK-105), no conjunto laboratorial CEATEC da Pontifícia Universidade Católica de Campinas. A disposição espacial dos NS em relação ao Nó Sorvedouro e PM é ilustrada na Figura 21. Todos os NS estavam a uma distância de aproximadamente 2,5 metros do chão. A distância em linha visada dos NS ao Nó Sorvedouro também é apresentada na Tabela 13.

Figura 21 Disposição espacial dos NS nos testes preliminares



Fonte: Elaboração própria

Tabela 13 Distância (linha visada) entre NS e Nó Sorvedouro

NS	Distância (em metros)
1	3
2	16
3	12
4	19
5	29

Fonte: Elaboração própria

Os endereços dos NS foram, respectivamente, 1, 2 , 3 , 4 e 5. A PER Máxima Tolerável, bem como o Período da PER não foram considerados, pois o teste em questão era referente apenas ao protocolo de roteamento. Os NS foram dispostos de maneira a permitir que uma rede de múltiplos saltos fosse estabelecida, sendo que alguns estavam através de paredes de concreto (NS 2 e 4) e longa distância (NS 5), em relação ao Nó Sorvedouro.

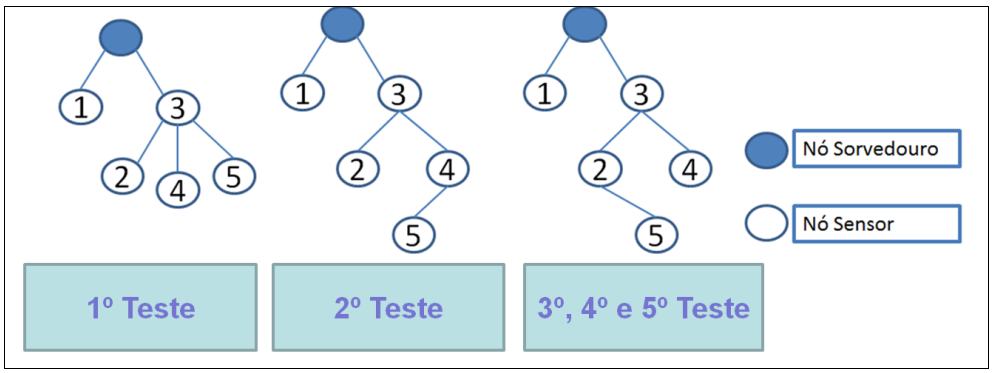
Os parâmetros informados via *web service* referente ao roteamento foram:

- **Limite máximo de saltos permitidos na RSSF:** devido a quantidade de NS utilizadas durante os testes, o número de saltos foi estabelecido como 5;
- **Custo limiar para enlaces:** a fim de tentar obter rotas com enlaces com uma RSSI considerada no minímo Boa [39], o que implicaria em uma PER média de 1,6%, considerada satisfatória para uma RSSF [55], o custo limiar para os enlaces foi configurado como 2;
- **Custo de um único salto:** cada NS que atuar como roteador acrescentará 1 ao custo da rota;
- **Tabela de Classificação de RSSI para enlaces:** para classificação dos enlaces, foram utilizados os dados contidos na Tabela 6. Os custos da mesma foram dispostos de forma a priorizar enlaces com RSSI maior;

A fim de demonstrar a execução, foram realizados cinco processos consecutivos de descoberta de rotas. Após cada teste, foram enviados mil pacotes de solicitação de dados para cada NS, a fim de medir a PER. O número de mil pacotes foi escolhido, pois em testes preliminares os resultados se mostraram estáveis quando chegaram próximo a este número.

A Figura 22 apresenta a árvore de escoamento obtida nos testes.

Figura 22 Árvores de escoamento obtidas nos testes

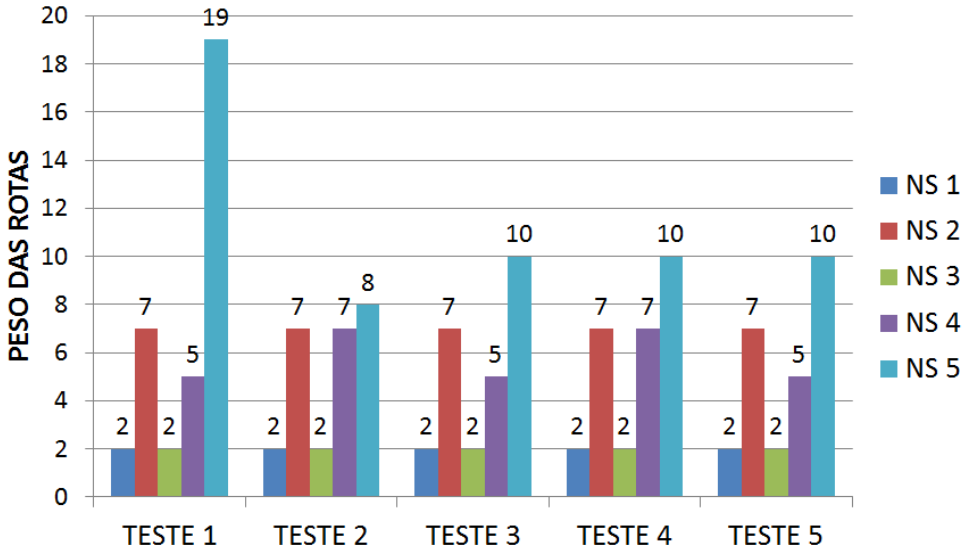


Fonte: Elaboração própria

Analisando as árvores de escoamento obtidas, é possível verificar que os NS que estavam a uma distância muito grande ou atrás de paredes de concreto foram descobertos através do processo de roteamento.

A Figura 23 apresenta os custos das rotas encontradas para os cinco NS nos cinco testes realizados. Os custos mais altos (NS 2, 4 e 5) refletem o maior número de saltos utilizados para atingir os mesmos.

Figura 23 Custos das Rotas encontradas



Fonte: Elaboração própria

A PER foi calculada em cada um dos testes obtidos, para cada um dos NS. A Tabela 14 apresenta estes resultados.

Tabela 14 Resultado de PERs nos testes executados

NS	TESTES				
	1º	2º	3º	4º	5º
1	0,30%	0,60%	0%	0,20%	0,10%
2	0,60%	0,50%	1,20%	1%	1%
3	0%	0,20%	0,10%	0,10%	0,20%
4	20,10%	26,50%	15,90%	33,60%	0,40%
5	10,70%	25,70%	2,60%	2%	2,40%

Fonte: Elaboração própria

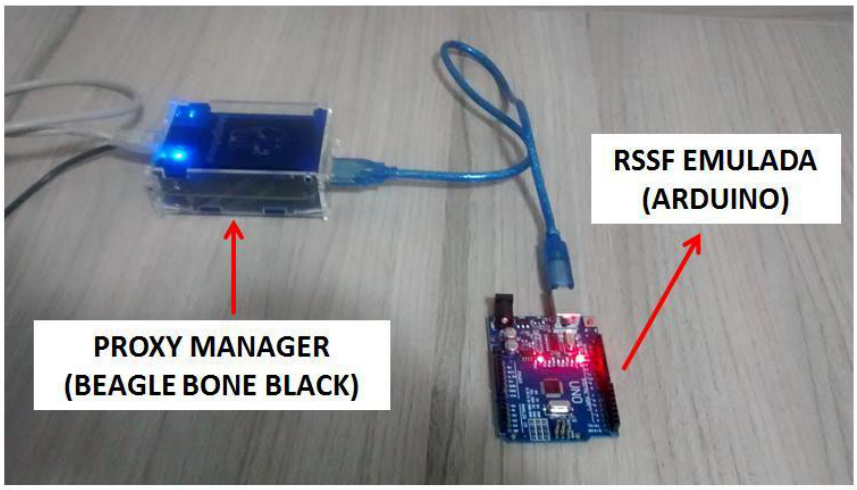
É importante salientar que, no momento em que estes testes foram realizados, o processo *PERAvaliador* não estava em execução. Caso contrário, os dados obtidos poderiam ser utilizados pelo mesmo para tratamento de falhas, identificadas através de uma PER relativamente alta. No mais, o processo de

estabelecimento de rotas se mostrou funcional.

8.2 Adequação do processo de Gerenciamento de Falhas e Desempenho

Para testar a adequação do processo de Gerenciamento de Falhas e Desempenho (através do processo *PERAvaliador*), conforme discutido no Tópico 7.2, fez-se uso do algoritmo de emulação de RSSF. Para tal, 5 Módulos NS foram implementados, sendo 1,2,3,4 e 5 os respectivos endereços de cada NS. Este foi processado na plataforma Arduino UNO. A comunicação deste com o PM foi realizado através de uma interface USB (*Universal Serial Bus*). A Figura 24 apresenta estes dois elementos conectados durante o experimento.

Figura 24 Beagle Bone Black e Arduino conectados



Fonte: Elaboração própria

Para testar a manutenção das rotas com base na PER, foram estabelecidos três cenários diferentes no algoritmo de emulação da RSSF. No momento inicial dos testes, um dos cenários era escolhido pela plataforma. A cada 3000 pacotes, o algoritmo escolhia um cenário diferente. Este valor foi escolhido para que a PER calculada pudesse se estabilizar antes de uma nova mudança de cenário. A probabilidade de perda de pacote estimada em cada um dos enlaces possíveis da RSSF (sendo cada enlace composto por um NS de origem e outro de destino), para cada um dos testes implementados, são apresentados na Tabela 15, 16 e 17.

Tabela 15 Probabilidade de Perda de Pacotes no Cenário 1

	PM	NS 1	NS 2	NS 3	NS 4	NS 5
PM		0,6%	0,6%	16,4%	33,3%	33,3%
NS 1	0,6%		7,5%	0,6%	7,5%	16,4%
NS 2	0,6%	7,5%		3,3%	0,6%	0,6%
NS 3	16,4%	0,6%	3,3%		7,5%	0,6%
NS 4	33,3%	7,5%	0,6%	7,5%		33,3%
NS 5	33,3%	16,4%	16,4%	0,6%	33,3%	

Fonte: Elaboração própria

Tabela 16 Probabilidade de Perda de Pacotes no Cenário 2

	PM	NS 1	NS 2	NS 3	NS 4	NS 5
PM		0,6%	0,6%	16,4%	33,3%	33,3%
NS 1	0,6%		7,5%	0,6%	7,5%	16,4%
NS 2	0,6%	7,5%		3,3%	0,6%	33,3%
NS 3	16,4%	0,6%	3,3%		7,5%	0,6%
NS 4	33,3%	7,5%	0,6%	7,5%		33,3%
NS 5	33,3%	16,4%	33,3%	0,6%	33,3%	

Fonte: Elaboração própria

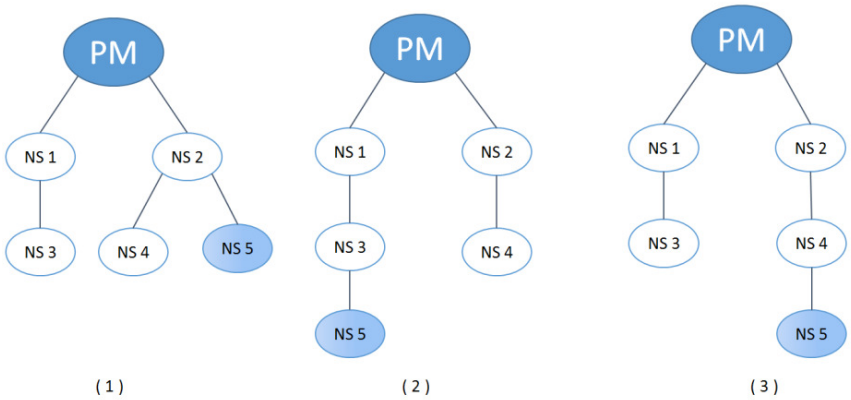
Tabela 17 Probabilidade de Perda de Pacotes no Cenário 3

	PM	NS 1	NS 2	NS 3	NS 4	NS 5
PM		0,6%	0,6%	16,4%	33,3%	33,3%
NS 1	0,6%		7,5%	0,6%	7,5%	16,4%
NS 2	0,6%	7,5%		3,3%	0,6%	33,3%
NS 3	16,4%	0,6%	3,3%		7,5%	33,3%
NS 4	33,3%	7,5%	0,6%	7,5%		0,6%
NS 5	33,3%	16,4%	33,3%	33,3%	0,6%	

Fonte: Elaboração própria

As variações dos cenários foram realizadas de forma a criar modificações na RSSI lida na rota para o NS 5, consequentemente aumentando sua PER e fazendo com que o PM realiza-se uma tentativa de encontrar uma rota alternativa para o mesmo. Desta forma, levando em consideração as melhores rotas para o NS em questão, as arvores de escoamento estimadas para cada cenário são apresentadas na Figura 25.

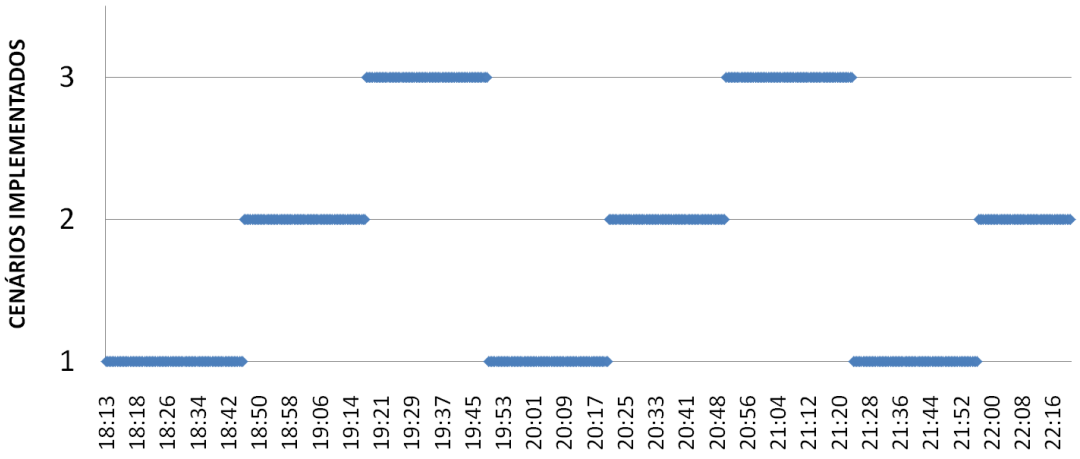
Figura 25 Árvores de escoamento estimadas para cada cenário



Fonte: Elaboração própria

A PER Máxima Tolerável configurada para todos os NS foi de 5%, uma PER considerada tolerável para uma RSSF [55]. O valor do Período da PER foi de 20 minutos, tempo suficiente para que a PER dos NS possa se estabilizar, já que a taxa de solicitação de pacotes é de aproximadamente 1 pacote por segundo (as mudanças de cenário ocorreram a aproximadamente cada 40 minutos). O tempo de execução total foi de aproximadamente 4 horas, com inicio as 18h13 e finalização as 22h16. Neste período, o algoritmo de emulação mudou o cenário da RSSF sete vezes, conforme é ilustrado na Figura 26.

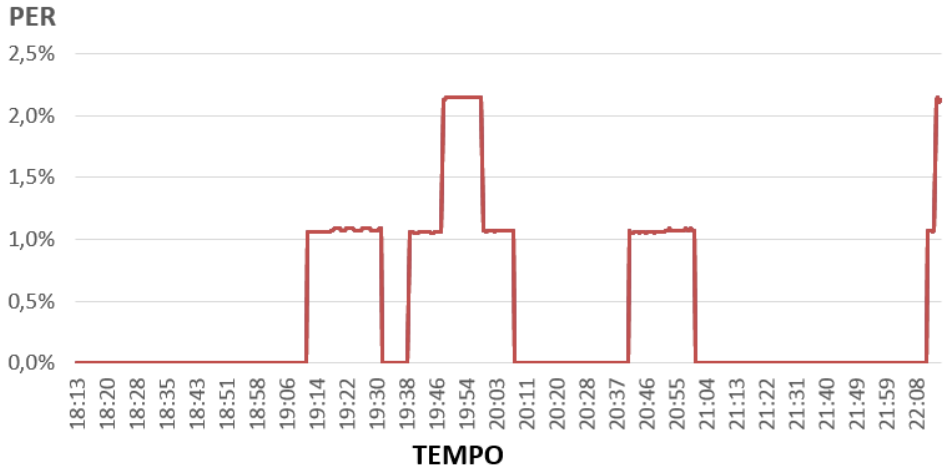
Figura 26 Alternância entre os cenários no algoritmo de emulação da RSSF no decorrer do tempo de testes



Fonte: Elaboração própria

A Figura 27 apresenta o histórico da PER do NS 1. Como é possível observar, em nenhum momento a PER ultrapassou a PER Máxima Tolerável de 5%.

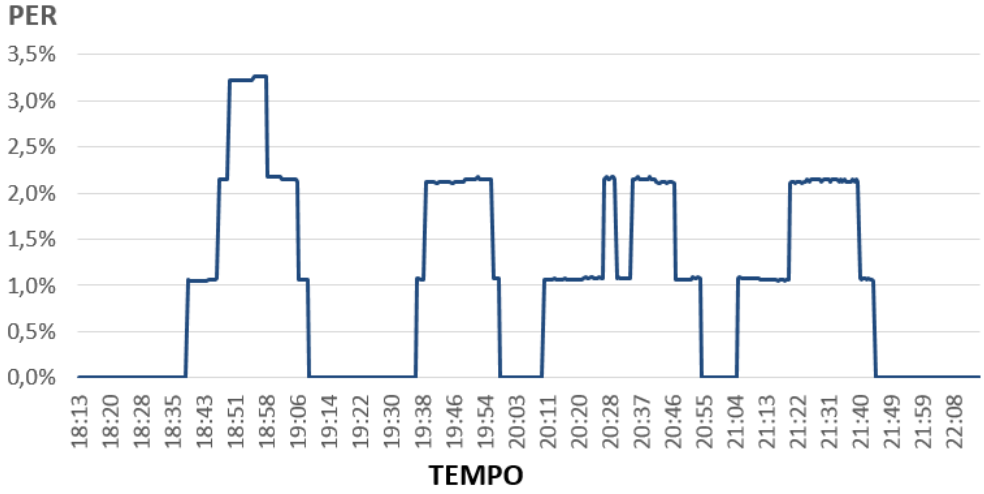
Figura 27 Histórico da PER (NS 1)



Fonte: Elaboração própria

A Figura 28 apresenta o histórico da PER do NS 2. Assim como NS 1, a PER do mesmo também não ultrapassou a PER Máxima Tolerável de 5%.

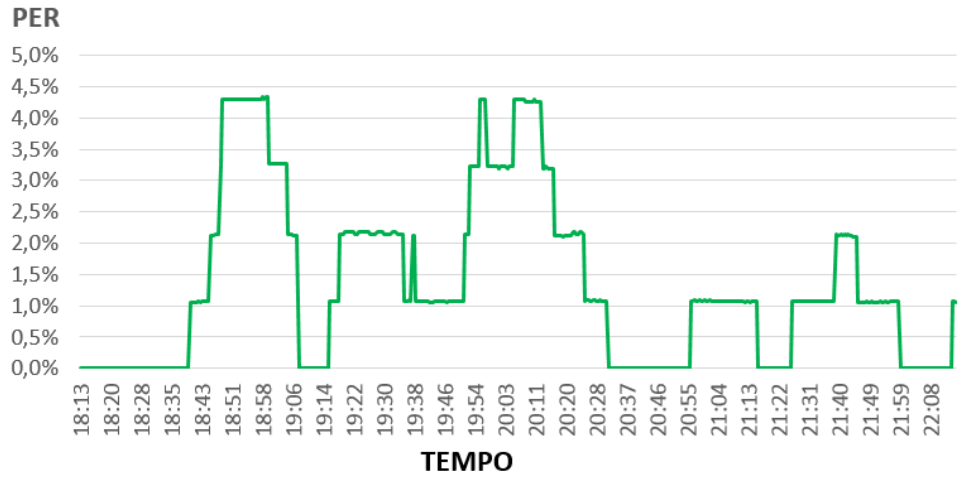
Figura 28 Histórico da PER (NS 2)



Fonte: Elaboração própria

Assim como o NS 1 e NS 2, a PER do NS 3 não ultrapassou a PER Máxima Tolerável, como é demonstrado no gráfico da Figura 29.

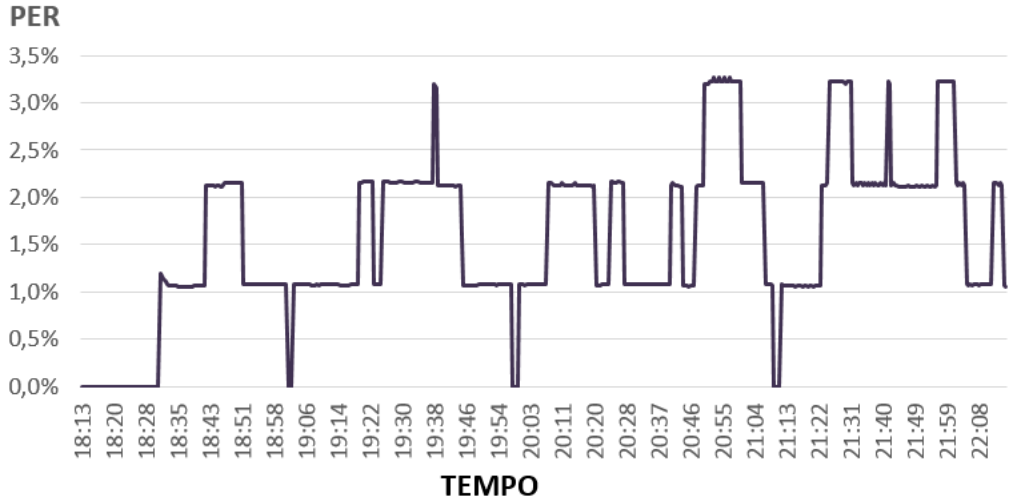
Figura 29 Histórico da PER (NS 3)



Fonte: Elaboração própria

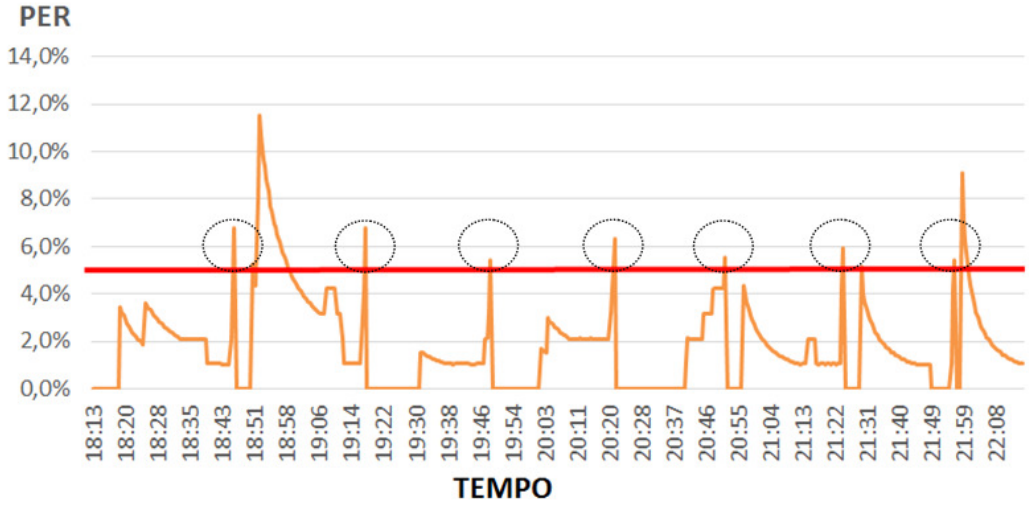
A PER do NS 4 também não ultrapassou a PER Máxima Tolerável de 5%, conforme o gráfico da Figura 30.

