UNIVERSIDADE FEDERAL FLUMINENSE
ESCOLA DE ENGENHARIA
Departamento de Engenharia de Telecomunicações

Cledson Oliveira de Sousa

STELE - Uma Técnica Simples Para Estimativa Local do Atraso de Transmissão em RSSF.

Niterói
2013

STELE - Uma Técnica Simples Para Estimativa Local do Atraso de Transmissão em RSSF.

Dissertação apresentada ao Curso de Mestrado em Engenharia de Telecomunicações da Universidade Federal Fluminense como requisito parcial para obtenção do Grau de Mestre. Área de concentração: Comunicação de Dados Multimídia.

Orientador: Prof. Luiz Claudio Schara Magalhães, Ph.D.

Coorientador: Prof. Ricardo Campanha Carrano, MSc.

Niterói-RJ
2013

Cledson Oliveira de Sousa

STELE - Uma Técnica Simples Para Estimativa Local do Atraso de Transmissão em RSSF.

Dissertação apresentada como exigência parcial para a obtenção do grau de Mestre. Área de concentração: Comunicação de Dados Multimídia.

Aprovada em:_____

_____
Prof. Luiz Claudio Schara Magalhães, Ph.D.
Universidade Federal Fluminense
Orientador

_____
Prof. Ricardo Campanha Carrano, M.Sc.
Universidade Federal Fluminense
Coorientador

_____
Prof. Carlos Alberto Malcher Bastos, D.Sc.
Universidade Federal Fluminense
Avaliador

_____
Prof. Alexandre Sztajnberg, D.Sc.
Universidade Estadual do Rio de Janeiro
Avaliador

Niterói-RJ
2013

A Fé inabalável é somente aquela que pode
encarar a razão face a face.

AGRADECIMENTOS

Aos pais, sempre!

Aos professores, pelo conhecimento e orientação.

À família, pelo apoio e paciência.

“Que hoje seja um bom dia para a Ciência!”

Dexter

RESUMO

Sousa, Cledson. **STELE - Uma Técnica Simples Para Estimativa Local do Atraso de Transmissão em RSSF**: 2013. 69fls. Dissertação (Mestrado em Engenharia de Telecomunicações) – Universidade Federal Fluminense, Niterói, 2013.

As técnicas tradicionais para o cálculo do atraso de transmissão entre nós vizinhos implicam troca recíproca de quadros e pressupõem a simetria dos enlaces. Além de energeticamente custosa, essa abordagem não é adequada para redes de sensores sem fio, onde os enlaces são tipicamente assimétricos. Por outro lado, técnicas como a introdução de marcas de tempo pela camada MAC eliminam o problema da assimetria, mas são dependentes de *hardware* específico. Neste artigo, é apresentada uma técnica simples de estimativa local do atraso de transmissão (STELE), que é independente de *hardware* e que elimina a necessidade de comunicação bidirecional, sendo, portanto, aplicável a enlaces assimétricos. Resultados experimentais com sensores reais mostram a técnica como vantajosa para redes densas.

Palavras-chave: redes de sensores, atraso de transmissão, economia de energia e sincronização.

ABSTRACT

Traditional techniques for delay calculation between neighbor nodes involve the exchange of frames and assume link symmetry. This approach, besides being costly in terms of energy, is not a good fit for wireless networks, where links are typically asymmetric. On the other hand, MAC layer timestamps eliminate the asymmetry problem, but depend on specific hardware. In this paper, we present a simple technique for local delay estimation (STELE) which is hardware independent and eliminates the need for bidirectional communication, and is therefore applicable to asymmetric links. Experimental results with sensors motes prove this technique advantageous for dense networks.

Key-words: sensor networks, transmission delay, energy saving and synchronization.

LISTA DE FIGURAS

Figura 1a. *Mote* modelo MicaZ 16

Figura 1b. *Mote* modelo Iris 16

Figura 2. Componentes básicos de hardware de um nó sensor..... 16

Figura 3. Placa de programação MIB520 17

Figura 4. CC2531 USB*Dongle* 18

Figura 5. Tela SmartRF IEEE802.15.4/ZigBee Packet Sniffer. 18

Figura 6. Sobreposição de histogramas dos atrasos em *mote* reais Micaz e no WSIM. . 20

Figura 7. Comportamento do escorregamento nos *motes* Mica2 [Lenzen et al. 2009]. 24

Figura 8. Momento de adição do *timestamp*. [Ping 2003]..... 36

Figura 9. Escorregamento de relógio em relação a um *mote"A"*.[Lenzen et al. 2009] 39

Figura 10. Comparação do erro de sincronização entre o FTSP e PulseSync em uma RSSF *multihop*.[Lenzen *et al.* 2009] 40

Figura 11. Diagrama de tempo da troca de mensagem do mecanismo proposto. 44

Figura 12. Componentes da aplicação do TinyOS. 46

Figura 13. Diagrama de tempo detalhado do experimento..... 48

Figura 14. Sobreposição de histogramas dos atrasos reais. 49

Figura 15. Dispersão do atraso *mote* MicaZ em ambiente simulado TOSSIM. 52

Figura 16. Sobreposição de histogramas dos atrasos no mote MicaZ e no TOSSIM..... 53

