UNIVERSIDADE FEDERAL FLUMINENSE

**GUSTAVO ZANATTA BRUNO**

# SISTEMA PARA MONITORAMENTO TERMOENERGÉTICO DE CPDs

Niterói

2013

UNIVERSIDADE FEDERAL FLUMINENSE

**GUSTAVO ZANATTA BRUNO**

# SISTEMA PARA MONITORAMENTO TERMOENERGÉTICO DE CPDs

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Redes e Sistemas Distribuídos e Paralelos.

Orientador:
Julius Leite, Ph.D.

Co-orientador:
Raphael Guerra, Dr.-Ing.

Niterói

2013

GUSTAVO ZANATTA BRUNO

SISTEMA PARA MONITORAMENTO TERMOENERGÉTICO DE CPDs

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Redes e Sistemas Distribuídos e Paralelos

Aprovada em Agosto de 2013.

BANCA EXAMINADORA

---

Prof. Raphael Guerra, Dr.-Ing.(IC/UFF) – Presidente

---

Prof. Orlando Gomes Loques Filho (IC/UFF)

---

Prof. Alexandre Sztajnberg (UERJ)

Niterói

2013

*Dedicatória(s): Dedico à minha mãe.*

# Agradecimentos

Agradeço ao meu orientador, Julius Leite, e ao meu Co-orientador Raphael Guerra, pelo apoio prestado durante o desenvolvimento do trabalho.

# Resumo

A sociedade é cada vez mais dependente de grandes centros de processamento de dados (CPDs). Sistemas de busca, sistemas de comércio eletrônico e computação em nuvem são exemplos de aplicações que necessitam de grandes CPDs. Um dos grandes desafios da Computação Verde é conciliar a demanda computacional desses centros com a necessidade de reduzir o seu custo de operação e o impacto ambiental causado pelo seu alto consumo de energia. Entre os principais responsáveis por esse consumo está o sistema de arrefecimento dessas instalações. Nesse contexto, propõe-se um sistema de monitoramento termoenergético para CPDs através de uma rede de sensores sem fio (RSSF) distribuída pela instalação. O sistema proposto deve ser capaz de exibir e prover para terceiros, em tempo real, os dados térmicos do CPD. A proposta aqui apresentada foi validada através de experimentos em um *cluster* real para avaliação de sua eficiência.

**Palavras-chave**: Economia de energia, centros de processamento de dados, computação em nuvem, sistemas de monitoramento, redes de sensores sem fio.

# Abstract

Society is becoming increasingly dependent on large data processing centers. Search systems, ecommerce systems and cloud computing are examples of applications that require large data centers. One of the major challenges of the Green Computing area is to reconcile the computational requirements of these centers with the need to reduce their operating costs and environmental impact caused by its high energy consumption. The cooling system of these facilities is one of the main cause of consumption. In this context, we propose a thermoenergetic monitoring system for datacenters through a wireless sensor network (WSN) distributed by the installation. The proposed system is able to display and provide to third parties, in real time, the thermal data of the datacenter. The proposal presented here has been validated through experiments on a real cluster to evaluate its efficiency.

**Keywords**: Energy saving, data center, cloud computing, monitoring systems, wireless sensor networks.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Abreviaturas e Siglas

| | | |
|---|---|---|
| AMQP | : | Advanced Message Queuing Protocol; |
| BD | : | Base de Dados; |
| COP | : | Coefficient of Performance; |
| CPD | : | Centro de Processamento de Dados; |
| CRAC | : | Computer Room Air Conditioning; |
| CRC | : | Cyclic Redundancy Check; |
| CSMA | : | Carrier Sense Multiple Access; |
| CTP | : | Collection Tree Protocol; |
| DSF | : | Django Software Foundation; |
| DVFS | : | Dynamic Voltage and Frequency Scaling; |
| ETX | : | Expected Transmission Count; |
| FTP | : | File Transfer Protocol; |
| FTSP | : | Flooding time Synchronization Protocol; |
| GPRS | : | General Packet Radio Service; |
| HTML | : | HyperText Markup Language; |
| IEEE | : | Institute of Electrical and Electronics Engineers; |
| JSON | : | JavaScript Object Notation; |
| LPL | : | Low Power Listening; |
| MAC | : | Media Access Control; |
| MVC | : | Model View Controler; |
| OLAP | : | Online Analytical Processing; |
| RSSF | : | Redes de Sensores sem Fio; |
| SO | : | Sistema Operacional; |
| TAPO | : | Thermal-Aware Power Optimization; |
| TAPO-dc | : | Thermal-Aware Power Optimization for datacenters; |
| TIC | : | Tecnologias da Informação e Comunicação; |
| URL | : | Uniform Resource Locator; |
| USB | : | Universal Serial Bus; |
| WRAP | : | Wireless Reliable Acquisition Protocol; |

XML : Extensible Markup Language;

# Capítulo 1

# Introdução

Com o passar dos anos a sociedade foi se tornando cada vez mais dependente de grandes centros de processamento de dados (CPDs), principalmente com popularização da Internet. Um CPD, também conhecido pelo nome *Data Center*, é o local onde são concentrados os equipamentos de processamento e armazenamento de dados de uma empresa ou organização. No entanto, estudos mostram que muitos desses CPDs têm gestão energética ineficiente e, portanto, oferecem várias oportunidades para o emprego de técnicas de economia energética [8]. Assim, um dos grandes desafios da Computação Verde é conciliar a demanda computacional dos CPDs com a necessidade de reduzir o seu consumo operacional, devido ao impacto ambiental causado pelo seu alto consumo de energia [16]. O aumento da eficiência energética acarreta na diminuição da demanda por geração de energia, levando à redução da emissão de gases causadores do efeito estufa.

As Tecnologias de Informação e Comunicação (TIC) contribuem diretamente para mais de 2% das emissões globais de $CO_2$. A tendência é que esta quantidade dobre até 2020 [8]. Neste cenário, as TIC ultrapassariam emissões de indústrias altamente poluentes, como a de aviação. Um estudo realizado pela *U.S. Environmental Protection Agency* [11] em 2007 estimou a demanda de pico desses sistemas em 7 gigawatts. Um estudo publicado em 2008 estimou que os CPDs de todo mundo estariam emitindo aproximadamente 116 milhões de toneladas anuais de carbono, mais que toda emissão da Nigéria [31]. Estimativas de 2010 indicaram que os CPDs consumiram cerca de 1.5% de toda energia consumida no mundo [23].

Por conta do calor dissipado pelos diversos equipamentos presentes em um CPD, os servidores precisam de refrigeração adequada para operar de forma confiável. Assim, sistemas de arrefecimento em muitos CPDs são ajustados para operarem em temperaturas muito baixas. Grandes *clusters* (um conjunto de computadores, que utiliza um tipo

especial de sistema operacional classificado como sistema distribuído) comerciais requerem milhares de processadores e uma grande área para a sua instalação, e na maioria das vezes a instalação, os servidores e a distribuição de carga são heterogêneas. Desta forma, surge naturalmente um desbalanceamento térmico em diversas localidades do CPD. Estima-se que para cada watt gasto com o processamento de dados, outro watt é consumido com arrefecimento [41]. Além disso, os gerentes de CPDs, por falta de informações para diagnosticar a causa real do problema, tendem a diminuir ainda mais a temperatura quando os servidores se encontram próximos a um limiar térmico operacional.

Dentre as abordagens já existentes para tratar o monitoramento termoenergético em CPDs predomina a característica delas serem altamente intrusivas (e.g. [34]), ou seja, não são transparentes para as aplicações que rodam nos servidores. Esse é o caso das políticas de gerenciamento de energia para CPDs que empregam alteração de frequência de operação do processador, ou desligamento de partes do hardware, por exemplo. Tais abordagens requerem uma avaliação caso-a-caso, de modo a respeitar os requisitos da aplicação. Diante disso, este trabalho tem como objetivo o desenvolvimento de uma solução genérica para monitoramento térmoenergético para CPDs utilizando uma solução para monitoramento baseada em redes de sensores sem fio (RSSF). Dados coletados dessa maneira podem futuramente auxiliar no desenvolvimento de soluções eficientes de controle energético.

Em um outro tipo de abordagem, soluções para monitoramento em CPDs fazem o uso de sensores com fio e necessitam de cabeamento para leitura dos dados [29]. Utilizar sensores internos, como aqueles presentes nas placas mãe dos servidores não é uma boa opção, já que esses sensores refletem a atividade da carga computacional do servidor e não as condições ambientais do CPD como um todo. Por fim, um problema fundamental é que não se pode depender da energia dos servidores para alimentar esses sensores (e.g. através de portas USB), pois os servidores podem ser desligados para manutenção, por economia de energia ou ainda por defeito. Feitas dessa forma essas soluções também são intrusivas.

## 1.1 Motivação

A dinâmica térmica em CPDs é de extrema importância, principalmente quando deseja-se detectar possíveis ilhas de calor, e assim realocar a carga computacional para evitá-las, ou até mesmo quando é necessário uma redução dos custos gerados pelo consumo energético

dos sistemas de arrefecimento nessas instalações. Esse tipo de fenômeno (*e.g.* ilhas de calor) é bem complexo e difícil de prever sem uma quantidade de informações precisas da instalação. A Figura 1.1 mostra a variabilidade de temperaturas através de uma imagem térmica de um conjunto de servidores [30]. Tal variação térmica, em conjunto com as diferentes topologias que os CPDs podem assumir, acarreta na necessidade de uma fonte de informações precisa da distribuição térmica na sala e que, ao mesmo tempo, não seja intrusiva. Sendo assim, escolheu-se desenvolver uma solução baseada em redes de sensores sem fio, uma vez que tal abordagem opera de forma eficiente e genérica (*i.e.* independente das características do CPD monitorado) face às inúmeras restrições (*i.e.* acessibilidade, interação) que podem ser encontradas em CPDs. Além disso, uma RSSF possibilita uma boa granularidade nos dados coletados de forma autônoma, isto é, informações mais precisas independentemente do estado (*e.g.* ligado, dormindo ou desligado) dos servidores do CPD.

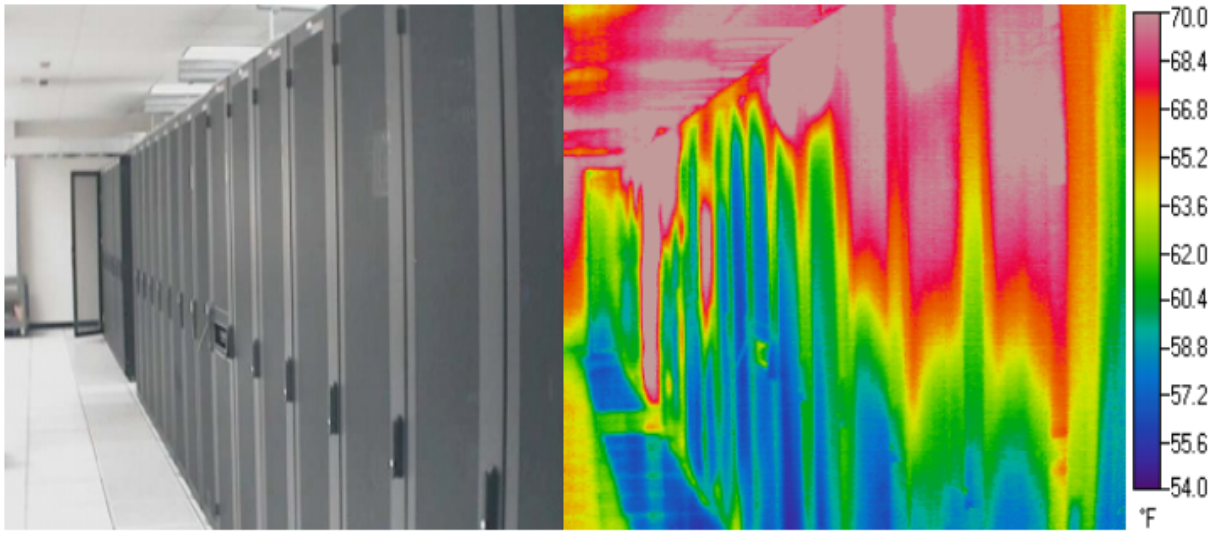



Figura 1.1: Dinâmica térmica em um CPD [30].

RSSFs para monitorar variáveis ambientais, como temperatura e umidade, têm sido amplamente utilizadas em diversas áreas, desde lavouras agropecuárias até estações espaciais [2]. Esse tipo de abordagem tem a principal vantagem de ser independente do ambiente a ser monitorado (*não-intrusivo*), pois a alimentação dos dispositivos é autônoma, através de baterias. Adicionalmente, por conta do baixo consumo, apresentam um grande tempo de vida. Em CPDs, principalmente aqueles que lidam com informações sensíveis (e.g., CPDs de instituições financeiras), a baixa intrusão é condição fundamental para a adoção de qualquer solução de monitoramento.

## 1.2 Objetivos

Nesse contexto, o objetivo deste trabalho é prover um sistema de monitoramento termoenergético para CPDs, através da utilização de uma rede de sensores sem fio. Essa solução é genérica, pois pode atender uma grande variedade de tipos de CPDs, pelo fato de não necessitar de nenhuma interação direta com os sistemas da instalação. O sistema proposto é capaz de coletar dados térmicos relativos ao ambiente do CPD com periodicidade de coleta com precisão de milissegundos, e disponibilizá-los a administradores do CPD por meio de uma interface clara e amigável. O sistema também provê acesso aos dados coletados para outras aplicações para que elas possam realizar otimizações no CPD com base na análise desses dados, como: detecção de anomalias que possam ocorrer nesse ambiente ou sugerir ao administrador do CPD possíveis melhorias com relação ao balanceamento de carga computacional para impedir a criação de ilhas de calor.

## 1.3 Organização

Este trabalho está organizado da seguinte forma: no Capítulo 2 é apresentado um levantamento sobre trabalhos relacionados à utilização de RSSFs para economia de energia em CPDs; no Capítulo 3 é descrita a infraestrutura de suporte utilizada; no Capítulo 4 é feito um detalhamento geral da arquitetura; no Capítulo 5 são descritos os detalhes de implementação. No Capítulo 6 são apresentados experimentos feitos para avaliar e validar vários aspectos do sistema. Finalmente, no Capítulo 7, são apresentadas as conclusões, contribuições e propostas para trabalhos futuros.

# Capítulo 2

# Trabalhos Relacionados

Este capítulo aborda alguns dos principais trabalhos que possuem relação com a solução aqui proposta. São apresentadas algumas propostas específicas para CPDs que levam em conta a dinâmica térmica no ambiente interno da instalação com o objetivo de aumentar a eficiência energética. Por fim serão descritas propostas que buscam monitorar os CPDs por meio de RSSFs e com base nas informações ambientais coletadas.

O capítulo está organizado da seguinte forma: na Seção 2.1 são descritas algumas soluções que buscam tratar o problema da dinâmica térmica dos CPDs de forma indireta (*e.g.* através de otimização da distribuição de carga); na Seção 2.2 estão as soluções que buscam tratar o problema termoenergético em CPDs utilizando RSSFs.

## 2.1 Trabalhos com Foco em Otimizações Termoenergéticas para CPDs

Nessa seção serão discutidas algumas propostas que têm foco na otimização da distribuição de carga visando uma redução da potência gasta pelo sistema de arrefecimento, o que consequentemente acarreta em economia energética.

### 2.1.1 *PowerTrade*

*PowerTrade* [1] é uma abordagem que faz uma otimização conjunta para equilibrar o consumo de energia dos servidores e do sistema de arrefecimento com o objetivo de reduzir o consumo de energia elétrica total do CPD. Essa solução para economia de energia em CPDs foi uma das primeiras que usaram esse tipo de otimização, que aborda de forma conjunta características dos servidores e do ambiente. Para fazer a otimização

o *PowerTrade* propôs que o CPD seja dividido em zonas térmicas, como ilustrado na Figura 2.1. A partir daí, foram empregados diversos métodos para alocação da carga de trabalho minimizando o consumo global de energia da instalação.

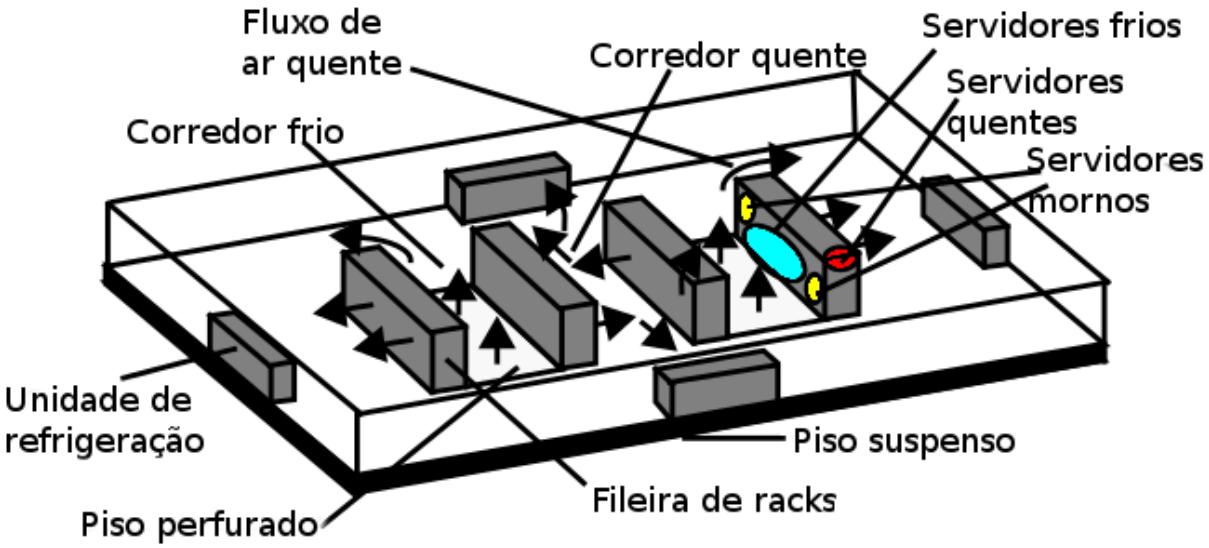



Figura 2.1: Layout do CPD e suas zonas térmicas [1].

Para facilitar o entendimento, os autores definem a energia total como a soma da energia gasta pelos servidores, acrescida da energia gasta pelo sistema de refrigeração e pelas perdas da distribuição de energia. Adicionalmente, define-se PUE (*power usage effectiveness*) como a potência total consumida pela instalação dividida pela potência gasta somente pelos equipamentos de TI, sendo que na potência consumida pela instalação entram o consumo dos servidores, do sistema de arrefecimento e as perdas na distribuição da energia.

O *PowerTrade* propõe reduzir a soma da energia consumida pelos servidores no estado ocioso com a energia consumida pelo sistema de arrefecimento. A energia necessária para o sistema de arrefecimento remover o calor da instalação é dada pela relação da energia consumida pelos servidores dividida pelo COP (*coefficient of performance*). O COP refere-se a uma métrica para calcular a eficiência energética de unidades do sistema de arrefecimento em CPDs, cujo valor é a razão entre a energia consumida pelos equipamentos de TI da instalação e a energia gasta pelo sistema de arrefecimento [47].

A subdivisão em zonas térmicas reduz o consumo energético total da instalação, concentrando a carga em menos servidores, desativando os desnecessários e consequentemente reduzindo o consumo de energia dos gasto por servidores ociosos. No entanto, a diminuição da quantidade de calor a ser removida por conta dos servidores inativos proporciona a concentração da carga de trabalho em certos pontos do CPD, gerando ilhas de calor.

O aumento da temperatura nesses pontos específicos reduzem o COP e aumentam a potência gasta pelo sistema de arrefecimento. Por outro lado, a distribuição das cargas nas áreas mais frias, melhora o COP e reduz a temperatura ao distribuir a carga entre os servidores para evitar as ilhas de calor. Entretanto essa melhora do COP acarreta no aumento o consumo por conta do aumento da quantidade de servidores ociosos, especialmente em períodos de cargas baixas (*i.e.* quando o CPD está operando bem abaixo da sua capacidade máxima).

Para reduzir a soma da potência gasta pelo sistema de arrefecimento com a potência gasta pelos servidores ociosos, o *PowerTrade* ativa mais servidores do que o estimado pela subdivisão em zonas térmicas, com objetivo de reduzir a carga em cada servidor e consequentemente a temperatura. Entretanto, com mais servidores ativos, a distribuição de carga tem menos impacto na redução do consumo dos servidores ociosos. Para este fim, o ideal é comparar o consumo do sistema de arrefecimento, mantendo um servidor ativo com o ganho de energia em ter um servidor no modo ocioso, redistribuindo a carga deste entre os servidores restantes ativos. Esta comparação permite reduzir a soma do consumo da energia gasta pelo servidor ocioso e pelo sistema de arrefecimento. No entanto, realizar esta comparação para cada servidor, em tempo de execução, no momento de cada requisição a esses servidores, ou ainda periodicamente ao longo do tempo, não é nada trivial para se implementar tendo em conta o grande número de servidores que esse tipo de instalação costuma possuir. Por isso, recorre-se a uma granularidade mais grossa (*i.e* um nível acima), explorando o fato do CPD poder ser dividido em zonas térmicas que se formam naturalmente devido aos padrões de fluxo de ar. Em configurações de *layout* tipicamente encontradas em CPDs, o ar quente sobe e o fluxo do ar é fraco nas extremidades dos corredores, fazendo com que os servidores localizados em cima e nas laterais sofram um maior aquecimento (ver Figura 2.1).

Devido à complexidade da divisão da carga entre as zonas térmicas, os autores separaram o problema de distribuição da carga em dois: um global, entre as zonas, e outro local, dentro de cada zona. O primeiro trata de como é distribuída a carga global entre as zonas (*i.e.* qual é a razão de distribuição de carga entre as zonas) e depois em como é a participação de um servidor na carga distribuída para sua zona térmica. Enquanto o primeiro problema trata somente a distribuição da carga entre as zonas, que são poucas, o segundo, isto é, a distribuição local, lida com os servidores dentro de cada zona, que são em grande número trazendo de volta o problema original de atribuir a carga aos inúmeros servidores. Para evitar esse problema, a proposta usa uma única abordagem para distribuição de carga local em cada zona e mantém o problema tratável. A partir daí, são

propostas três implementações para otimizar o consumo energético, duas estáticas e uma dinâmica, através da distribuição de cargas entre as zonas.

O *SurgeGuard* [1], é um método para manter o tempo de resposta em níveis satisfatórios. Ele consiste em uma técnica que provisiona o número de servidores ativos além do necessário para a carga corrente, de modo que absorva os possíveis aumentos inesperados de carga. O *SurgeGuard* é um esquema de duas camadas, uma que usa o bem conhecido excesso de provisionamento com uma precisão de tempo grosseira (*e.g.* uma hora) para absorver os extras (*i.e.* as oscilações de cargas mais suaves). Uma segunda camada é utilizada para absorver picos de carga inesperados, mais intensos e de menor quantidade. Essa última usa um esquema de provisionamento temporal mais fino (*e.g.* cinco minutos) para lidar com as cargas incomuns e oscilações bruscas.

Os autores é demonstram por meio de simulações utilizando dados reais que o *PowerTrade* e o *SurgeGuard* conseguem reduzir em 30% o consumo de energia total da instalação com relação outras soluções para redução de consumo energético em CPDs, aumentando apenas 1,7% o tempo de resposta. A proposta utiliza para comparação de resultados obtidos o trabalho denominado *PowerNap* [34], que difere muito da solução proposta e não proporciona uma comparação justa, e também não deixa claro o perfil das cargas utilizadas para execução dos testes.