Figura 30 Histórico da PER (NS 4)



Fonte: Elaboração própria

Figura 31 Histórico da PER (NS 5)



Fonte: Elaboração própria

No gráfico da Figura 31, é possível observar nas áreas circuladas os momentos em que a PER do NS 5 ultrapassou a PER Máxima Tolerável. Após isso acontecer, a PER do mesmo caiu para níveis aceitáveis, comprovando que uma rota alternativa para o NS em questão foi encontrada. É possível observar que, logo após o primeiro e último tratamento, a PER atingiu picos de aproximadamente 11,8% e 9%, respectivamente. Nestes momentos, a rota alternativa estava no início do processo de avaliação, e um número pequeno de pacotes perdidos se refletiria em uma PER alta. Contudo, é possível observar que logo após esses momentos a PER se estabilizou para níveis mais baixos.

Ainda analisando o gráfico da Figura 31, é possível observar que o PM realizou 7 tratamentos de falhas referente ao NS 5, Se confrontado o gráfico em questão com o da Figura 26, é possível observar que os horários de mudança dos cenários neste são aproximados ao tratamento de falhas daquele.

Em suma, após os testes demonstrados, é possível afirmar que a funcionalidade de gerenciamento de falhas e desempenho da infraestrutura se mostrou funcional.

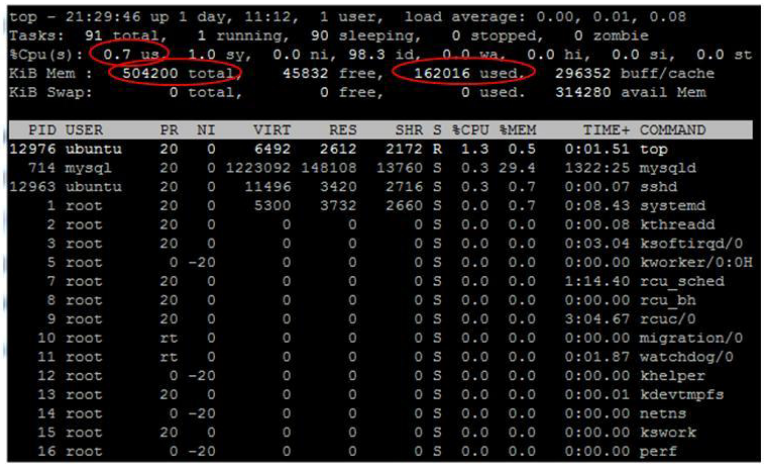
8.3 Eficiência em relação aos recursos de hardware

Para testar a eficiência em relação aos recursos de *hardware*, optou-se por medir a porcentagem de tempo ocupado na CPU (*Central Processing Unit* – Unidade

Central de Processamento) por processos do usuário (tais como os processos *FluxoPrincipal*, *PERAvaliador*, o SGBD MySQL, o servidor *web* Apache e todos os processos que não dizem respeito ao *kernel* do sistema operacional GNU/Linux) e a porcentagem de Memória RAM (*Random Access Memory*) ocupada. A preocupação em medir estes indicadores se deve ao fato de que a plataforma foi implementada através de *softwares* tipicamente voltados para ambientes de servidores e *desktops*, os quais poderiam utilizar uma grande quantidade da Memória RAM e tempo da CPU da plataforma *Beagle Bone Black*, impossibilitando o desenvolvimento de novas funcionalidades nesta implementação da plataforma no futuro, já que a mesma teria atingido seu limite de desempenho.

A medição de ambos os indicadores foi realizada em um período de aproximadamente 24 horas, com coleta de dados feita a cada 10 segundos. Durante os testes, os processos *FluxoPrincipal* e *PERAvaliador* estavam em execução, e, além deles, requisições via *web service,* referente a PER média dos NS (explicado com mais detalhes no tópico 8.4), eram realizadas a cada 15 segundos. A coleta dos dados foi realizada através de um *shell script*, o qual executava o comando *top* [56]. Este comando permite visualizar informações sumarizadas de desempenho dos processos que estão sendo executados, bem como informações gerais do sistema. A Figura 32 apresenta o comando sendo executado em tempo real. Os indicadores utilizados na coleta são destacados através de círculos.

Figura 32 Execução do comando top



Fonte: Elaboração própria

O indicador de porcentagem de tempo de utilização da CPU por processos do

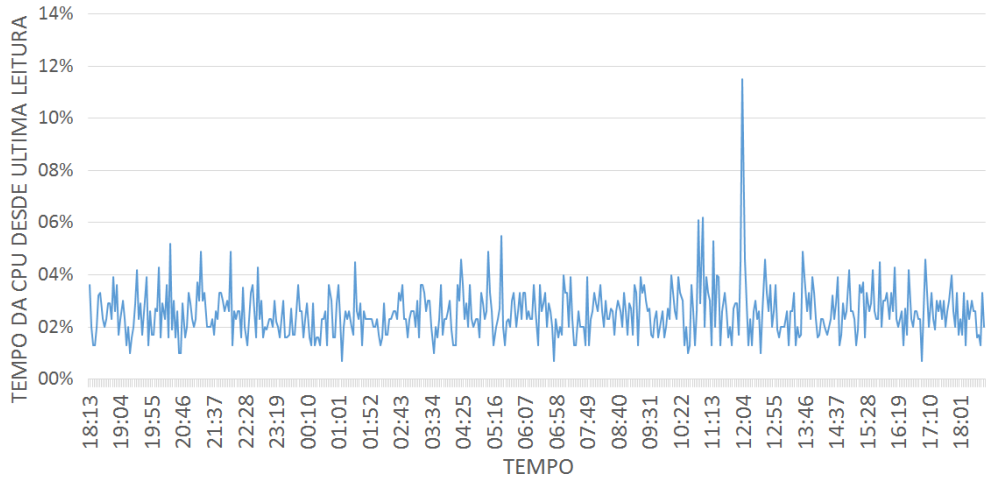
usuário é identificado por *us*. De acordo com o manual do comando, o tempo a qual se refere a porcentagem é referente ao intervalo desde a última medida realizada pelo comando.

Para o cálculo da porcentagem de total de memória utilizada, foi utilizada a expressão demonstrada na Equação 6.

$$\text{Porcentagem Memória Utilizada} = \text{Memória Usada} / \text{Total de Memória} \quad (6)$$

As Figuras 33 apresentam um gráfico com a porcentagem de tempo da CPU ocupada por processos do usuário.

Figura 33 Porcentagem do tempo da CPU ocupada por processos do usuário

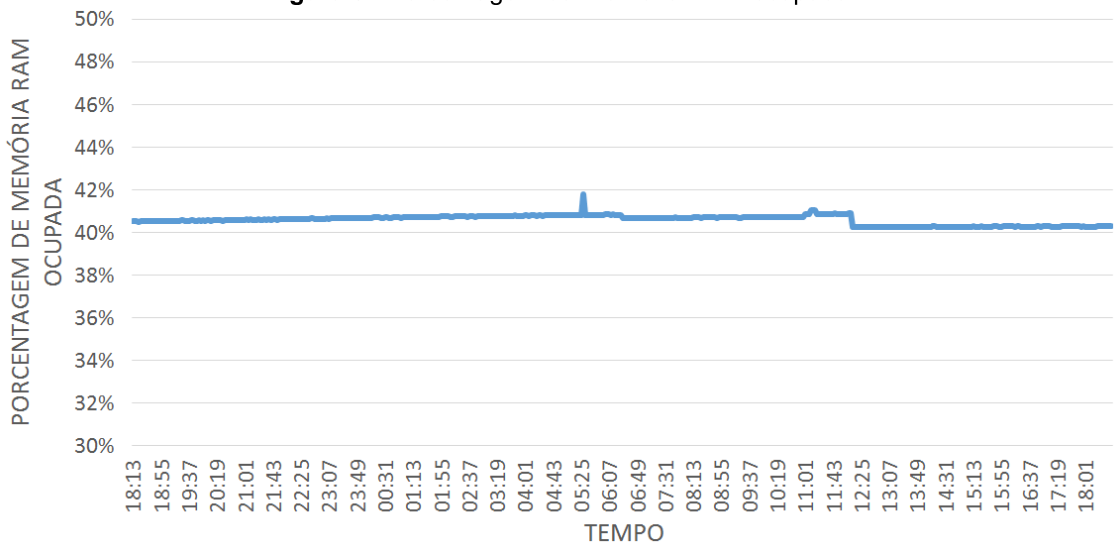


Fonte: Elaboração própria

A média de tempo de utilização da CPU foi de 2,5%, com um desvio padrão de 1%.

Por sua vez, a Figura 34 apresenta a porcentagem de memória RAM ocupada durante os testes.

Figura 34 Porcentagem da memória RAM ocupada



Fonte: Elaboração própria

É possível observar que a utilização da Memória RAM foi praticamente constante durante o período de testes. A média de utilização da mesma foi de 40,5%, com um desvio padrão de 0,2%.

Ao analisar os dados, é possível afirmar que, aliado a novos testes de desempenho, novas funcionalidades podem ser implementadas, visto que o tempo de CPU e a memória RAM não foram utilizados em seu limite.

8.4 Eficiência em relação ao tempo de resposta do Módulo Informante

A fim de avaliar a eficiência do Módulo Informante referente ao quesito tempo de resposta, foi desenvolvido, para fins de teste, um *web service* que retorna a média da PER de determinado NS nos últimos 60 minutos. Para tal, o *web service* realiza uma consulta SQL na tabela *historyPERTable*. A Figura 35 apresenta o trecho do código fonte ao qual esta consulta se refere.

Figura 35 Trecho do código fonte do *web service* implementado para testes

```
//Cria SQL - Retorna a média da PER do NS nos ultimos 60 minutos
$sql = "SELECT AVG(per) as mediaPER FROM historyPERTable
        WHERE addressNS = $addressNS and
        date > NOW() - INTERVAL 60 MINUTE";
//Realiza a consulta
$resultado = mysql_query($sql);
```

Fonte: Elaboração própria

Para realizar a requisição ao serviço do *web service* em questão, um script em PHP foi desenvolvido. O *script* foi executado em um computador com o servidor Web Apache instalado, localizado na mesma rede LAN (*Local Area Network*) do PM. A rede em questão trabalhava com o padrão *Fast Ethernet*, a uma taxa de transferência de 100 Mbps.

O teste em questão foi executado no mesmo período de tempo dos testes de eficiência em relação aos recursos de *hardware* (Tópico 8.3), ou seja, aproximadamente 24 horas. As requisições foram realizadas em intervalos de 15 segundos, totalizando 5853 requisições. Para medir o tempo de resposta, a função *microtime()* do PHP [57] foi utilizada. A mesma retorna um *timestamp* (tempo padrão do Unix) em microssegundos. Sendo assim, a função foi utilizada antes e após a requisição via *web service*, sendo o tempo de resposta a subtração dos dois valores retornados. A Figura 36 apresenta o trecho do código do script que implementa o procedimento em questão.

Figura 36 Trecho do código fonte para calculo do tempo de resposta

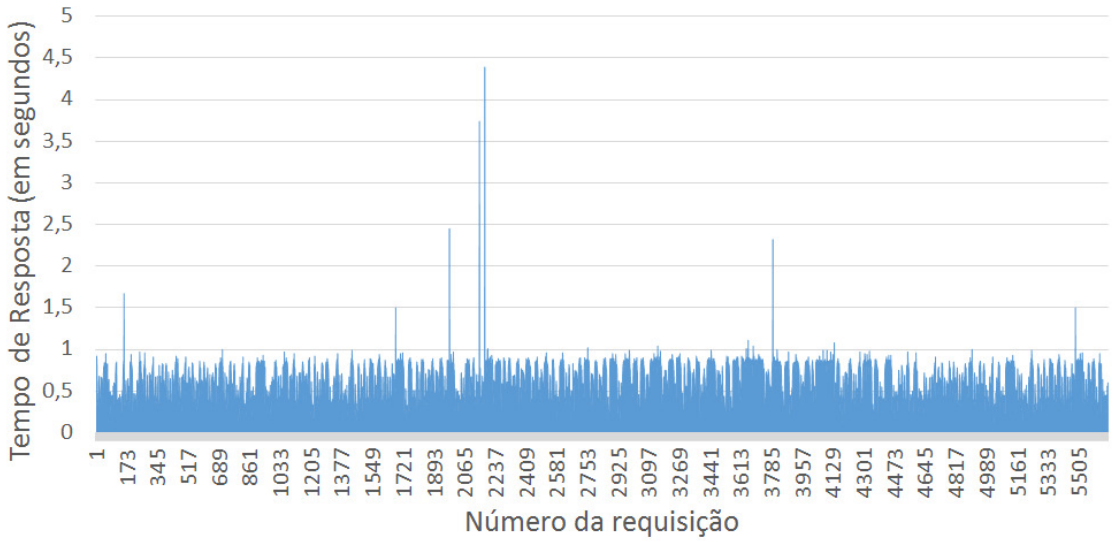
```
//Pega momento de execução inicial
$inicio = microtime(True);

    //Executa o web service
    $resultado = $clientNS->call("getAVGPer", array("addressNS" => "$enderecoNS"));

//Pega momento de execução final
$fim = microtime(True);

$tempoExecucao = number_format(($fim - $inicio), 2);
```

Fonte: Elaboração própria

O gráfico da Figura 37 apresenta o tempo de resposta (em segundos) das requisições realizadas. O tempo de resposta médio para as requisições foi de 0.59 segundos, com desvio padrão de 0.22 segundos, o que satisfaria aplicações de trabalho intensivo, como por exemplo, criação de gráficos em tempo real, conforme

é tratado em [9]. Sendo assim, para o ambiente em questão, o tempo de resposta se mostrou satisfatório.

Figura 37 Tempo de resposta – Coleta de dados via Web Service



Fonte: Elaboração própria

8.5 Adequação da coleta de dados dos sensores

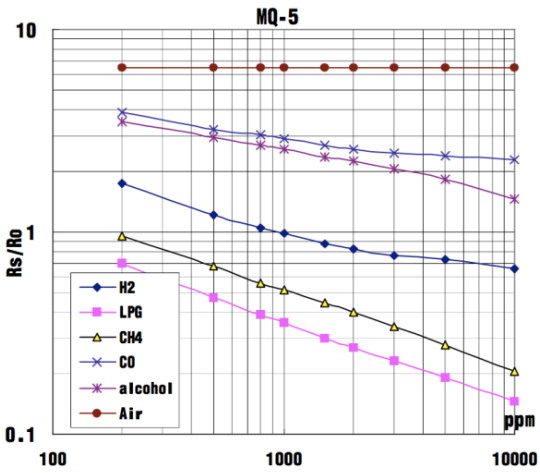
Para testar a funcionalidade de coleta e armazenamento do dados gerados pelos sensores, foi configurada e instalada uma RSSF com 2 NS Radiuino modelo DK-105, com endereços 1 e 2, respectivamente. O objetivo principal do teste era verificar se o PM conseguiria armazenar os dados gerados por NS e sensores diferentes.

No NS 1 foi instalado um sensor Grove de temperatura [58]. Este sensor utiliza um termisistor para ler a temperatura do ambiente, sendo que a resistência do mesmo sobe quando a temperatura do ambiente cai. O intervalo de temperatura do sensor é de -40ºC até 125ºC. O sensor em questão retorna um valor analógico, o qual, através de cálculos indicados no site oficial do sensor, é possível obter a temperatura em Cº.

Já no NS 2 foi instalado um sensor Grove de gás MQ5 [59]. Este tipo de sensor pode ser utilizado para detectar vazamento de gás em indústrias e residências, sendo capaz de detectar álcool, H2, LPG, CH4 e CO. O sensor retorna um valor analógico, o qual cresce conforme a presença destes gases no ambiente aumenta. Através de cálculos demonstrados no site oficial do sensor, é possível

obter um valor de resistência (RS/R0), o qual pode ser mapeado em uma concentração de gás (dada em ppm – partes por milhão), conforme é apresentado na Figura 38.

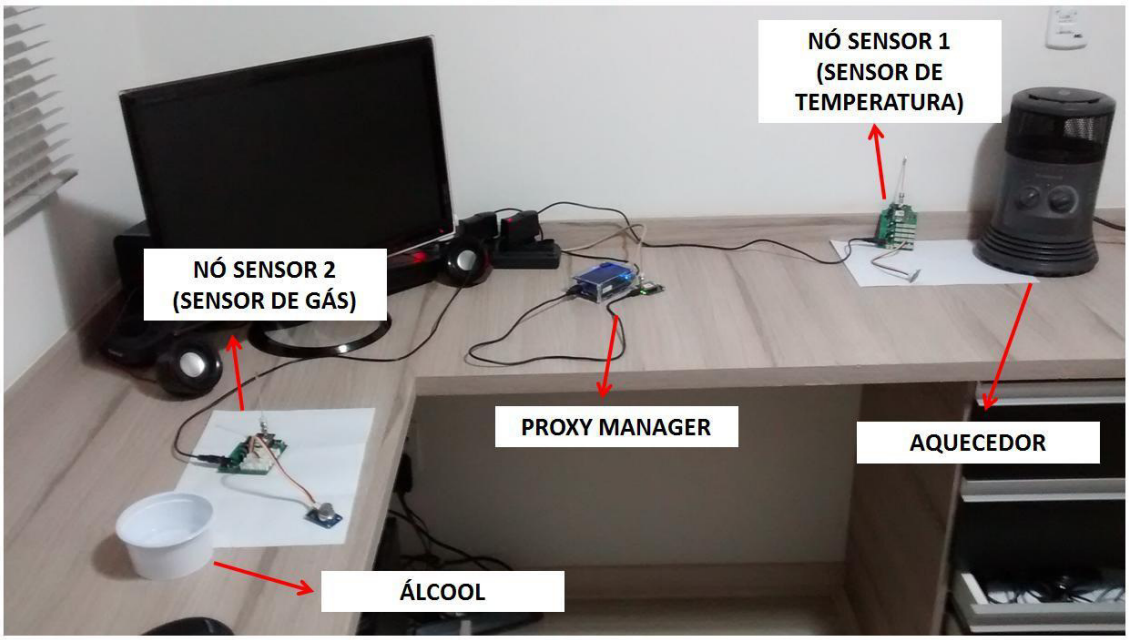
Figura 38 Concentração de gás pela resistência (RS/R0)



Fonte: [59]

A RSSF implementada é apresentada na Figura 39.

Figura 39 RSSF implementada para coleta de dados



Fonte: Elaboração própria

A coleta dos dados foi realizada em um período de aproximadamente 9 horas,

das 15h07 do dia 16/10/2016, até as 00h07 do dia 17/10/2016. Durante um intervalo de aproximadamente 1 hora um pote contendo álcool foi colocado próximo ao NS 2, a fim de que o sensor de gás pudesse ter uma variação em sua resistência. A mesma ideia e intervalo foram aplicados ao NS 1, através de um aquecedor. Por razões de segurança, o aquecedor não foi ligado na presença do pote com álcool.

A Tabela 18 apresenta uma síntese dos pacotes enviados, recebidos, perdidos e a PER para cada NS, referente ao período total dos testes.

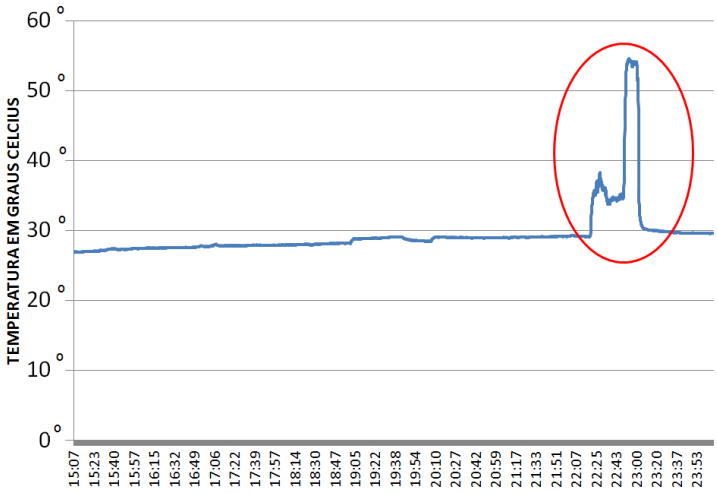
Tabela 18 Pacotes de requisição de dados durante os testes

NS	Pacotes enviados	Pacotes recebidos	Pacotes perdidos	PER
1	5645	5603	42	0,7%
2	5644	5617	27	0,4%

Fonte: Elaboração própria

A Figura 40 apresenta o histórico da temperatura coletada pelo sensor de temperatura. A área circulada indica o período onde o aquecedor foi ligado.

Figura 40 Dados coletados e armazenados pelo sensor de temperatura



Fonte: Elaboração própria

A Figura 41 apresenta o código em SQL utilizado para recuperar os dados do Banco de Dados do PM.

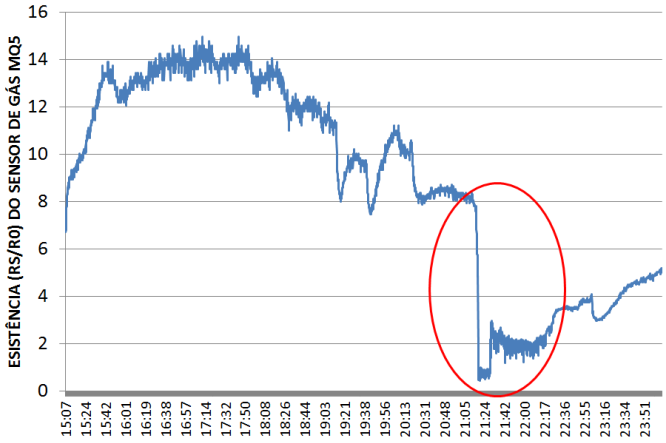
Figura 41 Código SQL utilizado para recuperar os dados do sensor de temperatura

```
SELECT packetReceivedTable.date as 'datahorario',collectedData.dataCollected as 'dado coletado', collectedData.idSensorNS as 'sensor'
FROM collectedData,packetReceivedTable,nsSensorTable
WHERE collectedData.idSensorNS = nsSensorTable.idSensorNS
  AND collectedData.idSensorNS = 1
  AND collectedData.idPacket = packetReceivedTable.idPacket
  AND packetReceivedTable.date >= '2016-10-16 00:00:00'
  AND packetReceivedTable.date <= '2016-10-17 00:08:00'
```

Fonte: Elaboração própria

A Figura 42 apresenta o histórico da resistência referente a presença de gases no ambiente, coletada pelo sensor MQ-5. A área circulada indica o período onde álcool foi posto próximo ao sensor. Confrontando os dados coletados com o gráfico da Figura 38, é possível ver que a presença de álcool no ambiente diminui a resistência do sensor.

Figura 42 Dados coletados e armazenados pelo sensor de gás MQ5



Fonte: Elaboração própria

A Figura 43 apresenta o código em SQL utilizado para recuperar os dados do Banco de Dados do PM.

Figura 43 Código SQL utilizado para recuperar os dados do sensor de gás MQ5

```
SELECT packetReceivedTable.date as 'datahorario',collectedData.dataCollected as 'dado coletado', collectedData.idSensorNS as 'sensor'
FROM collectedData,packetReceivedTable,nsSensorTable
WHERE collectedData.idSensorNS = nsSensorTable.idSensorNS
  AND collectedData.idSensorNS = 2
  AND collectedData.idPacket = packetReceivedTable.idPacket
  AND packetReceivedTable.date >= '2016-10-16 00:00:00'
  AND packetReceivedTable.date <= '2016-10-17 00:08:00'
```

Fonte: Elaboração própria

Através da análise dos dados coletados pelo PM, conclui-se que a plataforma coletou e armazenou os dados corretamente, inclusive em relação ao tempo.