Figura 17a. Dispersão do atraso *mote* MicaZ 55

Figura 17b. Dispersão do atraso *mote* Iris 55

Figura 18a. Dispersão do erro no *mote* MicaZ 57

Figura 18b. Dispersão do erro no *mote* Iris..... 57

Figura 19. Exemplo de uso do STELE com técnica de sincronização 58

Figura 20a. Consumo de energia no uso de CPU. Simulação com cinco nós. 62

Figura 20b. Consumo de energia no uso de CPU. Simulação com vinte nós. 62

Figura 20c. Consumo de energia no uso do rádio. Simulação com cinco nós. 62

Figura 20d. Consumo de energia no uso de rádio. Simulação com vinte nós. 62

LISTA DE TABELAS

Tabela 1. Comparações entre parâmetros estatísticos em *motes* reais e no WSIM com intervalo de confiança de 0,95 20

Tabela 2. Caracterização das componentes de tempo do atraso de transmissão. 34

Tabela 3. Principais diferenças entre as especificações dos motes. 45

Tabela 4. Comparação dos parâmetros estatísticos dos atrasos T_X medidos nos dois modelos de sensores. 49

Tabela 5. Valores dos parâmetros e seus impactos na média e variância dos atrasos em ambiente simulado. 52

Tabela 6. Comparações entre a estimativa T_E em *motes* reais e no TOSSIM. 53

Tabela 7. Tipos de correlações entre variáveis 54

Tabela 8. Comparação entre atraso e estimativa nos modelos Iris e MicaZ (IC=95%) .. 56

Tabela 9. Gasto de energia referente à CPU e ao Rádio com 5 nós 61

Tabela 10. Gasto de energia referente à CPU e ao Rádio com 20 nós. 61

LISTA DE SIGLAS

ASCII	*American Standard Code for Information Interchange*
CPU	*Central Processing Unit*
ECDF	*Empirical Cumulative Distribution Function*
FTSP	*Floodind Time Synchronization Protocol*
GPS	*Global Positioning System*
IEEE	*Institute of Electrical and Electronic Engineers*
kbps	*kilobits* por segundo
LAN	*Local Area Network*
nesC	*network embedded system C*
NS-2	*Network Simulator*
NTP	*Network Time Protocol*
MAC	*Medium Access Control*
PC	*Personal Computer*
PCAP	*Packet Capture*
PPM	Partes por milhão
PHY	*Physical Layer*
PTP	*Precision Time Protocol*
RBS	*Reference-Broadcast Synchronization*
RF	Rádio Frequência
RSSF	Redes de Sensores Sem Fio
RTT	*Round Trip Time*
SFD	*Start Frame Delimiter*
STELE	*Simple Technique for Local Delay Estimation*
USB	*Universal Serial Bus*

SUMÁRIO

AGRADECIMENTOS ..... **iv**

RESUMO ..... **vi**

ABSTRACT ..... **vii**

LISTA DE FIGURAS ..... **viii**

LISTA DE TABELAS ..... **ix**

LISTA DE SIGLAS ..... **x**

Capítulo 1 – Introdução **13**

Capítulo 2 – Metodologia e ferramentas **15**

2.1 Metodologia 15
2.2 *Motes* e outros dispositivos 15
2.3 O Sistema operacional dos *motes* 18
2.4 Ferramentas de simulação 19
2.5 A ferramenta estatística 22

Capítulo 3 - Desafios da sincronização **24**

3.1 Relógios de baixo custo 24
3.2 Comunicação sem fio 26
3.3 Escassez de recursos. 26
3.4 Falhas nos nós 27

Capítulo 4 – Trabalhos relacionados **28**

4.1 A sincronização em RSSF 29
4.2 RBS – Reference Broadcast Synchronization 29
4.3 TPSN – Timing-Sync Protocol for Sensor Networks 31
4.4 FTSP – Flooding Time Synchronization Protocol 32
4.4.1 Marca de tempo (*timestamp*) 35
4.4.2 Escorregamento de relógio 36
4.4.3 Sincronização multissalto 37
4.5 PulseSync 39

Capítulo 5 – *STELE* - Uma Técnica Simples Para a Estimativa do Atraso Local em RSSF **42**

5.1 Desafio 42
5.2 O mecanismo 42
5.3 Avaliação do mecanismo 45
5.4 Diferenças entre os modelos testados 48
5.5 O ambiente simulado 50
5.6 Resultados em ambiente simulado 51
5.7 Resultados com motes reais 53
5.8 Exemplo de uso do STELE como técnica de sincronização 57

Capítulo 6 – Consumo de energia **60**

6.1 Consumo de energia em ambiente simulado 60
6.2 Análise da simulação do consumo de energia 61

Capítulo 7 – Considerações finais..... 64
7.1 Cenários de uso 64
7.3 Conclusões e trabalhos futuros 64
Referências..... 66

Capítulo 1 –Introdução

O cálculo do atraso de transmissão entre dois nós é fundamental para diversos mecanismos utilizados em redes sem fio, sendo o exemplo mais evidente a sincronização entre dois dispositivos, que por sua vez, é componente essencial em outros tantos mecanismos, passando pelo *duty cycling* da interface do rádio [Yick et al. 2008] até as técnicas de redução de colisão [Anastasi *et* al. 2006]. Dentro deste contexto, a estimativa do atraso é um objetivo básico e deve ser alcançado sem negligenciar outras restrições importantes destas redes, como a alta probabilidade de falha nos sensores (*motes*) e os escassos recursos de computação. Além disso, em dispositivos de capacidades limitadas, como os nós de uma rede de sensores, é preciso observar também outras limitações de hardware, como a imprecisão dos relógios, sua baixa autonomia energética e algumas já bem caracterizadas como a assimetria dos enlaces em dispositivos munidos de transmissores de baixa potência [Woo, Alec *and* Culler, D. 2002], essa restrição é ainda mais severa nas redes de sensores.