O maior relação entre esse trabalho e a solução aqui apresentada é o fato dele monitorar as zonas térmicas para dar suporte a uma política de distribuição de carga entre os servidores. Entretanto o método para caracterização dessas zonas térmicas é feito indiretamente, isto é, através de estimativas. Além disso, o a solução proposta necessita previamente de traçar um perfil dos servidores para o cálculo do seu PUE, e também precisa controlar a distribuição de carga. Características como estas tendem a ser bem negativas no ponto de vista dos administradores de CPDs que lidam com informações sensíveis, por conta do nível de intrusividade que elas acarretam.

### 2.1.2 *TAPO*

O *Thermal-Aware Power Optimization* (*TAPO*) [20] é uma solução que interage somente com os componentes do sistema de refrigeração do CPD (*i.e.* sistema de refrigeração da instalação e dos servidores). Primeiramente o autor faz uma demonstração dos potenciais ganhos possíveis em economia de energia quando se ajusta de uma maneira mais fina o *setpoint* térmico (temperatura máxima de operação no ambiente do CPD). Depois propõe técnicas de gerenciamento de energia para procurar *setpoints* térmicos ótimos do ponto de

vista de consumo energético e em tempo de execução. Estas técnicas atuam basicamente em dois níveis: no nível do CPD e no nível do servidor.

Ao nível do CPD é proposta uma otimização que visa a redução do consumo energético do sistema de arrefecimento da instalação (*TAPO-dc*, isto é, TAPO para *datacenter*), que alterna entre dois diferentes *setpoints* (alto e baixo) para a zona de resfriamento, com base no seu nível de utilização. Essa abordagem otimiza o sistema de arrefecimento e agregados, e pode alcançar uma redução entre 12,4% e 17% no total de energia consumida pelo CPD sem perda de desempenho, segundo testes executados pelo autor. No nível do servidor, é medido o consumo energético em tempo de execução para ajustar o seu *setpoint* térmico e otimizar a potência do seu *cooler*.

O *TAPO-dc* é proposto para reduzir o consumo energético do CPD trabalhando com as configurações do sistema de arrefecimento. A solução busca otimizar o consumo energético da instalação mantendo o equilíbrio entre o consumo energético do sistema que é usado para levar o fluxo de ar frio para a entrada da zona de arrefecimento do CPD e a potência dos *coolers* utilizados para refrigeração dos interna dos servidores. A lógica básica é que quanto mais potência gasta pelo sistema de arrefecimento maior é o fluxo de ar frio nos servidores. Isso faz os *coolers* internos dos servidores trabalharem menos, resultando em economia de energia. Da mesma forma, um sistema de arrefecimento trabalhando em uma menor potência, uma maior quantidade de ar quente permanece ambiente, fazendo com que os *coolers* internos dos servidores trabalhem mais e consequentemente consumam mais energia. O *TAPO-dc* busca uma ponto de equilíbrio entre a potência consumida por essas duas partes visando minimizar o consumo energético de ambas.

Já o *TAPO-servidor* propõe uma técnica de otimização em tempo de execução que reduz a potência do *cooler* de um servidor que opera em potência máxima sem comprometer o seu desempenho. Dinamicamente e moderadamente ajusta-se o valor do *setpoint* térmico do processador. Isto é feito para atingir um estado operacional em tempo de execução que minimize o consumo energético dos *coolers* responsáveis pela dissipação de calor desse processadores.

O trabalho demonstra o quanto o sistema de arrefecimento, tanto o interno do servidor e o de toda a instalação consomem porções significantes de energia da instalação. Com base nisso propõe duas técnicas de gerenciamento de energia térmica (TAPO-dc, TAPO-servidor) que não dependem uma da outra e, portanto, pode ser implementadas de forma independente ou em combinação para atuar nessas partes. Implementando essa técnicas, embasados em um protótipo de trabalho tendo como servidores IBM POWER 750, os

autores conseguiram obter 5,4% de redução no total da energia pelo CPD sem perda de desempenho para uma carga de trabalho intensa, lembrando que interagindo somente com o sistema de arrefecimento. Por conta disso, a proposta tem um grau de intrusão médio, menos que a proposta da Seção 2.1.1, pois não atua em áreas como consolidação de carga dinamicamente e controle de potência do processador (*i.e.* DVFS).

## 2.2 Trabalhos com Foco em Otimizações Termoenergéticas para CPDs baseados em RSSFs

Nesta seção são abordados dois trabalhos que mais se assemelham com a solução aqui proposta. Ambas abordagens são baseadas em RSSF e visam economizar energia em CPDs atuando na complexa dinâmica térmica dessas instalações.

### 2.2.1 *ThermoCast*

O *ThermoCast* [29] é uma modelagem para previsão do comportamento térmico com o foco em grandes CPDs. Essa abordagem utiliza informações em tempo real da carga de trabalho, medidas de um conjunto de sensores de temperatura e fluxo de ar no ambiente para modelar e prever temperaturas em torno dos servidores. Assume-se que se tem conhecimento da temperatura de entrada de ar frio e saída de ar quente de cada servidor. Estes dados são obtidos a partir de sensores de temperatura colocados em alguns servidores. A partir daí foram identificados dois grandes desafios que estão presentes na construção de qualquer modelo em tempo real adaptativo: previsibilidade e escalabilidade. O modelo deve ser capaz de prever o superaquecimento cedo o suficiente, sem muitos falsos positivos/negativos, mesmo quando se alteram características do CPD (por exemplo, a carga e a potência do sistema de arrefecimento). O modelo também deve ser capaz de lidar com milhões de pontos de informações para monitorar em um CPD.

Com o intuito de obter alta previsibilidade, *ThermoCast* utiliza uma abordagem híbrida que leva em consideração a termodinâmica e as informações de carga. A proposta aprende e adapta os valores de vários parâmetros a partir dos dados obtidos em tempo real de sensores (Figura 2.2) e informações de carga de trabalho. Assim, é capaz de fornecer a previsão do comportamento térmico da instalação online mesmo quando as configurações sofrem mudança, tais como quando os servidores são ligados ou desligados, existência de um conjunto de servidores com carga de trabalho variável, configuração do sistema de arrefecimento alterada e manutenção de equipamentos da instalação. Para alcançar esca-

labilidade, utiliza-se a percepção de que a temperatura em torno de um servidor é afetada principalmente por características de seus servidores vizinhos, e conclui-se que os mais afastados não possuem tanto impacto. Portanto, *ThermoCast* baseia-se num modelo de zonas térmicas que constrói uma relação entre a temperatura de ventilação no corredor frio, a localização do servidor, a distribuição de temperatura local e a carga de trabalho dos servidores próximos, para prever a temperatura da entrada de cada servidor. Por causa da natureza local, *ThermoCast* distribui a tarefa de modelagem entre os servidores: cada servidor aprende e modela a temperatura em torno de si, usando medições dos sensores próximos e cargas de trabalho de servidores vizinhos. Com base nessas características, pode-se se dizer que o *ThermoCast* é computacionalmente e fisicamente escalável para um grande CPD.

Figura 2.2: Sensoriamento no *ThermoCast* [29].

A proposta foi implantada e avaliada em um CPD de laboratório com alguns *racks* e 40 servidores. Devido a densa instrumentação desse CPD, resultados mostraram a complexa dinâmica térmica que varia conforme o horário e a carga de trabalho sendo realizada pela

instalação. Com *traces* reais, mostrou-se que o *ThermoCast* pode prever picos térmicos com antecedência de 4,2 minutos, em comparação com 2,3 minutos usando um modelo baseado em regressão linear. Entretanto, a abordagem é considerada altamente intrusiva, pois o algoritmo para previsão térmica executa distribuído entre os servidores. Além disso, utiliza sensores cabeados para coleta das informações do ambiente em torno dos servidores, como também coleta as informações da carga de trabalho dos servidores em tempo real e distribuí a tarefa de modelagem para previsão térmica de anomalias a todos servidores.

### 2.2.2 *RacNet*

O *RACNet* [30] consiste em uma RSSF de grande escala e alta fidelidade para monitoramento ambiental (principalmente térmico) de CPDs. *RACNet* usa *nós* sensores, do tipo *Genomote*, que empregam uma combinação de comunicações com e sem fio. Como contribuição essencial do trabalho, desenvolveu-se um protocolo sem fio escalável e confiável para coleta de dados *WRAP* (*Wireless Reliable Acquisition Protocol*). Para solucionar o problema auto-interferência comum em redes sem fio densas, *WRAP* aproveita a frequência e o tempo de multiplexação, o que é conceitualmente diferente de abordagens anteriores para controlar o trafego e o congestionamento. Em particular, o *WRAP* usa vários canais de frequência do padrão IEEE 802.15.4 simultaneamente, de forma adaptativa, e equilibra o número de *nós* em cada canal com base no tráfego. *WRAP* também implementa a coordenação da coleta de dados através de um protocolo de passagem de *token*, permitindo que apenas um número controlado de nós esteja transmitindo por canal de frequência. Nota-se que o *WRAP* é diferente de outros protocolos de controle de congestionamento existentes na época de sua publicação. Enquanto os protocolos existentes detectam e reagem ao congestionamento, *WRAP* pró ativamente evita problemas que acarretam o congestionamento.

O *RACNet* coleta dados térmicos do ambiente a cada 30 segundos para detectar e mitigar condições térmicas anormais. A taxa de amostragem pode ser ainda maior quando for necessário tratar ilhas de calor ou quando são coletados outros tipos de informações (*e.g.* o monitoramento do consumo de energia do servidor). O *nó root* (mostrado à esquerda na Figura 2.3) e vários sensores com fio (mostrado no lado direito na Figura 2.3) formam uma cadeia. Cada uma dessa cadeia é colocada na entrada e na saída de ar de um *rack* em diferentes alturas. Este *layout* aumenta a cobertura de coleta e reduz as disputas pelo meio de transmissão, sem sacrificar a flexibilidade de implantação.


Figura 2.3: Modelo sensores RACnet [30].

O *WRAP* é o ponto central do *RACNet*. Como muitos protocolos de coleta de dados, o WRAP tem uma camada de rede que controla a topologia, e uma camada de transporte para recuperação dos dados. Especificamente, a camada de rede constrói árvores de coleta através de múltiplos canais, de uma forma distribuída. Já a camada de transporte depende de um mecanismo de passagem de *token* centralizado para prevenir o congestionamento da rede e de maneira confiável recuperar dados de cada um dos *nós* da rede. Observa-se também que o *WRAP* aproveita a oferta de energia a partir de portas USB dos servidores e por conta disso não tem maiores problemas com o gerenciamento do *duty-cycle* do rádio.

Os resultados da avaliação com base em uma simulação de um CPD de tamanho médio sugerem que *WRAP* é razoavelmente melhor que os protocolos existentes de coleta de dados (*e.g.* CTP [17], RCRT [42]). Especificamente, conforme a quantidade total de tráfego cresce, *WRAP* consegue alcançar rendimentos mais elevados. Além disso, os resultados de uma implantação da RSSF em larga escala mostram que o *WRAP* oferece desempenho estável com entrega de pacotes superior a 99%. Todavia, a solução tem 2 características que deixam a desejar. A primeira delas é o fato de usar fio, mesmo que seja para interligar as cadeias de monitoramento, "poluindo"o ambiente dos CPDs, e consequentemente aos seus administradores. O segundo ponto é o fato de usarem alimentação USB, característica essa que soluciona o problema da economia de energia (*duty-cycle*) em RSSF e proporciona uma granularidade fina nos dados coletados (*i.e.* uma coleta periódica a cada 30 segundos), porém, acaba tornando a solução mais intrusiva.

# Capítulo 3

# Infraestrutura de Suporte

Neste capítulo são discutidos detalhes da infraestrutura de suporte utilizada para desenvolvimento da RSSF e do sistema para monitoramento, disponibilização e exibição dos dados. Serão descritos o hardware, a plataforma de desenvolvimento e detalhes dos protocolos utilizados.

Durante os estudos iniciais para o desenvolvimento da proposta aqui apresentada optou-se por utilizar "soluções de prateleira"[1]. Por conta dos prazos disponíveis, verificou que esse era o meio com maior probabilidade de atingirmos o os resultados esperados. O que de fato foi verificado ao término do trabalho. Diante disso, foram escolhidas para desenvolvimento as soluções que possuem maior aceitação da comunidade que trabalha com RSSF.

O capítulo está organizado da seguinte forma, primeiramente na Seção 3.1 é feita uma descrição dos componentes e tecnologias utilizadas para desenvolvimento da RSSF. Por fim, na Seção 3.2 e 3.3 são descritos detalhes das bibliotecas e tecnologias usadas para desenvolvimento do sistema de disponibilização e exibição em tempo real das informações coletadas pela rede.

## 3.1 Rede de Sensores Sem Fio

Uma rede de sensores sem fio baseada no padrão do *Institute of Electrical and Electronic Engineers* (IEEE) 802.15.4 [46] foi utilizada para coleta dos dados. Esse padrão estabelece um protocolo de interconexão compatível com dispositivos de comunicação de dados, cujas

[1]Um software ou hardware pré-desenvolvido por um fornecedor terceiro. Ele pode ser alugado, comprado ou gratuito (*e.g.* de código aberto).

faixas de potência de transmissão utilizadas são baixas, e a comunicação é de curto alcance por meio de rádio frequência.

A Seção está organizada na seguinte maneira, na Subseção 3.1.1 encontra-se detalhado o hardware e na Subseção 3.1.2 o software, utilizado e desenvolvido para a RSSF.

### 3.1.1 *Hardware*

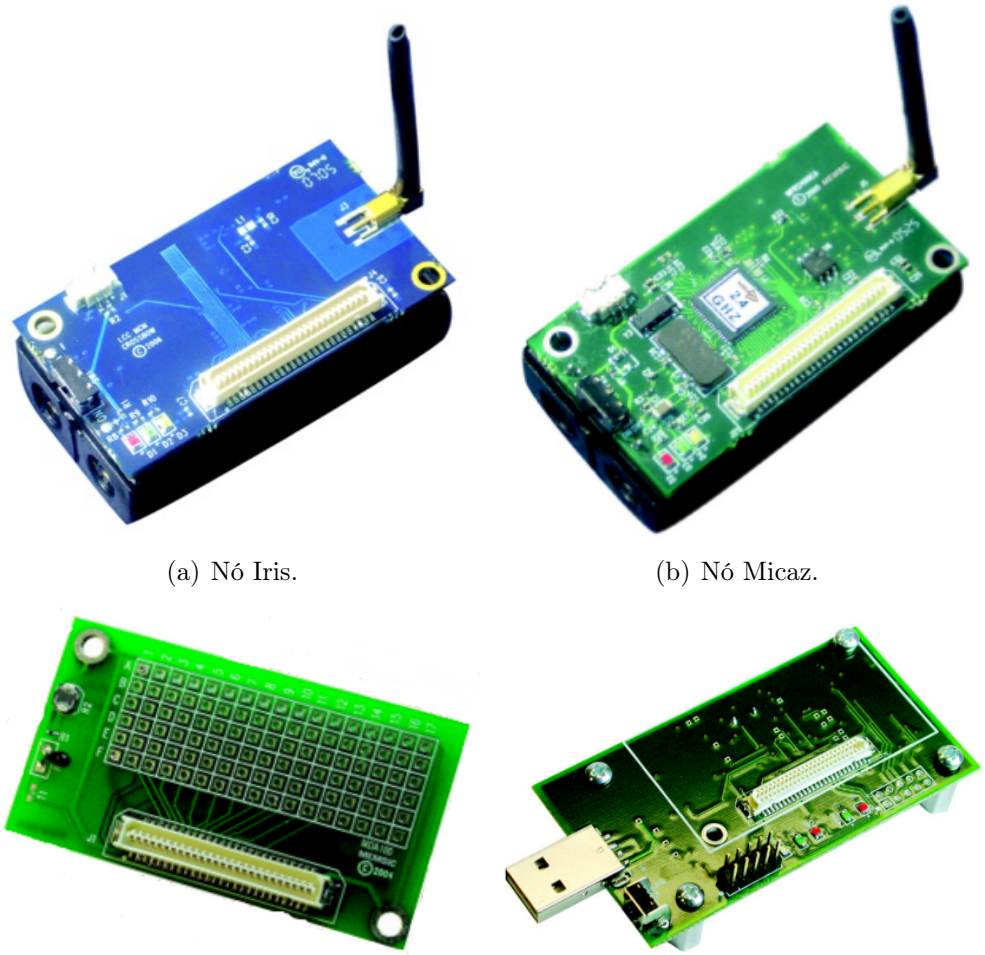
#### 3.1.1.1 Dispositivos Utilizados na RSSF

A RSSF proposta neste trabalho pode ser composta por dois tipos de dispositivos de comunicação e processamento, que podem ser do modelo *Iris* [35], exibido na Figura 3.1(a), ou *Micaz* [38] exibido na Figura 3.1(b), nos quais fica armazenada e é executada a aplicação embarcada responsável pela coleta das informações do ambiente. Os dispositivos que enviam as informações para os computadores base, isto é, os *nós root*, estão acoplados a uma placa *gateway* modelo MIB 520 [37], exibida na Figura 3.1(d), que possibilita a comunicação entre a RSSF por meio de uma porta no padrão *Universal Serial Bus* (USB). Nos demais dispositivos são acoplados a placa de sensoriamento MDA 100 [36], exibida na Figura 3.1(c), que provê sensoriamento de temperatura e luminosidade.

#### 3.1.1.2 Topologia da RSSF

Uma organização comum empregada em CPDs é a disposição dos *racks* em corredores frios e quentes. A Figura 3.2 ilustra um corte transversal em um centro de dados típico. Os *racks* de servidores são instalados em um piso elevado, o ar frio é injetado pelo sistema de arrefecimento através de grelhas nos corredores frios. Por sua vez, os servidores sugam ar frio e expelem ar quente (nos chamados corredores quentes) através de suas ventoinhas. De forma a diminuir a recirculação de ar, os *racks* são dispostos de forma que as saídas dos servidores (ar quente) fiquem face a face.

Além dos desafios impostos pelo sistema de arrefecimento (*e.g.* diferenças na distribuição do ar refrigerado), também existem as diferenças térmicas geradas pelas heterogeneidades do CPD. Essas heterogeneidades causadas pelas diferenças de *hardware*, *software* e carga computacional podem gerar comportamentos térmicos indesejáveis no ambiente (*e.g.* ilhas de calor). Visando capturar as informações térmicas desse tipo de ambiente, os *nós* sensores são dispostos verticalmente em várias alturas nos *racks*. A quantidade de sensores por *rack* varia em função da granularidade da informação térmica desejada. Nos casos em que o *nó* consumidor estiver muito longe dos *nós* coletores são inseridos *nós*



(a) Nó Iris. (b) Nó Micaz.

(c) Placa MDA100. (d) Placa MIB520.

Figura 3.1: Dispositivos da RSSF.

roteadores.

Uma vez definida a distribuição física dos sensores, o grafo da topologia de comunicação da RSSF é montado de maneira automática pelo protocolo de rede *Collection Tree Protocol* (CTP) [17], no qual os *nós* são organizados de maneira a minimizar o número de retransmissões (por perdas de mensagens ou falhas nos *nós* da rede).

Cada *nó* de sensoriamento da RSSF possui capacidade de processamento, armazenamento e comunicação *wireless*, além de sensoriamento de temperatura e luminosidade. Esses nós podem desempenhar diversos papéis como produtores, roteadores e consumidores. Os *nós* produtores são responsáveis por fazer as leituras dos sensores e as enviar para o *nó* consumidor. Os *nós* roteadores são responsáveis por repassar mensagens entre *nós* que não têm alcance para se comunicar diretamente com o *nó* consumidor. Por fim, o *nó* consumidor (*root*) tem a função de receber todas as medições e repassá-las para um

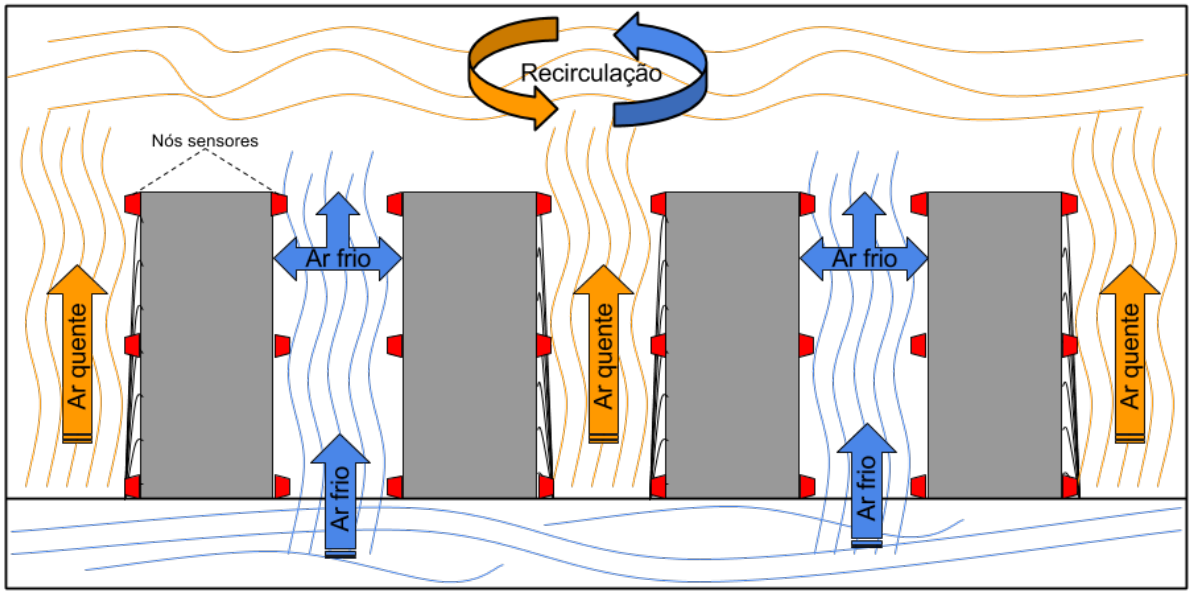



Figura 3.2: Dinâmica do fluxo de ar em um CPD típico.

servidor. O número de *nós* consumidores vai depender da escala do CPD. Além disso, um *nó* produtor pode desempenhar também o papel de roteador. Uma ilustração da topologia da RSSF empregada pode ser vista na Figura 3.3.

#### 3.1.1.3 Classificação da RSSF

No início da década passada, quando estavam inciando-se as pesquisas com as RSSF, uma metodologia para classificação destas de forma eficiente, com base em suas características, ainda não havia sido desenvolvida. Esse fato levava os desenvolvedores/pesquisadores a classificar as RSSF utilizadas de maneiras distintas, ou muitas vezes nem as classificavam. Os analistas, principalmente de requisitos, tinham uma certa dificuldade para descrever as características básicas que a RSSF a ser projetada teria. Durante congressos especializados verificou-se a necessidade da padronização de uma metodologia para descrição das características, que possibilitasse a classificação das RSSF. Em [45] é proposta uma metodologia que atende a grande parte das RSSF, muito utilizada em trabalhos posteriores a sua publicação. A classificação da RSSF utilizada na solução de monitoramento proposta neste trabalho, segundo a metodologia proposta por RÖMER em [45], é detalhada na Tabela 3.1.