9 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo apresentar uma proposta de *Proxy Manager* para Internet das Coisas de baixo custo, implementada no *gateway* da rede, que tenha como cerne a gerência de Redes de Sensores sem Fio instaladas em ambientes previamente conhecidos. A colaboração do trabalho em questão se deve ao fato de que soluções de RSSF carecem de processos e ferramentas de gerência, o que dificultam a implantação maciça de RSSF e, consequentemente, da Internet das Coisas. Aproveitando o maior poder computacional e energético que o *gateway* da RSSF oferece, a implantação de uma alternativa de gerência no mesmo se mostra vantajosa, visto que uma gerência centralizada pode ter uma visão total da RSSF, auxiliando no processo de analise de desempenho e tratamento de falhas, além de simplificar o *hardware* e *software* dos NS, diminuindo o custo de implantação da RSSF. O gerenciamento local (próximo da RSSF) também é vantajoso do ponto de vista de envio de dados para aplicações na Internet, visto que possibilita que informações sumarizadas e tratadas sejam encaminhadas para aplicações finais, diminuindo a vazão de dados brutos para a Internet.

Os focos de gerência implementados foram a da infraestrutura da rede e os dados coletados pela aplicação. As áreas de gerência abordadas pela proposta foram as áreas de configuração da infraestrutura (cadastro dos NS, estabelecimento de rotas e parametrização do PM); configuração dos dados (cadastro dos sensores que compõem a RSSF); desempenho da infraestrutura (monitoramento da PER de cada NS) e falha da infraestrutura (detecção de uma PER alta relacionada a cada NS).

O PM foi implementado através de alternativas gratuitas e *open source*, tanto em nível de *hardware* quando de *software*. A solução se mostrou viável, apresentando resultados satisfatórios do ponto de vista de funcionalidades e desempenho.

É de fundamental importância que, em momentos e trabalhos futuros, novas funcionalidades sejam implementadas, tanto do ponto de vista de gerência da infraestrutura quando dos dados, como, por exemplo, a análise do comportamento dos enlaces baseado na RSSI e na LQI (*Link Quality Indicator*) e sumarização dos

dados coletados pelos sensores. A integração da mesma com padrões diferentes de RSSF também é desejável, desde que as mesmas ofereçam suporte a implantação de novas funcionalidades (padrões abertos).

10 REFERÊNCIAS BIBLIOGRÁFICAS

[1] M. A. Feki, F. Kawsar, M. Boussard, e L. Trappeniers, “The Internet of Things: The Next Technological Revolution”, *IEEE Comput. Soc.*, vol. 35, nº 5, p. 24–25, 2013.

[2] H. Karl e A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. Chinchester: John Wiley & Sons, 2006.

[3] W. Dargie e C. Poellabauer, *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Chinchester: John Wiley & Sons, 2010.

[4] I. F. Akyildiz e M. C. Vuran, *Wireless sensor networks*. Chinchester: John Wiley & Sons, 2010.

[5] E. Alessio, A. Bragagnini, G. Perbellini, e D. Quaglia, “Gateway and middleware design: Trusted WSN-TLC network communication and enhanced WSN management”, *Proc. IEEE Int. Conf. Electron. Circuits, Syst.*, p. 637–640, 2007.

[6] M. Aazam e E. N. Huh, “Fog computing and smart gateway based communication for cloud of things”, in *Proceedings - 2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014*, 2014, p. 464–470.

[7] L. Karim, Q. H. Mahmoud, N. Nasser, e N. Khan, “An integrated framework for wireless sensor network management”, *Wirel. Commun. Mob. Comput.*, vol. 14, nº 12, p. 1143–1159, 2014.

[8] ITU-T, “ITU-T Recommendation M.3400: TMN Management Functions”, vol. 3400. 2000.

[9] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, 3th ed. Boston: Addison-Wesley Professional, 2013.

[10] S. Yi, Z. Hao, Z. Qin, e Q. Li, “Fog Computing: Platform and Applications”, *2015 Third IEEE Work. Hot Top. Web Syst. Technol.*, p. 73–78, 2015.

[11] I. Stojmenovic, “Fog computing: A cloud to the ground support for smart things and machine-to-machine networks”, *2014 Australas. Telecommun. Networks Appl. Conf. ATNAC 2014*, p. 117–122, 2015.

[12] D. A. Gomes, “Arquitetura para interconexão de Redes de Sensores sem Fio e a Internet através de Web Services e o protocolo HTTP”, M.S. thesis, Faculdade de Engenharia Elétrica, Pontífica Universidade Católica de Campinas, Campinas, 2015.

[13] D. Georgakopoulos e P. P. Jayaraman, “Internet of things: from internet scale sensing to smart services”, *Computing*, vol. 98, nº 10, p. 1–18, 2016.

[14] T. Renner, M. Meldau, e A. Kliem, “Towards Container-Based Resource Management for the Internet of Things”, *2016 Int. Conf. Softw. Netw.*, p. 5, 2016.

[15] D. Pasini, S. Mastrolembo Ventura, S. Rinaldi, P. Bellagente, A. Flammini, e A. L. C. Ciribini, “Exploiting Internet of Things and building information modeling framework for management of cognitive buildings”, *2016 IEEE Int. Smart Cities Conf.*, vol. 40545387, nº 40545387, p. 1–6, 2016.

[16] R. Alur, E. Berger, A. W. Drobnis, L. Fix, K. Fu, G. D. Hager, D. Lopresti, K. Nahrstedt, E. Mynatt, S. Patel, J. Rexford, J. A. Stankovic, e B. Zorn, “Systems Computing Challenges in the Internet of Things”, *Ccc*, nº June, 2015.

[17] A. Botta, W. de Donato, V. Persico, e A. Pescapé, “Integration of Cloud computing and Internet of Things: A survey”, *Futur. Gener. Comput. Syst.*, vol. 56, p. 684–700, 2014.

[18] A. Ahmed e E. Ahmed, “A Survey on Mobile Edge Computing (#16)”, *Isco*, nº JANUARY, 2016.
[19] B. K. Maharrey, A. S. Lim, e S. Gao, “Interconnection between IP networks and wireless sensor networks”, *Int. J. Distrib. Sens. Networks*, vol. 2012, 2012.
[20] A. Dunkels, J. Alonso, T. Voigt, H. Ritter, e J. Schiller, “Connecting Wireless Sensornets with TCP/IP Networks”, *Proc. Second Int. Conf. Wired/Wireless Internet Commun. (WWIC2004}*, p. 143–152, 2004.
[21] M. Patil e R. C. Biradar, “A survey on routing protocols in Wireless Sensor Networks”, *Networks (ICON), 2012 18th IEEE Int. Conf.*, p. 86–91, 2012.
[22] J. N. Al-Karaki e A. E. Kamal, “Routing Techniques in Wireless Sensor Networks”, vol. 2, nº 7, p. 344–348, 2012.
[23] N. A. Pantazis, S. A. Nikolidakis, e D. D. Vergados, “Energy-Efficient Routing Protocols in Wireless Sensor Networks: A Survey”, *IEEE Commun. Surv. Tutorials*, vol. 15, nº 2, p. 551–591, 2013.
[24] R. Ogier, F. Templin, e M. Lewis, “Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)”, *IETF RFC 3684*, p. 1–47, 2004.
[25] K. Sohrabi, J. Gao, V. Ailawadhi, e G. J. Pottie, “Protocols for self-organization of a wireless sensor network”, *IEEE Pers. Commun.*, vol. 7, nº 5, p. 16–27, 2000.
[26] W. L. Lee, A. Datta, e R. Cardell-Oliver, “Network management in wireless sensor networks”, *Handb. Mob. Ad Hoc Pervasive Commun.*, p. 1 – 18, 2006.
[27] M. Yu, H. Mokhtar, e M. Merabti, “A survey of network management architecture in wireless sensor network”, *Proc. Sixth Annu. Postgrad. Symp. Converg. Telecommun. Netw. Broadcast.*, p. 1–5, 2006.
[28] O. Salman, I. Elhajj, A. Kayssi, e A. Chehab, “An architecture for the Internet of Things with decentralized data and centralized control”, *Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl. AICCSA*, vol. 2016-July, 2016.
[29] H. Song, D. Kim, K. Lee, e J. Sung, “UPnP-based sensor network management architecture”, *Proc. ICMU Conf*, p. 6, 2005.
[30] N. Ramanathan, E. Kohler, L. Girod, e D. Estrin, “Sympathy: A debugging system for sensor networks”, in *Proceedings - Conference on Local Computer Networks, LCN*, 2004, p. 554–555.
[31] G. Tolle e D. Culler, “Design of an application-cooperative management system for wireless sensor networks”, *Wirel. Sens. Networks, 2005. Proceeedings Second Eur. Work.*, p. 121–132, 2005.
[32] M. Turon, “MOTE-VIEW: A sensor network monitoring and management tool”, in *Second IEEE Workshop on Embedded Networked Sensors*, 2005, p. 11–17.
[33] S. R. Madden, M. J. Franklin, J. M. Hellerstein, e W. Hong, “TinyDB: an acquisitional query processing system for sensor networks”, *ACM Trans. Database Syst.*, vol. 30, nº 1, p. 122–173, 2005.
[34] W. Naruephiphat, R. Prom-Ya, e C. Chansripinyo, “A web-based management system design for wireless sensor network monitoring”, *2013 Int. Comput. Sci. Eng. Conf.*, p. 281–285, 2013.
[35] M. Navarro, D. Bhatnagar, e Y. Liang, “An Integrated Network and Data Management System for Heterougeneous WSNs”, *Eighth IEEE Int. Conf. Mob. Ad-Hoc Sens. Syst.*, p. 819–824, 2011.
[36] S. L. R. Barreto, “Método para Averição de distâncias entre Nós Sensores Baseado em RSSI”, M.S. thesis, Faculdade de Engenharia Elétrica, Pontífica Universidade Católica de Campinas, 2014.
[37] N. Baccour, A. Koubâa, L. Mottola, M. A. Zúñiga, H. Youssef, C. A. Boano, e

M. Alves, “Radio Link Quality Estimation in Wireless Sensor Networks: A Survey”, *ACM Trans. Sen. Netw.*, vol. 8, nº 4, p. 34:1–34:33, 2012.
[38] K. Srinivasan e P. Levis, “RSSI is Under Appreciated”, *Proc. Third Work. Embed. Networked Sensors*, 2006.
[39] C. R. Real, “Medida e análise de comportamento da RSSI de uma rede de sensor sem fio em ambiente industrial”, M.S. thesis, Faculdade de Engenharia Elétrica, Pontífica Universidade Católica de Campinas, 2015.
[40] H. Garcia-Molina, J. D. Ullman, e J. Widom, “Database Systems: The Complete Book”, *Education*, p. 1248, 2008.
[41] M. Maksimović, V. Vujović, N. Davidović, V. Milošević, e B. Perišić, “Raspberry Pi as Internet of Things hardware : Performances and Constraints”, *Des. Issues*, vol. 3, nº JUNE, p. 8, 2014.
[42] “Arduino Board UNO”, 13-jan-2016. [Online]. Available at: https://www.arduino.cc/en/Main/ArduinoBoardUno.
[43] “BeableBone Black”, 13-jan-2016. [Online]. Available at: https://beagleboard.org/black.
[44] “Raspberry Pi”, 13-jan-2016. [Online]. Available at: https://www.raspberrypi.org/.
[45] “Appropriate Uses For SQLite”, 15-jan-2016. [Online]. Available at: https://www.sqlite.org/whentouse.html.
[46] C. E. S. Pires, R. O. Nascimento, e A. C. Salgado, “Comparativo de desempenho entre bancos de dados de codigo aberto”, *Cent. Informatica-Universidade Fed. Pernambuco. Recife*, 2008.
[47] T. Suzumura, S. Trent, M. Tatsubori, A. Tozawa, e T. Onodera, “Performance comparison of Web service engines in PHP, Java, and C”, *Proc. IEEE Int. Conf. Web Serv. ICWS 2008*, nº May 2016, p. 385–392, 2008.
[48] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, e W. Zwaenepoel, “Performance Comparison of Middleware Architectures for Generating Dynamic Web Content”, *Middlew. ’03 Proc. ACM/IFIP/USENIX 2003 Int. Conf. Middlew.*, p. 242–261, 2003.
[49] S. Trent, M. Tatsubori, T. Suzumura, A. Tozawa, e T. Onodera, “Performance comparison of PHP and JSP as server-side scripting languages”, *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5346 LNCS, p. 164–182, 2008.
[50] “Python”, 14-jan-2016. [Online]. Available at: https://www.python.org/.
[51] “Radiuino”, 15-jan-2016. [Online]. Available at: http://radiuino.cc/.
[52] “Grove System”, 14-set-2016. [Online]. Available at: http://wiki.seeed.cc/Grove_System/.
[53] V. Q. Pereira, “Estimativa da PER, Protocolo de Coleta de RSSI e Determinação de Melhoras Rotas em RSSF”, M.S. thesis, Faculdade de Engenharia Elétrica, Pontífica Universidade Católica de Campinas, 2016.
[54] International Organization For Standardization Iso, “Iso 9126”, *Software Process: Improvement and Practice*, vol. 2, nº 1. p. 1–25, 2001.
[55] R. M. Assumpção, “Avaliação do Impacto em Redes de Sensores sem Fio com Utilização de Rádio sobre Fibra”, M.S. thesis, Faculdade de Engenharia Elétrica, Pontífica Universidade Católica de Campinas, 2011.
[56] “Manual for command top”, 10-out-2016. [Online]. Available at: http://man7.org/linux/man-pages/man1/top.1.html.
[57] “PHP - Função microtime()”, 11-out-2016. [Online]. Available at: http://php.net/manual/pt_BR/function.microtime.php.

[58] “Grove - Temperature Sensor”, 16-out-2016. [Online]. Available at: http://wiki.seeedstudio.com/wiki/Grove_-_Temperature_Sensor.
[59] “Grove - Gas Sensor(MQ5)”, 16-out-2016. [Online]. Available at: http://wiki.seeedstudio.com/wiki/Grove_-_Gas_Sensor(MQ5).

11 ANEXOS

11.1 Código Fonte – Módulo Gerente Local

Arquivo: fluxoPrincipal.py

```
# coding=UTF-8
#!/usr/bin/python

import MySQLdb
import serial
import time
import sys
from time import localtime, strftime
from datetime import datetime, date
import struct
from timeit import default_timer as timer
import subprocess
import shlex

from conectaDB import *
from collectedFunctions import *
from routeFunctions import *


#Cria conexão com o banco de dados
db, cursor = conectaDB()

#Pega data e hora atual
date_time = time.strftime("%Y-%m-%d %H-%M-%S")
#Registra inicio das atividades na tabela de log
cursor.execute("INSERT INTO logTable
VALUES(%s,%s,%s,%s)",("*",date_time,"BEGIN","Begin of activities"))
db.commit()

#Testa existência da porta serial
#Executa consulta no banco de dados
cursor.execute("SELECT * from severalTable where statusConfig = 1")

#Pega resultado da execucao e passa para o vetor linha
linha = cursor.fetchone()

#Passa os parametros cadastrados no DB para variáveis
#Cada posição do vetor linha é relacionado a um campo de resultado do
SELECT
serialPort = linha["serialPort"] #Pega a porta serial cadastrada

#Cria objeto para porta serial
try:
   serialPort = serial.Serial(serialPort, 9600, timeout=0.5,
parity=serial.PARITY_NONE)
except Exception:

   #Pega data e hora atual
   date_time = time.strftime("%Y-%m-%d %H-%M-%S")

   #Registra incidente na tabela de logs
   cursor.execute("INSERT INTO logTable
VALUES(%s,%s,%s,%s)",("*",date_time,"ERROR_SERIAL","Error opening
serial port"))
```

```
    db.commit()
    cursor.close()

    #Encerra programa
    sys.exit()

#Registra ínicio das atividades do Proxy Manager na tabela de log
#Pega data e hora atual
date_time = time.strftime("%Y-%m-%d %H-%M-%S")
#Registra incidente na tabela de logs
cursor.execute("INSERT INTO logTable
VALUES(%s,%s,%s,%s)",("*",date_time,"REPORT_ROUTE","Beginning of the
All Route Discovery Process"))
db.commit()

#Realizar a descoberta de rotas para todos os NS cadastrados no Banco
de Dados
#Este passo só será dado no momento de inicialização do sistema
discoverAllRoutes()

#Insere atividade na tabela de log
#Pega data e hora atual
date_time = time.strftime("%Y-%m-%d %H-%M-%S")
#Registra incidente na tabela de logs
cursor.execute("INSERT INTO logTable
VALUES(%s,%s,%s,%s)",("*",date_time,"REPORT_ROUTE","Ending of the All
Route Discovery Process"))
db.commit()

#cursor.close()
#db.close()

#Laço principal do programa
while True:

    #Cria conexão com o banco de dados
    #db, cursor = conectaDB()

    #Verifica se existe alguma interrupção para recalculo das rotas
    cursor.execute("SELECT flagInterrupt from severalTable")
    #Pega resultado da execucao e passa para o vetor linha
    linha = cursor.fetchone()
    flagInterrupt = linha["flagInterrupt"]

    #Verifica se a flag está ativada
    if (flagInterrupt == 1):

        #Trata a interrupção
        #Executa consulta no banco de dados - Irá apenas selecionar os NS
onde o status seja 'NR' (NR) e 'Inconsistente' (IN)
        cursor.execute("SELECT
address,route,externalFunction,location,actualPER from nsTable where
status = 'NR' or status = 'IN'")

        #Pega resultado da execucao e passa para o vetor linhas
        linhas = cursor.fetchall()

        #Cria vetor para armazenar os resultados.
        #1° Posição: Endereço do NS
        #2° Posição: Rota atual do NS
        ns = {}
```

```
      i = 0

      #Percorre todos os NS encontrados, a fim de tentar encontrar uma
rota alternativa para os mesmos
      for linha in linhas:
         ns[i,1] = linha["address"]
         ns[i,2] = linha["route"]
         ns[i,3] = linha["externalFunction"]
         ns[i,4] = linha["location"]
         ns[i,5] = linha["actualPER"]

         #Faz tentativa de descoberta de rota
         novaRota = discoverSingleRoute(ns[i,1])

         #Verifica o status da nova rota - Se ela for N/A (nenhuma rota
descoberta) ou se for igual a rota antiga, o NS será cadastrado como
problemático, e uma mensagem será enviada ao administrador
         if (novaRota == "N/A") or (novaRota == ns[i,2]):

            #Realiza a atualização na base...
            #Atualiza status do NS para Problemático...
            cursor.execute("UPDATE nsTable SET status=%s WHERE
address=%s",("PR",ns[i,1]))
            db.commit()
            #Registrar o log na tabela de logs
            date_time = time.strftime("%Y-%m-%d %H-%M-%S")
            cursor.execute("INSERT INTO logTable
VALUES(%s,%s,%s,%s)",(ns[i,1],date_time,"ERROR_NS","NS Ranked
Problematic (PER: "+str(ns[i,5])+")"))
            db.commit()

            if (ns[i,3] != 0):

               #Cria string para acionar script externo

string="/var/www/html/bubble/externalFunctions/"+ns[i,3]+" 'Relato de
Falha' '" + date_time + " - NS " + ns[i,1] + "( Localização: "+ ns[i,4]
+") cadastrado como PR (Problematico). PER: " + str(ns[i,5]) + "Tomar
providências.'"
               print string
               #Executa ação externa
               subprocess.call(shlex.split(string))


         #Caso contrário, alguma rota alternativa foi encontrada para o
NS em questão...
         else:

            #Realiza a atualização na base...
            #Atualiza status do NS para Em Avaliação...

            date_time = time.strftime("%Y-%m-%d %H-%M-%S")

            #Muda o status do NS para EA e registra o horário de inicio
do processo de Em Avaliação
            cursor.execute("UPDATE nsTable SET status=%s,
beginEvaluation=%s WHERE address=%s",("EA",date_time,ns[i,1]))
            db.commit()
            #Registrar o log na tabela de logs
            cursor.execute("INSERT INTO logTable
```

```
VALUES(%s,%s,%s,%s)",(ns[i,1],date_time,"NS_REPORT","NS Ranked to In
Evaluation (Route: "+novaRota+")"))
            db.commit()


         i = i + 1

      #Retira a flag de interrupção
      date_time = time.strftime("%Y-%m-%d %H-%M-%S")
      cursor.execute("UPDATE severalTable SET
flagInterrupt=%s,lastModify=%s",(0,date_time))
      db.commit()

   else:

      #Executa consulta no banco de dados
      cursor.execute("SELECT address from nsTable where status = 'OP'
or status = 'IN' or status = 'EA'")

      #Pega resultado da execucao e passa para o vetor linhas
      linhas = cursor.fetchall()

      #Fecha conexao com DB
      #cursor.close()

      #Percorre todos os registros encontrados
      for linha in linhas:
         addressNS = linha["address"]

         #Envia um pacote de solicitação de dados para o NS em questão
         sendPackets(addressNS,1)
```