Todo esse contraste nas condições dos rádio-enlaces costuma ser descrito em termos das diferenças de qualidade entre os links de ida e os de volta e, mesmo quando simétricos do ponto de vista da probabilidade de recepção de quadros, os enlaces em redes sem fio ainda apresentam tempos de ida e de volta distintos. Assim, estimar a latência como metade do RTT se torna impreciso e essas divergências se dão por diversos fatores. Entre eles:

- Disputas pelo meio;
- Necessidade de retransmissões;
- Diferenças nos tempos de processamento e enfileiramento entre os nós.

Em síntese, a diferença entre os tempos de ida e de volta é função das incertezas no tempo de transmissão em cada uma das etapas do processo de comunicação, desde o momento em que a aplicação de um nó solicita o envio de uma mensagem, até que esta seja recebida pelo destinatário. Análises detalhadas destas incertezas e suas amplitudes podem ser encontradas em [Maróti *et al.* 2004]. Desta forma, para obter a sincronização entre dois nós, precisamos então caracterizá-las, eliminá-las ou compensá-las. E são diversas as técnicas propostas para este fim, vejamos algumas delas.

Ao contrário das redes cabeadas, em redes *ad hoc*, as técnicas empregadas pelos protocolos NTP[Mills 1985] e PTP [Committee, T. 2008], não apresentam bons resultados, por conta dos obstáculos já mencionados. Por esta razão, novos mecanismos foram propostos, como o RBS [Elson et al. 2002], o TPSN [Ganeriwal *et al.* 2003], o FTSP [Maróti *et al.* 2004] e o PulseSync [Lenzen *et al.* 2009] e apesar da precisão alcançada, em geral da ordem de dezenas de microssegundos, alguns destes mecanismos precisam de $O(n^2)$ trocas de mensagens, onde n é o número de nós, tornando-se muito custosos em energia e outros se apóiam em recursos que não estão presentes universalmente em todos os nós sensores.

Neste trabalho, apresentaremos um método de estimativa do atraso de transmissão calculado inteiramente no transmissor, que alcança em média uma precisão da ordem de 80 microssegundos, sem dependência de elementos externos ou características especiais de hardware, se utilizando apenas de duas mensagens para alcançar a sincronização entre dois nós e ele está organizado da seguinte forma: O capítulo dois contém a metodologia. Os principais desafios à sincronização em redes de sensores estão no capítulo três e os trabalhos relacionados no capítulo quatro. O mecanismo STELE [Sousa et al. 2013], bem como suas validações, simulações e um exemplo de uso do STELE como técnica de sincronização são apresentados no capítulo cinco. O capítulo seis expõe os testes e análises sobre o consumo de energia do nosso mecanismo e as considerações finais. Futuros desenvolvimentos e desdobramentos da técnica estão dispostos no capítulo sete.

Capítulo 2 - Metodologia e ferramentas

2.1 Metodologia

A metodologia utilizada neste trabalho baseou-se na montagem experimentos e execução de simulações visando à obtenção de dados reais e simulados do atraso de transmissão em uma rede de sensores, de modo que pudéssemos estabelecer um comportamento estatístico destes atrasos, a fim de estimá-los e compensá-los, estas avaliações não foram feita em apenas um, mas em dois modelos diferentes de sensores. Após a obtenção dos resultados iremos comparar e relacionar tais dados, caracterizar as diferenças estatísticas entre as amostras obtidas de cada modelo, testar nosso mecanismo em ambos os ambientes, real e simulado, verificar sua eficiência e finalmente confrontar nosso mecanismo com outra técnica de sincronização. Observe que a motivação por trás da condução de experimentos em ambiente real e simulado é aproveitar as qualidades de ambos os métodos. Simulações são completamente reproduzíveis e muito flexíveis, sendo possível extrair informações sobre cada evento da rede. Por outro lado, o ambiente real fornece toda a complexidade dos sistemas de comunicação sem fio, sendo possível acreditar que seus comportamentos não são um resultado de modelagem e/ou simulação. A seguir detalharemos os dispositivos utilizados as ferramentas e aplicações.

2.2 *Motes* e outros dispositivos

Segundo [Estrin *et al*. 1999] redes de sensores sem fio (RSSF) são um emergente caso particular da área de aplicação para redes *ad hoc*, e a ideia é que essa rede seja uma coleção de dispositivos minúsculos (*motes*), de baixo

custo de fabricação, capazes de sensoriar, coordenar atividades e transmitir algumas características físicas sobre o meio ambiente para uma estação de base associada.

Abaixo descreveremos tais dispositivos.

Nós sensores ou *motes*: são os componentes básicos de uma RSSF. Possuem alguma capacidade de comunicação e uma pequena memória não volátil programável. Um *mote* consiste basicamente de um micro controlador, um rádio, uma fonte de energia, uma memória e pode conter também alguns sensores, como por exemplo: sensores de temperatura, acelerômetro *etc*.