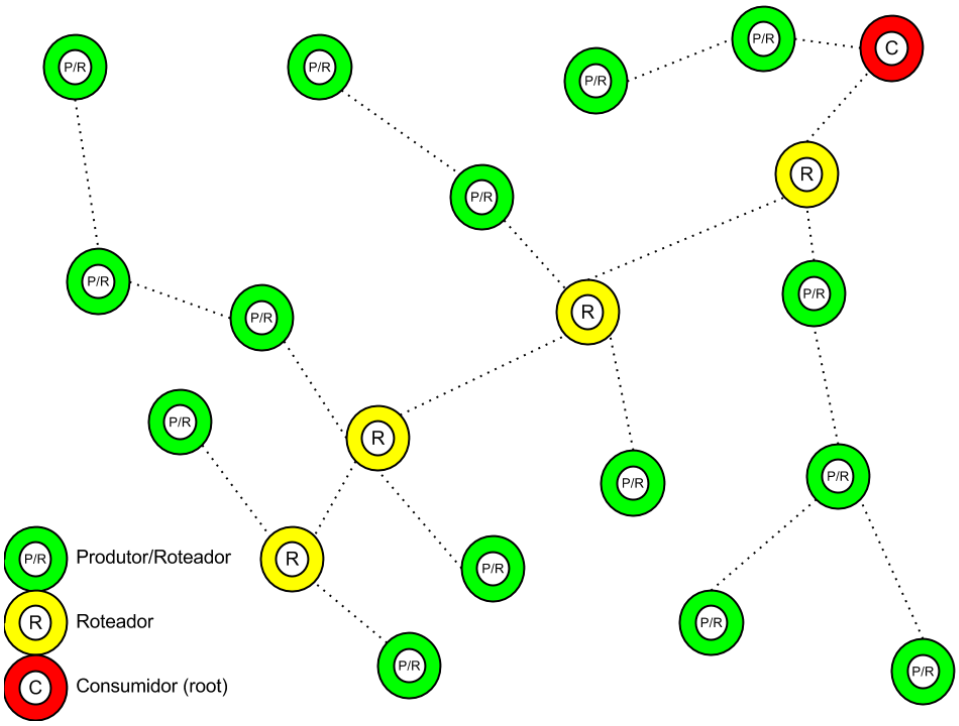



Figura 3.3: Grafo da RSSF.

### 3.1.2 *Software*

#### 3.1.2.1 Sistema Operacional

O SO adotado para desenvolvimento da aplicação embarcada para a RSSF é o *TinyOS* [28] na versão 2.1.2. Esse SO é empregado principalmente em RSSFs. A sua primeira versão foi escrita em 2000 e na época os usuários *TinyOS* eram apenas alguns pesquisadores de Ciência da Computação. Uma década mais tarde, a média de downloads do *TinyOs* chegou a 25.000 por ano, e atualmente está presente em muitos produtos comerciais e continua a ser a principal plataforma usada para trabalhos que envolvam RSSFs, sistemas de baixo consumo de energia e pesquisas na área de comunicação sem fio [26].

O *TinyOS* tem um modelo de programação baseado em componentes, codificado na linguagem *NesC* [15], um dialeto da linguagem de programação C. O *TinyOS* não é como os SOs tradicionais para *desktop*. Ele é uma estrutura de programação para sistemas embarcados que agrega um conjunto de componentes que permitem a construção de uma aplicação específica para cada contexto. Segundo [28], uma aplicação típica tem cerca de 16 kilobytes enquanto o sistema operacional de base (componentes para abstração de hardware básico e para gerenciamento de *boot*) tem cerca de 400 bytes. Uma aplicação considerada grande, com sistema de consulta a uma base de dados, tem de cerca de 64 kilobytes.

Tabela 3.1: Classificação da RSSF.

| *Parâmetro* | *Valor* |
|---|---|
| Forma de instalação | Manual |
| Mobilidade | Imóvel |
| Tamanho dos *nós* | Caixa de Fósforo |
| Alimentação | Bateria |
| Heterogeneidade | Heterogêneos |
| Modalidade de comunicação | Rádio |
| Infra-estrutura | *Base Station e Gateway* |
| Topologia | Árvore |
| Cobertura | Densa |
| Conectividade | Intermitente |
| Tamanho da rede | Variável e escalável |

Um programa *TinyOS* é um grafo de componentes e cada um dos quais é uma entidade computacional independente que exporta uma ou mais interfaces. Cada componente possui três abstrações computacionais: comandos, eventos e tarefas. Comandos e eventos são mecanismos para comunicação entre componentes, enquanto as tarefas são usadas para expressar paralelismo.

Um comando é tipicamente um pedido feito a um componente para executar algum serviço, tal como a leitura de um sensor, enquanto que um evento sinaliza a conclusão do serviço. Eventos podem também ser sinalizados de modo assíncrono, por exemplo, devido a interrupções de hardware decorrente da chegada de alguma mensagem. Comandos e eventos não podem se bloquear, ao contrário, um pedido de serviço é dividido em duas fases. A primeira é requisição pedido de serviço (o comando) e a outra é o sinal de conclusão (o evento correspondente). No instante seguinte ao término do comando, o evento sinaliza que este foi concluído.

#### 3.1.2.2 Protocolo de Rede

O protocolo utilizado para manter a topologia da rede e possibilitar a comunicação por múltiplos saltos é o CTP [13,17]. Ele provê um serviço básico e bastante comum em redes de sensores, que é coletar dados de vários sensores espalhados no ambiente e transportá-los a um ou mais *nós* centrais (*i.e. roots*), como é ilustrado na Figura 3.4. A ideia básica é a construção automática de uma rede mantendo a topologia em árvore, com uma ou mais raízes dependendo da escala e densidade da RSSF. O protocolo de coleta constrói e mantém um caminho com custo mínimo para o(s) *nó*(s) que anunciam-se como raiz da

árvore. O protocolo é livre de endereço: quando existem múltiplas raízes, ele envia para uma com o custo mínimo e sem saber seu endereço. Não há especificação direta de destino nas mensagens trocadas pelos *nós*. O endereçamento de destino é indireto e sempre para uma raiz (*root*) da rede. Portanto, o endereçamento é *anycast*.

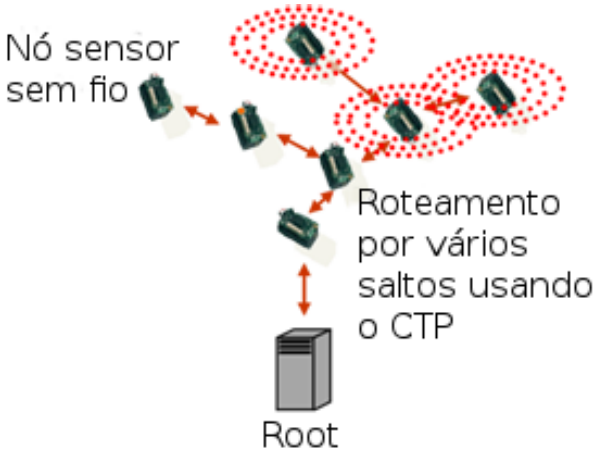



Figura 3.4: Funcionamento básico do CTP [17].

O CTP tem dois mecanismos que permitem que o protocolo de roteamento seja robusto para o gerenciamento das informações de rota antigas, e ágil e dinâmico ao detectar mudanças, mantendo a topologia estável e a um baixo custo energético. O primeiro mecanismo é a validação do caminho de dados: pacotes de dados (*beacons*) são enviados dinamicamente para investigar e validar a consistência da topologia de roteamento. O segundo é o *beaconing* adaptativo, baseado no algoritmo de propagação de código *Trickle* [27] aplicado no controle do tráfego gerado pelo mecanismo de validação da topologia de roteamento.

Para validar o caminho dos dados, cada *nó* mantém uma estimativa do custo de sua rota para um ponto de coleta (*i.e. root*). O custo de envio de um pacote para o *root* é o somatório dos custos da transmissão a cada salto, computada pela métrica *ETX*[2]. Cada pacote de dados contém a estimativa do custo do transmissor local. Quando um *nó* recebe um pacote para transmitir, ele compara o custo do *nó* transmissor com o seu próprio. Dado que o custo deve sempre diminuir, se o custo anunciado por um transmissor é menor do que o do *nó* receptor, então a informação do transmissor sobre a topologia da RSSF é obsoleta e pode haver um *loop* de roteamento. A validação do caminho de dados permite que o protocolo detecte possíveis *loops* de roteamento logo que eles aconteçam.

O mecanismo de coleta atualiza as informações de roteamento obsoletas enviando

[2]A métrica ETX, ou *expected transmission count*, é o número de transmissões necessárias para que um pacote possa ser recebido sem erro no seu destino. Este número varia um até o infinito [9].

*beacons* de controle. Tal como acontece com pacotes de dados, cada *beacon* contém a estimativa de custo do transmissor local. Mas ao contrário de pacotes de dados, os *beacons* de controle são transmissões periódicas. Um único *beacon* atualiza muitos *nós* próximos. Protocolos de coleta tipicamente transmitem *beacons* de controle em um intervalo fixo. O valor definido para esse intervalo tem os seguintes impactos: um pequeno intervalo reduz o quanto as informações podem estar desatualizadas e quanto tempo um *loop* de roteamento pode persistir, entretanto gera mais tráfego e consome mais energia. Um maior intervalo gera menos tráfego e consome menos energia, mas pode deixar problemas topológicos persistirem por um longo período de tempo.

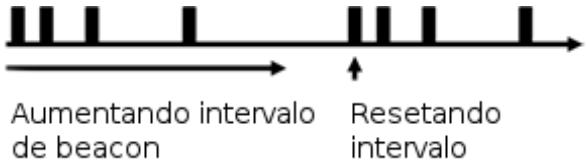



Figura 3.5: Periodicidade do envio de *beacons* pelo CTP [17].

Para manter um equilíbrio entre a robustez da topologia e o baixo consumo energético é utilizado o mecanismo de *beaconing* adaptativo, que é baseado no algoritmo *Trickle*. Esse algoritmo foi projetado para transmitir as informações da topologia da rede de forma confiável e eficiente em uma RSSF. O mecanismo básico do *Trickle* consiste na transmissão do número de versão e um identificador do *nó* usando um temporizador randômico. O algoritmo *Trickle* tem dois mecanismos para gerenciar essa transmissão: supressão e adaptação do intervalo do cronômetro. Se um *nó* ouve outro *nó* anunciar o mesmo número de versão, ele suprime a sua transmissão. Quando um intervalo de tempo terminar, o mecanismo irá enviar um *beacon* e o *Trickle* irá dobrar seu valor, isso até um valor máximo; quando o *nó* recebe um pacote de *beacon* com um número versão mais recente, reduz o intervalo para um valor mínimo. Se todos os *nós* têm o mesmo número de versão, seus intervalos do contador aumentarão exponencialmente, até um valor máximo. Além disso, apenas um pequeno subconjunto de *nós* transmitem por intervalo, pois uma única transmissão pode atingir muitos *nós* próximos fazendo com que eles suprimam a sua transmissão. Quando há alguma alteração, o intervalo diminui para o valor mínimo, fazendo com que os *nós* recebam rapidamente as novas informações, conforme é ilustrado na Figura 3.5. Outras formas que acarretam no reinício desses intervalos são descritas abaixo:

- Pedido para encaminhar um pacote de dados de um *nó* cujo custo de transmissão (ETX) não é superior ao seu próprio custo. Portanto é configurado um *looping* de

roteamento e as informações das tabelas de roteamento precisam ser atualizadas;

- Se o custo de roteamento através de outro *nó* é mais barato, sendo que essa redução de custo tem que ser maior que um determinado limiar. Esse limiar existe para manter a topologia mais estável;

- Recepção de um pacote com o bit *P*. Um *nó* define o bit *P* para envio no *beacon* quando ele não tem uma rota válida. Por exemplo, quando um *nó* percebe que a sua tabela de roteamento está vazia, por isso envia *beacons* com o bit *P* ativado.

Durante os testes iniciais com o CTP observou-se que ele continha uma falha que eventualmente ocasionava uma grande perda de pacotes na rede. A falha ocorria quando um *nó* perdia, por algum motivo, a comunicação com seu pai e ficava sem eleger um substituto até o próximo *beacon* da rede. Isso ocorria por que o CTP tentava $n$ vezes enviar pacotes para o pai incomunicável e, em vez de selecionar outro pai aumentava o ETX até a chegada do próximo *beacon*. Essa situação tornava-se mais grave quanto maior fosse o intervalo do *beacon* no momento da falha. Para corrigir o problema, o CTP foi alterado para toda vez que for identificado um ETX com valor maior que 100, o intervalo de *beacon* seja reiniciado e o bit *P* ativado para forçar os vizinhos a mandarem informações de suas tabelas de roteamento. Para não enviar *beacons* com uma frequência muito alta, o algoritmo foi alterado, conforme detalhado nó Código A.1 presente no Apêndice A, para enviar um *beacon* com o bit *P* apenas se o intervalo de *beacons* for maior que 5 segundos. O tempo máximo para envio do *beacon* ficou configurado para 10 minutos, com base nos resultados dos experimentos da Seção 6.1.3. Durante os testes observou-se que com esse valor é possível manter uma boa confiabilidade nas informações colhidas pela RSSF, não comprometendo de forma significativa o tempo de vida dos *nós*.

#### 3.1.2.3 Protocolo MAC

O protocolo utilizado na camada MAC é o Box-MAC-1 [39, 40], uma versão melhorada do popular Berkeley MAC (B-MAC) [43] para gerenciamento síncrono do *duty-cycle* do rádios dos *nós* da RSSF. O protocolo B-MAC utiliza informações somente da camada física, isto é, do meio, para detectar rapidamente alguma transmissão para si e ligar o rádio para recebe-la, durante suas verificações periódicas. Para efetivamente executar uma verificação de escuta os *nós* invertem a função de verificação do meio que é normalmente utilizada por protocolos baseados no *Carrier Sense Multiple Access* (CSMA) [22] para transmitir dados e a usa para receber. Estas verificações de canal livre permitem que o

rádio dos *nós* rapidamente retornem ao sono quando nenhuma transmissão dos vizinhos é detectada.

Para coincidir com as verificações periódicas do meio, os remetentes B-MAC são obrigados a enviar transmissões iniciais que consistem em longos preâmbulos de bits. Esses preâmbulos acordam todos os *nós* dentro da proximidade do transmissor, sendo ou não um determinado *nó* o destinatário do pacote a ser enviado. Depois de acordar, todos os receptores B-MAC permanecem ativos no modo de escuta e consomem energia durante toda a duração do longo preâmbulo de bits para despertar antes de adquirir o pacote. Isso resulta em um problema bem conhecido de escuta [6], que torna o B-MAC extremamente suscetível a ataques de negação de sono [5] e ataques simples de transmissores desonestos que podem acarretar em um grande desperdício energético. Verificou-se em na RSSF aqui proposta que utilizando o protocolo B-MAC a longevidade da RSSF não ultrapassava os seis dias, pois mesmo que os *nós* não estivessem transmitindo algo o rádio acabava sempre ligado e o consumo era semelhante ao exibido no gráfico da Seção 6.1.6.

O *Box-MAC-1* melhora o B-MAC ao adicionar informações da camada de enlace à transmissão do preâmbulo de despertar. Para adicionar esta informação da camada de enlace, a transmissão do B-MAC, que consiste em um fluxo de bits, é substituída por uma transmissão continua de pacotes de despertar. Esta substituição ajuda a prevenir que os *nós* próximos acordem desnecessariamente ao detectar transmissões destinadas a outros vizinhos. Assim, também torna-se desnecessário esperar o fim do preâmbulo, quando um *nó* detecta que tem um pacote desejando ser transmitido para ele, o remetente é avisado e pode enviar a informação instantaneamente.

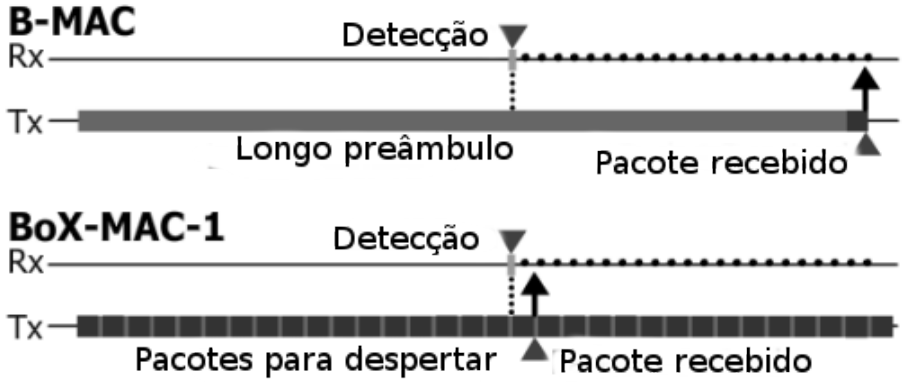



Figura 3.6: Comparação B-MAC e *Box-MAC-1* [40].

Conforme é ilustrado na Figura 3.6, quando um *nó* deseja enviar um pacote ele fica enviando pacotes de despertar até o *nó* destinatário fazer a verificação do meio e perceber que tem alguém desejando fazer uma transmissão. Assim que o *nó* destinatário checa o

meio e percebe que tem algum *nó* desejando enviar algo, ele ativa seu rádio para receber o pacote. Desta forma acontece a transmissão salto-a-salto até o pacote chegar na raiz da rede, o que gera uma latência e dependendo da aplicação, tem que ser tratada.

Um parâmetro do *Box-MAC-1* que deve ser configurado com cautela é o intervalo de verificação do meio, pois é necessário especificar um intervalo que forneça a melhor eficiência energética para as características da RSSF. Para a aplicação embarcada tratada neste trabalho, o intervalo escolhido foi de 300 milissegundos, o motivo desse intervalo esta detalhado no Capítulo 6. Vale lembrar também que quanto maior é esse intervalo, maior será o atraso da entrega salto-a-salto e não necessariamente é maior a eficiência energética.

Outro parâmetro não menos importante que também pode ser configurado é o tempo que o rádio fica ligado após o recebimento ou envio de um pacote. Em uma RSSF em que os *nós* frequentemente enviam informações que necessitam ser quebradas em mais de um pacote, esse parâmetro tem de ser calibrado de modo que o rádio não desligue após o recebimento de um pacote, mas também não fique tempo excessivo ligado consumindo energia. Como na RSSF abordada as informações que precisam ser transmitidas não extrapolam o tamanho de um pacote, esse valor é o mínimo possível para o *nó* receber uma confirmação de envio. Conforme testes executados, três milissegundos é o suficiente para esse parâmetro não interferir na taxa de entrega da rede e influenciar irrisoriamente no consumo energético.

Durante a implementação desse protocolo na aplicação embarcada ocorreram algumas dificuldades na integração dele com o protocolo de rede CTP. Foram necessárias algumas alterações triviais (*e.g.* alteração de nomes de variável). Esses problemas são provenientes de atualizações feitas pela comunidade desenvolvedora e que não foram testadas em todos os contextos (*e.g.* CTP integrado com o *Box-Mac-1*). Um e-mail foi enviado notificando a comunidade desenvolvedora.

#### 3.1.2.4 Protocolo para Sincronia de Relógios

Com intuito de coletar dados do ambiente de maneira mais precisa com relação ao instante em que eles são obtidos, optou-se por utilizar um protocolo de sincronia para manter os relógios dos *nós* na rede sincronizados. Para isso foi utilizado o protocolo FTSP [32]. Esse protocolo tem como objetivo prover sincronia entre os relógios de *nós* de RSSFs com erro máximo menor que um microssegundo, sendo escalável a redes com centenas de *nós* e mantendo a robustez independente de mudanças na topologia da rede e falhas diversas.

O algoritmo utilizado pelo protocolo compensa os erros ao empregar o conceito de MAC *timestamp* [33], isto é, a *timestamp* é inserida na camada MAC pouco antes do envio minimizando o risco de acumular atrasos devido a processos internos do *nó*. O desvio padrão entre os relógios é compensado utilizando uma regressão linear.

O principal objetivo do FTSP é prover um meio, confiável e eficiente energeticamente, para sincronizar todos os relógios dos *nós* da rede. Assume-se que cada *nó* possua um relógio local que apresenta os erros típicos devidos aos diferentes cristais e que podem-se comunicar de forma confiável com seus vizinhos. O protocolo sincroniza o tempo de um emissor para múltiplos receptores possíveis utilizando uma única mensagem de rádio enviada do emissor para os vários receptores. O *timestamp* inserido na camada MAC ajuda a eliminar muitos dos atrasos na transmissão. Além disso, um método baseado em regressão linear para compensação do desvio de tempo dos *nós* é empregado para conseguir uma alta precisão entre os pontos de sincronização mantendo o tráfego gerado pela sincronia baixo.

O FTSP utiliza transmissão de mensagens via rádio para sincronizar os possíveis destinatários com o tempo provido pelo remetente. O *nó*, no momento que recebe a *timestamp* global por meio das mensagens de sincronia, pega também o *timestamp* local para ser usado como ponto de referência no cálculo do desvio. Como os relógios dos *nós* não são exatamente iguais, por mínimas diferenças de frequência nos cristais utilizados, com passar do tempo, ocorre um desvio entre os relógios dos *nós* da rede. Para tratar esses desvios é feita uma estimativa utilizando uma regressão linear, que utiliza como fonte de dados um histórico recente de pontos de referência para determinar o desvio de cada *nó* com relação ao *root*. Essa estimativa é utilizada para obter mais precisão na obtenção do tempo global da rede, bem como reduzir o número de transmissões de mensagens de sincronia entre os *nós*. Ela também identifica a tendência da relação entre o tempo global e o tempo local e os dados recebidos no passado. Somente um número limitado de pontos de referência podem ser armazenados devido a limitações de memória dos *nós*.

No contexto de uma RSSF em que um pacote passa por vários *nós* para atingir um *nó root* (*i.e.* rede em malha ou *multihop*), o protocolo assume que cada *nó* da rede possui um único identificador e procede da seguinte forma para sincronizar a rede: os *nós* na rede utilizam os pontos de referência (o par formado pelo tempo global da rede e local do *nó*) para efetuar a sincronia e disseminar o tempo global da rede. O *nó root*, isto é, aquele que disseminará o tempo global é eleito e reeleito dinamicamente. Os *nós* que estão no raio de alcance da transmissão desse *nó* vão receber os pontos de referência diretamente dele. Os

que estão fora do raio de alcance receberão indiretamente dos *nós* que têm acesso ao *root*. Quando os *nós* coletam pontos de referência, eles estimam o *offset* do desvio do relógio global com relação ao local com base nas mensagens de sincronia recebidas anteriormente. A partir daí, o novo *nó* é considerado sincronizado e pode oferecer aos outros *nós* da rede mensagens de sincronização, e assim acontece sucessivamente.