Arquivo: perAvaliator.py

```
# coding=UTF-8
#!/usr/bin/python


import shlex
import sys
import subprocess
import MySQLdb
import serial
import time
from time import localtime, strftime
from datetime import datetime, date
import struct
from timeit import default_timer as timer

from conectaDB import *

#Verifica a PER de todos os NS cadastrados no Banco de Dados com status
OP (Operacional) e IN (Inconsistente)
def verifyAllNSPER():
#Status de retornos:
# 0 : NS testados com sucesso!

   #Cria uma flag para indicar ao processo se vai ser necessário gerar
uma interrupção no final de todas as verificações ou não
   #Começa com 0 para indicar que, a principio, não haverá uma
interrupção
   flagInterruptProcess = 0

   #Verifica se existe alguma tratativa em andamento. Se estiver
ocorrendo, não irá verificar a PER
   cursor.execute("SELECT flagInterrupt from severalTable")
   linha = cursor.fetchone()
   #Pega a flag de interrupção
   flagInterrupt = linha["flagInterrupt"]
   #Verifica se a flag está ativada
   if (flagInterrupt == 1):

      #Aguarda dois segundos antes de prosseguir
      time.sleep(2)

   else:

      #Executa consulta no banco de dados - Irá apenas selecionar os NS
onde o status seja 'Operacional' (OP), 'Inconsistente' (IN) e 'Em
Avaliação' (EA)
      cursor.execute("SELECT
address,maxPER,maxPERPeriod,status,beginEvaluation,externalFunction,flag
AlternativeRoute from nsTable where status = 'OP' or status = 'IN' or
status = 'EA'")

      #Pega resultado da execucao e passa para o vetor linhas
      linhas = cursor.fetchall()

      #Cria vetor para armazenar as informações relacionadas a PER
      #Primeira posição: Armazena o endereço do NS
      #Segunda posição: Armazena a máxima PER do NS
      #Terceira posição: Armazena o período máximo da PER (dado em
```

```
minutos)
      #Quarta posição: Armazena o status atual do NS cadastrado no BD
      #Quinta posiçao: Armazena a PER atual encontrada
      #Sexta posição: Novo status, se necessário (NR = Não Respondendo
ou IN = Inconsistente. Começa com 0
      #Sétima posição: Flag para indicar se a PER do NS será ou não
atualizado no DB independente do seu status (1 = Atualiza ; 0 = Não
atualiza
      #Oitava posição: Flag. Se estiver ativada, indica que o NS
utilizava uma rota alternativa, e esta está com uma PER menor do que o
limite. Caso seja necessário, será ativada no decorrer do algoritmo
      #Nona posição: Armazena o horário inicial do processo de avaliação
do NS, caso o mesmo se encontre neste status
      nsPER = {}

      #Cria variavel para identificar o NS no vetor
      i = 0

      #Percorre todos os NS encontrados, a fim de calcular a PER para os
mesmos
      for linha in linhas:
         nsPER[i,1]  = linha["address"]
         nsPER[i,2]  = linha["maxPER"]
         nsPER[i,3]  = linha["maxPERPeriod"]
         nsPER[i,4]  = linha["status"]
         nsPER[i,6]  = 0
         nsPER[i,7]  = 0
         nsPER[i,9]  = linha["beginEvaluation"]
         nsPER[i,10] = linha["externalFunction"]

         #Calcula a PER para o NS em questão
         #Executa consulta no banco de dados
         cursor.execute("SELECT (select count(status) as pcts_erros from
packetReceivedTable where addressNS = %s and status=%s AND date > NOW()
- INTERVAL %s MINUTE) / (select count(status) as total_pcts from
packetReceivedTable where addressNS = %s AND date > NOW() - INTERVAL %s
MINUTE) * 100 as
per",(nsPER[i,1],'ERRO',nsPER[i,3],nsPER[i,1],nsPER[i,3]))

         resultado = cursor.fetchone()

         #Se o resultado for diferente de nulo...
         if resultado["per"] != None:

            #Armazena a PER no vetor de resultados
            nsPER[i,5] = resultado["per"]

            #Indica na flag que irá ser realizado uma alteração na PER
atual do DB
            nsPER[i,7] = 1

            #Inicia a flag de utilização de rota alternativa com o valor
padrão (0)
            nsPER[i,8] = 0

            #Pega a hora atual
            horaAtual = time.time()

            #Transforma em minutos
            diferencaHora = int(horaAtual - horaInicial)
            minutosExecucao = diferencaHora // 60
```

```
            #Verifica se o NS em questão está em processo de avaliação.
Começa a fazer a analise da PER para estes casos. Se a mesma ultrapassou
o limite, NS será cadastrado como PR (Problematico)
            if (nsPER[i,4] == "EA"):

                #Calcula a PER de forma diferente para o NS em questão: o
período será composto a partir do inicio de seu processo de avaliação
                cursor.execute("SELECT (select count(status) as
pcts_erros from packetReceivedTable where addressNS = %s and status=%s
AND date > %s) / (select count(status) as total_pcts from
packetReceivedTable where addressNS = %s AND date > %s) * 100 as
per",(nsPER[i,1],'ERRO',nsPER[i,9],nsPER[i,1],nsPER[i,9]))
                resultadoEA = cursor.fetchone()
                nsPER[i,5] = resultadoEA["per"]

                #Verifica se o tempo de avaliação do NS em questão já
expirou
                fmt = '%Y-%m-%d %H:%M:%S'
                horario_atualTemp    =
datetime.strptime(time.strftime("%Y-%m-%d %H:%M:%S"), fmt)
                inicio_avaliacaoTemp = datetime.strptime(str(nsPER[i,9]),
fmt)

                horario_atual    =
time.mktime(horario_atualTemp.timetuple())
                inicio_avaliacao =
time.mktime(inicio_avaliacaoTemp.timetuple())

                #Pega o tempo descorrido em minutos
                tempo_descorrido = int(horario_atual-inicio_avaliacao) /
60

                #Se já passou o tempo, é hora de classificar o NS...
                if tempo_descorrido >= nsPER[i,3]:

                   #Se a PER é maior ou igual ao limite, o status do NS
será alterado para PR (Problematico)
                   if (nsPER[i,5] >= nsPER[i,2]):
                      nsPER[i,6] = "PR"
                   #Caso contrário, o status do mesmo será alterado para
OP (Operacional)
                   else:
                      nsPER[i,6] = "OP"
                #Caso contrario, o status do mesmo continua a ser EA
                else:
                   nsPER[i,6] = "EA"

            #Verifica se PER em questão é igual a 100% E o código já foi
executado por pelo menos o período da PER em questão...
            #Classifica o NS em questão como NR (Não Respondendo)
            elif ((nsPER[i,5] == 100) and (minutosExecucao >=
nsPER[i,3])):
                #O status do NS será alterado para NR
                nsPER[i,6] = "NR"

            #Verifica se a PER em questão é maior do que a PER Maxima
            #E o código já foi executado por pelo menos o período da PER
em questão...
            elif ((nsPER[i,5] >= nsPER[i,2]) and (minutosExecucao >=
nsPER[i,3])):
```

```
                #O status do NS será alterado para IN
                nsPER[i,6] = "IN"

            #Caso contrário, se a PER ainda estiver satisfatória, o
status ainda continua sendo OP...
            else:
                nsPER[i,6] = "OP"

         #Caso contrario se o valor da PER é nulo...
         else:
            #Indica que este NS não sofrerá atualização na base
            nsPER[i,7] = 0

         #Incrementa a variavel de identificação do NS para o próximo NS
         i = i + 1

      #Começa o processo de atualização da base com as informações em
questão, para cada NS...
      for x in range(0,i):

         #Se a flag indicar que a atualização deverá ser necessária...
         if nsPER[x,7] == 1:

            #Realiza a atualização na base...
            cursor.execute("UPDATE nsTable SET status=%s,actualPER=%s
WHERE address=%s",(nsPER[x,6],nsPER[x,5],nsPER[x,1]))
            db.commit()

            #Verifica se será necessário registrar a PER na tabela de
registros da PER (a cada 20 segundos)
            if contagem % 20 == 0:

               #Pega data e hora atual
               date_time = time.strftime("%Y-%m-%d %H-%M-%S")

               #Realiza a atualização na base...
               cursor.execute("INSERT INTO historyPERTable
values(%s,%s,%s)",(nsPER[x,1],nsPER[x,5],date_time))
               db.commit()

         #Se o status do NS for IN ou NR,  uma interrupção deverá ser
realizada
         if (nsPER[x,6] == 'IN' or nsPER[x,6] == 'NR'):

            #Aciona a flag de interrupção para este processo, para que,
no final de todas as verificações, a interrupção possa ser cadastrada no
Banco de Dados
            flagInterruptProcess = 1

         #Caso o status do NS for PR (Problematico), um alerta deverá
ser realizado via Whatsapp
         if (nsPER[x,6] == "PR"):

            date_time = time.strftime("%Y-%m-%d %H-%M-%S")

            #Verifica se existe algum script para executar
            if (nsPER[x,10] != 0):

               #Cria string para acionar script externo

string="/var/www/html/bubble/externalFunctions/"+nsPER[x,10]+" 'Relato
```

```
de Falha' '" + date_time + " - NS " + nsPER[x,1] + " cadastrado como PR
(Problematico). Tomar providencias.'"
                #Executa ação externa
                subprocess.call(shlex.split(string))

            cursor.execute("INSERT INTO logTable
VALUES(%s,%s,%s,%s)",(nsPER[x,1],date_time,"ERROR_NS","NS Ranked to
Problematic (PER"+str(nsPER[x,5])+")"))
            db.commit()

      #Caso a flag de interrupção for igual a 1...
      if flagInterruptProcess == 1:

         #Pega data e hora atual
         date_time = time.strftime("%Y-%m-%d %H-%M-%S")

         #Realiza a atualização na base...
         cursor.execute("UPDATE severalTable SET
flagInterrupt=%s,lastModify=%s",(1,date_time))
         db.commit()
         date_time = time.strftime("%Y-%m-%d %H-%M-%S")
         cursor.execute("INSERT INTO logTable
VALUES(%s,%s,%s,%s)",("*",date_time,"INTERRUPTION","An interruption
occurred for maintenance"))
         db.commit()

      #Encerra conexão com o DB
      #cursor.close()
      #db.close()

#Aguarda 6 segundos, para iniciar após a execução do fluxoPrincipal
#MODIFICADO PARA TESTES
time.sleep(6)

#Pega a hora inicial de execução
horaInicial = time.time()

#Cria objeto de conexao com o banco de dados. A função conectaDB retorna
dois valores: db e cursor.
#db é para fazer o commit dos updates
#cursor é para executar os SQLs
db, cursor = conectaDB()

#Execução principal do programa...
contagem = 0
while True:

   contagem = contagem + 1
   time.sleep(1)
   verifyAllNSPER()
```