Nas Figuras 1a e 1b, podemos ver as imagens dos dois tipos de nós sensores utilizados em nossos experimentos e na Figura 2, um diagrama de blocos detalhando os componentes de hardware principais destes *motes*.

Figura 1a. *Mote* modelo MicaZ

Figura 1b. *Mote* modelo Iris

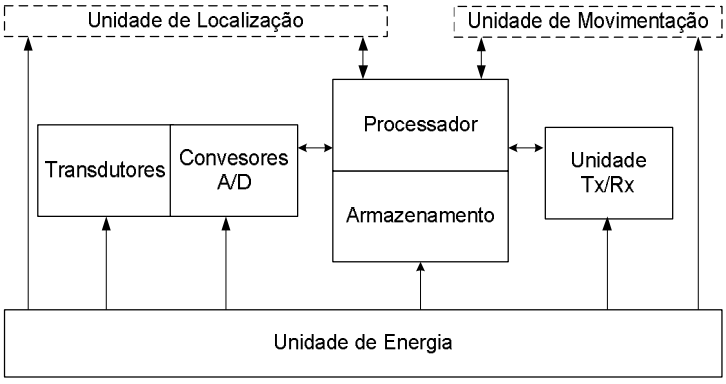

Figura 2. Componentes básicos de hardware de um nó sensor

Em nosso trabalho usamos os modelos de *motes* disponíveis em nosso laboratório: o MicaZ [Crossbow 2005a] e o Iris [Crossbow 2007]. Ambos possuem um rádio que opera nas frequências de 2,4 a 2,483 GHz compatíveis com o padrão IEEE 802.15.4 e necessitam de uma placa de programação.

Placa de programação: também conhecida como *gateway board* (Figura 3), pode prover entre outras interfaces, *ethernet*, USB e serial que conecta o *mote* a uma rede já infraestruturada ou a um *desktop*. A placa de programação usada foi a MIB520 [Crossbow 2005b] que possui apenas uma interface USB para conexão ao PC, que permite *upload* de programas ou captura de dados para processamento.

Figura3. Placa de programação MIB520

Capturador de pacotes: este dispositivo foi usado, em nossos experimentos, além de sua função primordial, de observar a troca de quadros, para garantir um referencial de tempo global.

Durante nossos testes, usamos o capturador de pacotes CC2531 da *Texas Instruments*, apresentado na Figura 4, e o software *SmartRF*® também da *Texas Instruments*, para a visualização dos conteúdos dos quadros. Este programa possui a funcionalidade de armazenar os pacotes capturados em um formato nativo da *Texas Instruments*, que pode eventualmente ser convertido em formato PCAP ou ASCII para uma análise mais detalhada.O aplicativo *SmartRF*® oferece ainda uma tela de visualização da linha de tempo do experimento, que facilita, entre outras coisas, o reconhecimento imediato da ocorrência de erros para futura depuração. Na Figura 5,

podemos ver o formato dos pacotes e a linha de tempo mostrando a troca de quadros entre os *motes*.

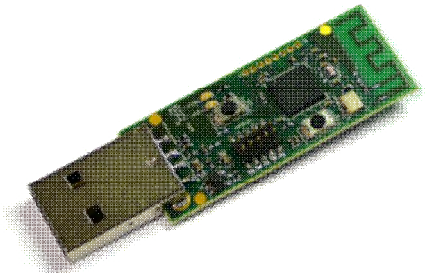

Figura 4. CC2531 USB*Dongle*

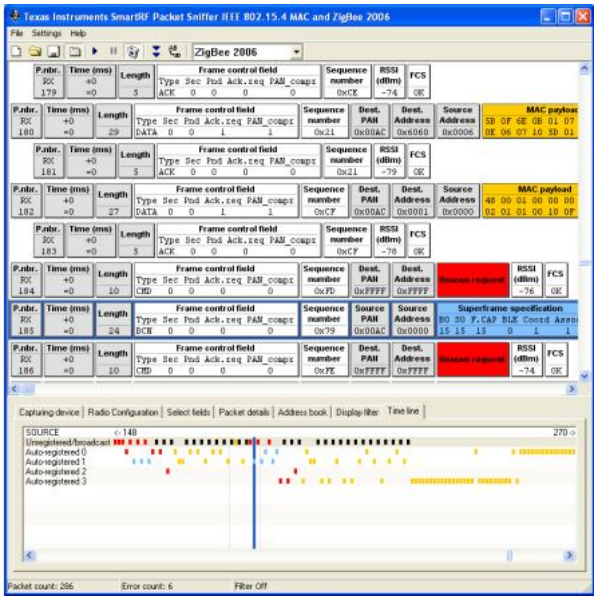

Figura 5. Tela SmartRF IEEE802.15.4 / ZigBee Packet Sniffer.

2.3 O Sistema operacional dos *motes*

Além das plataformas de *hardware*, muitas plataformas de *software* têm sido desenvolvidas especificamente para RSSF. Entre elas a mais difundida é o TinyOS [Levis *et al.* 2005]. Um sistema operacional de código aberto orientado a eventos e desenhado especificamente para operar sob os escassos recursos de memória, processamento e energia, que um *mote* oferece. Esses recursos e suas restrições serão apresentados em detalhes no capítulo três.