Neste trabalho fez-se algumas adaptações para adequar o protocolo as necessidades da aplicação, seguem os detalhes das alterações: (I) a frequência de envio de mensagens de sincronia é a cada dez minutos; (II) o período máximo para um *nó* se declarar *root* é a expiração de duas vezes o período de sincronia, isto é, se o *nó* ficar por 20 minutos sem receber a mensagem de sincronia ele se declara *root*, com exceção do *nó* com ID 0, que é o *nó* base, que assim que entra na RSSF já se declara *root*; (III) um *nó* qualquer da rede, após se declarar *root*, ignora por duas vezes o período de sincronia de mensagens de outro *nó root* qualquer, esse mecanismo provê mais estabilidade na eleição do *root* de sincronia; (IV) um *nó* só pode sinalizar como sincronizado caso tenha pelo menos duas entradas na tabela utilizada para regressão, mas pode disseminar mensagens de sincronia imediatamente após o recebimento da primeira; (V) por fim, se o erro entre o relógio local e o global for maior que 100 milissegundos, a tabela de pontos de referência temporais utilizada no cálculo da regressão linear é apagada. Vale observar que o *nó root* nesse contexto é o *nó* responsável pela disseminação do tempo na RSSF, não sendo necessariamente um nó consumidor.

#### 3.1.2.5 Protocolo para Disseminação de Código

Para facilitar os testes e as futuras atualizações no código da aplicação embarcada, utilizou-se o protocolo Deluge [21]. Esse protocolo fornece uma maneira eficiente para disseminação de grandes conjuntos de dados, como programas binários para muitos *nós* pertencentes a uma RSSF, eliminando o trabalho de inserir/atualizar o código em cada sensor individualmente. Combinando este mecanismo com um comando para disseminação e um mecanismo para gerenciamento de *boot*, ambos providos por ele, torna-se viável a programação remota dos *nós* da RSSF. As funcionalidades que tornam isso possível são as seguintes:

- **Suporte a redes *multihop***: Programar os *nós* em uma rede sem fio sem a necessidade de interagir diretamente com os *nós*;
- **Disseminação epidêmica**: Esforço para propagação contínua por todos os *nós*

ajuda a garantir a acessibilidade a *nós* com conectividade intermitente (*e.g.* *nós* utilizando o Box-MAC-1);

- **Verificação da integridade de dados redundantes**: *Cyclic Redundancy Check* (CRC) para garantir a integridade das imagens das aplicações que são disseminadas aos *nós*;
- **Armazenamento de imagens de várias aplicações**: Cada *nó* armazena imagens de várias aplicações distintas, facilitando a mudança da rede entre os diferentes programas sem a necessidade de inserir código para trafegar pela RSSF novamente;
- **Imagem de Ouro**: A imagem do programa com suporte mínimo para programação via rede fica armazenada em um local seguro na memória flash, possibilitando sempre haver um código que permita a recuperação do *nó*, em caso de falhas diversas;
- ***Bootloader* isolado**: Um pedaço de código que tem sua execução garantida após cada reinicialização independentemente da aplicação que o usuário está disseminando/executando. O *bootloader* é responsável pela programação do microcontrolador e também pela recuperação de erros de programação, carregando a Imagem de Ouro;
- **Rollback**: Um *nó* falhou e não conseguiu se recuperar automaticamente. O *nó* irá carregar a Imagem de Ouro quando o botão reiniciar for pressionado algumas vezes consecutivas. Assim o *nó* estará pronto para receber uma nova aplicação;
- **Baixo consumo de memória RAM**: menos de 150 bytes.

Os *nós* utilizados (**Iris**/**Micaz**), suportam o arquivamento de até 4 aplicações distintas na memória flash. No espaço zero, deixa-se a imagem de ouro para recuperação. No espaço um armazena-se a aplicação embarcada. O espaço contém uma aplicação para verificação dos *nós* denominada *Blink*, que acende e apaga periodicamente os Leds do dispositivo para verificar se o *nó* está funcional (*i.e.* com baterias carregadas e sem falhas de hardware). O espaço três não é utilizado.

#### 3.1.2.6 Aplicação Embarcada para Coleta de Dados

A aplicação desenvolvida é composta por cinco componentes principais. São eles os protocolos: *Box-Mac-1* [40], CTP [17], FTSP [32] o *Deluge* [21] e a aplicação embarcada. A Figura 3.7 mostra a hierarquia deles na aplicação. No início da pilha de protocolos utilizados, é executado o *Box-Mac-1* que atua como protocolo da camada MAC. Ele é

o responsável por fazer a comunicação salto-a-salto na RSSF. O *Box-Mac-1* conta com uma abordagem que tem como objetivo principal uma comunicação reativa visando uma maior eficiência energética.

Uma camada acima da MAC são executados os protocolos CTP, *Deluge*, e o FTSP. O CTP é o responsável por fazer o roteamento dos *nós* produtores até o *nó* consumidor/*root* mais próximo. O *Deluge* é um protocolo executado em segundo plano na aplicação, e tem como principal funcionalidade a atualização remota da aplicação embarcada nos *nós* da RSSF. Por fim, o FTSP é o responsável por manter os relógios dos *nós* da RSSF sincronizados.

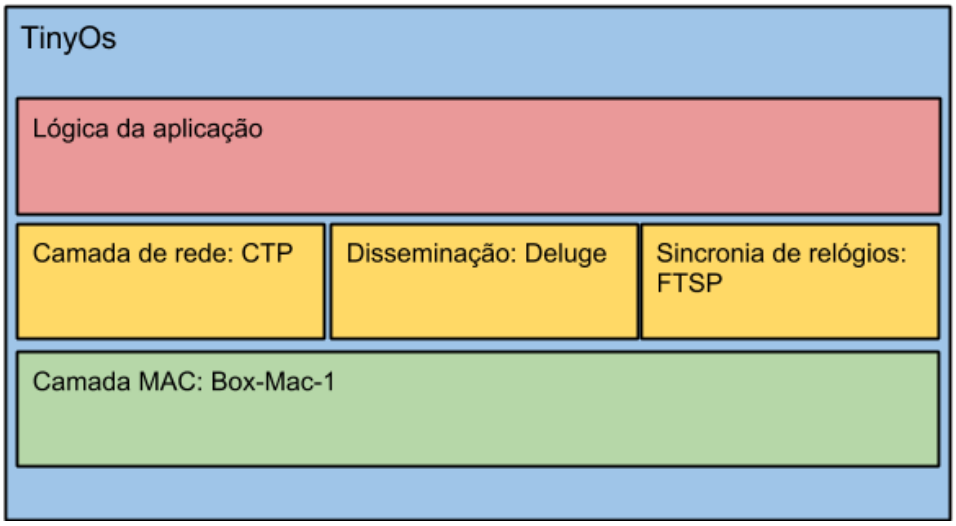



Figura 3.7: Principais componentes da aplicação embarcada.

A aplicação que implementa a lógica para a coleta das informações é executada sobre esses protocolos e utiliza as interfaces fornecidas por eles para usufruir das funcionalidades disponibilizadas e executar suas tarefas. O TinyOs [28], o sistema operacional (SO) executado nos *nós* da RSSF; interliga todos esses protocolos com a aplicação, além de fornecer outras abstrações.

## 3.2 Infraestrutura do Sistema de Disponibilização dos Dados

O sistema para disponibilização das informações é constituído basicamente por um servidor de eventos. Ele é responsável por disponibilizar os dados coletados pela RSSF para qualquer interessado em tempo real, sem a necessidade, por exemplo, de uma consulta a base de dados. Para prover essa funcionalidade é utilizado um mecanismo baseado na

troca de mensagens por meio do paradigma *Publish/Subscribe* [12].

O paradigma *publish/subscribe* pode ser visto como uma solução para o problema de sincronização com relação a notificação de eventos. Neste paradigma, os usuários publicam a ocorrência de eventos e declaram seus interesses para que sejam notificados. Por definição, aqueles que publicam eventos são chamados *Publishers* e são vistos como os produtores de eventos no paradigma. De forma similar, os consumidores de eventos são os *Subscribers*, aqueles que declaram seus interesses (*subscribe*). Conforme visto na Figura 3.8, as publicações e subscrições são enviadas a um servidor (*Event Manager*) que processa as informações que chegam. Quando encontra algum evento compatível com o interesse de algum *subscribe* é dito que ocorreu uma relação e, portanto, o *subscriber* que tem interesse em tal evento deve ser notificado imediatamente. Para implementação desse mecanismo é necessário um gerenciador de eventos centralizado para fazer o roteamento das mensagens.

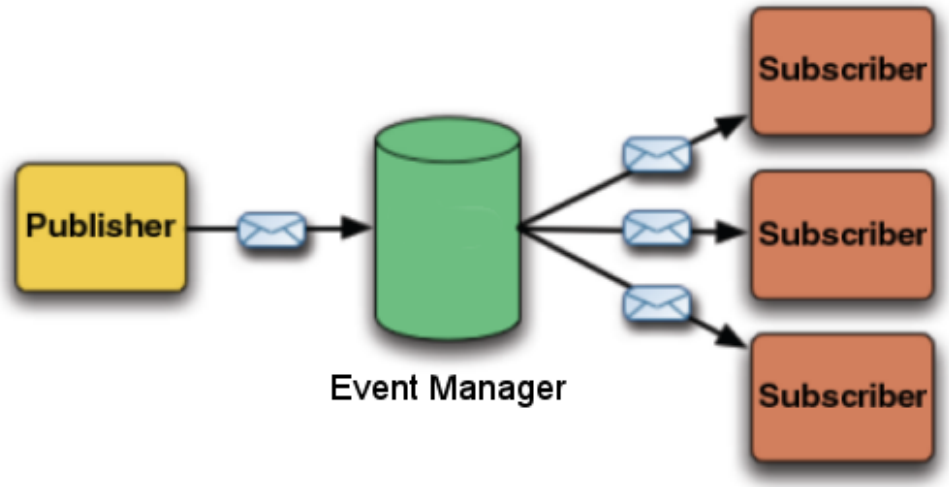



Figura 3.8: Funcionamento paradigma *Publish/Subscribe*.

O servidor de eventos utilizado foi o *RabbitMQ* [44]. Esse gerenciador implementa o *Advanced Message Queuing Protocol* (AMQP) [48] e fornece suporte para aplicações desenvolvidas em várias linguagens, inclusive o *Python*, utilizado neste trabalho.

O AMQP permite uma interoperabilidade funcional entre *middlewares* clientes e servidores de mensagens. O objetivo é que, através do AMQP, os recursos necessários para transmissão de mensagens possam ser abstraídos para a rede por meio de um *middleware* como o *RabbitMQ*.

A semântica para comunicação com o servidor é definida de forma explícita, já que a interoperabilidade exige que esta seja a mesma em qualquer servidor mensagens que use

o AMQP. Assim, a arquitetura AMQP especifica um conjunto modular de componentes e regras para conexão. Existem três tipos principais de componentes, os quais estão ligados às cadeias de processamento no servidor para criar a funcionalidade desejada pela aplicação:

- O **roteador** recebe mensagens de aplicativos geradores e os encaminha para as "filas de mensagens", com base em critérios arbitrários, normalmente propriedades da mensagem ou características do conteúdo;
- A **fila de mensagens** armazena as mensagens até que elas possam ser processados por uma ou várias aplicações cliente;
- O **conector** define a relação entre uma fila de mensagens e uma troca e fornece a mensagem de acordo com os critérios de encaminhamento.

Usando este modelo pode-se emular os conceitos clássicos de mensagens como o *publish/subscribe* (que é o utilizado neste trabalho) de forma simples. Pode-se também expressar conceitos menos triviais, como roteamento baseado em conteúdo, distribuição de carga de trabalho, e filas de mensagens sob demanda.

Em outros termos, um servidor baseado no AMQP (*e.g. RabbitMQ*), pode ser visto como um servidor de e-mail onde cada roteador atua como um agente de transferência de mensagens, e cada fila de mensagens é como uma caixa de correio. As assinaturas nos tópicos de interesse definem as tabelas de roteamento em cada agente roteador. Editores (*publishers*) enviam as mensagens para os agentes de roteamento, que então encaminham as mensagens para caixas de correio. Após isso, os consumidores (*subscribers*) pegam as mensagens das caixas de correio.

Outro ponto importante é a forma de comunicação do servidor de eventos *RabbitMQ*; ele trabalha através rede de computadores, por meio do *Transmission Control Protocol* (TCP/IP) [14]. Isso permite que ele seja divido em várias instâncias, em hardwares separados ou não, possibilitando o suporte a uma RSSF de grande escala, com grande tráfego de dados.

## 3.3 Infraestrutura do Sistema de Visualização dos Dados

Por conta da necessidade do sistema prover acesso remoto, verificou-se um meio para disponibilizar o sistema por meio da rede mundial de computadores, a Internet. Para isso, foi necessário um servidor para uma Aplicação Web. Aplicação *Web* é o termo utilizado para designar, de forma geral, sistemas de informática projetados para utilização através de um navegador, na Internet ou em redes privadas (Intranet).

Para tratar esse problema escolheu-se o Django [19]. O Django é um *framework* para aplicações web escrito em *Python*, gratuito e de código aberto, que segue a arquitetura *Model View Controler* (MVC) [24]. O Django é mantido pela Django Software Foundation (DSF) [10], uma organização sem fins lucrativos.

O núcleo do *framework* MVC consiste em um mapeador objeto-relacional que faz a mediação entre os modelos de dados (definos como classes Python) e o banco de dados relacional (***M****odel*); um sistema para processamento de requisições que conta com um sistema de *templates web* (***V****iew*) e um distribuidor de URLs baseado em expressões regulares (***C****ontroler*). Também são incluídas no núcleo do *framework* as seguintes funcionalidades:

- Servidor web leve e independente para desenvolvimento e testes;
- Sistema de validação e serialização que traduz os dados dos formulários HTML para valores que possam ser tratados e armazenados no banco de dados;
- Suporte para classes *middleware* que podem intervir em vários estágios do processamento dos requisições e realizar funções customizadas;
- Suporte à internacionalização, que incluí traduções dos próprios componentes do Django em diversos de idiomas;
- Sistema de serialização que pode produzir ou ler representações XML/JSON de instancias dos modelos Django;
- Interface para o *framework* de teste unitário que é incluído na linguagem Python;
- Sistema de autenticação extensível;
- Interface administrativa dinâmica;
- Ferramentas para a geração de *feeds* e distribuição de conteúdo;

- Sistema para comentários, entre outros.

O *framework* Django pode ser executado em conjunto com diversos servidores web, como o Apache, Nginx, e Cherokee. Ele também tem a capacidade de executar um servidor FastCGI[3], permitindo o uso em conjunto de qualquer servidor web que suporte FastCGI, tais como Lighttpd ou Hiawatha. Django suporta oficialmente quatro banco de dados: PostgreSQL, MySQL, SQLite e Oracle. O Microsoft SQL Server pode ser utilizado mas apenas em sistemas operacionais da Microsoft. Há também uma versão derivada da principal chamada *django-nonrel* que suporta bancos de dados NoSQL, como MongoDB e armazenamento de dados no Google App Engine.

[3]FastCGI Consiste numa importante tecnologia que permite gerar páginas dinâmicas, permitindo a um navegador passar parâmetros para um programa alocado em um servidor web. FastCGI é uma variação do *Common Gateway Interface* (CGI); o principal objetivo do FastCGI é reduzir a sobrecarga associada à interface entre o servidor web e programas CGI, permitindo que um servidor lide com mais requisições web ao mesmo tempo.

# Capítulo 4

# Arquitetura do Sistema de Monitoramento

O sistema de monitoramento termoenergético proposto neste trabalho consiste na integração de três áreas. A primeira é a de redes, que engloba todos os detalhes referentes à coleta dos dados e comunicação entre os sensores. A segunda área é a de engenharia de software, que é encarregada da disponibilização e organização dos dados. A terceira, a inteligência artificial, que é a responsável pelo monitoramento inteligente dos CPDs e implementação de técnicas de economia de energia com base nos dados coletados pela RSSF. Esta última está em fase de desenvolvimento e será tratada com mais detalhes em outro trabalho [4].

A Figura 4.1 descreve de forma simplificada a arquitetura do sistema. Ela é dividida em três partes: *Application Layer*, *Event Manager* e *Acquisition Layer* que estão descritos com mais detalhes nas Seções 4.1, 4.2 e 4.3, respectivamente. A camada de aplicação (*Application Layer*) contém os serviços que possibilitam monitorar o CPD, ou seja, as aplicações responsáveis pela exibição dos dados e a inteligência do sistema de monitoramento. O *Event Manager* tem a função de prover a conexão entre todos os módulos, de maneira assíncrona e em tempo real. Assim, ele controla a circulação de mensagens no sistema através do mecanismo *publish/subscribe* por meio de um servidor de eventos. A *Acquisition Layer* tem a função de coletar, processar os pacotes da RSSF, publicá-los e armazená-los na base de dados (BD). Estas camadas serão descritas com mais detalhes nas seções a seguir.

O fato da comunicação entre os módulos do sistema ser gerenciada por um servidor de eventos centralizado, permite que as camadas, ilustradas na Figura 4.1, estejam distribuídas em máquinas distintas. Isso possibilita que sejam executadas múltiplas instancias

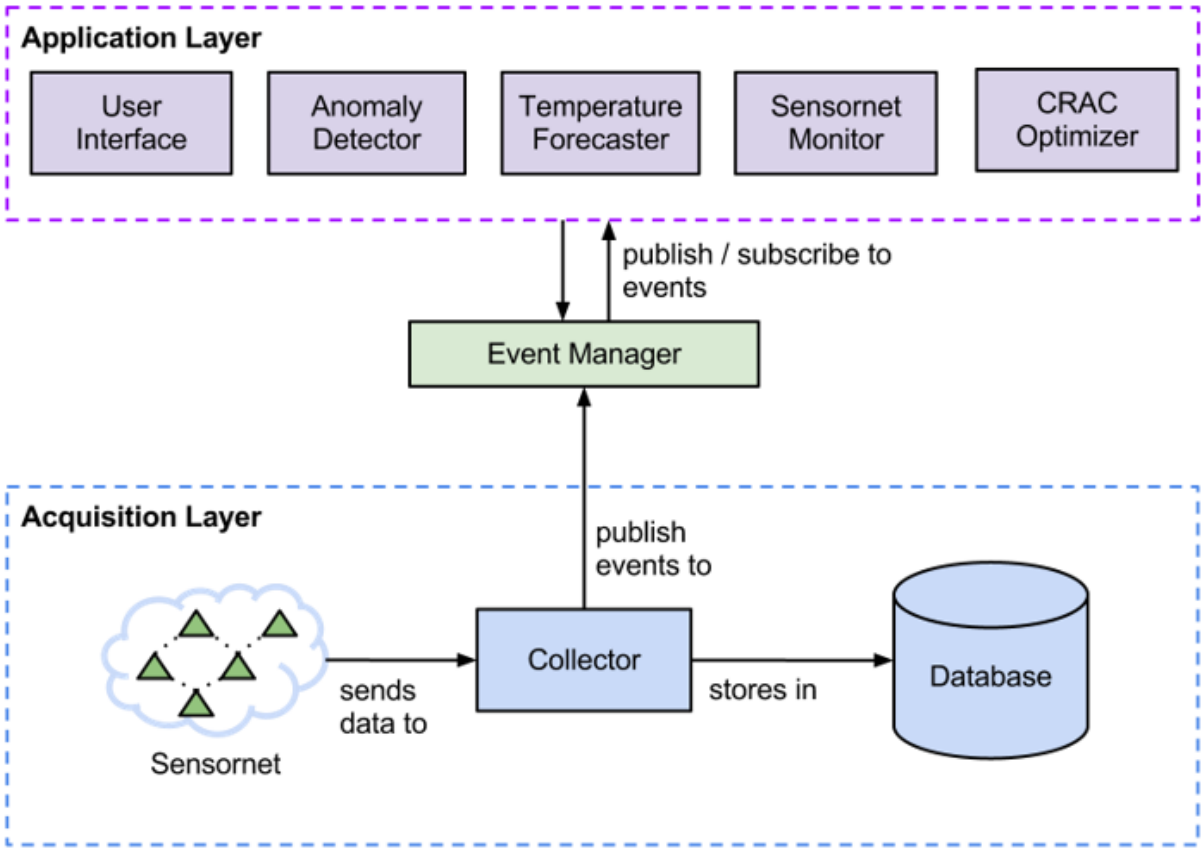



Figura 4.1: Arquitetura do Sistema para Monitoramento Termoenergético de CPDs.

em paralelo da *Acquisition Layer*. Essa característica provê redundância à falhas para as bases de coleta da RSSF pois, se caso uma dessas máquinas que contém a *Acquisition Layer* falhar o CTP nativamente reorganiza a topologia da RSSF para encaminhar os pacotes para uma base de coleta mais próxima que esteja funcional.

## 4.1 *Application Layer*

A *Application Layer* é composta por cinco módulos, são eles, *Sensornet Monitor*, *User Interface*, *Anomaly Detector*, *Temperature Forecaster* e *CRAC Optimizer*. O *Sensornet Monitor* é o módulo responsável por monitorar o status da RSSF (latência, *nós* ligados, estatdo da bateria, etc). Um previsor simples (baseado na média de consumo da bateria em função do tempo) é empregado para estimar a duração da bateria.

O *User Interface* é o modulo que tem como objetivo exibir informações coletadas pela RSSF e as providas pelos outros módulos. Ele também possibilita a visualização do estado da RSSF em tempo real(*e.g.* topologia, qualidade dos links de comunicação).

O *Anomaly Detector* [4] é o módulo responsável por detectar anomalias em tempo

real e também sob demanda (*e.g.* em um trecho do banco de dados). Por anomalias entende-se comportamentos incomuns nas várias séries temporais que compõem o sistema de monitoramento. Esses comportamentos incomuns podem estar relacionados com diversos problemas térmicos, como mal funcionamento de unidades de refrigeração, obstáculos nas saídas de ar, etc. Outras anomalias podem ser geradas pela rede (*e.g.* variação não usual de latência) e podem ser usadas para diagnosticar problemas.

O *Temperature Forecaster* [4] é o módulo responsável pela previsão de temperatura no CPD. As estimativas geradas serão empregadas em conjunto com um limiar de temperatura. Assim, alertas serão gerados no caso da previsão de algum evento que possa prejudicar a operação segura do CPD, para que uma providência seja tomada.

O *CRAC Optimizer* [4] tem como objetivo fazer previsões térmicas de longo prazo em conjunto com o *Temperature Forecaster* para determinar *setpoints* eficientes para operação do (*Computer Room Air Conditioning*) CRAC. Mudanças não podem ser feitas a todo instante, pois a configuração do CRAC é feita manualmente pelo operador. Portanto, é necessário um grau de confiança suficiente para realizar mudanças no sistema. Além disso, é importante ter cuidado ao aumentar a temperatura para não colocar o sistema em risco de ativar emergências térmicas.

## 4.2 *Event Manager*

O *Event Manager* tem o papel de controlar a circulação de mensagens no sistema e introduzir o mecanismo de *publish/subscribe* [12]. Este mecanismo é um padrão empregado em aplicações que trocam mensagens de maneira assíncrona. A ideia é realizar a comunicação entre duas entidades: produtor e consumidor. O produtor cria mensagens e o consumidor deseja acesso a essas mensagens.

Para realizar essa comunicação de forma assíncrona, o produtor publica suas mensagens em uma entidade (*i.e. Event Manager*). Essas mensagens ficam armazenadas de forma ordenada em um *buffer*. Eventualmente, um consumidor aparece no sistema e ele registra (*subscribe*) seu interesse em receber mensagens de um determinado tipo. Assim, o *Event Manager* agora sabe que deve repassar as mensagens do tipo correto que estão *buffer* para aquele consumidor. Essa relação pode ser "muitos para muitos", onde haveriam múltiplos consumidores e produtores. Além disso é possível haver múltiplos tipos de mensagens distintas, com seu conjunto de produtores e consumidores associado.