Arquivo: routeFunctions.py

```
# coding=UTF-8
#!/usr/bin/python

import MySQLdb
import serial
import time
from time import localtime, strftime
from datetime import datetime, date
import struct
from timeit import default_timer as timer

from conectaDB import *

#Descobre todas as rotas dos NS cadastrados no BD
def discoverAllRoutes():
#Códigos de Retorno:
#0 : Tabela atualizada com rotas
#1 : Não foi possível abrir porta serial

   #Cria objeto de conexao com o banco de dados. A função conectaDB
retorna dois valores: db e cursor.
   #db é para fazer o commit dos updates
   #cursor é para executar os SQLs
   db, cursor = conectaDB()

   #Executa consulta no banco de dados
   cursor.execute("SELECT * from severalTable where statusConfig = 1")

   #Pega resultado da execucao e passa para o vetor linha
   linha = cursor.fetchone()

   #Passa os parametros cadastrados no DB para variáveis
   #Cada posição do vetor linha é relacionado a um campo de resultado do
SELECT
   sinkAddress         = linha["sinkAddress"]         #Pega o endereço
do Nó Sorvedouro
   serialPort          = linha["serialPort"]          #Pega a porta
serial cadastrada
   hopLimit            = linha["hopLimit"]            #Pega o limite de
hops
   routeHopWeight      = linha["routeHopWeight"]      #Pega o peso de um
hop
   routeLinkWeightFine = linha["routeLinkWeightFine"] #Pega o indicador
de enlace bom
   routeNumPacketFind  = linha["routeNumPacketFind"]  #Pega o número de
pacotes a ser enviado na descoberta de rotas
   routeWeightGT50     = linha["routeWeightGT50"]     #Peso para enlaces
com RSSI maior que -50dBm
   routeWeightGE5999   = linha["routeWeightGE5999"]   #Peso para enlaces
com RSSI entre -50 e -59.99dBm
   routeWeightGE6999   = linha["routeWeightGE6999"]   #Peso para enlaces
com RSSI entre -60 e -69.99dBm
   routeWeightGE7999   = linha["routeWeightGE7999"]   #Peso para enlaces
com RSSI entre -70 e -79.99dBm
   routeWeightGE8999   = linha["routeWeightGE8999"]   #Peso para enlaces
com RSSI entre -80 e -89.99dBm
   routeWeightGT90     = linha["routeWeightGT90"]     #Peso para enlaces
com RSSI menor que -90dBm
```

```
   #Declarar matriz para receber os dados das rotas da rede
   #A matriz é composta por 5 partes partes:
   #1: Guarda o endereço do nó
   #2: Guarda o status da descoberta da rota. Se for igual a N, então a
rota não foi descoberta. Se for igual a S,  então uma rota já foi
descoberta.
   #3: Guarda a rota, no formato de "subtipo" do campo três. Exemplo: 2°
hop da rota do nó 4 da rede: rotaNosRede[4,3,2]
   #4: Guarda a quantidade de hops da rota em questão.
   #5: Guarda o peso que a rota recebeu. Quanto menor for o valor,
melhor é o mesmo
   rotaNosRede = {}

   #Pega os endereços de todos os nó Sensores cadastrados no banco de
dados
   #Primeiramente, cria um vetor para armazenar os mesmos
   nodesAddress = []
   qtdeNodes = 0

   #Realiza a consulta no banco
   cursor.execute("SELECT address from nsTable")

   #Pega resultado da execucao e passa para a matriz linhas
   linhas = cursor.fetchall()

   #Preenche o vetor nodesAddress com os endereços dos Nó Sensores. Acha
a quantidade de nó sensores
   #através da variavel qtdeNodes
   for linha in linhas:
      nodesAddress.append(linha["address"])
      qtdeNodes = qtdeNodes + 1

   #Cria uma flag para indicar se existem rotas para nós a serem
descobertas
   flagFaltaRotas = True

   #Estabelece o número de hops atuais, que deve ser 0, pois está
começando agora.
   hopsAtuais = 0

   #Flag para indicar que já existem rotas cadastradas. Caso não exista
nenhuma (como é o caso neste momento), ela
   #seguirá valendo False. Caso uma rota seja cadastrada, ela valerá
True
   flagJaExistemRotas = False

   #Inicializa o vetor, de acordo com a descrição da matriz identificada
acima
   for i in range(1,qtdeNodes + 1):
      rotaNosRede[i,1] = nodesAddress[i-1]
      #Rotas ainda não foram descobertas, então é identificado como 'N'
      rotaNosRede[i,2] = 'N'
      #Estabelece um valor nulo para as rotas
      rotaNosRede[i,3] = None
      #Estabelece a quantidade de hops para os nós
      rotaNosRede[i,4] = 0
      #Estabelece 10000 como peso primário da rota (10000 : rota ainda
não descoberta)
      rotaNosRede[i,5] = 10000

   #Cria objeto para porta serial
```

```
   try:
      serialPort = serial.Serial(serialPort, 9600, timeout=0.5,
parity=serial.PARITY_NONE)
   except Exception:
      #Retorna status de erro - Não foi possível abrir a porta serial
(Código 1)
      return 1

   #Cria uma flag para indicar se no hop atual (a partir do hop 2) não
foi descoberto nenhum nó sensor. Caso não tenha sido
   #descoberto, o programa para de procurar nos hops subsequentes, pois
não existe caminho para os sensores. Caso a flag
   #tenha valor True, significa que será realizado tentativas nos hops
subsequentes. Caso armazene False, a procura para
   #nos hops subsequentes.
   flagContinue = True


   #Repetição principal do programa. Enquanto o número de hops atuais
não atingir seu limite, ou enquanto ainda
   #existirem rotas a serem descobertas, o programa continua sendo
executado
   while (hopsAtuais+1 <= hopLimit) and (flagFaltaRotas and
flagContinue):

      #Parte do presuposto que não será encontrado nenhum nó neste hop.
Caso o mesmo encontre, ele mudará para True
      #e continuará analisando. Caso não encontre, ele terminará a
análise neste hop atual.
      flagContinue = False

      #Incrementa o hop atual
      hopsAtuais = hopsAtuais + 1

      #Quantidade de rotas já descobertas no momento
      quantidadeRotasDescobertas = 0

      #Quantidade de tentativas erradas de descobertas
      quantidadeTentativasErradas = 0

      #Começa o procedimento para descoberta das rotas dos nós. Vai
percorrer todos os nós, a principio.
      #Caso algum nó já tenha uma rota estabelecida (Cadastrada como
'S') e considerada OK, ele será pulado.
      for i in range(1,qtdeNodes+1):

         #Verifica se o nó em questão possui um peso maior do que o
limiar (se o peso é excelente ou não. Só vai
         #continuar se não for excelente.)
         if rotaNosRede[i,5] > routeLinkWeightFine * hopsAtuais - 1 +
hopsAtuais - 1 * routeHopWeight:

            #Cria um pacote de 52 bytes vazio para ser enviado
            pacoteSX = {}
            for j in range(0,52):
               pacoteSX[j] = 0

            #Começa a configurar o pacote que será enviado.
            #Configura o byte de posição 10 (referente ao endereço de
origem) com o endereço da base
            pacoteSX[10] = sinkAddress
```

```
            #Configura o byte na posição 11 (referente a solicitação de
roteamento) com o valor 128
            pacoteSX[11] = 128

            #Configura o byte na posição 23 (referente ao limite máximo
de hops que será
            #realizado nessa rodada). Este limite será incrementado em
cada hop
            pacoteSX[23] = hopsAtuais

            #Configura o byte na posição 24 (referente a quantidade de
hops atuais no ato de descoberta) Estabelece o
            #número de hops atuais para 0, já que o pacote está saindo
agora da base. Quando o mesmo for recebido por um
            #outro nó, o mesmo será incrementado em 1.
            pacoteSX[24] = 0

            #Configura o byte na posição 22 (referente ao endereço que
deve ser descoberto)
            pacoteSX[22] = rotaNosRede[i,1]

            #Se for o primeiro hop, a tratativa para enviar os pacotes é
diferente, já que não se sabe quem
            #são os nós diretamente ligados a base. Sendo assim, o
endereço de descoberta será igual ao endereço
            #do destinatário
            if hopsAtuais == 1:

               #Configura o byte de posição 8 (referente ao endereço de
destino) com o endereço do nó
               pacoteSX[8] = rotaNosRede[i,1]

               #Inicia a variavel flagPrimeiraVez como True. Esta flag
vai dizer quando o pacote for recebido da primeira vez, das tentativas
de entrega mais a baixo
               flagPrimeiraVez = True

               #Tenta realizar diversas tentativas de envio de pacote.
Quando um pacote for recebido,
               #as tentativas param imediatamente.
               for k in range(0,routeNumPacketFind):

                  #Limpa os buffers de entrada e saída da serial
                  serialPort.flushInput()
                  serialPort.flushOutput()

                  #Transmite o pacote na serial
                  for j in range(0,52):
                     serialPort.write(chr(int(pacoteSX[j])))

                  #Espera um tempo para que o pacote seja transmitido e
recebido novamente
                  time.sleep(0.1)

                  #Le a serial em busca do pacote
                  pacoteRX = serialPort.read(52)

                  #Verifica se o pacote está integro e se é um retorno
de rota (valor 64)
                  if len(pacoteRX) == 52 and ord(pacoteRX[11]) == 64:
```

```
                        #Já que pacote foi retornado, coloca a flag
flagContinue para TRUE, ou seja, vai tentar nos
                        #próximos hops
                        flagContinue = True

                        #Endereço do nó que está conectado ao nó final (é o
endereço do próprio nó em questão)
                        enderecoNoFinal = ord(pacoteRX[25])

                        #Calculo dos RSSI de Download e Upload
                        tempRSSIDownload = ord(pacoteRX[0])

                        if tempRSSIDownload > 128:
                           rssiDownload =((tempRSSIDownload-256)/2.0)-74
                        else:
                           rssiDownload =(tempRSSIDownload/2.0)-74


                        #Estabelece o peso da rota (com base no rssi de
Download)
                        if rssiDownload > -50:
                           pesoRota = routeWeightGT50
                        elif rssiDownload < -50 and rssiDownload >= -59.99:
                           pesoRota = routeWeightGE5999
                        elif rssiDownload < -60 and rssiDownload >= -69.99:
                           pesoRota = routeWeightGE6999
                        elif rssiDownload < -70 and rssiDownload >= -79.99:
                           pesoRota = routeWeightGE7999
                        elif rssiDownload < -80 and rssiDownload >= -89.99:
                           pesoRota = routeWeightGE8999
                        elif rssiDownload < -90:
                           pesoRota = routeWeightGT90

                        #Cadastra a rota para o nó como já descoberta
                        rotaNosRede[i,2] = 'S'

                        #Grava a rota para o nó em questão, que é o
endereço do próprio nó
                        rotaNosRede[i,3,1] = rotaNosRede[i,1]

                        #Grava a quantidade de hops para o nó em questão
                        rotaNosRede[i,4] = hopsAtuais

                        #Grava o peso da rota
                        if flagPrimeiraVez == True:
                           rotaNosRede[i,5] = pesoRota
                           flagPrimeiraVez = False
                        else:
                           if pesoRota < rotaNosRede[i,5]:
                              rotaNosRede[i,5] = pesoRota

                        #Configura uma flag para indicar que já existem
rotas
                        flagJaExistemRotas = True


            #Caso contrário, ou seja, o número de hops não é igual a 1,
a escrita do pacote é diferente.
            #Também é verificado se já existem rotas, pois se não
existir, o algoritmo não irá continuar
```

```
            #(pois não existem nós diretamente ligados ao proxy-
gateway).
            elif flagJaExistemRotas:

               #Percorre toda a matriz de rotas, procurando quem possua
rotas cadastradas, para
               #perguntar aos mesmos se estão conectados ao nó em
questão
               for a in range(1,qtdeNodes+1):

                  #Se o no em questão já possui uma rota já cadastrada,
se não é uma tentativa do próprio sensor tentar
                  #descobrir uma rota dele mesmo e também se o número de
hops atual - 1 é igual ao número de hops do sensor.
                  if rotaNosRede[a,2] == 'S' and rotaNosRede[a,1] !=
rotaNosRede[i,1] and rotaNosRede[a,4] == hopsAtuais-1:

                     #Pega o número de hops para o nó em questão...
                     numHops = rotaNosRede[a,4]
                     #Pega o endereço do primeiro nó na rota, e
configura como destino...
                     pacoteSX[8] = rotaNosRede[a,3,1]
                     #Pega o peso da rota em questão
                     pesoRotaTemp = rotaNosRede[a,5]
                     #E por fim, configura a rota no pacote e em uma
variavel para exibição...
                     exibicaoRota = ""
                     for j in range(1,numHops+1):
                        pacoteSX[27+j] = rotaNosRede[a,3,j]
                        exibicaoRota = exibicaoRota +
str(rotaNosRede[a,3,j]) + ','
                     exibicaoRota = exibicaoRota + str(rotaNosRede[i,1])

                     #Inicia a variavel flagPrimeiraVez como True. Esta
flag vai dizer quando o pacote for recebido da primeira vez, das
tentativas de
                     #entrega mais a baixo
                     flagPrimeiraVez = True

                     #Tenta realizar diversas tentativas de envio de
pacote. Quando um pacote for recebido, as
                     #tentativas param imediatamente.
                     for k in range(0,routeNumPacketFind):

                        #Limpa os buffers de entrada e saída da serial
                        serialPort.flushInput()
                        serialPort.flushOutput()

                        #Transmite o pacote na serial
                        for j in range(0,52):
                           serialPort.write(chr(int(pacoteSX[j])))

                        #Espera um tempo para que o pacote seja
transmitido e recebido novamente
                        time.sleep(0.1)

                        #Le a serial em busca do pacote
                        pacoteRX = serialPort.read(52)

                        #Verifica se o pacote está integro e se é um
retorno de rota
```

```
                        if len(pacoteRX) == 52 and ord(pacoteRX[11]) ==
64:

                            #Já que pacote foi retornado, coloca a flag
flagContinue para TRUE, ou seja, vai tentar nos
                            #próximos hops
                            flagContinue = True

                            #Endereço do nó que está conectado ao nó
final
                            enderecoNoFinal = ord(pacoteRX[25])

                            #Calculo dos RSSI de Download e Upload
                            tempRSSIDownload = ord(pacoteRX[26])

                            if tempRSSIDownload > 128:
                               rssiDownload =((tempRSSIDownload-
256)/2.0)-74
                            else:
                               rssiDownload =(tempRSSIDownload/2.0)-74

                            #Configura a nota da rota em questão
                            #Primeiro, aumenta a nota relacionado aos
hops
                            pesoRotaTemp = pesoRotaTemp + routeHopWeight

                            #Depois, aumenta o peso baseado no RSSI atual
                            if rssiDownload > -50:
                               pesoRotaTemp = pesoRotaTemp +
routeWeightGT50
                            elif rssiDownload < -50 and rssiDownload >= -
59.99:
                               pesoRotaTemp = pesoRotaTemp +
routeWeightGE5999
                            elif rssiDownload < -60 and rssiDownload >= -
69.99:
                               pesoRotaTemp = pesoRotaTemp +
routeWeightGE6999
                            elif rssiDownload < -70 and rssiDownload >= -
79.99:
                               pesoRotaTemp = pesoRotaTemp +
routeWeightGE7999
                            elif rssiDownload < -80 and rssiDownload >= -
89.99:
                               pesoRotaTemp = pesoRotaTemp +
routeWeightGE8999
                            elif rssiDownload < -90:
                               pesoRotaTemp = pesoRotaTemp +
routeWeightGT90

                            #Grava o peso da rota
                            if flagPrimeiraVez == True:
                               pesoRota = pesoRotaTemp
                               flagPrimeiraVez = False
                            else:
                               if pesoRotaTemp < pesoRota:
                                  pesoRota = pesoRotaTemp


                            #Verifica se essa nova rota é melhor que a
antiga já cadastrada. Caso não exista
```

```
                              #rota cadastrada, mesmo assim, ele irá
cadastrar, pois a nota inicial é 10000
                              if pesoRota < rotaNosRede[i,5]:

                                 #Grava a nova nota da rota
                                 rotaNosRede[i,5] = pesoRota

                                 #Grava a descoberta da rota
                                 rotaNosRede[i,2] = 'S'


                                 posicaoGravacao=1
                                 #A rota fica armazenada nas posições de 28
até 33...A rota será gravada até a quantidade
                                 #de hops atuais + 28. Exemplo: hops atuais
= 3 ; 28 + 3 = 31. A rota será gravada nas
                                 #posições 28, 29 e 30 (lembrando que o
ultimo valor não conta no for)
                                 for a in range(28,hopsAtuais+28):
                                    #Se for a ultima posição de
armazenamento da rota, armazena o endereço do nó de destino
                                    if a == hopsAtuais+27:
                                       rotaNosRede[i,3,posicaoGravacao] =
rotaNosRede[i,1]
                                    #Caso contrário, armazena a rota do nó
antigo, armazenada no pacote que foi recebido
                                    else:
                                       rotaNosRede[i,3,posicaoGravacao] =
ord(pacoteRX[a])
                                    posicaoGravacao = posicaoGravacao + 1

                                 #Grava a quantidade de hops para o nó em
questão
                                 rotaNosRede[i,4] = hopsAtuais

                              #Para as tentativas de comunicação com o
sensor em questão
                              break


      qtdeRotasFaltantes = 0
      #Verifica se existe alguma rota ainda para ser descoberta. Caso
alguma rota esteja cadastrada como N - Não existe rota OU possua um peso
que por ventura possa ser melhorado, o processo continua.
      for i in range(1,qtdeNodes + 1):
         if rotaNosRede[i,2] == 'N' or rotaNosRede[i,5] >
routeLinkWeightFine * rotaNosRede[i,4]  + routeHopWeight *
rotaNosRede[i,4]-1:
            qtdeRotasFaltantes = qtdeRotasFaltantes + 1
      if qtdeRotasFaltantes == 0:
         flagFaltaRotas = False

   #Imprime resultado final
   for i in range(1,qtdeNodes + 1):
      rota = ""
      for j in range(1,rotaNosRede[i,4]+1):
         if j == rotaNosRede[i,4]:
            rota = rota + str(rotaNosRede[i,3,j])
         else:
            rota = rota + str(rotaNosRede[i,3,j]) + ','
      print 'Rota para o Nó ',rotaNosRede[i,1], ':'
```

```
      print '\tNumero de Hops:', rotaNosRede[i,4]
      print '\tRota: ',rota
      print '\tPeso da Rota:', rotaNosRede[i,5]
      if rotaNosRede[i,2] == "S":
         cursor.execute("UPDATE nsTable SET
status=%s,route=%s,flagAlternativeRoute=%s,hops=%s,weightRoute=%s where
address=%s",("OP",rota,0,rotaNosRede[i,4],rotaNosRede[i,5],rotaNosRede[i
,1]))
         db.commit()
         date_time = time.strftime("%Y-%m-%d %H-%M-%S")
         cursor.execute("INSERT INTO logTable
VALUES(%s,%s,%s,%s)",(rotaNosRede[i,1],date_time,"ROUTE_FIND","Route
found in the All Route Discovery Process ("+rota+")"))
         db.commit()

      else:
         cursor.execute("UPDATE nsTable SET
status=%s,route=%s,hops=%s,weightRoute=%s where
address=%s",("NR",'',0,10000,rotaNosRede[i,1]))
         db.commit()
         date_time = time.strftime("%Y-%m-%d %H-%M-%S")
         cursor.execute("INSERT INTO logTable
VALUES(%s,%s,%s,%s)",(rotaNosRede[i,1],date_time,"ROUTE_ERROR","No route
found in the All Route Discovery Process"))
         db.commit()

   #Fecha conexao com DB
   cursor.close()
   db.close()

   #Retorna valor 0 para indicar que a execução terminou
   return 0



def discoverSingleRoute(nodeAddress):
#Códigos de Retorno:
#rota: Retorna a rota encontrada caso tenha obtido sucesso
#1 : Não foi possível abrir porta serial
#N/A : Nenhuma rota foi descoberta

   #Cria objeto de conexao com o banco de dados. A função conectaDB
retorna dois valores: db e cursor.
   #db é para fazer o commit dos updates
   #cursor é para executar os SQLs
   db, cursor = conectaDB()

   #Executa consulta no banco de dados
   cursor.execute("SELECT * from severalTable where statusConfig = 1")

   #Pega resultado da execucao e passa para o vetor linha
   linha = cursor.fetchone()

   #Passa os parametros cadastrados no DB para variáveis
   #Cada posição do vetor linha é relacionado a um campo de resultado do
SELECT
   sinkAddress          = linha["sinkAddress"]          #Pega o endereço
do Nó Sorvedouro
   serialPort           = linha["serialPort"]           #Pega a porta
serial cadastrada
   hopLimit             = linha["hopLimit"]             #Pega o limite de
```

```
hops
   routeHopWeight      = linha["routeHopWeight"]      #Pega o peso de um
hop
   routeLinkWeightFine = linha["routeLinkWeightFine"] #Pega o indicador
de enlace bom
   routeNumPacketFind  = linha["routeNumPacketFind"]  #Pega o número de
pacotes a ser enviado na descoberta de rotas
   routeWeightGT50     = linha["routeWeightGT50"]     #Peso para enlaces
com RSSI maior que -50dBm
   routeWeightGE5999   = linha["routeWeightGE5999"]   #Peso para enlaces
com RSSI entre -50 e -59.99dBm
   routeWeightGE6999   = linha["routeWeightGE6999"]   #Peso para enlaces
com RSSI entre -60 e -69.99dBm
   routeWeightGE7999   = linha["routeWeightGE7999"]   #Peso para enlaces
com RSSI entre -70 e -79.99dBm
   routeWeightGE8999   = linha["routeWeightGE8999"]   #Peso para enlaces
com RSSI entre -80 e -89.99dBm
   routeWeightGT90     = linha["routeWeightGT90"]     #Peso para enlaces
com RSSI menor que -90dBm

   #Vetor rotaNosRede: vetor com rotas e informações relacionadas

   #Declarar vetor para receber os dados das rotas da rede
   #O vetor é composto por 5 partes partes:
   #1: Guarda o endereço do nó
   #2: Guarda o status da descoberta da rota. Se for igual a N, então a
rota não foi descoberta. Se for igual a S,  então uma rota já foi
descoberta.
   #3: Guarda a rota, no formato de "subtipo" do campo três. Exemplo: 2°
hop da rota do nó 4 da rede: rotaNoRede[3,2]
   #4: Guarda a quantidade de hops da rota em questão.
   #5: Guarda o peso que a rota recebeu. Quanto menor for o valor,
melhor é o mesmo
   rotaNoRede = {}

   #Matriz rotascadastradas: matriz para armazenar as rotas do banco de
dados
   rotasCadastradas = {}

   #Realiza pesquisa no banco por NS com rotas já cadastradas
   cursor.execute("SELECT address,route,hops,weightRoute from nsTable
where status = 'OP'")

   #Pega resultado da execucao e passa para a matriz linhas
   linhas = cursor.fetchall()

   #Preenche o vetor rotasCadastradas com os endereços dos Nó Sensores.
Acha a quantidade de nó sensores
   #através da variavel qtdeNodes
   qtdeNodes=0
   for linha in linhas:
      rotasCadastradas[qtdeNodes,1] = linha["address"]
      rotasCadastradas[qtdeNodes,2] = "S"

      #Divide a string da rota cadastrada, tendo como base a divisão dos
elementos (uma virgula - , )
      rota = linha["route"].split(",")
      i=1
      for elemento_rota in rota:
         rotasCadastradas[qtdeNodes,3,i] = elemento_rota
         i = i+1
```

```
      rotasCadastradas[qtdeNodes,4] = linha["hops"]
      rotasCadastradas[qtdeNodes,5] = linha["weightRoute"]
      qtdeNodes = qtdeNodes + 1


   #Cria uma flag para indicar se continuará ou não a procurar por
rotas. Começa valendo True, ou seja, vai procurar por rotas
   flagFaltaRotas = True

   #Estabelece o número de hops atuais, que deve ser 0, pois está
começando agora.
   hopsAtuais = 0

   #Preenche o vetor rotaNoRede com valores padrões.
   rotaNoRede[1] = nodeAddress
   #Rotas ainda não foram descobertas, então é identificado como 'N'
   rotaNoRede[2] = 'N'
   #Estabelece um valor nulo para as rotas
   rotaNoRede[3] = None
   #Estabelece a quantidade de hops para os nós
   rotaNoRede[4] = 0
   #Estabelece 10000 como peso primário da rota (10000 : rota ainda não
descoberta)
   rotaNoRede[5] = 10000

   #Cria objeto para porta serial
   try:
      serialPort = serial.Serial(serialPort, 9600, timeout=0.5,
parity=serial.PARITY_NONE)
   except Exception:
      #Retorna status de erro - Não foi possível abrir a porta serial
(Código 1)
      return 1

   #Repetição principal do programa. Enquanto o número de hops atuais
não atingir seu limite, ou enquanto ainda
   #existirem rotas a serem descobertas, o programa continua sendo
executado
   while (hopsAtuais+1 <= hopLimit) and flagFaltaRotas:

      #Incrementa o hop atual
      hopsAtuais = hopsAtuais + 1

      #Quantidade de rotas já descobertas no momento
      quantidadeRotasDescobertas = 0

      #Quantidade de tentativas erradas de descobertas
      quantidadeTentativasErradas = 0


      #Verifica se o nó em questão possui um peso maior do que o limiar
(se o peso é excelente ou não. Só vai
      #continuar se não for excelente.)
      if rotaNoRede[5] > routeLinkWeightFine * hopsAtuais - 1 +
hopsAtuais - 1 * routeHopWeight:

         #Cria um pacote de 52 bytes vazio para ser enviado
         pacoteSX = {}
         for j in range(0,52):
            pacoteSX[j] = 0
```

```
         #Começa a configurar o pacote que será enviado.
         #Configura o byte de posição 10 (referente ao endereço de
origem) com o endereço da base
         pacoteSX[10] = sinkAddress

         #Configura o byte na posição 11 (referente a solicitação de
roteamento) com o valor 128
         pacoteSX[11] = 128

         #Configura o byte na posição 23 (referente ao limite máximo de
hops que será
         #realizado nessa rodada). Este limite será incrementado em cada
hop
         pacoteSX[23] = hopsAtuais

         #Configura o byte na posição 24 (referente a quantidade de hops
atuais no ato de descoberta) Estabelece o
         #número de hops atuais para 0, já que o pacote está saindo
agora da base. Quando o mesmo for recebido por um
         #outro nó, o mesmo será incrementado em 1.
         pacoteSX[24] = 0

         #Configura o byte na posição 22 (referente ao endereço que deve
ser descoberto)
         pacoteSX[22] = rotaNoRede[1]

         #Se for o primeiro hop, a tratativa para enviar os pacotes é
diferente, já que não se sabe quem
         #são os nós diretamente ligados a base. Sendo assim, o endereço
de descoberta será igual ao endereço
         #do destinatário
         if hopsAtuais == 1:

            #Configura o byte de posição 8 (referente ao endereço de
destino) com o endereço do nó
            pacoteSX[8] = rotaNoRede[1]

            #Inicia a variavel flagPrimeiraVez como True. Esta flag vai
dizer quando o pacote for recebido da primeira vez, das tentativas de
entrega mais a baixo
            flagPrimeiraVez = True

            #Tenta realizar diversas tentativas de envio de pacote.
Quando um pacote for recebido,
            #as tentativas param imediatamente.
            for k in range(0,routeNumPacketFind):

               #Limpa os buffers de entrada e saída da serial
               serialPort.flushInput()
               serialPort.flushOutput()

               #Transmite o pacote na serial
               for j in range(0,52):
                  serialPort.write(chr(int(pacoteSX[j])))

               #Espera um tempo para que o pacote seja transmitido e
recebido novamente
               time.sleep(0.1)

               #Le a serial em busca do pacote
```

```
                pacoteRX = serialPort.read(52)

                #Verifica se o pacote está integro e se é um retorno de
rota (valor 64)
                if len(pacoteRX) == 52 and ord(pacoteRX[11]) == 64:

                    #Já que pacote foi retornado, coloca a flag
flagContinue para TRUE, ou seja, vai tentar nos
                    #próximos hops
                    flagContinue = True

                    #Endereço do nó que está conectado ao nó final (é o
endereço do próprio nó em questão)
                    enderecoNoFinal = ord(pacoteRX[25])

                    #Calculo dos RSSI de Download e Upload
                    tempRSSIDownload = ord(pacoteRX[0])

                    if tempRSSIDownload > 128:
                        rssiDownload =((tempRSSIDownload-256)/2.0)-74
                    else:
                        rssiDownload =(tempRSSIDownload/2.0)-74


                    #Estabelece o peso da rota (com base no rssi de
Download)
                    if rssiDownload > -50:
                        pesoRota = routeWeightGT50
                    elif rssiDownload < -50 and rssiDownload >= -59.99:
                        pesoRota = routeWeightGE5999
                    elif rssiDownload < -60 and rssiDownload >= -69.99:
                        pesoRota = routeWeightGE6999
                    elif rssiDownload < -70 and rssiDownload >= -79.99:
                        pesoRota = routeWeightGE7999
                    elif rssiDownload < -80 and rssiDownload >= -89.99:
                        pesoRota = routeWeightGE8999
                    elif rssiDownload < -90:
                        pesoRota = routeWeightGT90

                    #Cadastra a rota para o nó como já descoberta
                    rotaNoRede[2] = 'S'

                    #Grava a rota para o nó em questão, que é o endereço
do próprio nó
                    rotaNoRede[3,1] = rotaNoRede[1]

                    #Grava a quantidade de hops para o nó em questão
                    rotaNoRede[4] = hopsAtuais

                    #Grava o peso da rota
                    if flagPrimeiraVez == True:
                        rotaNoRede[5] = pesoRota
                        flagPrimeiraVez = False
                    else:
                        if pesoRota < rotaNoRede[5]:
                            rotaNoRede[5] = pesoRota

                    #Configura uma flag para indicar que já existem rotas
                    flagJaExistemRotas = True
```

```
         #Caso contrário, ou seja, o número de hops não é igual a 1, a
escrita do pacote é diferente.
         else:

            #Percorre toda a matriz de rotas, procurando quem possua
rotas cadastradas, para
            #perguntar aos mesmos se estão conectados ao nó em questão
            for a in range(0,qtdeNodes):

               #Pega o número de hops para o nó em questão...
               numHops = rotasCadastradas[a,4]
               #Pega o endereço do primeiro nó na rota, e configura como
destino...
               pacoteSX[8] = rotasCadastradas[a,3,1]
               #Pega o peso da rota em questão
               pesoRotaTemp = rotasCadastradas[a,5]
               #E por fim, configura a rota no pacote...
               for j in range(1,numHops+1):
                  pacoteSX[27+j] = rotasCadastradas[a,3,j]


               #Inicia a variavel flagPrimeiraVez como True. Esta flag
vai dizer quando o pacote for recebido da primeira vez, das tentativas
de
               #entrega mais a baixo
               flagPrimeiraVez = True

               #Tenta realizar diversas tentativas de envio de pacote.
Quando um pacote for recebido, as
               #tentativas param imediatamente.
               for k in range(0,routeNumPacketFind):

                  #Limpa os buffers de entrada e saída da serial
                  serialPort.flushInput()
                  serialPort.flushOutput()

                  #Transmite o pacote na serial
                  for j in range(0,52):
                     serialPort.write(chr(int(pacoteSX[j])))

                  #Espera um tempo para que o pacote seja transmitido e
recebido novamente
                  time.sleep(0.1)

                  #Le a serial em busca do pacote
                  pacoteRX = serialPort.read(52)

                  #Verifica se o pacote está integro e se é um retorno
de rota
                  if len(pacoteRX) == 52 and ord(pacoteRX[11]) == 64:

                     #Já que pacote foi retornado, coloca a flag
flagContinue para TRUE, ou seja, vai tentar nos
                     #próximos hops
                     flagContinue = True

                     #Endereço do nó que está conectado ao nó final
                     enderecoNoFinal = ord(pacoteRX[25])

                     #Calculo dos RSSI de Download e Upload
                     tempRSSIDownload = ord(pacoteRX[26])
```

```
                        if tempRSSIDownload > 128:
                           rssiDownload =((tempRSSIDownload-256)/2.0)-74
                        else:
                           rssiDownload =(tempRSSIDownload/2.0)-74

                        #Configura a nota da rota em questão
                        #Primeiro, aumenta a nota relacionado aos hops
                        pesoRotaTemp = pesoRotaTemp + routeHopWeight

                        #Depois, aumenta o peso baseado no RSSI atual
                        if rssiDownload > -50:
                           pesoRotaTemp = pesoRotaTemp + routeWeightGT50
                        elif rssiDownload < -50 and rssiDownload >= -59.99:
                           pesoRotaTemp = pesoRotaTemp + routeWeightGE5999
                        elif rssiDownload < -60 and rssiDownload >= -69.99:
                           pesoRotaTemp = pesoRotaTemp + routeWeightGE6999
                        elif rssiDownload < -70 and rssiDownload >= -79.99:
                           pesoRotaTemp = pesoRotaTemp + routeWeightGE7999
                        elif rssiDownload < -80 and rssiDownload >= -89.99:
                           pesoRotaTemp = pesoRotaTemp + routeWeightGE8999
                        elif rssiDownload < -90:
                           pesoRotaTemp = pesoRotaTemp + routeWeightGT90

                        #Grava o peso da rota
                        if flagPrimeiraVez == True:
                           pesoRota = pesoRotaTemp
                           flagPrimeiraVez = False
                        else:
                           if pesoRotaTemp < pesoRota:
                              pesoRota = pesoRotaTemp


                        #Verifica se essa nova rota é melhor que a antiga
já cadastrada. Caso não exista
                        #rota cadastrada, mesmo assim, ele irá cadastrar,
pois a nota inicial é 10000
                        if pesoRota < rotaNoRede[5]:

                           #Grava a nova nota da rota
                           rotaNoRede[5] = pesoRota

                           #Grava a descoberta da rota
                           rotaNoRede[2] = 'S'


                           posicaoGravacao=1
                           #A rota fica armazenada nas posições de 28 até
33...A rota será gravada até a quantidade
                           #de hops atuais + 28. Exemplo: hops atuais = 3 ;
28 + 3 = 31. A rota será gravada nas
                           #posições 28, 29 e 30 (lembrando que o ultimo
valor não conta no for)
                           for a in range(28,hopsAtuais+28):
                              #Se for a ultima posição de armazenamento da
rota, armazena o endereço do nó de destino
                              if a == hopsAtuais+27:
                                 rotaNoRede[3,posicaoGravacao] =
rotaNoRede[1]
                              #Caso contrário, armazena a rota do nó
antigo, armazenada no pacote que foi recebido
```

```
                                else:
                                    rotaNoRede[3,posicaoGravacao] =
ord(pacoteRX[a])
                                posicaoGravacao = posicaoGravacao + 1

                            #Grava a quantidade de hops para o nó em questão
                            rotaNoRede[4] = hopsAtuais

                        #Para as tentativas de comunicação com o sensor em
questão
                        break

      if rotaNoRede[2] == 'N' or rotaNoRede[5] > routeLinkWeightFine *
rotaNoRede[4]  + routeHopWeight * rotaNoRede[4]-1:
         flagFaltaRotas = True
      else:
         flagFaltaRotas = False

   #Se alguma rota foi descoberta para o NS em questão, realiza o Update
da mesma na tabela
   if rotaNoRede[2] == 'S':
      rota = ""
      for j in range(1,rotaNoRede[4]+1):
         if j == rotaNoRede[4]:
            rota = rota + str(rotaNoRede[3,j])
         else:
            rota = rota + str(rotaNoRede[3,j]) + ','
      cursor.execute("UPDATE nsTable SET route=%s,hops=%s,weightRoute=%s
where address=%s",(rota,rotaNoRede[4],rotaNoRede[5],rotaNoRede[1]))
      db.commit()
      date_time = time.strftime("%Y-%m-%d %H-%M-%S")
      cursor.execute("INSERT INTO logTable
VALUES(%s,%s,%s,%s)",(rotaNoRede[1],date_time,"ROUTE_FIND","Route found
in the Single Route Discovery Process ("+rota+")"))
      db.commit()

      #Fecha conexao com DB
      cursor.close()

      #Retorna 0 (status positivo)
      return rota

   #Caso contrário, o pacote não tenha sido recebido
   else:
      cursor.execute("UPDATE nsTable SET
status=%s,route=%s,hops=%s,weightRoute=%s where
address=%s",("NR",'',0,10000,rotaNoRede[1]))
      db.commit()
      date_time = time.strftime("%Y-%m-%d %H-%M-%S")
      cursor.execute("INSERT INTO logTable
VALUES(%s,%s,%s,%s)",(rotaNoRede[1],date_time,"ROUTE_ERROR","No route
found in the Single Route Discovery Process"))
      db.commit()

      #Fecha conexao com DB
      cursor.close()
      db.close()
      #Retorna valor 0 para indicar que a execução terminou
      return "N/A"
```