O TinyOS utiliza uma arquitetura que minimiza o tamanho do código respeitando os poucos recursos de memória e CPU. Sua biblioteca de componentes inclui protocolos de rede e *drivers* dos sensores, além de possuir seu próprio protocolo de acesso ao meio. Associado intrinsecamente ao TinyOS está o TOSSIM, o simulador que o acompanha e que será detalhado na subseção 2.4.

2.4 Ferramentas de simulação

Ferramentas de simulação há muito vem sendo largamente utilizadas em redes de computadores e são também necessárias em redes de sensores, especialmente pelo fato de que ainda há pouquíssimas RSSF operacionais e disponíveis para experimentação. De fato, por este motivo, tem surgido uma profusão [Ranganathan and Nygard 2010] de simuladores no mercado, e todos prometem uma modelagem acurada da operação de uma rede de sensores, principalmente no que toca redes multissaltos e simulação com o gasto de energia, entre eles o **NS-2** ("The Network Simulator, NS–2"), o **TOSSIM** [Levis et al. 2003], o **WSIM** [Fraboulet et al. 2007], o **Avrora** [Titzer et al. 2005] e o **COOJA** (Contiki OS as Java) [Osterlind et al. 2006].

Nosso primeiro desafio foi escolher uma ferramenta, entre tantas, que se ajustasse bem a nossa proposta de caracterização e estimativa do atraso.Assim,em virtude do grande número de ferramentas de simulação, procuramos nos concentrar somente na avaliação das mais citadas entre os trabalhos relacionados.

Analisamos primeiro o NS-2, que tem sido um sucesso em modelagem de redes de computadores em meio confinado e que possui uma grande quantidade de extensões para RSSF, mas além dele não possuir um bom modelo para o consumo de energia, o código escrito em NS-2 não poderia ser reaproveitado para os testes em *motes* reais, tendo por isso sido descartado.

O segundo simulador analisado foi o WSIM. Esta ferramenta permite a simulação, usando o mesmo código objeto executado no sensor e é ótimo para propósitos de depuração e/ou para quando não é necessário o uso do

rádio, no entanto, durante os primeiros testes, os atrasos médios obtidos ficaram da ordem do dobro dos atrasos obtidos em sensores reais.Além disso, o comportamento da distribuição estatística desse atrasos e mostrou muito díspar da distribuição do atraso dos dados das amostras com *motes* reais, como podemos observar no histograma sobreposto da Figura 6 e na Tabela 1.

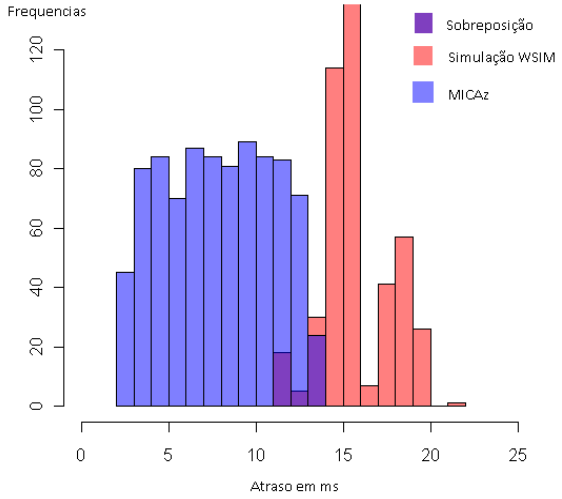

Figura 6. Sobreposição de histogramas dos atrasos em *mote* reais Micaz e no WSIM.

Tabela 1. Comparações entre parâmetros estatísticos em *motes* reais e no WSIM com intervalo de confiança de 0,95

	Micaz(ms)	WSIM(ms)
Valor mínimo	2,22	11,06
Valor máximo	11.9	21,94
Intervalo	9,62	10,88
Mediana	6,98	15,44
Média	7,88	15,79
Erro Padrão	0,13	0,09
Variância	7,5	7,77
Desvio padrão	2,73	1,93
Curtose	-1,20	-0,297
Assimetria	0,024	0,274
Coeficiente de Variação	0,39	0,122

Outro simulador analisado foi o TOSSIM, que implementa o padrão 802.15.4. A sua abordagem é a de simular a execução de uma aplicação real, através de *scripts* construídos em Python [Python Software Foundation 2000], mas sem o retrabalho de reescrita do código, como no NS-2, facilitando, desta forma, os testes e a depuração. Um fator limitante sobre o TOSSIM é que ele não permite simulações para sensores modelo Iris, tendo sido concebido para simular apenas o funcionamento dos *motes* Mica de Berkeley.Outra limitação encontrada foi que o TOSSIM apesar de emular o hardware real do MicaZ, mapeando interrupções de hardware a eventos discretos, não possui um modelo realista para simulação do consumo de energia. Contudo,o TOSSIM e o Mica vem se tornando a plataforma de testes *de facto* para RSSF [Alizai *et al.* 2008]e diversas aplicações periféricas vem sendo construídas para facilitar e popularizar o uso deste simulador. No entanto,o fator preponderante para a sua escolha foi que a distribuição dos atrasos obtidos em suas simulações foi consistente com as encontradas em medidas com *motes* reais,tendo sido necessário apenas alguns ajustes para que as medidas alcançassem, em valores médios, níveis que exprimissem os tempos reais de *back off* e, consequentemente um comportamento final, similar à distribuição estatística da latência que queremos observar.Na subseção 5.6 detalharemos tais ajustes.