Eventos podem ser gerados tanto pela camada de coleta (*i.e.* novos dados coletados)

quanto pela camada de aplicação. Os eventos da camada de aplicação, dependem da aplicação em si. Por exemplo pode-se pensar no evento de alerta, caso seja prevista uma temperatura muito elevada no CPD. Graças ao *Event Manager* aplicações podem publicar e se registrar em eventos de outras aplicações. Isso é especialmente útil para o *User Interface*, que precisa exibir diversos tipos de eventos em tempo real. Segue abaixo, a título de exemplo, uma breve descrição dos tipos de eventos que podem ocorrer no sistema:

- *New measurement*: Ocorre quando uma nova medida (luz, temperatura ou bateria) é publicada pelo *Collector*. Note que esse evento ocorre sempre que o *Collector* recebe um novo pacote de dados com número de sequência diferente do anterior. Essa diferenciação é importante para se estimar com que periodicidade os dados estão sendo coletados, por exemplo.
- *Alert*: Mensagem de alerta, geralmente requer intervenção ou supervisão humana. Subtipos podem ser: *Info*, *Warning* e *Critical*, dependendo da gravidade. Qualquer componente pode publicar alertas.
  - *Anomaly Alert (Warning)*: Indica que uma anomalia foi encontrada. A priori não há como saber se a anomalia é séria ou não. Por isso ela deve ser um *Warning*. Se a temperatura estiver alta, o *Forecast Alert* será enviado. Cabe ao operador correlacionar esses eventos para descobrir o que aconteceu.
  - *Forecast Alert (Critical)*: Indica que uma previsão foi feita e a temperatura prevista se encontra acima de um limiar considerado seguro. O evento deve conter os seguintes campos: *nós* afetados pela previsão, valor de temperatura previsto, tempo estimado até que a temperatura cruze o limiar máximo.
  - *Low Battery Alert (Warning)*: Indica que é necessário trocar as baterias de algum *nó*. O alerta é ativado com alguns dias de antecedência.
  - *Mote Down Alert (Warning)*: Indica que um *nó* não está mais se comunicando com a base. Pode ser por falta de bateria ou outros motivos. Esse alerta se baseia na latência (*i.e.* intervalo sem enviar dados) de um *nó*.
  - *WSN Down Alert (Warning)*: Indica que toda a RSSF caiu e não está mais se comunicando com a base. Pode ser por falta de bateria ou outros motivos. Esse alerta também se baseia na latência de entrega de um *nó*.
  - *Battery Replaced Alert (Info)*: Indica que a bateria de um *nó* foi trocada e ele está operante.

  - *Mote Boot Alert (Info)*: Indica que um novo *nó* foi detectado na RSSF. Isso pode ser percebido pelo contador de mensagens ou por mensagens do CTP.

- *Maintenance mode start e Maintenance mode finish*: Mensagem gerada pelo *User Interface* que indica um período de manutenção. Isso pode ser útil para que as outras aplicações não pensem que está ocorrendo alguma emergência (*e.g.* quebra do sistema de arrefecimento).

- *New forecast*: Indica uma nova previsão de temperatura.

## 4.3 *Acquisition Layer*

Essa camada é responsável pela coleta e persistência dos dados. O componente *Sensornet* representa a própria rede de sensores, incluindo a estação base. Os dados são agregados pelo *Collector*, os pacotes são armazenados no BD e publicados pelo servidor de eventos. As aplicações podem acessar a base diretamente ou no caso de necessitarem de dados em tempo real utilizam o *Event Manager* para obtê-los. Não há uma conexão na Figura 4.1 entre o BD e a camada de aplicação por questão de legibilidade. O BD, além dos dados coletados pela rede, também armazena informações sobre os sensores, como posicionamento, id (pelo id pode-se derivar o seu papel: *root*, roteador ou sensor) e modelo (Iris/Micaz). O BD é caracterizado por poucas escritas (dados são escritos uma vez e depois não são mais modificados) e muitas leituras. Isso ocorre pois o objetivo do banco é disponibilizar os dados para execução de análises em séries temporais (em busca de padrões, anomalias, etc.) e portanto requer-se leituras rápidas. Essas características o colocam como um BD do tipo OLAP [7].

# Capítulo 5

# Detalhes de Implementação

Neste capítulo é detalhada a implementação do Sistema de Monitoramento Termoenergético para CPDs. Conforme discutido anteriormente, o trabalho desenvolvido é uma abordagem para monitoramento de CPDs que realiza a coleta de dados térmicos de uma RSSF e as exibe de diversas maneiras aos interessados, possibilitando o acompanhamento em tempo real do ambiente do CPD e o estado da rede.

O capítulo está organizado da seguinte maneira, na Seção 5.1 tem-se uma descrição da implementação dos componentes da RSSF e na Seção 5.2 encontram-se os detalhes referentes a implementação dos módulos do sistema para visualização e monitoramento.

## 5.1 Rede de Sensores

Nesta seção são descritos alguns detalhes sobre os componentes que compõem a Aplicação desenvolvida. As interligações entre esses componentes podem ser descritas em um grafo, conforme ilustrado na Figura 5.1.

A seguir encontra-se uma descrição sucinta das funcionalidades providas por cada componente utilizado na aplicação embarcada.

- **MainC**: interface para o sistema de inicialização *TinyOs*. Ela conecta a sequência de inicialização com o escalonador e provê as abstrações para os recursos de hardware básicos;
- **LedsC**: provê abstração para interação com os leds dos *nós*;
- **ActiveMessageC**: provê abstração para envio de pacotes salto-a-salto pela RSSF;

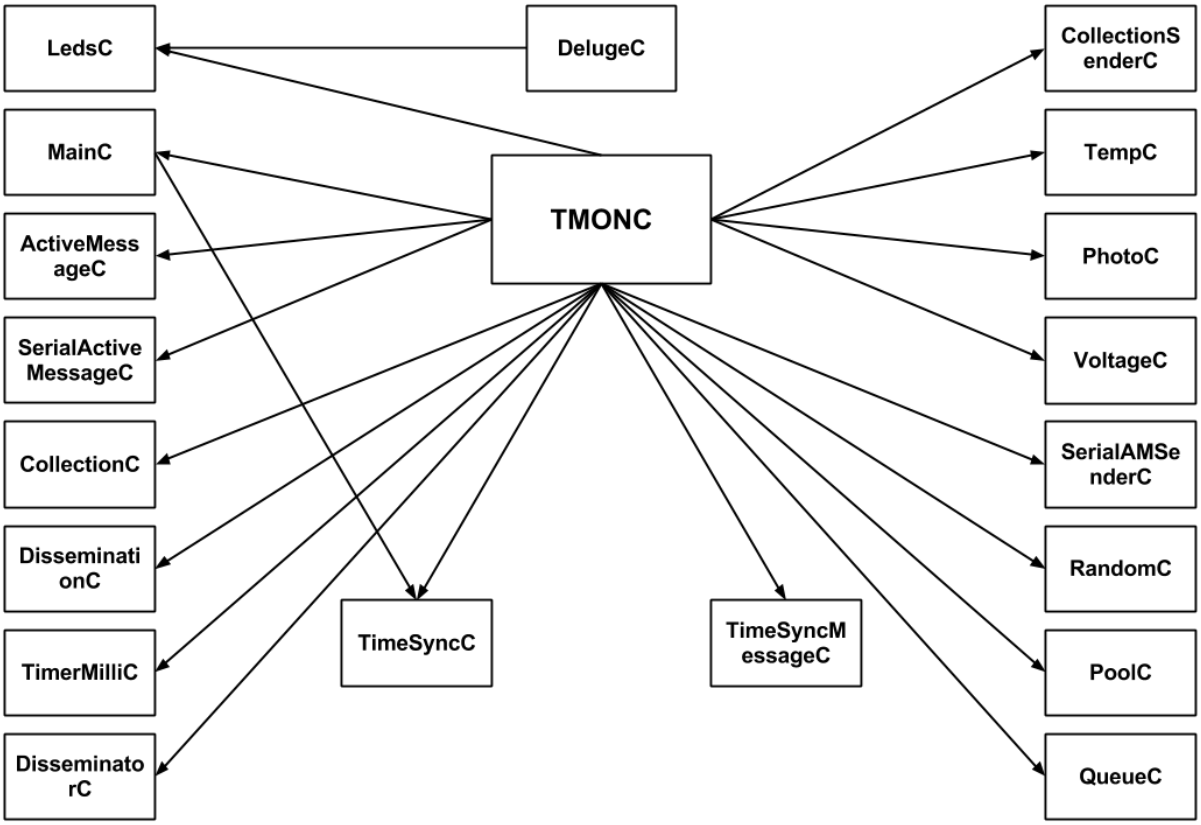



Figura 5.1: Grafo dos componentes da aplicação embarcada.

- **SerialActiveMessageC**: abstração para criação de pacotes para envio por meio de comunicação serial;
- **CollectionC**: provê abstração para coleta e envio dos pacotes para a base, faz parte do CTP;
- **DisseminationC**: interface para abstração de alto nível utilizada na comunicação salto-a-salto;
- **TimerMilliC**: interface que provê abstração para um contador virtualizado com precisão de milissegundos;
- **DisseminatorC**: serviço para estabelecer e disseminar de forma robusta informações compartilhadas pelos *nós* da rede;
- **CollecionSenderC**: abstração para envio de pacotes pelo CTP;
- **TempC**: abstração para leitura de valores do sensor de temperatura;
- **PhotoC**: abstração para leitura de valores do sensor de luminosidade;
- **VoltageC**: abstração para leitura de valores referentes ao status da bateria;

- **SerialAMSenderC**: abstração para envio de pacotes por meio de comunicação serial. Comumente utilizado para encaminhamento dos pacotes da base para o servidor de aplicação;
- **RandomC**: abstração para geração de números aleatórios;
- **PoolC**: abstração para gerenciamento dinâmico de memória;
- **QueueC**: abstração para gerenciamento de fila com tamanho limitado;
- **DelugeC**: componente responsável pelo protocolo Deluge. Utilizado para *upload* de código remotamente para os *nós* da RSSF;
- **TimeSyncC**: provê as interfaces necessárias para o gerenciamento do mecanismo de sincronia dos relógios FTSP;
- **TimeSyncMessageC**: abstração para envio de mensagens de sincronia para os relógios dos *nós* da RSSF pelo FTSP;
- **TMONC**: componente principal que contém lógica da aplicação embarcada dos dispositivos da RSSF. Tem como função principal a aquisição dos dados térmicos de forma confiável. Composta basicamente por uma *thread* que faz verificações periódicas na temperatura, na luminosidade, no estado da bateria e na topologia da rede e transmite esses dados em função de dois fatores: caso alguma dessas variáveis se alterem mais de que um determinado limiar (*e.g.* a temperatura varie 0,5°C, a luminosidade do ambiente ou topologia se altere), ou um *nó* fique sem enviar por um determinado limiar de tempo (*e.g.* 5 minutos). Esses critérios visam reduzir a quantidade de informações que são geradas pela rede e filtrar variações mínimas de temperatura do ambiente. A lógica é descrita com mais detalhes Código A.2 presente no Apêndice A.

### 5.1.1 Pacote de Dados

O pacote de dados da aplicação embarcada transporta as informações coletadas pela RSSF. É composto por dois grupos de informações, os dados coletadas pelos sensores e os dados coletados sobre a topologia da RSSF, como pode ser observado Código A.3 presente no Apêndice A. Nos dados coletados pelos sensores estão: o valor da temperatura, o valor da luminosidade e o valor da voltagem da bateria do *nó*. Nos dados sobre a topologia da RSSF estão: o *nó* de origem do pacote; o número de sequência do pacote (*i.e.* quantos

pacotes já foram enviados pelo *nó* originário do pacote somado com o atual); quem é o pai do pacote (*i.e.* para quem os dados são encaminhados no primeiro salto), o custo de transmissão (calculado através da métrica *expected transmission count* (ETX) [9]) para o *nó root* mais próximo; e o tempo que o pacote ficou trafegando pela RSSF até ser encaminhado pela porta serial pelo *nó root*.

## 5.2 Sistema para Monitoramento

Nesta seção serão fornecidos detalhes de implementação dos componentes utilizados para exibir as informações coletadas pela RSSF.

### 5.2.1 Interface Web

A lógica da aplicação web do sistema foi desenvolvida utilizando o Django [19]. A organização do *layout* e da aparência dos componentes da interface do sistema foi desenvolvida com auxilio da biblioteca *Twitter Bootstrap* [25].

No canto superior em todas as páginas que o usuário tem acesso ao sistema foi colocado o menu principal que provê acesso a todas as funcionalidades. No primeiro item do menu está um *link* para a página inicial do sistema; após esse *link* existem dois botões que exibem menus do tipo *drop down*. O primeiro, *WSN Monitor* provê acesso para as funcionalidades de monitoramento da RSSF e das condições ambientais do CPD, e o segundo, *Events System* dá acesso às funcionalidades responsáveis pelo gerenciamento dos eventos do sistema. No canto esquerdo da barra de ferramentas existem mais dois botões: o primeiro dá acesso à interface administrativa do sistema e o segundo disponibiliza as informações de contato.

Na página inicial do sistema, ilustrada pela Figura 5.2, encontra-se um gráfico com o histórico recente da média da temperatura de todos o sensores do CPD. Abaixo desse gráfico são exibidas três tabelas com a descrição dos cinco últimos eventos de cada tipo. A primeira exibe os do tipo *info*, a segunda do tipo *warning* e a terceira com os eventos do tipo *critical*, conforme descrição feita na Seção 4.2. No final da página estão os dados referentes a quantidade de nós que estão ativos e inativos no instante da visualização.

No botão *List of motes* do sistema, presente no menu *WSN Monitor*, conforme pode ser visto na Figura 5.3, é apresentado uma listagem contendo todos os *nós* da RSSF e detalhes sobre eles. Os *nós* são representados por meio de botões em uma tabela,

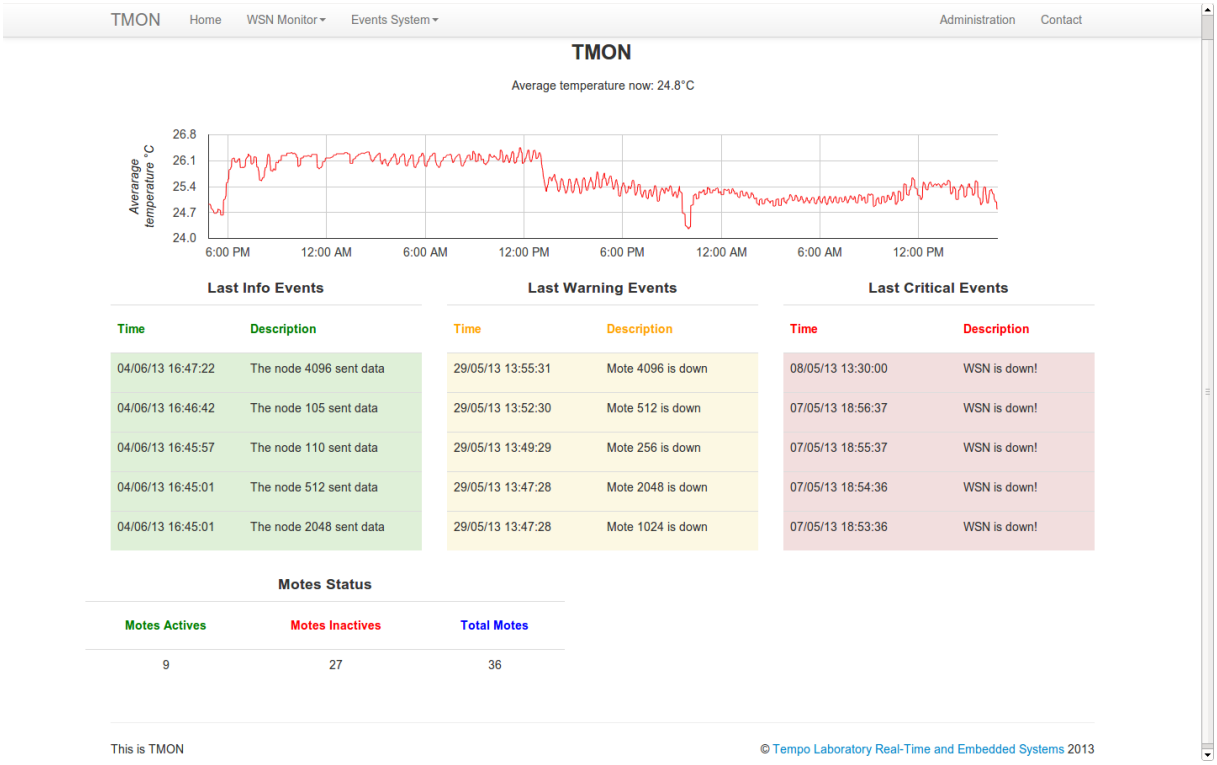



Figura 5.2: Página inicial do sistema.

que também contém os detalhes sobre a localização dos dispositivos. Os *nós* que são identificados por botões da cor verde estão ativos, os da cor vermelha estão com alguma falha e estão inativos. Também nessa página é possível consultar os detalhes dos sensores diretamente pelo identificador do *nó*. Essa funcionalidade foi inserida para facilitar o acesso a detalhes de um *nó* no caso de uma RSSF com uma escala de milhares de *nós*.

Por meio do botão *Network Graph* é exibido o grafo da árvore representando o estado da topologia da RSSF em tempo real, conforme ilustrado na Figura 5.4. Os *nós* exibidos na cor verde são os que estão ativos, os em vermelho os inativos, e os da cor azul representam os *nós root* (*i.e.* *nós* associados a bases, que fazem a coleta das informações da rede). Nas arestas desse grafo está descrito o custo do *link* por meio da métrica ETX [9], do *nó* considerado até o *nó root*. Clicando em algum *nó* é possível ocultar ou expandir todos os seus "descendentes"e clicando no texto abre-se uma página com detalhes sobre o *nó*. Para geração grafo foi utilizada a biblioteca *NetworkX* [18]. Para renderização do grafo na página do sistema foi utilizada uma biblioteca gráfica para *JavaScript* denominada *Data-Driven Documents* [3].

O botão *Thermal Map* dá acesso a um mapa térmico do CPD em tempo real, que usa como base para geração das imagens os dados coletados pela RSSF, como pode ser observado na Figura 5.5. No lado esquerdo da página há um formulário para configuração

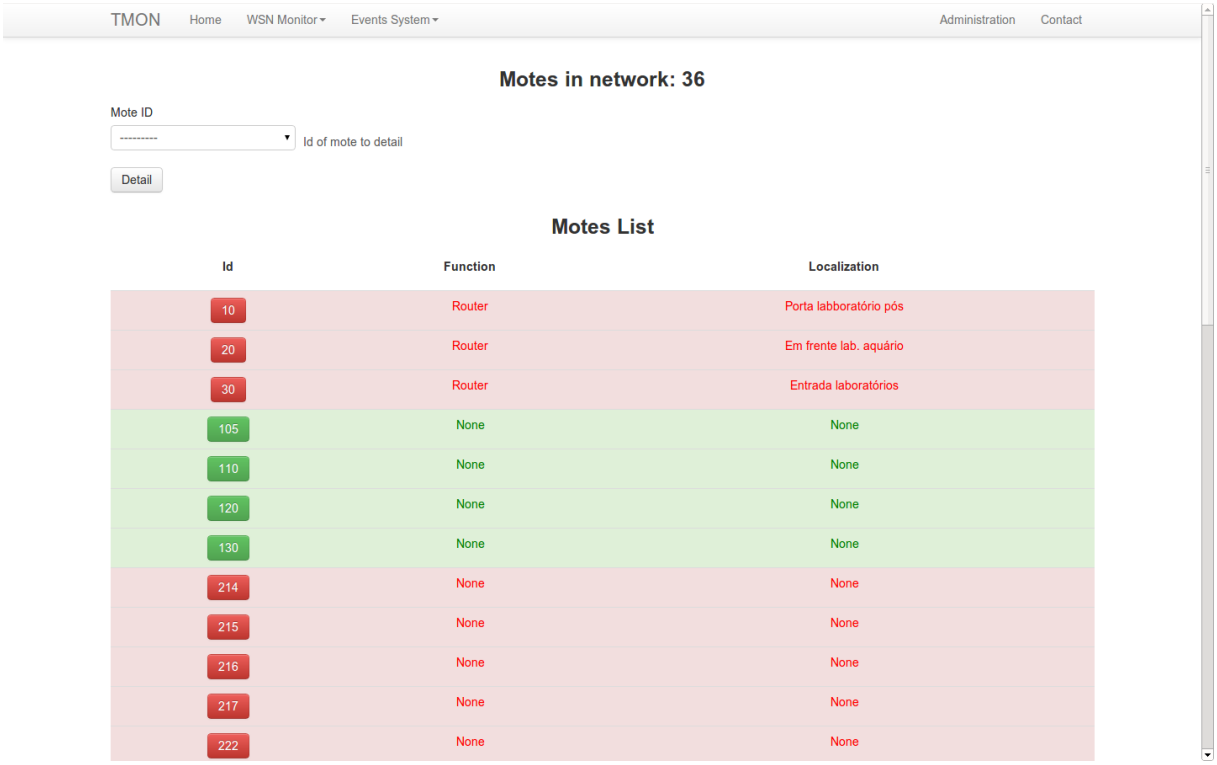



Figura 5.3: Tela listagem dos *nós* na RSSF.

do mapa que se deseja gerar; pode-se definir qual o ângulo de visão (*e.g.* superior, frontal e atrás) e o nível de visualização (esse nível varia em função do tamanho do CPD) da imagem.

O botão *Dynamic View* possibilita acesso do usuário a um formulário que permite configurar os parâmetros para visualização de um gráfico gerado dinamicamente. São necessários os seguintes parâmetros para geração do gráfico: id(s) do(s) sensor(es) (podem ser inseridos um ou mais sensores), especificação do tipo de dados que deseja-se visualizar (*e.g.* temperatura, luminosidade ou bateria), e por último define-se qual intervalo de dados que se deseja exibir. Com base nessas informações é gerado e exibido o gráfico, como se pode observar na Figura 5.6.

No botão *Events Query* presente no menu *Events System*, é apresentado um formulário para consulta de eventos no sistema. O usuário deve especificar o período que deseja-se exibir e o tipo de eventos que deseja visualizar (*e.g. info, warning e critical*). Também é disponibilizado um botão para consulta de todos os eventos que ocorreram no sistema até o momento da consulta (vale lembra que uma consulta desse tipo pode ser lenta em função da grande quantidade de dados que pode estar armazenada). Como resultado dessa consulta é exibida uma lista de eventos, como pode ser visto na Figura 5.7.