Arquivo: collectFunctions.py

```
# coding=UTF-8
#!/usr/bin/python

import MySQLdb
import serial
import time
from time import localtime, strftime
from datetime import datetime, date
import struct
from timeit import default_timer as timer

from conectaDB import *

#Envia numPackets pacotes de coleta para determinado NS
def sendPackets(nodeAddress,numPackets):
#Códigos de retorno:
#0 : Pacotes enviados com sucesso
#1 : Não foi possível abrir a porta serial
#2 : NS não cadastrado ou status diferente de OP (Operacional)

   #Cria objeto de conexao com o banco de dados. A função conectaDB
retorna dois valores: db e cursor.
   #db é para fazer o commit dos updates
   #cursor é para executar os SQLs
   db, cursor = conectaDB()

   #Executa consulta no banco de dados
   cursor.execute("SELECT * from severalTable where statusConfig = 1")

   #Pega resultado da execucao e passa para o vetor linha
   linha = cursor.fetchone()

   #Passa os parametros cadastrados no DB para variáveis
   #Cada posição do vetor linha é relacionado a um campo de resultado do
SELECT
   sinkAddress = linha["sinkAddress"]          #Pega o endereço do Nó
Sorvedouro
   serialPort  = linha["serialPort"]           #Pega a porta serial
cadastrada

   #Cria objeto para porta serial
   try:
      serialPort = serial.Serial(serialPort, 9600, timeout=0.5,
parity=serial.PARITY_NONE)
   except Exception:
      #Retorna status de erro - Não foi possível abrir a porta serial
(Código 1)
      return 1

   #Verifica se NS está cadastrado no Banco. Em caso negativo, retorna
erro
   cursor.execute("SELECT count(*) as qtde from nsTable where address =
%s and ( status = %s OR status = %s OR status = %s
)",(nodeAddress,"OP","EA","IN"))
   resultado = cursor.fetchone()
   if resultado["qtde"] != 1:
      return 2

   #Caso contrário, irá preparar o pacote para solicitar dados
   else:
```

```
      #Cria um pacote de 52 bytes vazio para ser enviado
      pacoteSX = {}
      for j in range(0,52):
         pacoteSX[j] = 0

      #Consulta a rota e a quantidade de hops para o NS
      cursor.execute("SELECT route,hops from nsTable where address =
%s",(str(nodeAddress)))
      resultado = cursor.fetchone()
      rota = resultado["route"].split(",")
      hops = resultado["hops"]

      #Configura o byte de posição 10 (referente ao endereço de origem)
com o endereço do Nó Sorvedouro
      pacoteSX[10] = sinkAddress

      #Configura o byte na posição 11 (referente a solicitação de dados)
com o valor 254
      pacoteSX[11] = 254

      #Pega o endereço do primeiro nó na rota, e configura como destino
      pacoteSX[8] = rota[0]

      #Configura o numero de hops para o nó em questão, e configura o
numero de hops inicial como 0
      pacoteSX[23] = hops
      pacoteSX[24] = 0

      #Configura o endereço de destino final
      pacoteSX[22] = nodeAddress


      #E por fim configura a rota para o mesmo
      for j in range(1,hops+1):
         pacoteSX[27+j] = rota[j-1]

      #Cria vetores para armazenar as informações referente ao dados dos
sensores do NS
      byteType         = {}
      byteINT          = {}
      byteFRACT        = {}
      expression       = {}
      idSensorNSVector = {}

      #Consulta os dados referente aos sensores para o NS que estão
cadastrados no BD
      cursor.execute("SELECT
idSensorNS,byteType,byteINT,byteFRACT,expression from nsSensorTable
where addressNS = %s and status = %s",(nodeAddress,"ON"))
      linhas = cursor.fetchall()
      qtde_sensores = 0
      for linha in linhas:
         qtde_sensores = qtde_sensores+1
         idSensorNSVector[qtde_sensores] = linha["idSensorNS"]
         byteType[qtde_sensores]         = linha["byteType"]
         byteINT[qtde_sensores]          = linha["byteINT"]
         byteFRACT[qtde_sensores]        = linha["byteFRACT"]
         expression[qtde_sensores]       = linha["expression"]
```

```
      #Envia a quantidade de pacotes solicitado
      for x in range(0,numPackets+3):

         #Transmite o pacote na serial
         for j in range(0,52):
            serialPort.write(chr(int(pacoteSX[j])))

         #Espera um tempo para que o pacote seja transmitido e recebido
novamente
         time.sleep(0.6)

         #Le a serial em busca do pacote
         pacoteRX = serialPort.read(52)

         #Pega data e hora atual
         date_time = time.strftime("%Y-%m-%d %H-%M-%S")

         #Verifica se o pacote está integro e se é uma resposta de dados
         if len(pacoteRX) == 52 and ord(pacoteRX[11]) == 255 and x > 2:
            print "Recebeu!!"
            #Cadastra no DB registro de pacote recebido corretamente
            cursor.execute("INSERT INTO
packetReceivedTable(addressNS,date,status)
values(%s,%s,%s)",(nodeAddress,date_time,"OK"))
            db.commit()


            #Pega a identificação do ultimo pacote registrado
            cursor.execute("SELECT idPacket from packetReceivedTable
order by idPacket desc limit 1")
            linha = cursor.fetchone()
            idPacket = linha["idPacket"]

            #Pega o número de hops para o NS em questão
            num_hops = ord(pacoteRX[23])

            #Para cada sensor cadastrado referente ao NS, irá cadastrar
no DB o valor obtido
            for y in range(0,qtde_sensores):
               intByte       = byteINT[y+1]
               fractByte     = byteFRACT[y+1]
               expressionEnd = expression[y+1]

               intValue   = ord(pacoteRX[int(intByte)])
               fractValue = ord(pacoteRX[int(fractByte)])

               expressionEnd =
expressionEnd.replace("%INT",str(intValue))
               expressionEnd =
expressionEnd.replace("%FRACT",str(fractValue))
               finalData = eval(expressionEnd)
               print 'Dado coletado: ', finalData, 'Dado inteiro: ',
intValue, 'Dado fracionado: ', fractValue

               #Pega a identificação do sensor
               idSensorNS = idSensorNSVector[y+1]

               #Grava no DB os registro em si - Primeiro, na tabela
collectedData
               cursor.execute("INSERT INTO collectedData
values(%s,%s,%s)",(idPacket,idSensorNS,finalData))
```

```
                db.commit()

                #E por fim na tabela nsSensorTable, informando apenas o
ultimo registro
                cursor.execute("UPDATE nsSensorTable SET
lastValueRead=%s,lastDate=%s WHERE
idSensorNS=%s",(finalData,date_time,idSensorNS))
                db.commit()


                #Cria um vetor para armazenar as RSSIs dos Links
                rssiVetor = {}

            #Percorre o pacote para ler as RSSIs do Enlace
            for j in range(0,num_hops):
                #Valores da RSSI são armazenados da posição 34 até a 39,
sendo um byte para cada link
                tempRSSIDownload = ord(pacoteRX[46+j])

                if tempRSSIDownload > 128:
                    rssiDownload =((tempRSSIDownload-256)/2.0)-74
                else:
                    rssiDownload =(tempRSSIDownload/2.0)-74
                #Grava o registro no Banco de Dados
                cursor.execute("INSERT INTO rssiLinkTable
values(%s,%s,%s)",(idPacket,j+1,rssiDownload))
                db.commit()


         #Caso contrário, se o pacote não for recebido...
         elif x > 2:
            print "Não recebeu"
            #Cadastra no DB registro de pacote recebido corretamente
            cursor.execute("INSERT INTO
packetReceivedTable(addressNS,date,status)
values(%s,%s,%s)",(nodeAddress,date_time,"ERRO"))
            db.commit()

             #Limpa os buffers de entrada e saída da serial
         serialPort.flushInput()
         serialPort.flushOutput()

      #Fecha conexao com DB
      cursor.close()
      db.close()

      #Retorna status positivo
      return 0
```

Arquivo: conectaDB.py

```
# coding=UTF-8
#!/usr/bin/python

import MySQLdb

def conectaDB():

   #Define variaveis para conexao com DB
   host     = "localhost"
   user     = "root"
   password = "ubuntu"
   database = "bubbleDB"

   # Abre conexao com banco de dados
   db = MySQLdb.connect(host,user,password,database)

   #Cria objeto para realizar futuros comandos SQL
   #cursor = db.cursor(MySQLdb.cursors.DictCursor)
   #Cria objeto para realizar comandos SQL utilizando SSdDictCursor -
Resolve problema de memória ?
   cursor = db.cursor(MySQLdb.cursors.SSDictCursor)

   #Retorna o objeto para o programa principal
   return (db, cursor)
```

11.2 Código Fonte – Banco de Dados

Arquivo: database.sql

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `bubbleDB`
--

-- --------------------------------------------------------

--
-- Estrutura da tabela `collectedData`
--

CREATE TABLE `collectedData` (
  `idPacket` int(11) NOT NULL,
  `idSensorNS` int(11) NOT NULL,
  `dataCollected` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Estrutura da tabela `historyPERTable`
--
```

```
CREATE TABLE `historyPERTable` (
  `addressNS` varchar(255) NOT NULL,
  `per` decimal(10,3) DEFAULT NULL,
  `date` datetime DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Estrutura da tabela `logTable`
--

CREATE TABLE `logTable` (
  `nsAddress` varchar(255) DEFAULT NULL,
  `date` datetime DEFAULT NULL,
  `type` varchar(255) DEFAULT NULL,
  `description` mediumtext
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Estrutura da tabela `nsSensorTable`
--

CREATE TABLE `nsSensorTable` (
  `idSensorNS` int(11) NOT NULL,
  `addressNS` varchar(255) NOT NULL,
  `description` varchar(255) NOT NULL,
  `typeSensor` varchar(255) NOT NULL,
  `byteType` char(3) DEFAULT NULL,
  `byteINT` char(3) DEFAULT NULL,
  `byteFRACT` char(3) DEFAULT NULL,
  `expression` varchar(255) DEFAULT NULL,
  `status` varchar(3) NOT NULL,
  `unit` varchar(255) NOT NULL,
  `lastValueRead` varchar(255) DEFAULT NULL,
  `lastDate` datetime DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Estrutura da tabela `nsTable`
--

CREATE TABLE `nsTable` (
  `address` varchar(255) NOT NULL DEFAULT '',
  `description` varchar(255) DEFAULT NULL,
  `location` varchar(255) DEFAULT NULL,
  `status` varchar(255) DEFAULT NULL,
  `route` varchar(255) DEFAULT NULL,
  `hops` int(11) DEFAULT NULL,
  `weightRoute` int(11) DEFAULT NULL,
  `maxPER` decimal(10,2) DEFAULT NULL,
  `maxPERPeriod` int(11) DEFAULT NULL,
  `actualPER` decimal(10,2) DEFAULT NULL,
  `beginEvaluation` datetime DEFAULT NULL,
  `externalFunction` varchar(512) DEFAULT NULL,
  `flagAlternativeRoute` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
-- --------------------------------------------------------

--
-- Estrutura da tabela `packetReceivedTable`
--

CREATE TABLE `packetReceivedTable` (
  `idPacket` int(11) NOT NULL,
  `addressNS` varchar(255) NOT NULL,
  `date` datetime NOT NULL,
  `status` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Estrutura da tabela `rssiLinkTable`
--

CREATE TABLE `rssiLinkTable` (
  `idPacket` int(11) NOT NULL,
  `hopNum` int(11) NOT NULL,
  `rssi` decimal(5,2) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Estrutura da tabela `severalTable`
--

CREATE TABLE `severalTable` (
  `idConfig` int(11) NOT NULL,
  `sinkAddress` varchar(255) NOT NULL,
  `serialPort` varchar(255) NOT NULL,
  `hopLimit` int(11) NOT NULL,
  `routeHopWeight` int(11) NOT NULL,
  `routeLinkWeightFine` int(11) NOT NULL,
  `routeNumPacketFind` int(11) NOT NULL,
  `routeWeightGT50` int(11) NOT NULL,
  `routeWeightGE5999` int(11) NOT NULL,
  `routeWeightGE6999` int(11) NOT NULL,
  `routeWeightGE7999` int(11) NOT NULL,
  `routeWeightGE8999` int(11) NOT NULL,
  `routeWeightGT90` int(11) NOT NULL,
  `limitDaysPacket` int(11) DEFAULT NULL,
  `flagInterrupt` int(11) DEFAULT NULL,
  `statusConfig` int(11) NOT NULL,
  `lastModify` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Indexes for dumped tables
--

--
-- Indexes for table `collectedData`
--
ALTER TABLE `collectedData`
  ADD KEY `indexIDPacketCollectedData` (`idPacket`);
```

```
--
-- Indexes for table `nsSensorTable`
--
ALTER TABLE `nsSensorTable`
  ADD PRIMARY KEY (`idSensorNS`),
  ADD KEY `addressNS` (`addressNS`);

--
-- Indexes for table `nsTable`
--
ALTER TABLE `nsTable`
  ADD PRIMARY KEY (`address`);

--
-- Indexes for table `packetReceivedTable`
--
ALTER TABLE `packetReceivedTable`
  ADD PRIMARY KEY (`idPacket`),
  ADD KEY `indexAddressNS` (`addressNS`),
  ADD KEY `indexIDPacket` (`idPacket`);

--
-- Indexes for table `rssiLinkTable`
--
ALTER TABLE `rssiLinkTable`
  ADD PRIMARY KEY (`idPacket`,`hopNum`),
  ADD UNIQUE KEY `rssiLinkIndex` (`idPacket`,`hopNum`);

--
-- Indexes for table `severalTable`
--
ALTER TABLE `severalTable`
  ADD PRIMARY KEY (`idConfig`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `nsSensorTable`
--
ALTER TABLE `nsSensorTable`
  MODIFY `idSensorNS` int(11) NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT for table `packetReceivedTable`
--
ALTER TABLE `packetReceivedTable`
  MODIFY `idPacket` int(11) NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=52218;
--
-- AUTO_INCREMENT for table `severalTable`
--
ALTER TABLE `severalTable`
  MODIFY `idConfig` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;
--
-- Constraints for dumped tables
--

--
-- Limitadores para a tabela `collectedData`
```

```
--
ALTER TABLE `collectedData`
  ADD CONSTRAINT `collectedData_ibfk_1` FOREIGN KEY (`idPacket`)
REFERENCES `packetReceivedTable` (`idPacket`) ON DELETE CASCADE ON
UPDATE CASCADE;

--
-- Limitadores para a tabela `nsSensorTable`
--
ALTER TABLE `nsSensorTable`
  ADD CONSTRAINT `nsSensorTable_ibfk_1` FOREIGN KEY (`addressNS`)
REFERENCES `nsTable` (`address`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Limitadores para a tabela `rssiLinkTable`
--
ALTER TABLE `rssiLinkTable`
  ADD CONSTRAINT `rssiLinkTable_ibfk_1` FOREIGN KEY (`idPacket`)
REFERENCES `packetReceivedTable` (`idPacket`) ON DELETE CASCADE ON
UPDATE CASCADE;

DELIMITER $$
--
-- Eventos
--
CREATE DEFINER=`root`@`localhost` EVENT `cleanOldPacketReceived` ON
SCHEDULE EVERY 1 DAY STARTS '2016-05-23 00:00:00' ON COMPLETION NOT
PRESERVE ENABLE DO DELETE FROM packetReceived WHERE date < NOW() -
INTERVAL (SELECT limitDaysPacket FROM severalTable) DAY$$

DELIMITER ;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

11.3 Código Fonte – Módulo Informante

Arquivo: soap.php

```
<?php

//Utiliza biblioteca nusoap para criação do Web Service
require_once "lib/nusoap.php";

//Inclui os arquivos referente as funções

include "ns.php";
include "nsSensor.php";
include "several.php";
include "dataAvaiable.php";
include "managementNetwork.php";

//Cria as instâncias para o Web Service
$server = new soap_server();

//Registrar funções do Nó Sensor
$server->register("insertNS");
$server->register("alterNS");
```

```
$server->register("alterPERDataNS");
$server->register("excludeNS");
$server->register("searchStatusNS");
$server->register("alterStatusNS");

//Registrar funções dos Sensores
$server->register("insertSensorNS");
$server->register("alterStatusSensorNS");
$server->register("excludeSensorNS");
$server->register("alterSensorNS");

//Registrar funções gerais
$server->register("updateRouteConfig");
$server->register("updateTableLinkRSSI");
$server->register("updateSinkAddress");
$server->register("updateSerialPort");
$server->register("updateLimitDaysPacket");
$server->register("forceInterrupt");

//Registrar funções de coleta de dados referente a aplicações
$server->register("getLastData");

//Registrar funções de coleta de dados referente a gerência da rede
$server->register("getPERLiveNS");

$server->service($HTTP_RAW_POST_DATA);


?>
```

Arquivo: ns.php

```
<?php

//Função para inserir NS no Proxy Manager
function
insertNS($addressNS,$descriptionNS,$locationNS,$maxPER,$maxPERPeriod,$e
xternalFunction) {
//Códigos de Retorno (Web Service):
//0 : NS inserido com sucesso
//1 : Endereço do NS OU período máximo da PER OU PER máxima não foi
informado
//2 : Erro de gravação no banco de dados

      //Verifica se o endereço do NS, a PER máxima e o período da PER
são nuloo. Em caso positivo, retorna erro
      if (empty($addressNS)||empty($maxPER)||empty($maxPERPeriod)) {
            return 1;
      }
      else {

            //Caso a descrição ou localização do NS for nula, as mesmas
serão preenchidas com "N/A"
            if (empty($descriptionNS)) {
                  $descriptionNS = "N/A";
            }
            if (empty($locationNS)) {
                  $locationNS = "N/A";
            }
            if (empty($externalFunction)) {
```

```
                  $externalFunction = "N/A";
            }

            //Cria status para o NS em questão: Código NR (Não
Respondendo)
            $statusNS = 'NR';

            //Realiza conexão com banco de dados através do arquivo
conectaDB.php
            include "conectaDB.php";

            //Cria o comando SQL para inserção do dado no banco de
dados
            $sql = "insert into
nsTable(address,description,location,status,maxPER,maxPERPeriod,externa
lFunction)

values(UPPER('$addressNS'),UPPER('$descriptionNS'),UPPER('$locationNS')
,UPPER('$statusNS'),

      '$maxPER','$maxPERPeriod','$externalFunction')";

            //Executa a query no banco de dados
            $resultado = mysql_query($sql);

            //Efetua a gravação do NS no banco de dados
            if ($resultado) {
                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
                  mysql_close();

                  //Retorna status positivo
                  return 0;
                }
            else {
                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
                  mysql_close();

                  //Retorna status negativo
                  return 2;
            }


            }
}

//Função para alterar dados do NS no Proxy Manager
function
alterNS($oldAddressNS,$newAddressNS,$newDescriptionNS,$newLocationNS) {
//Códigos de Retorno (Web Service):
//0 : NS alterado com sucesso
//1 : Endereço do NS não foi informado
//2 : Erro de gravação no banco de dados
//3 : Nenhum registro foi alterado

      //Verifica se o endereço antigo ou novo é nulo. Caso seja,
retorna erro
      if (empty($oldAddressNS) || empty($newAddressNS)) {
            return 1;
      }
```

```
      else {

            //Caso a descrição ou localização do NS for nula, as mesmas
serão preenchidas com "N/A"
            if (empty($newDescriptionNS)) {
                  $newDescriptionNS = "N/A";
            }
            if (empty($newLocationNS)) {
                  $newLocationNS = "N/A";
            }

            //Altera o status para Não Descoberto (NS)
            $statusNS = "ND";

            //Realiza conexão com banco de dados através do arquivo
conectaDB.php
            include "conectaDB.php";

            //Cria o comando SQL para alteração do dado no banco de
dados
            $sql = "UPDATE nsTable SET
address=UPPER('$newAddressNS'),description=UPPER('$newDescriptionNS'),
            location=UPPER('$newLocationNS'),status=UPPER('$statusNS')
where address=$oldAddressNS";

            $resultado = mysql_query($sql);

            //Efetua a alteração do NS no banco de dados
            if ($resultado) {

                  //Pega número de registros afetados pela alteração
                  $numRegistrosAfetados = mysql_affected_rows();

                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
                  mysql_close();

                  if ($numRegistrosAfetados == 0)
                        //Retorna erro 3 (nenhum registro foi alterado)
                        return 3;
                  else
                        //Retorna status positivo
                        return 0;
                }
            else {
                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
                  mysql_close();

                  //Retorna status negativo
                  return 2;
            }


            }
}

//Função para alterar dados referente a PER de um NS no Proxy Manager
function alterPERDataNS($addressNS,$newMaxPER,$newMaxPERPeriod) {
//Códigos de Retorno (Web Service):
//0 : Dados alterados com sucesso
```

```
//1 : Dados não foram informados
//2 : Erro de gravação no banco de dados
//3 : Nenhum registro foi alterado

	//Verifica se os dados passados são nulos. Caso seja, retorna
erro
	if (empty($addressNS) || empty($newMaxPER) ||
empty($newMaxPERPeriod)) {
		return 1;
	}
	else {

		//Realiza conexão com banco de dados através do arquivo
conectaDB.php
		include "conectaDB.php";

		//Cria o comando SQL para alteração do dado no banco de
dados
		$sql = "UPDATE nsTable SET
maxPER='$newMaxPER',maxPERPeriod='$newMaxPERPeriod' where
address='$addressNS'";

		$resultado = mysql_query($sql);

		//Efetua a alteração do NS no banco de dados
		if ($resultado) {

			//Pega número de registros afetados pela alteração
			$numRegistrosAfetados = mysql_affected_rows();

			//Encerra a conexão com o banco de dados
			mysql_free_result($resultado);
			mysql_close();

			if ($numRegistrosAfetados == 0)
				//Retorna erro 3 (nenhum registro foi alterado)
				return 3;
			else
				//Retorna status positivo
				return 0;
		    }
		else {
			//Encerra a conexão com o banco de dados
			mysql_free_result($resultado);
			mysql_close();

			//Retorna status negativo
			return 2;
		}


		}
}

//Função para excluir NS no Proxy Manager
function excludeNS($addressNS) {
//Códigos de Retorno (Web Service):
//0 : NS excluido com sucesso
//1 : Endereço do NS não foi informado
//2 : Erro de gravação no banco de dados
//3 : Nenhum registro foi excluído do banco de dados
```

```
	//Verifica se o endereço é nulo. Caso seja, retorna erro
	if (empty($addressNS)) {
		return 1;
	}
	else {

		//Realiza conexão com banco de dados através do arquivo
conectaDB.php
		include "conectaDB.php";

		//Cria o comando SQL para exclusão do NS
		$sql = "DELETE FROM nsTable where address =
UPPER('$addressNS')";

		$resultado = mysql_query($sql);


		//Efetua a exclusão do NS no banco de dados
		if ($resultado) {

			//Pega número de registros afetados pela exclusão
			$numRegistrosAfetados = mysql_affected_rows();

			//Encerra a conexão com o banco de dados
			mysql_free_result($resultado);
			mysql_close();

			if ($numRegistrosAfetados == 0)
				//Retorna erro 3 (nenhum registro foi exclúido)
				return 3;
			else
				//Retorna status positivo
				return 0;
		    }
		else {
			//Encerra a conexão com o banco de dados
			mysql_free_result($resultado);
			mysql_close();

			//Retorna status negativo
			return 2;
		}


		}
}

//Função para consultar endereços de NS pelo status dos mesmos
//Status possíveis:
//ND: Não Descoberto
//OP: Operacional
//SP: Dormindo
function searchStatusNS($statusNS) {
//Códigos de retorno (Web Service):
//1 : Status não foi informado
//2 : Erro de consulta com o banco de dados
//3 : Nenhum registro encontrado

	//Verifica se o status é nulo. Caso seja, retorna erro
	if (empty($statusNS)) {
```

```
            return 1;
      }
      else {

            //Realiza conexão com banco de dados através do arquivo
conectaDB.php
            include "conectaDB.php";

            //Cria SQL para consulta
            $sql = "SELECT address,description,location from nsTable
where status=UPPER('$statusNS')";

            //Realiza a consulta
            $resultado = mysql_query($sql);

            // Se não foi possível realizar a consulta...
            if (! $resultado) {

                  // Fecha a conexão e retorna status 2
                  mysql_close();
                  return 2;
            }
            else {
                  //Verifica o número de registros encontrados. Caso
seja igual a 0, retorna status 3
                  $num_resultados = mysql_num_rows($resultado);
                  if ($num_resultados == 0){
                        mysql_close();
                        return 3;
                  }
                  else {
                        $i=0;
                        //Caso esteja tudo certo, retorna o vetor de
resposta (matriz)
                        while ($linha = mysql_fetch_array($resultado))
{
                              $vetor_resposta[$i][0]=$linha["address"];

      $vetor_resposta[$i][1]=$linha["description"];
                              $vetor_resposta[$i][2]=$linha["location"];

                              $i++;
                        }
                        mysql_free_result($resultado);
                        mysql_close();
                        return $vetor_resposta;
                  }
            }

      }

}

//Função para alterar o status de determinado NS manualmente
function alterStatusNS($addressNS,$newStatus) {
//Códigos de Retorno (Web Service):
//0 : Dados alterados com sucesso
//1 : Dados não foram informados
//2 : Erro de gravação no banco de dados
//3 : Nenhum registro foi alterado
```

```
	//Verifica se os dados passados são nulos. Caso seja, retorna
erro
	if (empty($addressNS) || empty($newStatus)) {
		return 1;
	}
	else {

		//Realiza conexão com banco de dados através do arquivo
conectaDB.php
		include "conectaDB.php";

		//Cria o comando SQL para alteração do dado no banco de
dados
		$sql = "UPDATE nsTable SET status='$newStatus'
			 where address='$addressNS'";

		$resultado = mysql_query($sql);

		//Efetua a alteração do NS no banco de dados
		if ($resultado) {

			//Pega número de registros afetados pela alteração
			$numRegistrosAfetados = mysql_affected_rows();

			//Encerra a conexão com o banco de dados
			mysql_free_result($resultado);
			mysql_close();

			if ($numRegistrosAfetados == 0)
				//Retorna erro 3 (nenhum registro foi alterado)
				return 3;
			else
				//Retorna status positivo
				return 0;
		    }
		else {
			//Encerra a conexão com o banco de dados
			mysql_free_result($resultado);
			mysql_close();

			//Retorna status negativo
			return 2;
		}

		}
}

?>
```