O quarto e último simulador analisado foi o **Avrora.** Diversas das suas características chaves foram preponderantes para sua escolha:

- Sua acurácia e precisão na simulação dos eventos;
- Implementa o hardware do MicaZ;
- Usa um modelo bastante realista do consumo de energia [Haas *et al*. 2012];
- É capaz de manter um *clock* local diferente e isolado para cada nó;
- É capaz de emular os programas compilados em TinyOS rodando diretamente o código objeto;

- Possui uma grande granularidade em seus diversos monitores de consumo (gasto de energia com CPU, gasto de energia no uso do rádio *etc*.).

Além das acima mencionadas,outras facilidades oferecidas pelo Avrora são:a simulação uma rede de sensores executando códigos diferentes em cada nó e a possibilidade de grande escalabilidade, que permite simular redes com até 10.000 nós [Sundani 2011].Suas maiores limitações para o nosso trabalho são:

- Não conseguir simular o escorregamento de relógio;
- Não modelar o comportamento do atraso com uma distribuição de que espelhasse a distribuição encontrada em *motes* reais.

E embora o Avrora não tenha se ajustado **perfeitamente** as nossas necessidades para simulações multissalto e na estimativa do atraso devido, principalmente, a grande variância do erro da estimativa, quando comparada aos motes reais, melhor detalhada no capítulo cinco. Ele nos foi muito útil nas avaliações com o consumo de energia. Por último, os dados apurados nas simulações e nos experimentos e foram tratados e tabulados através de *scripts* montados com os aplicativos *sed* e *awk*.

2.5 A ferramenta estatística

Utilizamos como ferramenta de análise e compilação e plotagem das amostras obtidas o *"R"* [R Development Core Team 2007]. O "*R"* consiste de um arcabouço de computação estatística e construção de gráficos, de código aberto, compatível com diversos sistemas operacionais e que permite a importação e exportação de dados para diversos formatos e interfaces diferentes.

Os itens analisados para a escolha dessa ferramenta foram:

- Farta coleção de bibliotecas;

- Interface de linha de comando,que confere à manipulação de dados facilidade e velocidade que supera os concorrentes com interfaces gráficas;
- A facilidade na montagem de análises bivariadas e na repetição e comparação dessas análises de com diferentes amostras;
- A rapidez e precisão na plotagem de gráficos;
- A possibilidade de, através de poucos comandos, sobrepor curvas,mantendo suas escalas originais;
- A vasta biblioteca de testes de hipóteses que permite a análise do comportamento de suas distribuições, parte fundamental neste trabalho;
- Documentação farta e acessível.

Capítulo 3 - Desafios da sincronização

Dentre os principais desafios enfrentados pelos protocolos de sincronização, alguns fatores que afetam a correção dos *offsets* são mais preponderantes que outros.Nas seções 3.1 a 3.4, conceituaremos quatro deles: a baixa qualidade dos relógios internos dos *motes*, os obstáculos inerentes à comunicação sem fio, os escassos recursos de CPU, memória e bateria destes dispositivos e a alta probabilidade de falhas nos nós.

3.1 Relógios de baixo custo

Nós em redes de sensores são tipicamente fabricados com materiais de baixo custo, especialmente os cristais dos seus relógios, que são imprecisos.Esta imprecisão, evidenciada pelas diferentes frequências destes cristais resulta no que chamamos de escorregamento de relógio e *jitter.* Influências que explicaremos em seguida. A Figura 7, por exemplo, mostra como a frequência do relógio de um *mote* mica2 se comporta em função do aumento da temperatura. Adiante explicaremos também como o escorregamento e o *offset* afetam a sincronização em redes de sensores.

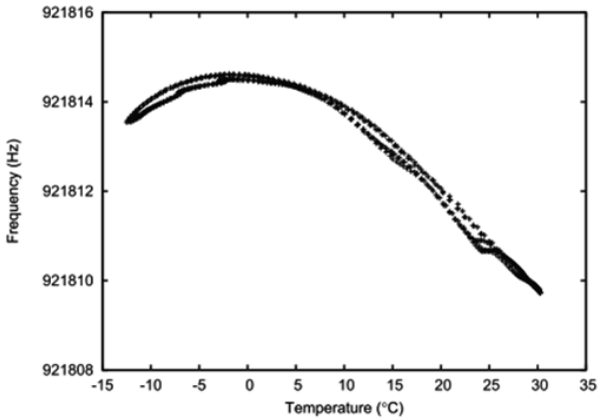

Figura7. Comportamento do escorregamento nos *motes* MicaZ [Lenzen et al. 2009].