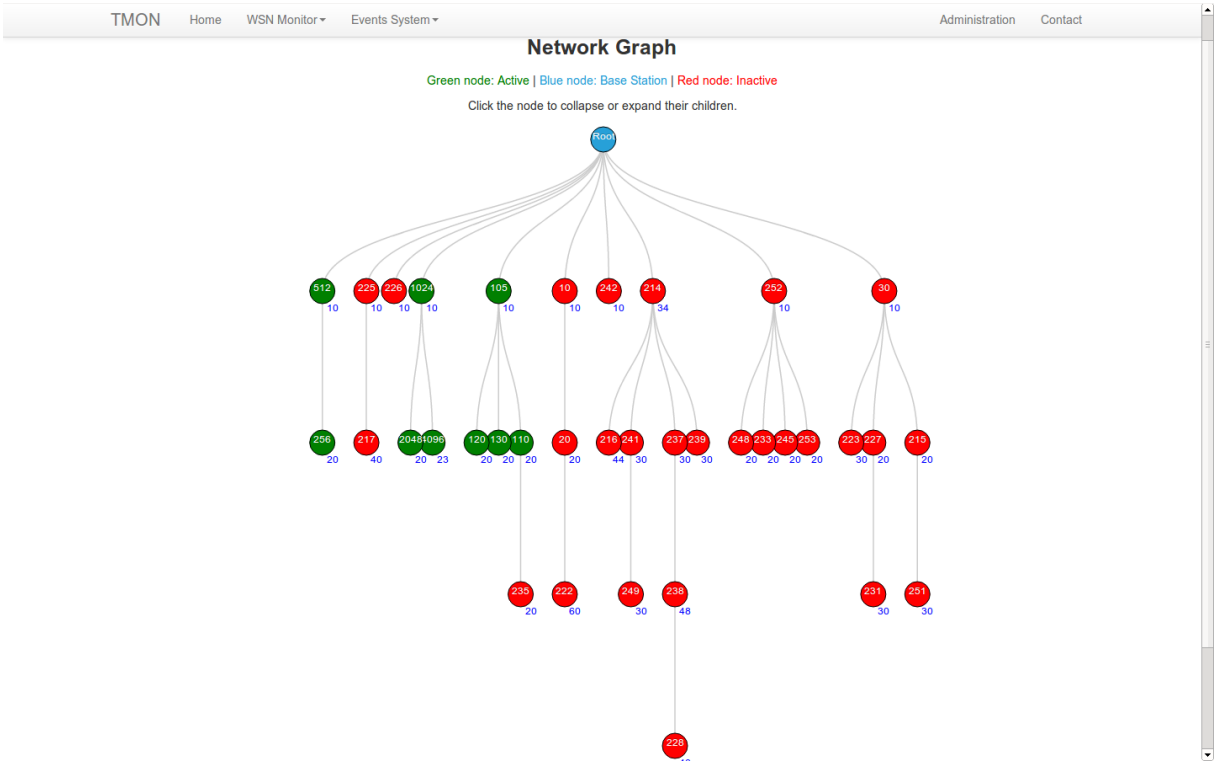



Figura 5.4: Tela do grafo em tempo real da RSSF.

Ainda no menu *Events System*, no botão *System Status* é possível habilitar ou desabilitar o sistema de notificação de eventos. Essa funcionalidade existe com objetivo de desativar as notificações em momentos excepcionais, como manutenção do sistema de monitoramento, ou do próprio CPD. Pelo fato do risco que essa funcionalidade pode causar ao CPD, ela somente pode ser acessada por usuários autorizados.

Ainda no menu superior, no botão *Administration* é provido acesso a interface de administração de sistema. Por meio desta é possível gerenciar usuários, sensores e objetos do CPD (*e.g.* sala, *rack*, servidor). Na administração do usuário é possível efetuar gestão dos usuários (*i.e.* remover, alterar, visualizar e inserir), bem como, gerenciar os privilégios de acesso a páginas restritas do sistema (*e.g.* página de administração e a página do estado do sistema de eventos). O gerenciamento dos sensores, permite gerir as informações desses dispositivos (*i.e.* localização, modelo, descrição). Vale lembrar que quando um novo sensor entra na rede ele é automaticamente inserido no sistema (é lançada uma notificação do tipo *warning ao usuário*), faltando apenas a inserção das informações adicionais (*i.e.* modelo e localização). Existe também um gerenciamento dos objetos do CPD, onde são mantidas as informações necessárias para geração do mapa térmico da instalação (*i.e.* localização e tamanho dos objetos).

Por fim, no botão *Contact* é possível ver as informações de contato do Laboratório

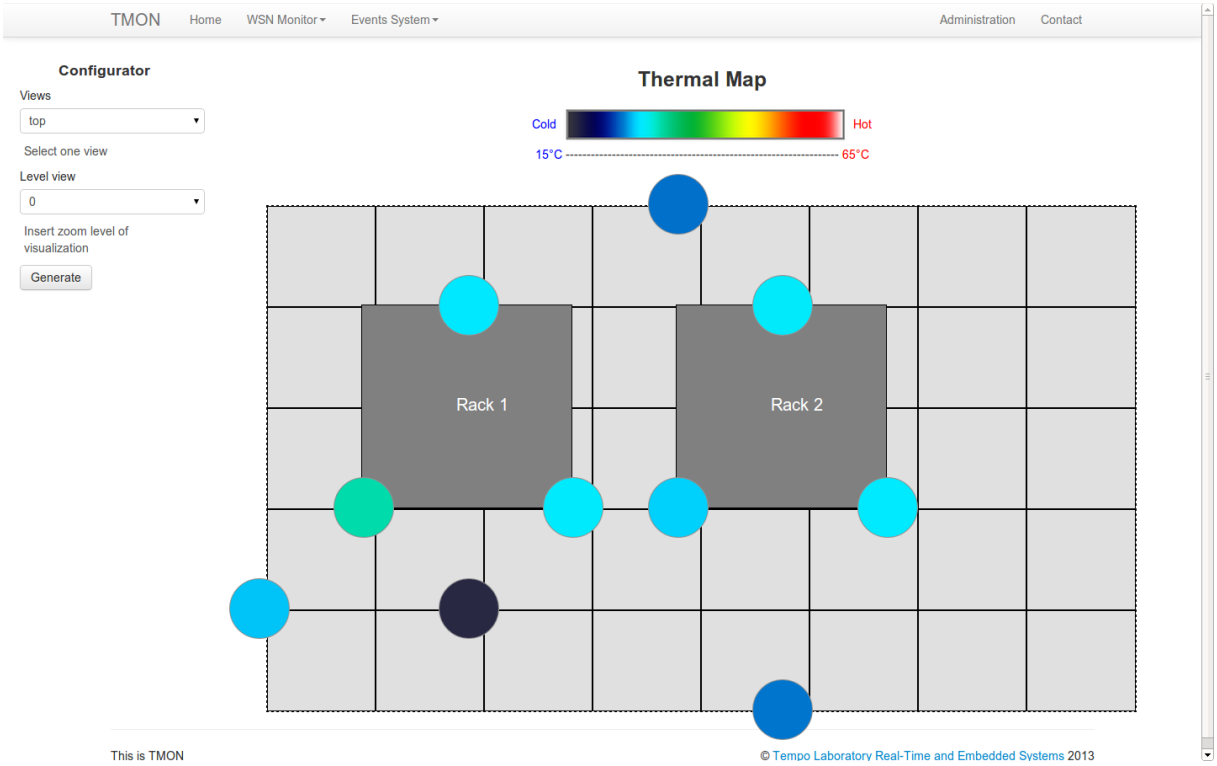



Figura 5.5: Tela mapa térmico do CPD.

responsável pelo desenvolvimento do projeto, bem como um mapa interativo para visualização do endereço.

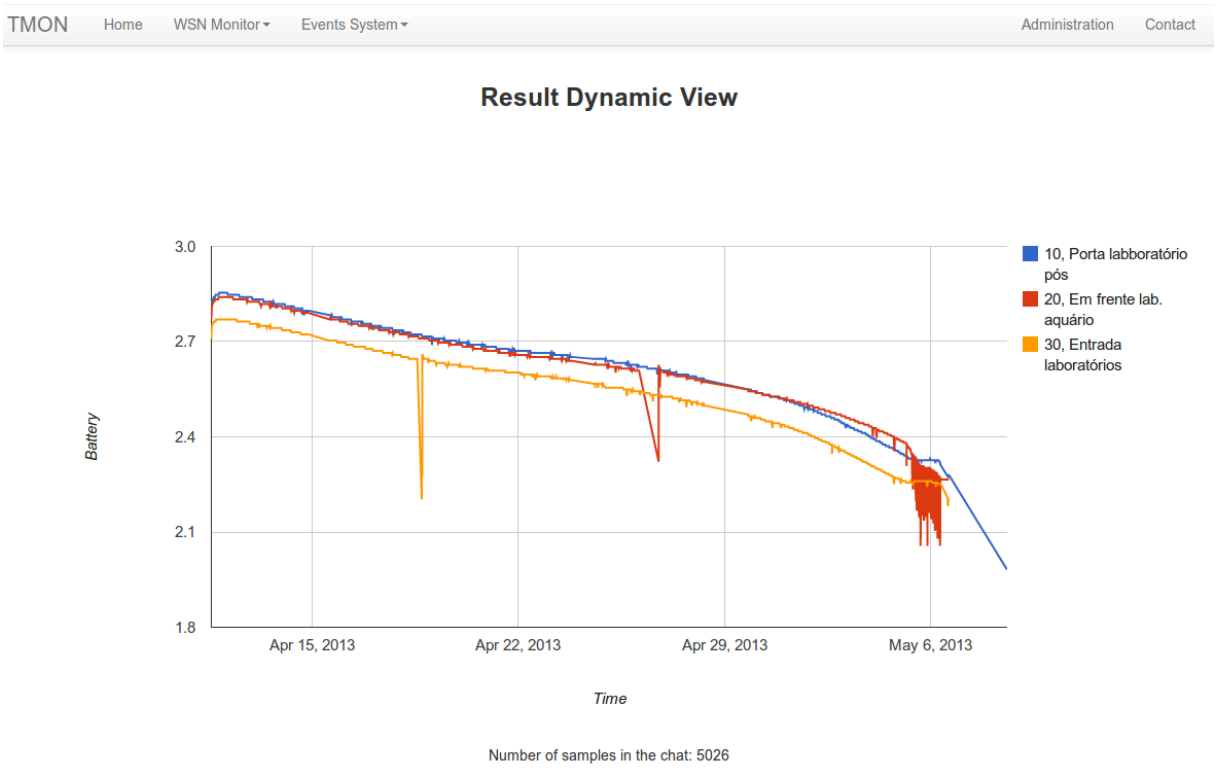



Figura 5.6: Gráfico resultante da consulta dinâmica.

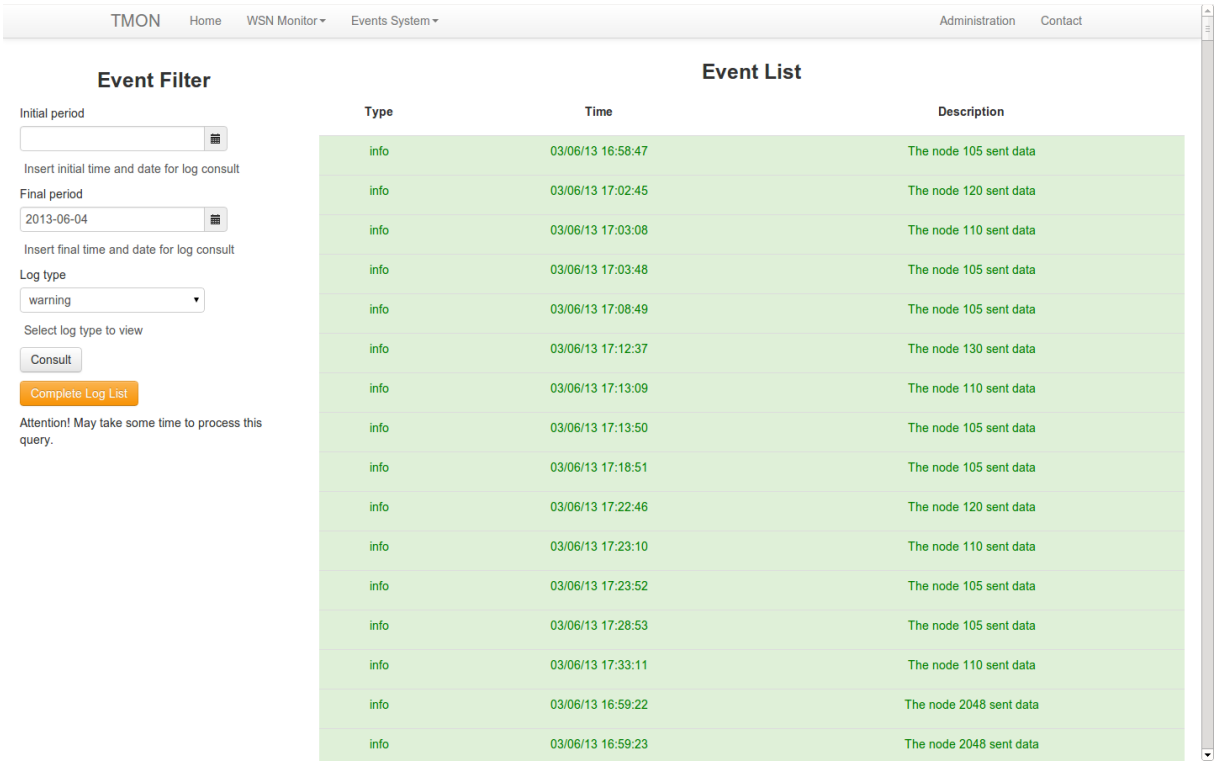



Figura 5.7: Tela lista de eventos.

# Capítulo 6

# Experimentos

Como visto anteriormente, este trabalho tem o objetivo coletar e disponibilizar dados térmicos obtidos por meio de uma RSSF do ambiente de um CPD, para os administradores dessas instalações e para outras aplicações. Portanto, neste capítulo são descritos experimentos realizados com o intuito de avaliar a eficácia do trabalho na coleta e disponibilização dos dados e a influência das tecnologias adotadas para coleta na qualidade das informações coletadas. O Capítulo está organizado da seguinte maneira: na Seção 6.1 encontram-se os testes e simulações executados durante o desenvolvimento do trabalho para validação de alguns parâmetros adotados e, na Seção 6.2 são validados alguns parâmetros, por meio de modelagens matemáticas, para provar a precisão do dados coletados.

## 6.1 Experimentos

### 6.1.1 Impacto do Tempo de Verificação do Meio do Box-MAC-1 no Consumo Energético

Esse experimento tem o objetivo de analisar o impacto do intervalo de verificação do meio que o *Box-MAC-1* executa no consumo energético dos *nós* da RSSF. Ele foi realizado para possibilitar a identificação de qual é o tempo ideal desse parâmetro, de modo a minimizar o consumo energético dos *nós* da RSSF.

Durante o início do desenvolvimento da aplicação embarcada, verificou-se que o tempo de vida dos nós da RSSF era muito baixo, e a redução do tráfego da rede não impactava o consumo energético dos *nós* da RSSF. Nesse contexto, verificou-se a necessidade da medição do consumo energético do rádio com o protocolo padrão, o B-MAC, e um protocolo para gerenciamento do rádio com foco na eficiência energética, o *Box-MAC-1*.

De acordo com as informações obtidas, como pode ser observado na Tabela 6.1, o rádio é de fato o principal responsável pelo consumo energético dos *nós*.

Tabela 6.1: Consumo energético *Iris*.

| **Modo de operação do rádio com *nó* ativo** | **Consumo (mA)** |
|---|---|
| Rádio ligado gerenciado pelo B-MAC sem transmissão | 18 |
| Rádio ligado gerenciado pelo B-MAC com transmissão | 21 |
| Rádio gerenciado pelo *Box-MAC-1* com transmissão | 3 com picos frequentes de 21 |
| Rádio gerenciado pelo *Box-MAC-1* sem transmissão | 3 com picos de 21 |
| Rádio desligado (microprocessador ocioso) | 0,04 |

Como pode ser visto no resultado do consumo energético do *Box-MAC-1*, na Tabela 6.1, como era esperado, o desempenho do *Box-MAC-1* é superior ao do B-MAC. Também durante a execução do experimento foi observado que ele possui picos de consumo que estão associados ao intervalo de verificação do meio e à transmissão de pacotes. Nesse contexto, foi identificada a necessidade de um experimento com o objetivo de verificar o tempo ideal do intervalo de verificação do meio a ser configurado no protocolo MAC *Box-Mac-1*, de modo a minimizar o consumo energético. Para isso, foram utilizados 5 *nós* do modelo *Iris*, cada um com o tempo de verificação do meio diferente. Foram utilizados os seguintes intervalos: 250, 500, 1000, 2000 e 4000 milissegundos. É importante observar que quanto maior esse intervalo, maior poderá ser o tempo para entrega do pacote para o *nó root*, conforme demonstrado na Seção 6.2.1.

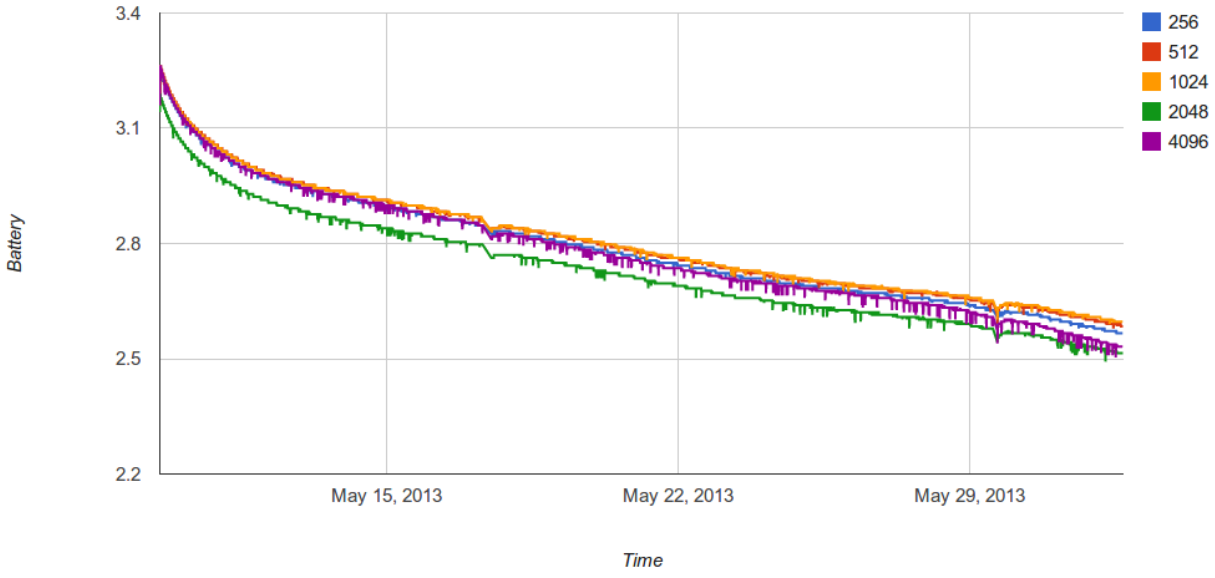



Figura 6.1: Resultado do experimento variando o intervalo de verificação do meio no *Box-MAC-1*.

Tabela 6.2: Detalhamento dos resultados do experimento para cálculo de consumo energético com o Box-MAC-1, variando o intervalo de verificação do meio (Obs.: Os valores referentes ao consumo energético estão em volts).

| **Intervalo de Verificação (ms)** | **Bat. inicial** | **Bat. final** | **Consumo médio/dia** | **Tempo de vida estimado (dias)** |
|---|---|---|---|---|
| 250 | 3,24 | 2,57 | 0,029 | 41,15 |
| 500 | 3,26 | 2,58 | 0,029 | 41,09 |
| 1000 | 3,26 | 2,60 | 0,028 | 41,83 |
| 2000 | 3,19 | 2,51 | 0,029 | 41,36 |
| 4000 | 3,24 | 2,53 | 0,030 | 39,13 |

Diante dos resultados obtidos, que estão ilustrados na Figura 6.1, não houve nenhuma variação significativa no consumo energético dos *nós* variando o intervalo de verificação do meio no *Box-MAC-1*. Na Tabela 6.2, onde os dados são exibidos com mais detalhes, pode ser observado que apenas o intervalo de quatro segundos de verificação apresenta um valor pior com relação aos outros intervalos. Portanto, em virtude dos outros valores estarem muito semelhantes[1], optou-se por definir o intervalo de verificação do meio de acordo com o valor genérico eficiente para um tráfego baixo[2] é de 300 milissegundos [40].

### 6.1.2 Avaliação de Estratégias de Agregação de Dados

Para execução dessa avaliação foram implementadas algumas estratégias de envio para simular uma RSSF. A leitura de dados do ambiente pelos *nós* da RSSF foi emulado utilizando um *trace* contendo dados de temperatura reais que foram coletados anteriormente em um CPD. O objetivo dessa simulação é verificar qual estratégia de envio gera menos tráfego na RSSF, no contexto do monitoramento térmico de CPDs, e consequentemente menos consumo energético. Para isso é feita uma análise do comportamento do número de transmissões em função da estratégia de envio utilizada.

O arquivo utilizado para simular os dados coletados pela RSSF continha três valores: uma identificação do *nó* que fez a leitura do dado, uma *timestamp* contendo a hora que foi coletado e a temperatura coletada. Foi simulado que cada *nó* tem visibilidade para todos os outros *nós*.

Foram abordadas as seguintes estratégias no experimento: *Ingênua*, *Aditiva* e *Faixa*.

[1] A variação apresentada pode ser em função do estados iniciais das cargas das pilhas, que por conta de suas propriedades químicas não são iguais.

[2] Um tráfego menor que um pacote por segundo por *nó*; o tráfego gerado pela RSSF monitorando um CPD é estimado na média de um pacote enviado a cada 8,7 minutos.

Tabela 6.3: Tráfego gerado pelas estratégias de envio.

| Estratégia | Parâmetros (segundos) | Transmissões |
|---|---|---|
| *Ingênua* | Coleta/envio a cada 1 | 6065538 (100%) |
| *Aditiva* | min: 1, max: 30, incr: 1 | 252093 (4,2%) |
| *Aditiva* | min: 1, max: 10, incr: 1 | 562455 (9,3%) |
| *Aditiva* | min: 1, max: 8, incr: 1 | 688039 (11,3%) |
| *Aditiva* | min: 1, max: 2, incr: 1 | 2934492 (48,4%) |
| *Aditiva* | min: 2, max: 2, incr: 2 | 3032773 (50,0%) |
| *Aditiva* | min: 3, max: 3, incr: 3 | 2021850 (33,3%) |
| *Faixa* | delta = 0,001°C; max = 10 | 684811 (11,3%) |
| *Faixa* | delta = 0,1°C; max = 10 | 571899 (9,4%) |
| *Faixa* | delta = 0,2°C; max = 10 | 554298 (9,1%) |
| *Faixa* | delta = 0,2°C; max = 30 | 202227 (3,3%) |
| *Faixa* | delta = 0,5°C; max = 60 | 100544 (16,6%) |
| *Faixa* | delta = 0,5°C; max = 600 | 13420 (0,02%) |
| *Faixa* | delta = 1,0°C; max = 30 | 195835 (3,2%) |

A estratégia *Ingênua* é a que coleta e envia as amostras térmicas a cada segundo. A estratégia *Aditiva* consiste em um valor mínimo e máximo para o intervalo de coleta/envio das amostras e um valor para incremento do intervalo do ciclo. Conforme a temperatura permanece estável, o valor do intervalo é incrementado com o valor de incremento, se a temperatura se alterar o valor do incremento volta para o mínimo e o envio volta a ser frequente. A estratégia *Faixa* consiste em coletar amostras a cada segundo e somente enviá-las quando houver uma variação mínima entre as amostras ou quando atingir um intervalo máximo para envio.