Arquivo: nsSensor.php

```
<?php

//Função para inserir NS no Proxy Manager
function
insertSensorNS($addressNS,$typeSensor,$descriptionSensor,$byteType,
         $byteINT,$byteFRACT,$expression,$status,$unit) {
//Códigos de retorno (Web Service):
//0 : Registro inserido como sucesso
//1 : Endereço não foi informado
//2 : Erro ao inserir registro no DB

      //Verifica se o endereço do NS é nulo ou o tipo de sensor. Em caso
      //positivo, retorna erro
      if (empty($addressNS)||empty($typeSensor)) {
            return 1;
      }

      //Verifica se o status é nulo. Caso seja, cadastrar o mesmo como
"OFF"
      if (empty($status)) {
            $status = "OFF";
      }

      //Realiza conexão com banco de dados através do arquivo
conectaDB.php
      include "conectaDB.php";

      //Cria o comando SQL para inserção do dado no banco de dados
      $sql = "insert into
nsSensorTable(addressNS,description,typeSensor,byteType,byteINT,byteFRAC
T,
               expression,status,unit)

values(UPPER('$addressNS'),UPPER('$descriptionSensor'),UPPER('$typeSenso
r'),
               UPPER('$byteType'),UPPER('$byteINT')

,UPPER('$byteFRACT'),UPPER('$expression'),UPPER('$status'),'$unit')";

      //Executa a query no banco de dados
      $resultado = mysql_query($sql);

      //Efetua a gravação do NS no banco de dados
      if ($resultado) {
            //Encerra a conexão com o banco de dados
            mysql_free_result($resultado);
            mysql_close();

            //Retorna status positivo
            return 0;
        }
      else {

            //Encerra a conexão com o banco de dados
            mysql_free_result($resultado);
            mysql_close();

            //Retorna status negativo
            return 2;
```

```
	}
}

//Função para alterar status (ON/OFF) de determinado NS no Proxy Manager
function alterStatusSensorNS($addressNS,$typeSensor,$newStatus) {
//Códigos de retorno (Web Service):
//0 : Registro alterado como sucesso
//1 : Endereço não foi informado
//2 : Novo Status não foi informado
//3 : Tipo (Type) do sensor não foi informado
//4 : Erro com alteração do Banco de Dados

	//Verifica se o endereço do NS é nulo. Em caso positivo, retorna
erro
	if (empty($addressNS)) {
		return 1;
	}

	//Verifica se o status novo é nulo. Em caso positivo, retorna erro
	if (empty($newStatus)) {
		return 2;
	}

	//Verifica se o byte Type (Tipo) é nulo. Em caso positivo, retorna
erro

	if (empty($typeSensor)) {
		return 3;
	}

	//Cria o comando SQL para alteração do dado no banco de dados
	$sql = "UPDATE nsSensorTable SET status=UPPER('$newStatus') WHERE
		addressNS=$addressNS AND typeSensor=$typeSensor";


	//Realiza conexão com banco de dados através do arquivo
conectaDB.php
	include "conectaDB.php";

	//Executa a query no banco de dados
	$resultado = mysql_query($sql);

	//Efetua a alteração no banco de dados
	if ($resultado) {
		//Encerra a conexão com o banco de dados
		mysql_free_result($resultado);
		mysql_close();

		//Retorna status positivo
		return 0;
	  }
	else {

		//Encerra a conexão com o banco de dados
		mysql_free_result($resultado);
		mysql_close();

		//Retorna status negativo
		return 4;

	}
```

```
}

//Função para excluir Sensor de determinado NS no Proxy Manager
function excludeSensorNS($addressNS,$typeSensor) {
//Códigos de retorno (Web Service):
//0 : Registro alterado como sucesso
//1 : Endereço não foi informado
//2 : Tipo (type) do Sensor não foi informado
//3 : Erro com alteração do Banco de Dados
//4 : Nenhum registro foi excluido

      //Verifica se o endereço do NS é nulo. Em caso positivo, retorna
erro
      if (empty($addressNS)) {
            return 1;
      }

      //Verifica se o Type (Tipo) é nulo. Em caso positivo, retorna erro

      if (empty($typeSensor)) {
            return 2;
      }

      //Cria o comando SQL para alteração do dado no banco de dados
      $sql = "DELETE FROM nsSensorTable WHERE addressNS=$addressNS AND
               typeSensor=$typeSensor";

      //Realiza conexão com banco de dados através do arquivo
conectaDB.php
      include "conectaDB.php";

      //Executa a query no banco de dados
      $resultado = mysql_query($sql);

      //Efetua a exclusão no banco de dados
      if ($resultado) {

            //Pega número de registros afetados pela exclusão
            $numRegistrosAfetados = mysql_affected_rows();

            //Encerra a conexão com o banco de dados
            mysql_free_result($resultado);
            mysql_close();

            //Se o número de registros excluidos for igual a 0, então
            //retorna 4. Caso contrário, retorna 0
            if ($numRegistrosAfetados == 0)
                  return 4;
            else
                  return 0;
        }
      else {

            //Encerra a conexão com o banco de dados
            mysql_free_result($resultado);
            mysql_close();

            //Retorna status negativo
            return 3;
```

```
	}

}

//Função para alterar status (ON/OFF) de determinado NS no Proxy Manager
function
alterSensorNS($addressNS,$typeSensor,$newTypeSensor,$newByteType,
                        $newByteINT,
$newByteFRACT,$newExpression,$newDescription,
                        $newStatus,$newUnit) {
//Códigos de retorno (Web Service):
//0 : Registro alterado como sucesso
//1 : Endereço não foi informado
//2 : Type (Tipo) não foi informado
//3 : Erro no Banco de Dados
//4 : Nenhum registro foi alterado

	//Verifica se o endereço do NS é nulo. Em caso positivo, retorna
erro
	if (empty($addressNS)) {
		return 1;
	}

	//Verifica se o Type (Tipo) é nulo. Em caso positivo, retorna erro

	if (empty($typeSensor)) {
		return 2;
	}

	//Cria o comando SQL para alteração do dado no banco de dados
	$sql = "UPDATE nsSensorTable SET
typeSensor=UPPER('$newTypeSensor')byteType=UPPER('$newByteType'),

byteINT=UPPER('$newByteINT'),byteFRACT=UPPER('$newByteFRACT'),expression
=UPPER('$newExpression'),

	description=UPPER('$newDescription'),status=UPPER('$newStatus'),
			unit='$newUnit'
			WHERE addressNS=$addressNS AND
typeSensor=$newTypeSensor";
	//Realiza conexão com banco de dados através do arquivo
conectaDB.php
	include "conectaDB.php";

	//Executa a query no banco de dados
	$resultado = mysql_query($sql);

	//Efetua a alteração no banco de dados
	if ($resultado) {

		//Pega número de registros alterados
		$numRegistrosAfetados = mysql_affected_rows();

		//Encerra a conexão com o banco de dados
		mysql_free_result($resultado);
		mysql_close();

		//Se o número de registros excluidos for igual a 0, então
retorna 4.
		 //Caso contrário, retorna 0
```

```
            if ($numRegistrosAfetados == 0)
                  return 4;
            else
                  return 0;
        }
      else {

            //Encerra a conexão com o banco de dados
            mysql_free_result($resultado);
            mysql_close();

            //Retorna status negativo
            return 3;

      }

}

?>
```

Arquivo: several.php

```
<?php

//Função para alterar configurações relacionadas a roteamento na tabela
Several
//SUBSTITUI qualquer configuração anterior relacionada aos parâmetros
utilizados
function
updateRouteConfig($newHopLimit,$newRouteHopWeight,$newRouteLinkWeightFi
ne,$newRouteNumPacketFind){
//Códigos de retorno:
//0 : Registros atualizados com sucesso
//1 : Não foi possível executar a QUERY


      //Realiza conexão com banco de dados através do arquivo
conectaDB.php
      include "conectaDB.php";

      //Cria comando SQL para verificar se já existe algum registro no
banco (verificar se vai inserir ou alterar o registro existente.
Aproveita e, se já existir, pega os dados da tabela
      $sql = "SELECT hopLimit, routeHopWeight, routeLinkWeightFine,
routeNumPacketFind from severalTable";

      //Executa a query no banco de dados
      $resultado = mysql_query($sql);

      // Verifica se foi possível realizar a query
      if ($resultado == FALSE){
            //Encerra a conexão com o banco de dados
            mysql_free_result($resultado);
            mysql_close();

            //Retorna status negativo (1)
            return 1;
      }
      else {
```

```
            //Pega o resultado e passa para um vetor linha
            $linha = mysql_fetch_assoc($resultado);

            //Se algum parametro informado foi nulo, pegar valor que
está cadastrado no banco de dados
            if (empty($newHopLimit))
                  $hopLimit = $linha["hopLimit"];
            else
                  $hopLimit = $newHopLimit;

            if (empty($newRouteHopWeight))
                  $routeHopWeight = $linha["routeHopWeight"];
            else
                  $routeHopWeight = $newRouteHopWeight;

            if (empty($newRouteLinkWeightFine))
                  $routeLinkWeightFine = $linha["routeLinkWeightFine"];
            else
                  $routeLinkWeightFine = $newRouteLinkWeightFine;

            if (empty($newRouteNumPacketFind))
                  $routeNumPacketFind = $linha["routeNumPacketFind"];
            else
                  $routeNumPacketFind = $newRouteNumPacketFind;

            //Pega data e hora atual
            $date_time = date("Y-m-d G:i:s");

            //Cria novo SQL para update
            $sql = "UPDATE severalTable SET
hopLimit=$hopLimit,routeHopWeight=$routeHopWeight,routeLinkWeightFine=$
routeLinkWeightFine,routeNumPacketFind=$routeNumPacketFind,
lastModify='$date_time'";

            //Executa a query no banco de dados
            $resultado = mysql_query($sql);

            //Verifica se foi possível realizar o UPDATE
            if ($resultado) {

                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
                  mysql_close();
                  return 0;
                }
            else {
                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
                  mysql_close();

                  //Retorna status negativo
                  return 1;
            }

      }
}

//Atualiza os pesos referente ao RSSI dos enlaces da WSN
function
updateTableLinkRSSI($newRouteWeightGT50,$newRouteWeightGE5999,$newRoute
WeightGE6999,$newRouteWeightGE7999,$newRouteWeightGE8999,$newRouteWeigh
```

```
tGT90) {
//Códigos de retorno:
//0 : Registros atualizados com sucesso
//1 : Não foi possível executar a QUERY

      //Realiza conexão com banco de dados através do arquivo
conectaDB.php
      include "conectaDB.php";

      //Cria comando SQL para verificar se já existe algum registro no
banco (verificar se vai inserir ou alterar o registro existente.
Aproveita e, se já existir, pega os dados da tabela
      $sql = "SELECT routeWeightGT50, routeWeightGE5999,
routeWeightGE6999, routeWeightGE7999, routeWeightGE8999,
routeWeightGT90 from severalTable";

      //Executa a query no banco de dados
      $resultado = mysql_query($sql);

      // Verifica se foi possível realizar a query
      if ($resultado == FALSE){
            //Encerra a conexão com o banco de dados
            mysql_free_result($resultado);
            mysql_close();

            //Retorna status negativo (1)
            return 1;
      }
      else {

            //Pega o resultado e passa para um vetor linha
            $linha = mysql_fetch_assoc($resultado);

            //Se algum parametro informado foi nulo, pegar valor que
está cadastrado no banco de dados
            if (empty($newRouteWeightGT50))
                  $routeWeightGT50 = $linha["routeWeightGT50"];
            else
                  $routeWeightGT50 = $newRouteWeightGT50;

            if (empty($newRouteWeightGE5999))
                  $routeWeightGE5999 = $linha["routeWeightGE5999"];
            else
                  $routeWeightGE5999 = $newRouteWeightGE5999;

            if (empty($newRouteWeightGE6999))
                  $routeWeightGE6999 = $linha["routeWeightGE6999"];
            else
                  $routeWeightGE6999 = $newRouteWeightGE6999;

            if (empty($newRouteWeightGE7999))
                  $routeWeightGE7999 = $linha["routeWeightGE7999"];
            else
                  $routeWeightGE7999 = $newRouteWeightGE7999;

            if (empty($newRouteWeightGE8999))
                  $routeWeightGE8999 = $linha["routeWeightGE8999"];
            else
                  $routeWeightGE8999 = $newRouteWeightGE8999;

            if (empty($newRouteWeightGT90))
```

```
                  $routeWeightGT90 = $linha["routeWeightGT90"];
            else
                  $routeWeightGT90 = $newRouteWeightGT90;

            //Pega data e hora atual
            $date_time = date("Y-m-d G:i:s");

            //Cria novo SQL para update
            $sql = "UPDATE severalTable SET
routeWeightGT50=$routeWeightGT50,routeWeightGE5999=$routeWeightGE5999,r
outeWeightGE6999=$routeWeightGE6999,routeWeightGE7999=$routeWeightGE799
9,routeWeightGE8999=$routeWeightGE8999,routeWeightGT90=$routeWeightGT90
, lastModify='$date_time'";

            //Executa a query no banco de dados
            $resultado = mysql_query($sql);

            //Verifica se foi possível realizar o UPDATE
            if ($resultado) {

                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
                  mysql_close();
                  return 0;
                }
            else {
                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
                  mysql_close();

                  //Retorna status negativo
                  return 1;
            }
      }
}

//Atualiza o endereço do Nó Sorvedouro (Sink Address)
function updateSinkAddress($newSinkAddress) {
//Códigos de retorno:
//0 : Endereço atualizado com sucesso!
//1 : Nenhum Sink Address foi informado ou valor não é válido
//2 : Não foi possível realizar o update no banco de dados

      if (empty($newSinkAddress) || !(is_int($newSinkAddress))) {
            return 1;
      }
      else {


            //Realiza conexão com banco de dados através do arquivo
conectaDB.php
            include "conectaDB.php";

            //Pega data e hora atual
            $date_time = date("Y-m-d G:i:s");

            //Cria SQL para pegar endereço atual do banco
            $sql = "UPDATE severalTable SET
lastModify='$date_time',sinkAddress = $newSinkAddress";

            //Executa a query no banco de dados
```

```
            $resultado = mysql_query($sql);

            //Verifica se foi possível realizar o UPDATE
            if ($resultado) {

                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
                  mysql_close();
                  return 0;
                }
            else {
                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
                  mysql_close();

                  //Retorna status negativo
                  return 2;
            }

      }

}

//Atualiza a porta serial cadastrada
function updateSerialPort($newSerialPort) {
//Códigos de retorno:
//0 : Porta Serial atualizada com sucesso!
//1 : Nenhuma porta serial foi informada
//2 : Não foi possível realizar o update no banco de dados

      if (empty($newSerialPort)) {
            return 1;
      }
      else {


            //Realiza conexão com banco de dados através do arquivo
conectaDB.php
            include "conectaDB.php";

            //Pega data e hora atual
            $date_time = date("Y-m-d G:i:s");

            //Cria SQL para pegar endereço atual do banco
            $sql = "UPDATE severalTable SET
lastModify='$date_time',serialPort = $newSerialPort";

            //Executa a query no banco de dados
            $resultado = mysql_query($sql);

            //Verifica se foi possível realizar o UPDATE
            if ($resultado) {

                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
                  mysql_close();
                  return 0;
                }
            else {
                  //Encerra a conexão com o banco de dados
                  mysql_free_result($resultado);
```

```
                    mysql_close();

                    //Retorna status negativo
                    return 2;
              }

        }

}

//Atualiza o limite de dias que os pacotes ficarão registrados no DB
function updateLimitDaysPacket($newLimitDaysPacket) {
//Códigos de retorno:
//0 : Atualização realizada com sucesso!
//1 : Nenhuma quantidade de dias foi informad ou valor não é válido
//2 : Não foi possível realizar o update no banco de dados

      if (empty($newLimitDaysPacket) || !(is_int($newLimitDaysPacket)))
{
              return 1;
      }
      else {


              //Realiza conexão com banco de dados através do arquivo
conectaDB.php
              include "conectaDB.php";

              //Pega data e hora atual
              $date_time = date("Y-m-d G:i:s");

              //Cria SQL para pegar endereço atual do banco
              $sql = "UPDATE severalTable SET
lastModify='$date_time',limitDaysPacket = $newLimitDaysPacket";

              //Executa a query no banco de dados
              $resultado = mysql_query($sql);

              //Verifica se foi possível realizar o UPDATE
              if ($resultado) {

                    //Encerra a conexão com o banco de dados
                    mysql_free_result($resultado);
                    mysql_close();
                    return 0;
                  }
              else {
                    //Encerra a conexão com o banco de dados
                    mysql_free_result($resultado);
                    mysql_close();

                    //Retorna status negativo
                    return 2;
              }

      }
}

//Força processo de interrupção para tentativa de descoberta de rotas
para NS com status NR e IN
function forceInterrupt() {
```

```
//Códigos de retorno:
//0 : Atualização realizada com sucesso!
//1 : Não foi possível realizar o update no banco de dados

      //Realiza conexão com banco de dados através do arquivo
conectaDB.php
      include "conectaDB.php";

      //Pega data e hora atual
      $date_time = date("Y-m-d G:i:s");

      //Cria SQL para pegar endereço atual do banco
      $sql = "UPDATE severalTable SET
lastModify='$date_time',flagInterrupt = 1";

      //Executa a query no banco de dados
      $resultado = mysql_query($sql);

      //Verifica se foi possível realizar o UPDATE
      if ($resultado) {

            //Encerra a conexão com o banco de dados
            mysql_free_result($resultado);
            mysql_close();
            return 0;
        }
      else {
            //Encerra a conexão com o banco de dados
            mysql_free_result($resultado);
            mysql_close();
            //Retorna status negativo
            return 1;
      }


}

?>
```