O relógio de cada sensor *i* representa o tempo como um valor $\tau_i(t)$ em cada instante, como uma função do tempo *t*, considerando esse tempo *t* como o tempo de referência, mostrada na equação (3.1).

$$\tau_i(t) = \alpha_i(t)t + \theta_i(t) \quad (3.1)$$

Sendo $\alpha_i(t)$ o escorregamento do relógio e $\theta_i(t)$ o *offset* do relógio. Em um relógio perfeito $\alpha_i(t)=1$ e $\theta_i(t)=0$. Em redes de sensores, entretanto, estes cristais introduzem tanto o escorregamento quanto o *offset*, *i.e.*, $\alpha_i(t)\neq 1$ *e* $\theta_i(t) \neq 0$. Então, como o escorregamento e o *jitter* podem ser diferentes para cada sensor, os nós eventualmente perderão o sincronismo. Além disso, o relógio local de um nó *i* também pode variar em relação a um nó ***j.*** Esta variação é dada pela equação (3.2).

$$\tau_i(t) = \alpha_{ij'}(t)\ \tau_j(t) + \theta_{ij}(t) \quad (3.2)$$

onde$\alpha_{ij}(t)$ *e* $\theta_{ij}(t)$ são respectivamente o escorregamento e o atraso relativos. Um protocolo de sincronização tenta corrigir estes fatores e consequentemente, prover uma referência de tempo comum, minimizando as diferenças entre os nós.

Através da troca de mensagens os relógios dos dois nós podem ser corrigidos, entretanto, uma única mensagem pode apenas corrigir seu $\theta_{ij}(t)$.Mas note que o escorregamento de cada nó é função da frequência de seu cristal,*i.e.* $\alpha_{ij}(t)\neq 0$**,**assim, estes relógios eventualmente virão a perder a sincronização. Desta forma, troca de mensagens periódicas ou estimativas deste escorregamento relativo devem ser usadas para correção. Já dissemos que a baixa qualidade cristal causa o escorregamento de relógio e esse fator introduz erros de até 40 PPM (parte por milhão) no pior caso [Akyildiz and Vuran 2010].Assim, mesmo a uma taxa de 1PPM, esse desvio causaria um erro de aproximadamente 90ms por dia, o que leva a uma frequente necessidade de novas sincronizações.

3.2 Comunicação sem fio

A sincronização requer que os nós se comuniquem uns com os outros para a troca de mensagens de referência de tempo. Enquanto em redes cabeadas essa comunicação é mais simples,em redes sem fio, maiores desafios se apresentam, resultados da alta taxa de erros de transmissão,de relevantes ocorrências de atrasos não determinísticos durante a transmissão e de diferentes relações sinal ruído nas proximidades dos nós, entre outros.

Devemos considerar, ainda, que a natureza da transmissão por difusão em redes sem fio exige algum tipo de protocolo MAC para a ordenação do uso do canal.Tudo isso introduz atrasos aleatórios, que formam as componentes do atraso total de transmissão: tempo entre o envio da mensagem pela aplicação e o momento em que finalmente a mensagem realmente ganha o meio.

Veremos adiante na subseção 4.4 que estes tempos aleatórios respondem por um percentual significativo da contabilização total das fontes de atraso. E que sua característica aleatória, se não pode ser determinada, precisa ser estimada e compensada. Finalmente, a assimetria dos enlaces sem fio é ainda mais acentuada em tecnologias que se utilizam de transmissores de baixa potência, como as RSSF. Esta assimetria introduz mais incertezas, pois o tempo de transmissão pode variar devido a erros no canal e variações das condições do meio nos arredores de cada nó.

Em redes cabeadas, a maioria das soluções de sincronização assume que o RTT (*Round-Trip Time*) é aproximadamente duas vezes o atraso de transmissão em uma direção e essa suposição não é verdadeira em redes sem fio.

3.3 Escassez de recursos.

Os nós de uma RSSF operam sob restrições de energia muito rigorosas e seus recursos para processamento e memória são igualmente muito escassos. Assim, muitas abordagens usadas em redes cabeadas não podem

ser empregadas naquelas redes. Por exemplo,transmitir e receber quadros são operações que consomem praticamente a mesma energia.Além disso, apenas manter o rádio ligado (*idle*) custa quase tanto como transmitir.Então, trocas de mensagens para sincronização devem ser reduzidas ao mínimo possível,ligando o rádio, idealmente, apenas quando houver necessidade real de transmissão ou recepção.

3.4 Falhas nos nós

O *hardware* de baixo custo dos *motes* e sua limitada fonte de energia os leva frequentemente a falhas. Consequentemente as soluções de sincronização que dependem de um único nó, somente devem ser aplicadas a casos específicos onde o nó raiz esteja menos sujeito a defeitos. Soluções distribuídas ou robustas que não são afetadas por tais falhas, são mais adequadas.

Introduziremos no capítulo quatro diversas abordagens que têm sido adotadas para criação de protocolos de sincronização. Todas elas propõem algumas soluções para as restrições mencionadas.

Capítulo 4 – Trabalhos relacionados

Muitos métodos de sincronização têm sido propostos e usados ao longo dos anos, mas virtualmente todos eles compartilham o mesmo modelo: um servidor envia uma ou mais mensagens para o cliente, que por sua vez, calcula o *offset* e ato contínuo sincroniza seu relógio. Contudo, em RSSF, com poucos recursos de energia, mais sujeitas a interferências, com enlaces assimétricos e mudanças abruptas da relação sinal ruído, certas estratégias que usam tráfego intensivo de quadros se tornam impraticáveis.