Como se pode observar na Tabela 6.3, a melhor abordagem para o nosso cenário é a *Faixa*. Pode-se observar nos resultados das simulações com essa estratégia que quanto maior o valor do *Delta* e do tempo máximo para envio menos transmissões ocorrem. Entretanto, quanto maior for o valor do *Delta*, maior é a granularidade dos dados coletados e mais informações são perdidas. A precisão dos dados coletados pelos *nós* da RSSF é de 0,2°C, portando, valores para um *Delta* menor que esse foram desconsiderados. Para obter um equilíbrio entre a necessidade de informações precisas e a necessidade de reduzir o tráfego da RSSF foi escolhido o *Delta* de 0,5°C. Por fim, o tempo limite máximo para envio é o mínimo possível que atenda as restrições de consumo energético da RSSF sem comprometer a segurança do CPD. Esse parâmetro é estudado com mais detalhes na Seção 6.1.3.

### 6.1.3 Limite de Tempo para Verificação de Atividade dos Nós da RSSF

O objetivo desse teste é identificar qual é o tempo limite que um ou mais *nós* podem ficar sem enviar nenhuma informação sem correr o risco de comprometer a supervisão do CPD, pela falta de alerta ao administrador do CPD por falha na coleta dos dados pela RSSF. Para a análise foi levada em conta como o pior caso uma situação em que o sistema de arrefecimento falhe completamente. Esse intervalo merece uma atenção especial pois, enquanto estiver dentro desse período, não é possível a identificação de falha na RSSF nem da instalação caso os *nós* estejam com problemas, por conta disso, quanto maior ele for, pior.

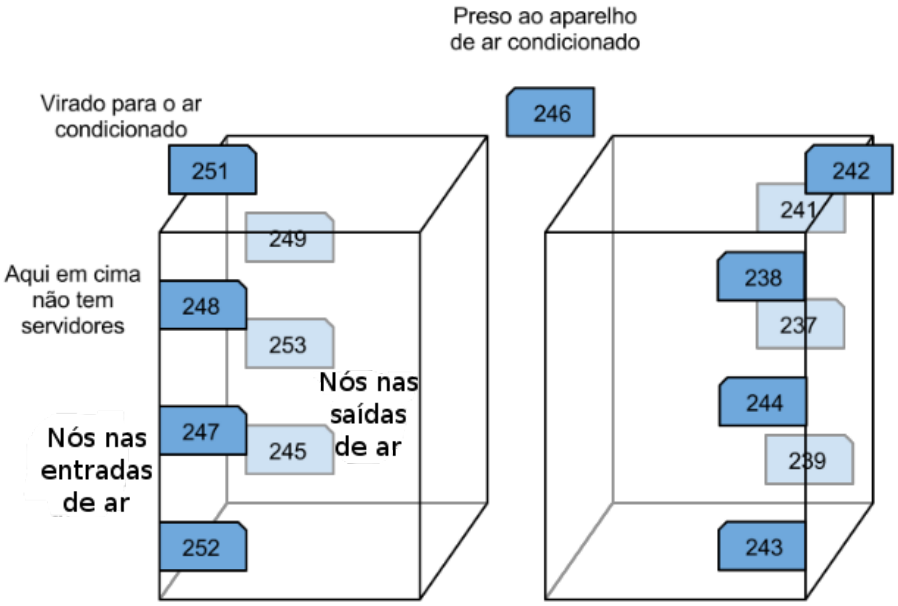



Figura 6.2: Layout da RSSF no momento da falha do sistema de arrefecimento.

Os dados críticos para execução desse experimento correspondem a uma falha elétrica que desligou o sistema de arrefecimento do CPD monitorado. Os servidores ficaram ativos até que um mecanismo de proteção de hardware os desligassem. A RSSF colocada em torno dos servidores está descrita no layout ilustrado na Figura 6.2. A taxa de coleta/envio no contexto era de uma amostra de temperatura a cada minuto.

Como se pode observar na Figura 6.3, os servidores levaram aproximadamente 40 minutos para desligar, isto é, paralisar todos os serviços que estavam sendo providos pelo CPD naquele instante. O *nó* que registrou a maior variação foi o 251. Por estar localizado em cima do *rack* da esquerda e receber o fluxo de ar direto do sistema de arrefecimento, ele registra um aumento de aproximadamente 10°C em dez minutos. Esse *nó*, durante todo

o período de operação sem refrigeração, detecta um aumento de 16°C, pois inicialmente estava registrando temperaturas em torno de 17 °C e, quando os servidores falharam chegou a registrar 33°C. O *nó* que registrou as temperaturas mais elevadas foi o 252 que, por estar na saída de um servidor bem em baixo do *rack*, teve maior dificuldade de receber o fluxo de ar do sistema de arrefecimento, e no instante da falha registrava 39°C chegou a registrar 50°C.

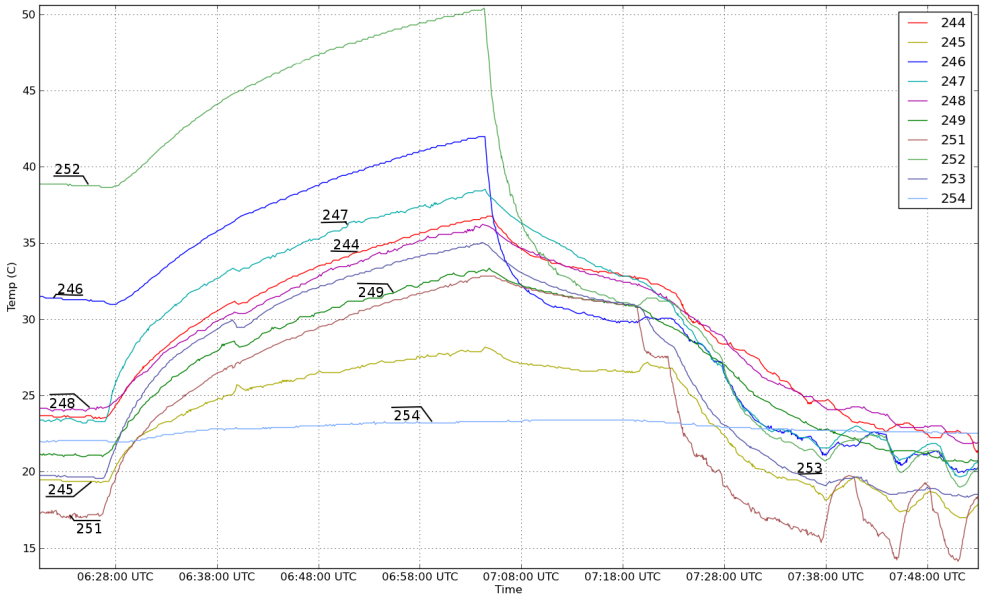



Figura 6.3: Perfil do comportamento térmico no instante da falha do sistema de arrefecimento.

Com base nesse experimento, pode-se observar que o ideal é enviar informações com a maior frequência possível, a fim de detectar a falha de um ou mais *nós* o quanto antes. Entretanto, na contra mão dessa necessidade, tem-se o requisito básico para toda RSSF alimentada a bateria, o de economizar energia, o que é feito principalmente minimizando o número de transmissões de dados dos *nós* da RSSF. Com objetivo de manter esse compromisso e por conta desse experimento não avaliar esse impacto na periodicidade de transmissão no consumo energético, foi executado o experimento descrito na Seção 6.1.4.

### 6.1.4 Impacto do Tempo Limite de Envio dos Dados Coletados no Consumo Energético do Nó

Esse experimento tem como objetivo verificar qual o tempo limite máximo possível para envio das informações coletadas de modo que não comprometa o tempo de vida dos

*nós* sensores da RSSF. Para execução desse teste, foram utilizados quatro *nós*, cada um com um tempo limite para envio distinto, sendo eles: 5, 10, 15 e 30 minutos. Foram escolhidos esses valores pois, em experimentos realizados para intervalos menores que 5 minutos, o consumo energético aumenta de forma significativa, e consequentemente o tempo de vida dos *nós* cai. Para valores acima que 30 minutos, em caso de falhar a RSSF e coincidentemente acontecer alguma anomalia na instalação, um aviso com esse atraso pode ser muito tarde, como pode ser visto da Figura 6.3.

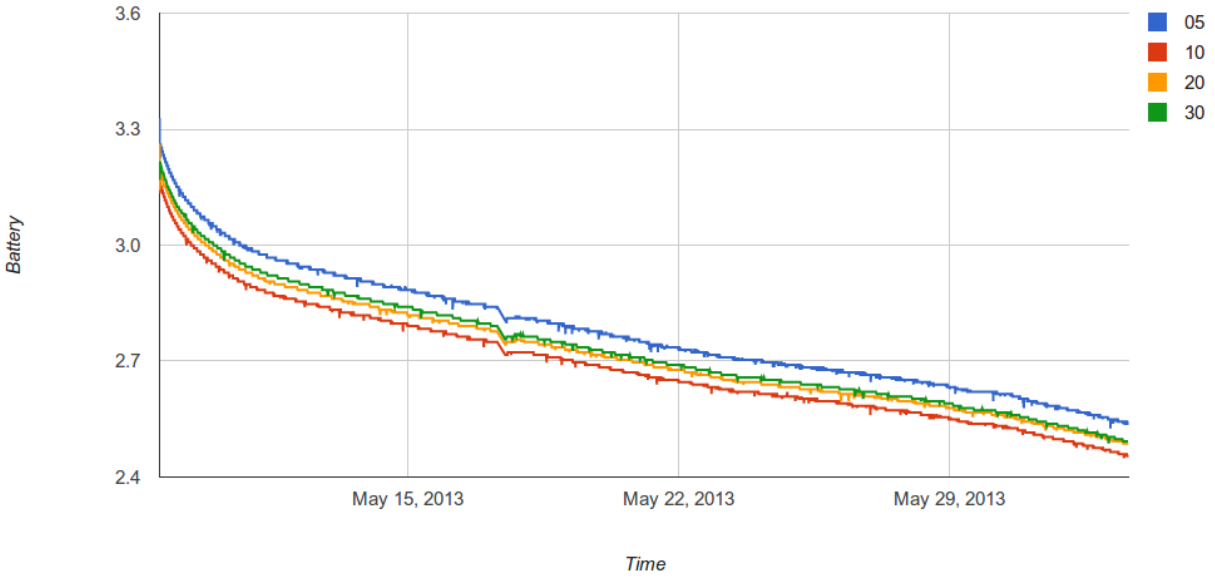



Figura 6.4: Resultado do experimento variando o intervalo máximo para envio dos dados coletados.

Como se pode observar no gráfico da Figura 6.4 e diante dos dados exibidos na Tabela 6.4, houve um ganho aproximado de 9,5% em tempo de vida dos *nós* utilizando o maior intervalo, ou seja 30 minutos, com relação ao menor intervalo, que é de cinco minutos. Assim, optou-se por perder esse ganho em tempo de vida e utilizar o menor intervalo para o tempo máximo para envio do *nó*, pelo fato desse valor possibilitar uma detecção precoce de falha nos *nós* da RSSF e no CPD.

### 6.1.5 Quantidade de Pacotes Perdidos Variando o Alcance do Rádio

Essa simulação tem como objetivo variar a potência de transmissão do rádio dos *nós* da RSSF para verificar o efeito no número de saltos dos pacotes que trafegam pela RSSF. Para efeito de cálculo levou-se em conta as transmissões e retransmissões de pacotes da aplicação (para detalhes sobre o pacote vide o Apêndice A.3) no trajeto dos *nós*

Tabela 6.4: Detalhamento de resultados do experimento para cálculo do consumo energético do *nó*, variando o intervalo máximo para envio dos pacotes (Obs.: Os valores referentes ao consumo energético estão em volts).

| **Intervalo de Verificação (s)** | **Bat. inicial** | **Bat. final** | **Consumo médio/dia** | **Longevidade estimada (dias)** |
|---|---|---|---|---|
| 5 | 3,33 | 2,53 | 0,031 | 38,05 |
| 10 | 3,33 | 2,50 | 0,030 | 38,50 |
| 20 | 3,26 | 2,49 | 0,030 | 38,88 |
| 30 | 3,21 | 2,50 | 0,028 | 41,69 |

consumidores até o *nó root* e também considerou-se o número de pacotes recebidos. Com base nesses valores é possível calcular o número médio de saltos, bem como as perdas[3] de pacotes.

Para execução da simulação, foi utilizado o simulador de aplicações nativas do *TinyOs*, o *Tossim* [27]. A aplicação utilizada para simulação foi o *Multihope Osciloscope*, presente nos aplicativos exemplos do *TinyOs*.

Tabela 6.5: Resultados da simulação para contagem do número de pacotes encaminhados perdidos variando a potência do rádio dos *nós* na RSSF.

| Atenuação de potência | Transmissões | Média de saltos | Recepções | Perdas % |
|---|---|---|---|---|
| 50 dB a cada 6 metros. | 23737 | 2.25 | 10507 | 42.62 |
| 50 dB a cada 3 metros. | 45787 | 2.66 | 17154 | 43.93 |
| 30 dB a cada metro. | 33637 | 2.68 | 12510 | 41.21 |
| 50 dB a cada metro. | 32985 | 3.41 | 9672 | 67.69 |
| 60 dB a cada metro. | 0 | 0 | 965 | 100 |

Com base nas informações mostradas na Tabela 6.5, pode-se observar que, quanto maior a atenuação do sinal, maior é o número de saltos e de perdas. Perda implica em retransmissão e, como consequência, em maior consumo energético. Concluiu-se então que para fazer sentido reduzir a potência de transmissão, ela deve fornecer um aumento de eficiência energética maior que o gasto com retransmissões na RSSF. Para isso, foi feito um experimento para medição do consumo energético dos *nós* na potência máxima e mínima permitida disponível em cada dispositivo, conforme descrito na Sessão 6.1.6

[3]Essa perda não implica necessariamente no não recebimento das informações pelo *nó root*, mas sim em reenvio dos pacotes a cada salto por conta de uma transmissão mal sucedida.

### 6.1.6 Impacto da Potência de Transmissão no Consumo Energético

Este experimento tem como objetivo verificar o impacto da potência de transmissão do rádio no consumo energético dos *nós*. Para essa verificação, foi desenvolvida uma aplicação para o *TinyOs* que captura e envia a voltagem para o *nó root* uma vez a cada segundo. Para realização dos experimentos foram utilizados *nós* do modelo Iris [35], configurados com potência de transmissão distintas.

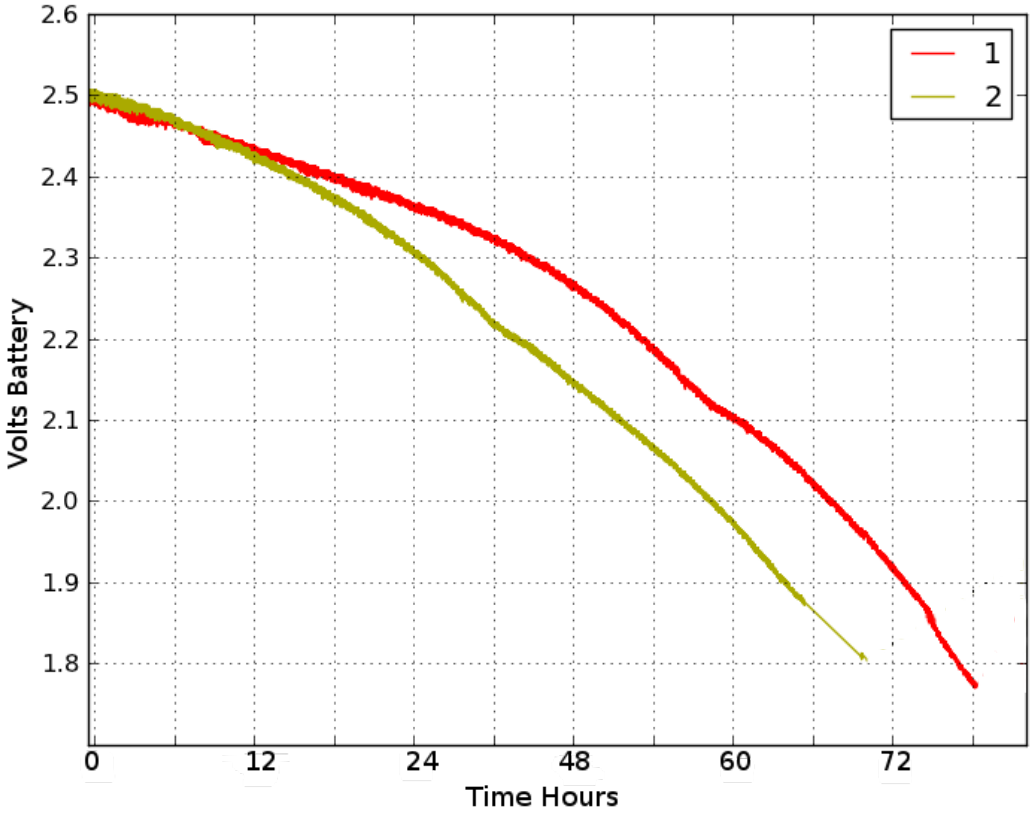



Figura 6.5: Gráfico do consumo energético variando-se a potência dos *nós*.

O *nó* com identificador 1 está na potência mínima e o com identificador 2 está na máxima. Como se pode notar na Figura 6.5, o *nó* com a potência mínima teve seu tempo de vida aproximadamente oito horas maior que o *nó* com potência máxima.

Em um CPD de pequeno porte, como o utilizado neste experimento, não existe a necessidade de um protocolo para gerenciamento do pacote por vários saltos e, portanto, o emprego de redução da potência pode trazer ganhos que goram em torno 11,5%, na longevidade dos *nós*. Entretanto em ambientes onde o pacote deve percorrer mais de um salto para chegar ao *root*, pelo fato da diferença de consumo girar em torno de 11,5% não

é viável do ponto de vista energético aumentar o número de saltos e reduzir a potência. Conforme medições realizadas, um *nó* que está configurado na potência mínima tem um alcance para comunicação de aproximativamente 20 metros e um na potência máxima tem alcance aproximado de 100 metros[4]. Portanto, com os nós configurados na potência mínima em um CPD de grande escala o número de saltos pode ser até 5 vezes maior, aumentando o consumo energético global da RSSF consideravelmente. Uma vez que o foco desse experimento é verificar os ganhos energéticos usando a redução de potência em uma rede com transmissões com mais de um salto, reduzir a potência de transmissão não é uma opção para poupar a bateria dos nós da RSSF.

## 6.2 Modelagens

### 6.2.1 Atraso atribuído pelo Box-MAC-1

Esse cálculo tem o objetivo de modelar e estimar o atraso máximo que pode ocorrer na entrega dos pacotes, em decorrência do uso do protocolo *Box-MAC-1*. Para isso foi necessário uma modelagem para identificar e estimar os parâmetros que influenciam nesse atraso. O primeiro é o tempo necessário para o *nó* destinatário acordar, definido pelo parâmetro *Tw*, que está relacionado diretamente com o intervalo de verificação do meio definido nas configurações do *Box-Mac-1*. O segundo parâmetro é o tempo gasto para envio do pacote, definido pelo parâmetro *Tx*. Ele depende diretamente do tamanho do pacote, que além das informações coletadas, tem cabeçalhos do protocolo de rede (CTP) e do protocolo MAC. E por último, mas não menos importante, é o raio da rede $R$ (*i.e.*, o número de saltos necessários para um pacote percorrer do *nó* origem até o *root*, como pode ser observado na Figura 6.6). Como base nesses parâmetros, o atraso no envio dos pacotes no pior caso pode ser modelado de acordo com a Equação 6.1.

$$\Delta T = R * (Tw + Tx) \tag{6.1}$$

Para mensurar esse atraso, precisa-se então estimar o valor de *Tx*. Para isso, primeiramente calcula-se o tamanho do pacote, que é formado pelos dados e pelos cabeçalhos do CTP e do Box-MAC-1. O pacote tem 16 Bytes, o cabeçalho do CTP tem 8 Bytes e o cabeçalho do *Box-Mac-1* tem 31 Bytes, que é o tamanho máximo do cabeçalho da camada MAC e física regulamentada pelo padrão IEEE 802.15.4 [46], totalizando 55 Bytes.

[4]Em um ambiente que o meio de comunicação esteja livre de obstáculos, isto é, cada *nó* tem visão direta do outro.

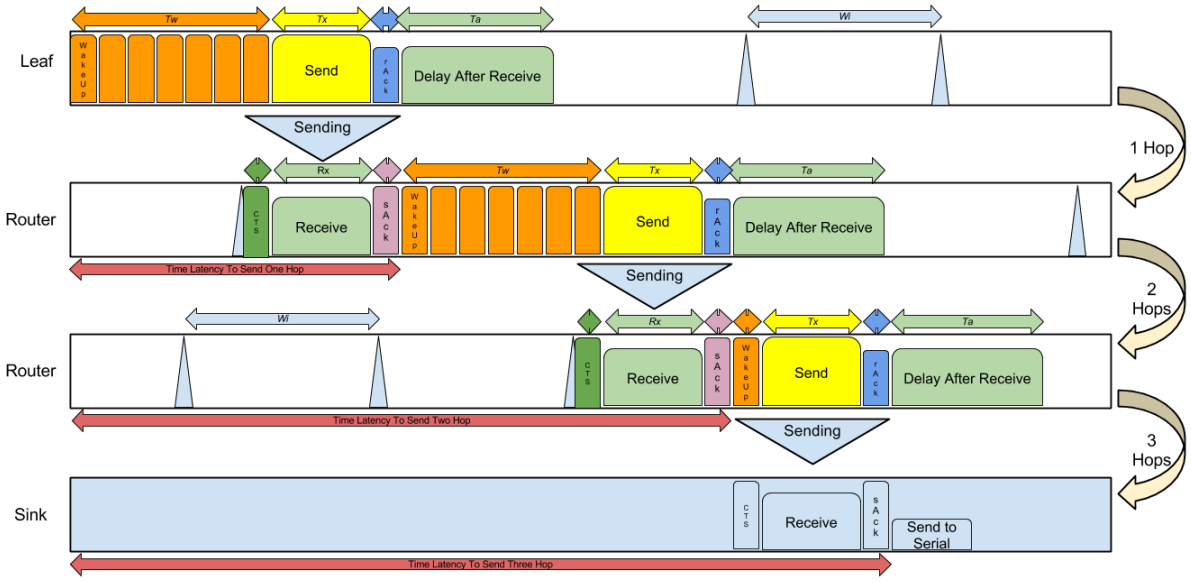



Figura 6.6: Modelagem do atraso atribuído pelo protocolo Box-MAC-1.

Levando-se em conta que a taxa de transmissão dos *nós* utilizados é de 250 kbps (Iris) a transmissão do pacote levaria aproximadamente 1,71 milissegundos. O valor do $Tw$ foi configurado, de acordo com o experimento da Seção 6.1.1, como sendo 300 milissegundos. E o $R$, no pior caso verificado na RSSF, é quatro saltos. Com base nessas informações, o atraso para recebimento das informações pelo *nó root*, no pior caso, é de aproximadamente 1,206 segundos.

### 6.2.2 *Duty-cycle* da Aplicação Embarcada

Esse experimento tem como objetivo estimar o tempo que o rádio, maior consumidor energético do *nó*, fica ligado, isto é, o *duty-cycle*. Para isso foi desenvolvida uma modelagem baseada na que foi apresentada no trabalho [40], em que é descrito o protocolo *Box-MAC-1* e foi contextualizada para a aplicação embarcada proposta.