11.4 Algoritmo de emulação da RSSF (Arduino)

Arquivo: emulacao.ino

```
//Cria a estrutura do pacote
byte pacote[52]; // Pacote de 52 bytes que será recebido na serial e
encaminhado para os NS

//Cria a matriz de Probabilidade de perda de pacote pelas RSSIs. Cada
enlace ficara associado a uma entrada desta tabela
// 1 Campo: Valor Inicial da RSSI
// 2 Campo: Valor Final da RSSI
// 3 Campo: Probabilidade de perda do pacote ao trafegar por um enlace
deste
// AS PROBABILIDADES DE PERDA NECESSITAM SER REVISTAS
float matrizProbabilidadePerda[6][3] = {
  {-40 , -49.9 , 0.1 }, // Indice 0
  {-50 , -59.9 , 1.4 }, // Indice 1
  {-60 , -69.9 , 3.3 }, // Indice 2
```

```
  {-70 , -79.9 , 7.5 }, // Indice 3
  {-80 , -89.9 , 16.4 }, // Indice 4
  {-90 , -100 , 33.3 }, // Indice 5
};

//Cria matriz para indicar a probabilidae de perda de pacote dos enlaces
da RSSF
// 1 Campo: Indica o 1 NS que compoem o enlace em questao
// 2 Campo: Indica o 2 NS que compoem o enlace em questao
// 3 Campo: Indice relacionado da matriz de probabilidade de perda

// Matriz de Enlaces - 1 Cenario
float matrizEnlaces1[15][3] = {
  {0, 1, 0},
  {0, 2, 0},
  {0, 3, 4},
  {0, 4, 5},
  {0, 5, 5},
  {1, 2, 3},
  {1, 3, 0},
  {1, 4, 3},
  {1, 5, 4},
  {2, 3, 2},
  {2, 4, 0},
  {2, 5, 0},
  {3, 4, 3},
  {3, 5, 0},
  {4, 5, 0},
};

// Matriz de Enlaces - 2 Cenario
float matrizEnlaces2[15][3] = {
  {0, 1, 0},
  {0, 2, 0},
  {0, 3, 4},
  {0, 4, 5},
  {0, 5, 5},
  {1, 2, 3},
  {1, 3, 0},
  {1, 4, 3},
  {1, 5, 4},
  {2, 3, 2},
  {2, 4, 0},
  {2, 5, 5},
  {3, 4, 3},
  {3, 5, 0},
  {4, 5, 0},
};

// Matriz de Enlaces - 3 Cenario
float matrizEnlaces3[15][3] = {
  {0, 1, 0},
  {0, 2, 0},
  {0, 3, 4},
  {0, 4, 5},
  {0, 5, 5},
  {1, 2, 3},
  {1, 3, 0},
  {1, 4, 3},
  {1, 5, 4},
  {2, 3, 2},
```

```
  {2, 4, 0},
  {2, 5, 5},
  {3, 4, 3},
  {3, 5, 5},
  {4, 5, 0},
};

// Registra a quantidade de enlaces disponiveis
int quantidadeEnlaces = 15;


// Registra a quantidade de NS
int quantidadeNS = 5;

// Registra qual o endereco do No Sorvedouro
int enderecoSorvedouro = 0;

//Inicia variavel para controlar looping - Comeca com o valor 0
int looping = 0;


//Inicia variavel para controlar a quantidade de vezes que cada cenario
ira rodar
int quantidadeVezesPorCenario = 3000;

//Define uma classe denominada NS (No Sensor)
class NS {

  // Define as variaveis da classe NS

  int ID; // Nesta variável deve estar o endereço do nó sensor
  int
LeituraADC0,LeituraADC1,LeituraADC2,LeituraADC3,LeituraADC4,LeituraADC5;
// Variáveis para cada um dos conversores analógico digital. Neste exemplo
só está usando o ADC 0
  int IO0_PIN,IO1_PIN,IO2_PIN,IO3_PIN,IO4_PIN,IO5_PIN;
  int IO0_STATUS,IO1_STATUS,IO2_STATUS,IO3_STATUS,IO4_STATUS,IO5_STATUS;
// Variáveis de status do IO correspondente. Será usado para funcionar com
o toggle (alternância) do ScadaBR
  int contador_pacote_TX; // Variável para contar os pacotes transmitidos
pelo nó sensor

  //Declara variavel para controlar status de retorno do processamento do
pacote
  int retorno;

  //Declara duas variaveis: origem e destino, para armazenar a origem e o
destino atuais;
  int origem;
  int destino;

  //Declara variavel para armazenar tensao relacionada a RSSI
  float rssiTensao;

  //Declara variavel para verificar se NS esta ativado ou desativado (ON -
1 / OFF - 0)
  int statusNS;

  //Cria o construtor da classe (responsavel por iniciar as variaveis da
classe)
  public:
```

```
    NS(int endereco) {

      //O ID recebe o endereco da classe
      ID = endereco;

      //Inicia o contador de pacotes recebidos como 0
      contador_pacote_TX = 0;

      //Inicia as variaveis de status dos IO como 0
      IO0_STATUS=IO1_STATUS=IO2_STATUS=IO3_STATUS=IO4_STATUS=IO5_STATUS=0;

      //Variavel de retorno comeca o valor com 10 (um valor neutro)
      retorno = 10;

      //Inicia valor de tensao para RSSI
      rssiTensao = 0;

      //Inicia o status do NS com 1 (ON - Ativado)
      statusNS = 1;
    }

    inline void receivePHY();
    inline void receiveMAC();
    inline void receiveNWK();
    inline void receiveTRANSP();
    inline void receiveAPP();

    inline void sendPHY();
    inline void sendMAC();
    inline void sendNWK();
    inline void sendTRANSP();
    inline void sendAPP();

    inline int processPacket();


  private:
};


  //Comeca a criar os metodos da classe
  //Cria um metodo receive e send para cada camada

  //Recebe camada fisica
  void NS::receivePHY() {

    //Se o status do NS estiver ON, ou seja, ele esta ligado...
    if (statusNS == 1) {

      //Armazena os enderecos de origem e destino.
      origem  = pacote[10];
      destino = pacote[8];


      // Percorre a matriz de enlaces para determinar a probabilidade de
entrega do pacote
      for (int k = 0; k < quantidadeEnlaces; k++) {

        if (looping <=  quantidadeVezesPorCenario ) {
```

```
          //Se os NS em questao pertece ao enlace a ser comparado...
          if (((origem == matrizEnlaces1[k][0])&&(destino ==
matrizEnlaces1[k][1])) || (destino == matrizEnlaces1[k][0])&&(origem ==
matrizEnlaces1[k][1])) {

            //Calcula a RSSI de Downlink
            float rssiMinimaPossivel =
matrizProbabilidadePerda[int(matrizEnlaces1[k][2])][1];
            float rssiMaximaPossivel =
matrizProbabilidadePerda[int(matrizEnlaces1[k][2])][0];
            float rssi = rssiMaximaPossivel+((float)rand()/(float)RAND_MAX)
* (rssiMinimaPossivel - rssiMaximaPossivel);

            //Calcula o valor de tensao relacionado
            rssiTensao = 2.0 * (rssi + 74);
            pacote[0] = rssiTensao;

          }
        }
        else if (looping <= quantidadeVezesPorCenario * 2) {

          //Se os NS em questao pertece ao enlace a ser comparado...
          if (((origem == matrizEnlaces2[k][0])&&(destino ==
matrizEnlaces2[k][1])) || (destino == matrizEnlaces2[k][0])&&(origem ==
matrizEnlaces2[k][1])) {

            //Calcula a RSSI de Downlink
            float rssiMinimaPossivel =
matrizProbabilidadePerda[int(matrizEnlaces2[k][2])][1];
            float rssiMaximaPossivel =
matrizProbabilidadePerda[int(matrizEnlaces2[k][2])][0];
            float rssi = rssiMaximaPossivel+((float)rand()/(float)RAND_MAX)
* (rssiMinimaPossivel - rssiMaximaPossivel);

            //Calcula o valor de tensao relacionado
            rssiTensao = 2.0 * (rssi + 74);
            pacote[0] = rssiTensao;

          }

        }
        else if (looping <= quantidadeVezesPorCenario * 3) {

          //Se os NS em questao pertece ao enlace a ser comparado...
          if (((origem == matrizEnlaces3[k][0])&&(destino ==
matrizEnlaces3[k][1])) || (destino == matrizEnlaces3[k][0])&&(origem ==
matrizEnlaces3[k][1])) {

            //Calcula a RSSI de Downlink
            float rssiMinimaPossivel =
matrizProbabilidadePerda[int(matrizEnlaces3[k][2])][1];
            float rssiMaximaPossivel =
matrizProbabilidadePerda[int(matrizEnlaces3[k][2])][0];
            float rssi = rssiMaximaPossivel+((float)rand()/(float)RAND_MAX)
* (rssiMinimaPossivel - rssiMaximaPossivel);

            //Calcula o valor de tensao relacionado
            rssiTensao = 2.0 * (rssi + 74);
            pacote[0] = rssiTensao;

          }
```

```
        }

      }

      //Envia para a camada MAC
      receiveMAC();
    }
    //Caso contrario, o NS estiver desativado...
    else {
      //Retorno recebe 3, significando que o NS esta desligado
      retorno = 3;
    }

  }

  //Recebe camada MAC
  void NS::receiveMAC() {

    //Verifica se o pacote recebido possui 52 bytes
    if (sizeof(pacote) == 52) {

      //Em caso positivo, chama a camada de rede
      receiveNWK();
    }

    //Caso contrario, o pacote tenha chegado corrompido...
    else {

      //Variavel de retorno recebe 1
      retorno = 1;
    }

  }

  //Recebe camada NWK
  void NS::receiveNWK() {

    //Verifica se o pacote e para o NS em questao...
    if (pacote[8] == ID) {


     int posicao_rota = 0;

     /*Verifica se o pacote e uma solicitacao de rota, e se o o no em
questao e o no de destino*/
     //if ((pacote[11] == 128)&&(pacote[8] == ID)) -
MODIFICACAO!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
     if ((pacote[11] == 128)&&(pacote[8] ==
ID)&&(pacote[24]+1==pacote[23]))
        {
          /* Zera o numero de hops atuais (indicando um retorno)*/
          pacote[24] = 0;
          /* Muda a flag de roteamento para indicar retorno positivo de
descoberta de rota*/
          pacote[11] = 64;
          /* Troca endereco de destino e de origem */
          pacote[8] = pacote[10];
          pacote[10] = ID;

          /* Envia para a camada inferior */
```

```
          sendMAC();
        }

      /* Verifica se e um no intermediario e se recebeu uma solicitacao de
rota */
      else if (pacote[11] == 128)
        {
          /* Incrementa o numero de hops atuais */
          pacote[24]++;
          /* Se o numero de hops atuais + 1 for menor que o numero de hops
maximo */
          if (pacote[24]+1 < pacote[23])
            {
              /* Determina qual sera o proximo endereco, e configura o
campo de destino com o mesmo */
              posicao_rota = pacote[24] + 28;
              pacote[8] = pacote[posicao_rota];
              pacote[10] = ID;
            }


          /* Caso contrario, o numero de hops atuais seja igual ao numero
de hops maximo */
          else if (pacote[24]+1 == pacote[23])
            {
              /* O endereco de destino sera o endereco a ser procurado.
Caso o mesmo nao responda, a base detectara pelo esgotamento do tempo */
              pacote[8] = pacote[22];
              pacote[10] = ID;
            }

          /* Envia para a camada inferior */
          sendMAC();
        }


      /* Caso contrario, a flag indicar resposta de rota e o numero de
hops seja igual a 0, ou seja, este e o no diretamente ligado ao no de
descoberta */
      else if ((pacote[11] == 64)&&(pacote[24] == 0))
        {
          /*Incrementa o numero de hops */
          pacote[24]++;
          /* Informa que e este o no que esta conectado ao no de destino
*/
          pacote[25] = ID;
          /* Guarda os RSSI de Download e Upload entre este no e o no
final, para o Proxy Manager avaliar */
          pacote[26] = pacote[0];
          pacote[27] = pacote[2];
          /* Se o numero de hops atuais for igual ao numero maximo de
hops, ou seja, vai enviar para a base */
          if (pacote[24]+1 == pacote[23])
           {
             pacote[8] = 0;
             pacote[10] = ID;
           }
          /* Caso contrario, caso o numero de hops seja menor, ou seja
ainda existem nos para passar */
          else if (pacote[24]+1 < pacote[23])
           {
```

```
              posicao_rota = 26 + pacote[23] - pacote[24] ;
              pacote[8]  = pacote[posicao_rota];
              pacote[10] = ID;
            }
           /* Envia para a camada inferior */
           sendMAC();
         }
       /* Caso contrario, caso a flag indique que e uma resposta de rota e
este seja um no que nao seja o no diretamente ligado ao no de destino */
       else if (pacote[11] == 64)
         {
           pacote[24]++;
           /* Se o numero de hops atuais for igual ao numero maximo de
hops, ou seja, vai enviar para a base */
           if (pacote[24]+1 == pacote[23])
            {
              pacote[8] = 0;
              pacote[10] = ID;
            }
           /* Caso contrario, caso o numero de hops seja menor, ou seja
ainda existem nos para passar */
           else if (pacote[24]+1 < pacote[23])
            {
              posicao_rota = 26 + pacote[23] - pacote[24] ;
              pacote[8]  = pacote[posicao_rota];
              pacote[10] = ID;
            }

           /* Envia para a camada inferior */
           sendMAC();

         }

       /* Caso contrario, caso a flag indique que e uma solicitacao de
dados para os sensores, e este no e apenas um no intermediario, na ida
para o sensor */
       else if ((pacote[11] == 254)&&(pacote[22] != ID))
         {
           /* Incrementa o numero de hops atuais */
           pacote[24]++;

           /* Determina qual sera o proximo endereco, e configura o campo
de destino com o mesmo */
           posicao_rota = pacote[24] + 28;
           pacote[8]  = pacote[posicao_rota];
           pacote[10] = ID;

           /* Envia para a camada inferior */
           sendMAC();

         }

       /* Caso contrario, caso a flag indique que e uma solicitacao de
dados para os sensores, e este no e o no de destino final */
       else if ((pacote[11] == 254)&&(pacote[22] == ID))
         {

           /* Zera o numero de hops atuais (indicando um retorno)*/
           pacote[24] = 0;

           /* Muda a flag de roteamento para indicar retorno positivo de
```

```
dados dos sensores*/
          pacote[11] = 255;

          /* Troca endereco de destino e de origem */
          pacote[8] = pacote[10];
          pacote[10] = ID;


          /* Envia para a camada superior */
          receiveTRANSP();

        }

      /* Caso contrario, caso a flag indique que e um retorno de dados
(para a base)  */
      else if (pacote[11] == 255)
        {
         /* Incrementa o numero de hops atuais */
         pacote[24]++;

          /* Se o numero de hops atuais for igual ao numero maximo de
hops, ou seja, vai enviar para a base */
         if (pacote[24]+1 == pacote[23])
          {
            pacote[8]  = enderecoSorvedouro;
            pacote[10] = ID;
          }

         /* Caso contrario, caso o numero de hops seja menor, ou seja
ainda existem nos para passar */
         else if (pacote[24]+1 < pacote[23])
          {
            posicao_rota = 26 + pacote[23] - pacote[24];
            pacote[8] = pacote[posicao_rota];
            pacote[10] = ID;
          }

         /* Envia para a camada inferior */
         sendMAC();
        }

    }

  }

  //Recebe camada TRANSP
  void NS::receiveTRANSP() {

    //A principio nao faz nada
    //Chama a camada de Applicacao
    receiveAPP();

  }

  //Recebe camada APP
  void NS::receiveAPP() {

    //A principio nao faz nada
    //Chama a camada de Applicacao
    sendAPP();
```

```
  }

  //Envia camada APP
  void NS::sendAPP() {

    //A principio nao faz nada
    //Chama a camada de Transporte
    sendTRANSP();
  }

  //Envia camada TRANSP
  void NS::sendTRANSP() {

    contador_pacote_TX = contador_pacote_TX + 1;
    pacote[12] = contador_pacote_TX;

    //Chama a camada de rede
    sendNWK();
  }

  //Envia camada NWK
  void NS::sendNWK() {

    // A principio nao faz nada

    //Envia para a camada MAC
    sendMAC();
  }

  //Envia camada MAC
  void NS::sendMAC() {

    //A principio nao faz nada

    //Envia para a camada PHY
    sendPHY();

  }

  //Envia camada fisica
  void NS::sendPHY() {

          //Registra a RSSI no pacote
          pacote[0] = rssiTensao;

          // Registra a RSSI no pacote para historico
          // Pega a posicao
          int posicao = 46 + pacote[24]; //- MODIFICACAO PARA A LINHA A
BAIXO?????????????????????/
          //int posicao = 45 + pacote[24];
          // Grava na posicao correta
          pacote[posicao] = pacote[0];

          //Zera o valor de tensao de RSSI
          rssiTensao = 0;

          //Configura status de saida para 0
          retorno = 0;

  }
```

```
  //Cria metodo para processar o pacote
  // Caso a mesma retornar 0, o pacote foi processado com sucesso
  // Caso a mesma retornar 1, o pacote chegou corrompido
  int NS::processPacket() {

    //Muda o valo da variavel retorno para um valor livre
    retorno = 10;

    //Chama o metodo receivePHY
    receivePHY();

    //Retorna o status de saida (obtido atraves do atributo retorno
    return retorno;
  }

//Cria os NS
NS ns1(1);
NS ns2(2);
NS ns3(3);
NS ns4(4);
NS ns5(5);

void setup() {

  //Inicia objeto da serial
  Serial.begin(9600);
}

void loop() {

   //Verifica se tem algo na serial
   if (Serial.available() >=52) {


      // Incrementa o valor de looping
      looping++;


      //Pega o pacote na serial...
      for (int i = 0; i < 52; i++) {
        pacote[i] = Serial.read();
        delay(1);
      }

      if (looping <=  quantidadeVezesPorCenario ) {

      //INICIO DO 1 CENARIO DE TESTES


      //Inicia a variavel resultado (para olhar a resposta da analise do
processamento dos NS
      //Inicia o valor da mesma com 10 ( um valor neutro, que nao vai
interferir com os valores retornados
      int resultado = 10;


      //Variavel acertoEntregaPacote comeca com o valor 1, significando
sucesso de entrega (ou seja, looping pode continuar)
      // Caso o mesmo durante o looping ter seu valor mudado para 0, houve
uma perda de pacote.
```

```
      int acertoEntregaPacote = 1;

      //Declara variaveis para armazenar os enderecos de origem e destino.
      int origem  = pacote[10];
      int destino = pacote[8];

      // Declara uma variavel para receber a quantidade de NS que
processaram o pacote e NAO era para eles
      int quantidadeRecebidosNS = 0;


      //Entra em um loop de tentativa de entrega...
      while ((acertoEntregaPacote == 1)&&(destino !=
enderecoSorvedouro)&&(quantidadeRecebidosNS < quantidadeNS)) {

         //Zera a variavel quantidadeRecebidosNS
         quantidadeRecebidosNS = 0;

         // Percorre os enlaces para determinar a probabilidade de entrega
do pacote
         for (int k = 0; k < quantidadeEnlaces; k++) {

            //Se os NS em questao pertecem ao enlace a ser comparado...
            if (((origem == matrizEnlaces1[k][0])&&(destino ==
matrizEnlaces1[k][1])) || (destino == matrizEnlaces1[k][0])&&(origem ==
matrizEnlaces1[k][1])) {

               //Pega a porcentagem da probabilidade de erro da entrega de
pacote...
               float probabilidadeErro =
matrizProbabilidadePerda[int(matrizEnlaces1[k][2])][2];

               //Calcula se ocorreu um erro ou nao...
               float randNumber = ((float)rand()/(float)RAND_MAX) * 100;

               //Se o erro NAO ocorreu...
               if (!(randNumber <= probabilidadeErro)) {


                 //Envia para  todos os NS

                 resultado = ns1.processPacket();
                 if (resultado == 0) {
                    //Encerra, pois o NS de destino ja processou o pacote
                    break;
                 }
                 else {
                   quantidadeRecebidosNS++;
                 }


                 resultado = ns2.processPacket();
                 if (resultado == 0) {
                    //Encerra, pois o NS de destino ja processou o pacote
                    break;
                 }
                 else {
                   quantidadeRecebidosNS++;
                 }
```

```
                  resultado = ns3.processPacket();
                  if (resultado == 0) {
                     //Encerra, pois o NS de destino ja processou o pacote
                     break;
                  }
                  else {
                    quantidadeRecebidosNS++;
                  }


                  resultado = ns4.processPacket();
                  if (resultado == 0) {
                     //Encerra, pois o NS de destino ja processou o pacote
                     break;
                  }
                  else {
                    quantidadeRecebidosNS++;
                  }

                  resultado = ns5.processPacket();
                  if (resultado == 0) {
                     //Encerra, pois o NS de destino ja processou o pacote
                     break;
                  }
                  else {
                    quantidadeRecebidosNS++;
                  }


               }
               //Caso contrario, o erro OCORREU...
               else {

                  //Variavel acertoEntregaPacote passara a ter o valor 0,
indicando um erro de entrega
                  acertoEntregaPacote = 0;

               }

               //Ja que descobriu qual e o enlace, quebra o laco de
repeticao...
               break;
            }

         }

         //Por fim, atualiza o endereco de origem e destino do proximo
pacote
         origem  = pacote[10];
         destino = pacote[8];
      }

      //Caso o destino seja o No Sorvedouro, escrever na serial
      if ((destino == enderecoSorvedouro )&&(resultado == 0)) {
         for (int i = 0; i < 52; i++) {
            //Escreve o pacote na serial
            Serial.write(pacote[i]);

         }

       Serial.flush();
```

```
         // Zera o vetor pacote
         for (int i = 0; i < 52; i++) {
            pacote[i] = 0;
         }
      }

      quantidadeRecebidosNS = 0;
      acertoEntregaPacote = 1;
      resultado = 10;
   }

 else if (looping <=  quantidadeVezesPorCenario*2) {

      //INICIO DO 2 CENARIO DE TESTES

      //Inicia a variavel resultado (para olhar a resposta da analise do
processamento dos NS
      //Inicia o valor da mesma com 10 ( um valor neutro, que nao vai
interferir com os valores retornados
      int resultado = 10;

      //Variavel acertoEntregaPacote comeca com o valor 1, significando
sucesso de entrega (ou seja, looping pode continuar)
      // Caso o mesmo durante o looping ter seu valor mudado para 0, houve
uma perda de pacote.
      int acertoEntregaPacote = 1;

      //Declara variaveis para armazenar os enderecos de origem e destino.
      int origem  = pacote[10];
      int destino = pacote[8];

     // Declara uma variavel para receber a quantidade de NS que
processaram o pacote e NAO era para eles
     int quantidadeRecebidosNS = 0;

      //Entra em um loop de tentativa de entrega...
      while ((acertoEntregaPacote == 1)&&(destino !=
enderecoSorvedouro)&&(quantidadeRecebidosNS < quantidadeNS)) {

         //Zera a variavel quantidadeRecebidosNS
         quantidadeRecebidosNS = 0;

         // Percorre os enlaces para determinar a probabilidade de entrega
do pacote
         for (int k = 0; k < quantidadeEnlaces; k++) {

            //Se os NS em questao pertecem ao enlace a ser comparado...
            if (((origem == matrizEnlaces2[k][0])&&(destino ==
matrizEnlaces2[k][1])) || (destino == matrizEnlaces2[k][0])&&(origem ==
matrizEnlaces2[k][1])) {

               //Pega a porcentagem da probabilidade de erro da entrega de
pacote...
               float probabilidadeErro =
matrizProbabilidadePerda[int(matrizEnlaces2[k][2])][2];

               //Calcula se ocorreu um erro ou nao...
               float randNumber = ((float)rand()/(float)RAND_MAX) * 100;

               //Se o erro NAO ocorreu...
               if (!(randNumber <= probabilidadeErro)) {
```

```
                //Envia para  todos os NS

                resultado = ns1.processPacket();
                if (resultado == 0) {
                   //Encerra, pois o NS de destino ja processou o pacote
                   break;
                }
                else {
                  quantidadeRecebidosNS++;
                }


                resultado = ns2.processPacket();
                if (resultado == 0) {
                   //Encerra, pois o NS de destino ja processou o pacote
                   break;
                }
                else {
                  quantidadeRecebidosNS++;
                }

                resultado = ns3.processPacket();
                if (resultado == 0) {
                   //Encerra, pois o NS de destino ja processou o pacote
                   break;
                }
                else {
                  quantidadeRecebidosNS++;
                }


                resultado = ns4.processPacket();
                if (resultado == 0) {
                   //Encerra, pois o NS de destino ja processou o pacote
                   break;
                }
                else {
                  quantidadeRecebidosNS++;
                }

                resultado = ns5.processPacket();
                if (resultado == 0) {
                   //Encerra, pois o NS de destino ja processou o pacote
                   break;
                }
                else {
                  quantidadeRecebidosNS++;
                }


              }
              //Caso contrario, o erro OCORREU...
              else {

                 //Variavel acertoEntregaPacote passara a ter o valor 0,
indicando um erro de entrega
                 acertoEntregaPacote = 0;

              }
```

```
                //Ja que descobriu qual e o enlace, quebra o laco de
repeticao...
                break;
             }

          }

          //Por fim, atualiza o endereco de origem e destino do proximo
pacote
          origem  = pacote[10];
          destino = pacote[8];
       }

       //Caso o destino seja o No Sorvedouro, escrever na serial
       if ((destino == enderecoSorvedouro )&&(resultado == 0)) {
          for (int i = 0; i < 52; i++) {
             //Escreve o pacote na serial
             Serial.write(pacote[i]);


          }

          // Zera o vetor pacote
          for (int i = 0; i < 52; i++) {
             pacote[i] = 0;
          }
       }

       quantidadeRecebidosNS = 0;
       acertoEntregaPacote = 1;
       resultado = 10;

       Serial.flush();

 // Fim do 2 Cenario
 }

  else if (looping <=  quantidadeVezesPorCenario*3) {


       //INICIO DO 3 CENARIO DE TESTES


       //Inicia a variavel resultado (para olhar a resposta da analise do
processamento dos NS
       //Inicia o valor da mesma com 10 ( um valor neutro, que nao vai
interferir com os valores retornados
       int resultado = 10;

       //Variavel acertoEntregaPacote comeca com o valor 1, significando
sucesso de entrega (ou seja, looping pode continuar)
       // Caso o mesmo durante o looping ter seu valor mudado para 0, houve
uma perda de pacote.
       int acertoEntregaPacote = 1;

       //Declara variaveis para armazenar os enderecos de origem e destino.
       int origem  = pacote[10];
       int destino = pacote[8];

      // Declara uma variavel para receber a quantidade de NS que
processaram o pacote e NAO era para eles
```

```
      int quantidadeRecebidosNS = 0;


       //Entra em um loop de tentativa de entrega...
       while ((acertoEntregaPacote == 1)&&(destino != 
enderecoSorvedouro)&&(quantidadeRecebidosNS < quantidadeNS)) {

          //Zera a variavel quantidadeRecebidosNS
          quantidadeRecebidosNS = 0;

          // Percorre os enlaces para determinar a probabilidade de entrega 
do pacote
          for (int k = 0; k < quantidadeEnlaces; k++) {

             //Se os NS em questao pertecem ao enlace a ser comparado...
             if (((origem == matrizEnlaces3[k][0])&&(destino == 
matrizEnlaces3[k][1])) || (destino == matrizEnlaces3[k][0])&&(origem == 
matrizEnlaces3[k][1])) {

                //Pega a porcentagem da probabilidade de erro da entrega de 
pacote...
                float probabilidadeErro = 
matrizProbabilidadePerda[int(matrizEnlaces3[k][2])][2];

                //Calcula se ocorreu um erro ou nao...
                float randNumber = ((float)rand()/(float)RAND_MAX) * 100;

                //Se o erro NAO ocorreu...
                if (!(randNumber <= probabilidadeErro)) {
                  //Envia para  todos os NS

                  resultado = ns1.processPacket();
                  if (resultado == 0) {
                     //Encerra, pois o NS de destino ja processou o pacote
                     break;
                  }
                  else {
                    quantidadeRecebidosNS++;
                  }


                  resultado = ns2.processPacket();
                  if (resultado == 0) {
                     //Encerra, pois o NS de destino ja processou o pacote
                     break;
                  }
                  else {
                    quantidadeRecebidosNS++;
                  }
                  resultado = ns3.processPacket();
                  if (resultado == 0) {
                     //Encerra, pois o NS de destino ja processou o pacote
                     break;
                  }
                  else {
                    quantidadeRecebidosNS++;
                  }


                  resultado = ns4.processPacket();
                  if (resultado == 0) {
```

```
                    //Encerra, pois o NS de destino ja processou o pacote
                    break;
                  }
                  else {
                    quantidadeRecebidosNS++;
                  }

                  resultado = ns5.processPacket();
                  if (resultado == 0) {
                     //Encerra, pois o NS de destino ja processou o pacote
                     break;
                  }
                  else {
                    quantidadeRecebidosNS++;
                  }
                }
                //Caso contrario, o erro OCORREU...
                else {

                   //Variavel acertoEntregaPacote passara a ter o valor 0,
indicando um erro de entrega
                   acertoEntregaPacote = 0;
                }

                //Ja que descobriu qual e o enlace, quebra o laco de
repeticao...
                break;
             }
          }
          //Por fim, atualiza o endereco de origem e destino do proximo
pacote
          origem  = pacote[10];
          destino = pacote[8];
       }

       //Caso o destino seja o No Sorvedouro, escrever na serial
       if ((destino == enderecoSorvedouro )&&(resultado == 0)) {
          for (int i = 0; i < 52; i++) {
             //Escreve o pacote na serial
             Serial.write(pacote[i]);
          }
          // Zera o vetor pacote
          for (int i = 0; i < 52; i++) {
             pacote[i] = 0;
          }
       }
       quantidadeRecebidosNS = 0;
       acertoEntregaPacote = 1;
       resultado = 10;
     Serial.flush();
  // Fim do 3 Cenario
  }
  // Caso contrario, se ele nao pertencer a nenhum teste, comeca a resetar
os cenarios
  else {
    looping = 0;
  }
  //Fim da verificacao da serial
  }
}
```