Cada *nó* da RSSF gasta um certo tempo verificando o meio para identificar se existem transmissões para ele. Na Equação 6.2 é modelado esse tempo, onde $R$ representa o número de verificações do meio que são realizadas no intervalo de uma hora e *0,78ms* é a duração dessas verificações.

$$T_c = R * (0,78[ms]) \tag{6.2}$$

Os *nós* também gastam um certo tempo recebendo pacotes válidos e inválidos. O

tempo médio consumido por cada protocolo, em uma hora, para recepção de pacotes válidos é expresso na Equação 6.3, onde $V$ representa o número de pacotes recebidos no intervalo de uma hora, $T$ o intervalo entre as verificações do meio e *50ms* é o tempo médio para recebimento de um pacote válido.

$$T_v = V * (\frac{T}{2} + (50[ms])) \tag{6.3}$$

O tempo gasto para o recebimento dos pacotes inválidos é expresso na Equação 6.4, onde $I$ representa o número de pacotes inválidos no intervalo de uma hora (*i.e.* que foram recebidos mas continham falhas) e *20ms* é o tempo médio para recebimento de um pacote inválido.

$$T_i = I * (20[ms]) \tag{6.4}$$

E, por fim, cada nó gasta um determinado tempo transmitindo dados. Esse tempo é descrito pela Equação 6.5, onde $D$ representa o número de transmissões de pacotes para despertar no intervalo de uma hora.

$$T_x = D * T \tag{6.5}$$

Com base nessas equações, o *duty-cycle* dos nós da RSSF pode ser aproximado com base na Equação 6.6.

$$T_{on/h} = T_c + T_v + T_i + T_x \tag{6.6}$$

O parâmetro $R$ é definido com o valor de 1200 checagens por hora do meio, com base em um $T$ de 300 milissegundos que é ó valor definido por meio do experimento descrito na Seção 6.1.1.

O parâmetro $V$ depende da quantidade de pacotes gerados pela aplicação e pelos protocolos CTP e FTSP. Por conta do trafego da aplicação e do CTP variarem em função das condições de comunicações no ambiente, foi necessária a medição local nos *nós* da RSSF. O trafego médio gerado pela aplicação em cada *nó* é em torno de $6,9pck/h$ e o gerado pelo CTP fica em torno de $8,37pck/h$. O trafego gerado pelo FTSP é estático de $7,2pck/h$. Assim, $V = 22,47pck/h$.

O parâmetro $I$ foi estimado com base no experimento da Seção 6.1.5, foi utilizado o

resultado da simulação em que a atenuação da potência do rádio é de 50dB a cada metro, resultando numa perda de *42,62%* para um ambiente similar a um CPD. Esse perda foi aplicada para trafego gerado resultando em $I = 9,6pck/h$.

E, por fim, para $D$ assume-se como sendo a soma da quantidade de pacotes recebidos e inválidos, isto é, $I + V$, resultando em $D = 32,07$ transmissões por hora.

Com base nesses dados, pode-se estimar que o rádio de um nó pertencente a RSSF proposta fica ligado aproximadamente *23,67 segundos* a cada hora. Ou seja, o *duty-cycle* da aplicação embarcada proposta é de *0,65%*.

# Capítulo 7

# Considerações Finais

A solução apresentada neste trabalho teve como objetivo monitorar e prover dados que possibilitem monitorar e eventualmente prever o comportamento térmico, para reduzir o consumo energético dos sistemas de arrefecimento em CPDs, bem como aumentar a eficiência energética da instalação como um todo. Como mencionado anteriormente, os sistemas de arrefecimento são responsáveis por grande parte do consumo energético nos CPDs. Além disso, o sistema aqui proposto também provê informações sobre o estado da RSSF usada para monitorar o ambiente do CPD. É importante enfatizar que os testes para desenvolvimento da solução aqui proposta foram todos executados em um CPD real gerido pelo Instituto de Computação da Universidade Federal Fluminense.

O ambiente do CPD configura uma ampla fonte de interferências a qualquer tipo de comunicação sem fio, principalmente em dispositivos de baixa potência, como os utilizados neste trabalho. Para tratar esse problema, empregou-se o protocolo CTP que contorna este desafio através da manutenção da árvore de coleta de dados dinamicamente. Além disso temos o desafio que é comum para todas RSSF que possuem alimentação por meio de baterias, baixa longevidade dos *nós*, para tratar esse problema utilizou-se um protocolo assíncrono para um baixo consumo energético na comunicação, o *Box-MAC-1*. Por fim, para garantir uma maior precisão do instante em que os dados foram coletados, é também usado o protocolo FTSP, para tratar sincronização dos relógios locais dos *nós* da RSSF.

No início do desenvolvimento verificou-se que a proposta aqui apresentada era um dos primeiros trabalhos a agregar diversas tecnologias e conceitos (*e.g.* RSSF, servidor de eventos e sistema para visualização web) para coletar e disponibilizar dados ambientais de CPDs em tempo real. Além disso, tudo isso tinha que ser feito garantindo alta precisão e confiabilidade nas informações disponibilizadas. Características como essa tornaram o desenvolvimento da trabalho um tanto quanto desafiador.

A solução também provê dados térmicos do ambiente do CPD em tempo real, por meio de um servidor de eventos (*RabbitMQ*), para qualquer aplicação, de forma simplificada, e de fácil integração com o mecanismo de notificações disponível na interface web. Por meio dessa funcionalidade, no trabalho [4] estão sendo desenvolvidas formas de prever em tempo real o comportamento térmico, bem como identificar anomalias que podem ocasionalmente ocorrer no CPD. Através dessas previsões pode-se estimar o comportamento térmico nas diversas regiões do CPD. Assim, o gerente do CPD poderá consolidar a carga de trabalho prioritariamente em regiões com mais acesso ao resfriamento extra sem aumentar ou até mesmo deixando margens para redução da potência do sistema de arrefecimento e, em consequência disso, conseguir a redução dos custos energéticos.

## 7.1 Contribuições

Este trabalho apresentou a definição de uma abordagem de monitoramento termoenegético para CPDs. Para isso foram realizados passos para embasamento teórico, definição, implementação e análise da abordagem. Portando, as principais contribuições deste trabalho são:

- Definição e implementação de uma RSSF para coleta de dados de um CPD;
- Definição e implementação de um sistema para monitoramento termoenergético para CPDs;
- Definição e implementação de abordagem não intrusiva para monitoramento de CPDs - a abordagem não necessita de nenhuma interação com os equipamentos do CPD;
- Definição e implementação de sistema para prover em tempo real dados coletados pela RSSF do CPD para sistemas terceiros - por meio do servidor de eventos;
- Experimentação de uma RSSF em ambientes de CPDs.

## 7.2 Trabalhos Futuros

O trabalho aqui proposto possui diversas funcionalidades que foram desenvolvidas conforme o escopo definido para o projeto. Entretanto outras funcionalidades foram identificadas sem que fossem implementadas nesta primeira versão. Essas funcionalidades

identificadas são descritas nesta seção para que futuramente possam ser implementadas e incorporadas ao trabalho, resultando em uma solução mais completa, fornecendo um maior grau de confiança sobre as informações coletadas pela RSSF.

### 7.2.1 Mecanismo para Tolerância a Falhas na Base de Coleta

Os *nós root ou base* são essenciais para a RSSF pelo fato de desempenharem papéis como recepção e encaminhamento de pacotes da rede para o computador e sincronia do tempo na rede. Caso aconteça algum problema na comunicação serial entre o *nó root* e o computador base podem ocorrer graves problemas decorrentes da perdas de dados de toda a RSSF. Por conta desses fatores fez-se necessário um mecanismo para prover tolerância à falhas.

Embora não esteja no escopo do trabalho, uma possível solução poderia ser implementada da seguinte maneira: cada computador base poderá ter conectado a ele *n nós root*, estando esses, por padrão, no modo inativo com exceção de um que ficará ativo e será de fato um *root* para a rede naquele momento. O *nó* root no estado ativo conectado ao computador base enviará periodicamente um pacote de notificação para sinalizar que está funcional. Destes *n nós* conectados ao computador, todos devem ter um identificador que satisfaça as necessidades dos protocolos utilizados para assumir o papel de *root*. No caso do FTSP e do CTP, o *nó* com menor identificador tem a prioridade de assumir a tarefa de *root* da rede para ambos os contextos. Caso o *nó root* ativo falhe, por um motivo qualquer, e não envie mais pacotes para sinalizar que seu estado é de funcional, imediatamente o computador base desativará este *nó* considerado falho (ou não confiável), e ativará um dos outros *nós* conectados para substituir o *nó* problemático.

### 7.2.2 Mecanismo para Modelagem Tridimensional do CPD

A solução proposta não provê aos usuários uma visualização completa das informações. Por conta disso, sugere-se o desenvolvimento de um mecanismo web que possibilite ao administrador do CPD a modelagem tridimensional do ambiente. Esse mecanismo terá como objetivo prover uma visualização mais precisa, interativa e intuitiva dos dados capturados pela RSSF. Essa interface deve conter todos os itens comumente encontrados em CPD (*e.g. racks*, servidores, salas), possibilitando que o usuário simplesmente arraste os objetos em uma ambiente de modelagem deixando a instalação virtual o mais semelhante à sua realidade possível.

Com base nessa modelagem tridimensional propõe-se também prover maneiras mais intuitivas de exibir as informações coletadas pela RSSF e os alertas do sistema aos usuários. Tudo com o objetivo de tornar a solução mais completa e intuitiva possível.

### 7.2.3 *Framework* para Construção/manutenção de CPDs

Depois da implantação da solução em diversos tipos de CPDs, espera-se identificar os principais motivos, com relação aos aspectos térmicos, que levam os CPDs a terem uma baixa eficiência energética. Com as principais causas para a ineficiência energética em CPDs levantadas, será possível elaborar um *framework* para construção/manutenção dessas instalações. Posteriormente, com base nesses padrões identificados, desenvolver uma metodologia padrão para fiscalização e certificação da eficiência energética de CPDs com relação ao sistema de arrefecimento.

### 7.2.4 Integração de Protocolos para RSSFs

Fica também como sugestão o desenvolvimento de protocolos integrados, que forneçam as funcionalidades básicas (*e.g.* coleta, controle do rádio para aumento da eficiência energética, criptografia e sincronia de relógios), para a aplicação em RSSFs, que minimizem o *overhead* gerado pelo uso de vários protocolos distintos em paralelo. Por exemplo, melhorando alguns aspectos (*e.g.* eficiência energética e segurança), principalmente por meio da integração das funcionalidades semelhantes (*e.g.* integração da disseminação de *beacons* de controle para manutenção de rede do CTP com os *beacons* para atualização de relógio do FTSP) providas por esses protocolos.

Outra possível integração no nível dos protocolos da RSSF, seria o uso do nível da bateria no cálculo do ETX. Isso seria feito visando reduzir o tráfego em *nós* que fazem o papel de roteadores e que estejam com a bateria perto do fim. Para isso seria proposto algum algoritmo que aumentasse o custo do ETX conforme o nível da bateria fosse caindo.

# Referências


[1] Ahmad, F.; Vijaykumar, T. Joint optimization of idle and cooling power in data centers while maintaining response time. In *ACM Sigplan Notices* (New York, EUA, Março 2010), pp. 243–256.

[2] Akyildiz, I. F.; Su, W.; Sankarasubramaniam, Y.; Cayirci, E. Wireless sensor networks: a survey. *Elsevier Computer Networks 38*, 4 (Março 2002), 393–422.

[3] Bostock, M.; Ogievetsky, V.; Heer, J. $D^3$ data-driven documents. *Visualization and Computer Graphics 17*, 12 (2011), 2301–2309.

[4] Bottari, G. Técnicas inteligentes para monitoramento termoenégético de CPDs. Dissertação de Mestrado em andamento, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro, 2013.

[5] Brownfield, M.; Gupta, Y.; Davis, N. Wireless sensor network denial of sleep attack. In *IEEE Information Assurance Workshop* (West Point, EUA, Junho 2005), pp. 356–364.

[6] Buettner, M.; Yee, G. V.; Anderson, E.; Han, R. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *ACM 4th International Conference on Embedded Networked Sensor Systems* (Boulder, EUA, Outubro 2006), pp. 307–320.

[7] Chaudhuri, S.; Dayal, U. An overview of data warehousing and OLAP technology. *ACM Sigmod Record 26*, 1 (Março 1997), 65–74.

[8] Christensen, K. Green networks: Opportunities and challenges. In *34th IEEE Conference on Local Computer Networks* (Zurich, Suíça, Outubro 2009), p. 13.

[9] De Couto, D. S.; Aguayo, D.; Bicket, J.; Morris, R. A high-throughput path metric for multi-hop wireless routing. *Wireless Networks 11*, 4 (Julho 2005), 419–434.

[10] Django Software Foundation. About the Django Software Foundation, 2013. `https://www.djangoproject.com/foundation/`.

[11] EPA. Report to congress on server and data center energy efficiency, 2007. `http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf`.

[12] Eugster, P. T.; Felber, P. A.; Guerraoui, R.; Kermarrec, A.-M. The many faces of publish/subscribe. *ACM Computing Surveys 35*, 2 (Junho 2003), 114–131.

[13] Fonseca, R.; Gnawali, O.; Jamieson, K.; Kim, S.; Levis, P.; Woo, A. TEP 123: collection tree protocol. Tech. rep., 2006.

[14] Forouzan, B. A. *TCP/IP protocol suite*. McGraw-Hill, Inc., 2002.

[15] Gay, D.; Levis, P.; Von Behren, R.; Welsh, M.; Brewer, E.; Culler, D. The nesc language: A holistic approach to networked embedded systems. 1–11.

[16] Glanz, J. Power, Pollution and the Internet. In *The New York Times*. Setembro 2012, p. 5.

[17] Gnawali, O.; Fonseca, R.; Jamieson, K.; Moss, D.; Levis, P. Collection tree protocol. In *7th ACM Conference on Embedded Networked Sensor Systems* (Novembro 2009), pp. 1–14.

[18] Hagberg, A.; Swart, P.; S Chult, D. Exploring network structure, dynamics, and function using NetworkX. Tech. rep., Los Alamos National Laboratory, 2008.

[19] Holovaty, A.; Kaplan-Moss, J. *The definitive guide to Django: Web development done right*. Apress, 2009.

[20] Huang, W.; Allen-Ware, M.; Carter, J. B.; Elnozahy, E.; Hamann, H.; Keller, T.; Lefurgy, C.; Li, J.; Rajamani, K.; Rubio, J. Tapo: thermal-aware power optimization techniques for servers and data centers. In *2nd IEEE Green Computing Conference and Workshops* (Orlando, EUA, Julho 2011), pp. 1–8.

[21] Hui, J. Deluge 2.0 - TinyOs network programming, 2005. `http://www.cs.berkeley.edu/~jwhui/deluge/deluge-manual.pdf`.

[22] Jubin, J.; Tornow, J. D. The darpa packet radio network protocols. *Proceedings of the IEEE 75*, 1 (Janeiro 1987), 21–32.

[23] Koomey, J. Growth in data center electricity use 2005 to 2010. *Analytics Press 1*, 1 (Agosto 2011).

[24] Krasner, G. E.; Pope, S. T., et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming 1*, 3 (1988), 26–49.

[25] Lerner, R. M. At the forge: Twitter bootstrap. *Linux Journal 2012*, 218 (Junho 2012), 6.

[26] Levis, P. Experiences from a decade of tinyos development. In *USENIX 10th Conference on Operating Systems Design and Implementation* (Hollywood, EUA, Outubro 2012), pp. 207–220.

[27] Levis, P.; Lee, N.; Welsh, M.; Culler, D. TOSSIM: Accurate and scalable simulation of entire tinyos applications. In *1st ACM International Conference on Embedded Networked Sensor Systems* (Los Angeles, California, EUA, Outubro 2003), pp. 126–137.

[28] Levis, P.; Madden, S.; Polastre, J.; Szewczyk, R.; Whitehouse, K.; Woo, A.; Gay, D.; Hill, J.; Welsh, M.; Brewer, E., et al. TinyOs: an operating system for sensor networks. In *Ambient intelligence*. Springer, 2005, pp. 115–148.

[29] Li, L.; Liang, C.-J. M.; Liu, J.; Nath, S.; Terzis, A.; Faloutsos, C. Thermocast: a cyber-physical forecasting model for data centers. In *17th ACM Conference on Knowledge Discovery and Data Mining* (San Diego, EUA, Agosto 2011), vol. 11.

[30] Liang, C.-J. M.; Liu, J.; Luo, L.; Terzis, A.; Zhao, F. Racnet: a high-fidelity data center sensing network. In *7th ACM Conference on Embedded Networked Sensor Systems* (Novembro 2009), pp. 15–28.

[31] Mankoff, J.; Kravets, R.; Blevis, E. Some computer science issues in creating a sustainable world. *IEEE Computer 41*, 8 (Agosto 2008), 102–105.

[32] Maróti, M.; Kusy, B.; Simon, G.; Lédeczi, Á. The flooding time synchronization protocol. In *2nd ACM International Conference on Embedded Networked Sensor Systems* (Baltimore, EUA, Novembro 2004), pp. 39–49.

[33] Maroti, M.; Sallai, J. TEP 133: packet-level time synchronization. Tech. rep., 2008. `http://www.tinyos.net/tinyos-2.1.0/doc/html/tep133.html`.

[34] Meisner, D.; Gold, B. T.; Wenisch, T. F. Powernap: eliminating server idle power. In *Architectural Support for Programming Languages and Operating Systems* (Março 2009), pp. 205–216.

[35] MEMSIC. Iris datasheet. `http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf`.

[36] MEMSIC. MDA 100 datasheet. `http://www.memsic.com/userfiles/files/Datasheets/WSN/mts_mda_datasheet.pdf`.

[37] MEMSIC. MIB 520 datasheet. `http://www.memsic.com/userfiles/files/Datasheets/WSN/6020-0091-04_a_mib520cb-t.pdf`.

[38] MEMSIC. Micaz datasheet. `http://www.memsic.com/userfiles/files/Datasheets/WSN/MICAZ_Datasheet.pdf`.

[39] Moss, D.; Hui, J.; Klues, K. TEP 116: low power listening. `http://www.tinyos.net/tinyos-2.x/doc/html/tep116.html`.

[40] Moss, D.; Levis, P. Box-macs: Exploiting physical and link layer boundaries in low-power networking. Tech. rep., Stanford University, 2008.

[41] Mossé, D.; Leite, J.; de Barros, A. Introdução aos Clusters Verdes de Servidores. In *Atualizações em Informática*, W. M. Jr. and A. de Carvalho, Eds. Sociedade Brasileira de Computação, Porto Alegre, RS, Brasil, Junho 2010, pp. 41–50.

[42] Paek, J.; Govindan, R. RCRT: rate-controlled reliable transport for wireless sensor networks. In *5th ACM International Conference on Embedded Networked Sensor Systems* (Sydney, NSW, Australia, Novembro 2007), pp. 305–319.

[43] Polastre, J.; Hill, J.; Culler, D. Versatile low power media access for wireless sensor networks. In *ACM 2nd International Conference on Embedded networked Sensor Systems* (Baltimore, EUA, Novembro 2004), pp. 95–107.

[44] RabbitMQ. What can RabbitMQ do for you?, 2013. http://www.rabbitmq.com/features.html.

[45] Romer, K.; Mattern, F. The design space of wireless sensor networks. *IEEE Wireless Communications 11*, 6 (Dezembro 2004), 54–61.

[46] Salman, N.; Rasool, I.; Kemp, A. Overview of the IEEE 802.15.4 standards family for low rate wireless personal area networks. In *7th IEEE International Symposium on Wireless Communication Systems* (York, United Kingdom, Setembro 2010), pp. 701–705.

[47] VanGeet, O. Best practices guide for energy-efficient data center design. *US Department of Energy Federal Energy Management Program* (2011).

[48] Vinoski, S. Advanced message queuing protocol. *IEEE Internet Computing 10*, 6 (Janeiro/Fevereiro 2006), 87–89.

# APÊNDICE A - Códigos

Listagem A.1: Correção CTP.

```
if (!forcePBit && currentEtx != MAX_METRIC && currentEtx > 100 &&
    currentInterval > 1024 * 5)){
  forcePBit = TRUE;
  resetInterval();
}
```

Listagem A.2: Lógica da aplicação embarcada.

```
event void Timer.fired() {
  float tempDiff = 0.0f;
  int16_t lightDiff;
  //Coleta a tenperatura
  call Temp.read();
  //Se a temperatura atual e a ultima coletada não for nula é feito o
     cálculo da variação entre elas
  if (lastTemperature != NULL_TEMP_READING && temperatureNow !=
     NULL_TEMP_READING)
  {
    tempDiff = abs(adc_to_celsius(lastTemperature) - adc_to_celsius(
       temperatureNow))
  }else if (temperatureNow == NULL_TEMP_READING) // Assegura que não vai
     enviar se a temperatura coletada é incoerente
  {
    suppressedTemperatures = suppressedTemperatures == suppressionMsgs ?
       suppressedTemperatures - 1: suppressedTemperatures;
  }else // (last_reading <= 0) certificar que irá enviar independente da
     variação de tempo
  {
    suppressedTemperatures = suppressionMsgs;
  }
  //Faz a leitura da luminosidade e calcula a diferença entre a leitura
     atual e a ultima
```

```
18   call Light.read();
19   lightDiff = lastLight - lightNow;
20   lightDiff = lightDiff < 0 ? -lightDiff : lightDiff;
21   //Coleta o status da bateria e verifica se é necessário notificar que
        está com pouca carga
22   call Volt.read();
23   if (voltageNow > 624) {
24     call Leds.led2Toggle();
25   }
26   //Faz a leitura da qualidade do da comunicação com seu pai, dado provido
        pelo CTP
27   call CtpInfo.getEtx(&etxNow);
28   //Cas haver variação de temperatura, luminosidade ou do ETX, ou ficou um
        período de tempo sem enviar, é feito o envio dos dados.
29   if((tempDiff > TEMPERATURE_DELTA) | (suppressedTemperatures ==
        suppressionMsgs) | (lightDiff > LIGHT_DIFF_THRESHOLD) | (lastEtx !=
        etxNow) ) {
30     atomic{
31       if (!sendBusy)
32         sendMessage();
33         //Controla o período de envio inicial.
34         initialSendPhase++;
35         if (initialSendPhase == NUMBER_INITIAL_MSGS){
36           suppressionMsgs = MAX_SUPPRESSION_MSGS;
37           }
38     }
39     suppressedTemperatures = 0;
40     lastTemperature = temperatureNow;
41     lastLight = lightNow;
42     lastEtx = etxNow;
43   }else
44   {
45     suppressedTemperatures++;
46   }
47 }
```

Listagem A.3: Descrição do pacote de dados da RSSF.

```
1 typedef nx_struct TMON {
2   nx_uint16_t readingTemperature; /* Valor da Temperatura */
3   nx_uint16_t readingLight; /* Valor da Luminosidade */
4   nx_uint16_t readingVoltage; /* Valor da bateria */
5   /* variáveis de informações sobre a topologia WSN */
6   nx_am_addr_t source; /* Id do nó origem do pacote */
```

```
7   nx_uint16_t seqno;  /* Número de sequencia do pacote*/
8   nx_am_addr_t parent; /* Nó pai do nó que originou o pacote*/
9   nx_uint16_t cost; /* Custo da transmissão pela métrica ETX*/
10  nx_uint32_t delay; /* Atraso na coleta dos dados */
11 } TMON_t;
```