O protocolo mais largamente utilizado para sincronização na Internet é o NTP (*Network Time Protocol*). Nele,os clientes sincronizam seus relógios com servidores NTP, alcançando uma precisão da ordem de milissegundos calculando o *offset* através da determinação estatística do RTT (*Round-Trip time*). Os servidores, por sua vez, mantêm seus relógios sincronizados através de fontes externas, sejam receptores de GPS (*Global Positioning System*) ou relógios atômicos, mas em RSSF o uso deste protocolo se torna proibitivo por diversos motivos:

1. O não determinismo, inerente aos meios não confinados;
2. O uso, mesmo moderado, de CPU, consome muita energia;
3. Muitas tentativas de acesso ao meio e transmissões eventuais também são custosos energeticamente.

Além disso, esse protocolo assume também que os enlaces dessas redes devam ser simétricos para estimativa correta do RTT, situação nem sempre verdadeira em RSSF.

O *Precision Time Protocol* (PTP), apesar de mais preciso que o NTP, também não se mostra apropriado para RSSF. E uma das razões é que os quadros PTP são geralmente mais longos que os quadros da camada MAC do

protocolo IEEE 802.15.4(IEEE 802.15.4 2007), padrão comumente empregado nos módulos de rádio dos nós sensores em RSSF. Também vale ressaltar que o PTP foi concebido para operar em LANs (*Local Area Networks*) devido ao maior determinismo de seus enlaces. Neste protocolo, a necessidade de troca de três mensagens para a sincronização inicial, somada a frequência das mensagens *Master Sync Message*, enviadas pelo nó Mestre, e *Master Delay Response*, enviadas pelo nó Escravo, tornam o método muito dispendioso em energia para RSSF.

4.1 A sincronização em RSSF

A maneira tradicional de tratar o problema da sincronização em redes cabeadas não se ajusta bem as redes sem fio, devido às várias razões já enumeradas nas seções anteriores. RSSF exigem dos novos protocolos uma abordagem diferente, que resolva os problemas do consumo de energia,do excesso de *overhead* durante a sincronização, da baixa largura de banda e dos limitados recursos de hardware. Na próxima subseção vamos examinar exemplos de protocolos de sincronismo propostos para redes de sensores.

4.2 RBS – Reference Broadcast Synchronization

Alguns protocolos de sincronização em redes de sensores são classificados como par-a-par: um exemplo é o RBS (*Reference-Broadcast Synchronization*) [Elson et al. 2002]. Nesse tipo de mecanismo de sincronização, qualquer nó pode se comunicar com outros nós da rede, isto é, ele não forma pares específicos para a sincronização. Essa conformação oferece mais flexibilidade e elimina o ponto de falha da existência de um único nó mestre, porém exige uma gerência mais complexa para a convergência do sincronismo.

O RBS emprega um nó de referência usando difusão e um conjunto de receptores que trocam seus *offsets* através de *unicast,* assumindo que a contribuição não determinística para o atraso total é muito maior durante a transmissão que durante a recepção, pois no decorrer do envio, as

componentes não previsíveis, sejam de acesso ao meio ou do escalonamento da CPU, somadas, podem chegar a 600ms (ver Tabela 2), enquanto o tempo de recepção é bem menor e não inclui o acesso ao meio, o *overhead* de chamadas ao sistema ou trocas de contexto [Elson *et al.* 2002].

O algoritmo deste protocolo opera em um regime de sincronização conhecido como *post-facto*, neste tipo de algoritmo dois relógios referenciam-se a um evento no passado, e só então calculam os desvios de seus relógios e ajustam o *timestamp* associado a esse evento, **nunca** corrigindo o relógio local.Ao invés disso, cada nó armazena localmente uma tabela contendo o desvio relativo a cada par dentro de seu alcance. Essa tabela é atualizada a cada mensagem recebida de outro **nó receptor**, dependendo de seu número de sequência e do *offset* já calculado.Note que essa abordagem mais simples funciona bem para situações de apenas um salto. Para situações de múltiplos saltos, o RBS precisará de nós que possam lidar com duas ou mais referências de emissores diferentes, chamados *translation nodes*, esses nós tradutores são usados para traduzir o tempo entre domínios de difusão diferentes.

Na prática, no entanto, mensagens em difusão em ambiente densos, possuem maior probabilidade de serem corrompidas e perdidas. Mais ainda, os nós receptores nem sempre estarão ociosos e prontos para gravar a referência,poderão eventualmente estar ocupados transmitindo quadros. Para mitigar estes fatores negativos, o protocolo RBS precisa de uma **sequência de mensagens** de referência, ao invés de apenas uma.

Outro problema a ser resolvido é o escorregamento de relógio. Por isso, o RBS emprega um método para estimativa destes deslizamentos. O mecanismo se baseia no emprego dos mínimos quadrados para ajustar a variações das defasagens. Essa técnica assume que os erros destes relógios devem variar em uma taxa constante ao longo do tempo.

As vantagens desta abordagem são:

- Eliminar as fontes não determinísticas no **transmissor**;

- Realizar os ajustes quando necessário, eliminando o gasto de energia caso o fizesse periodicamente;
- Devido a sua tabela de ajustes, mesmo com quadros perdidos é possível manter a coerência de tempo.

As desvantagens são:

- O excesso de troca de mensagens entre os nós receptores;
- Em redes com mais de um salto este protocolo requer $O(n^2)$ troca de mensagens;
- O tempo de convergência é longo devido ao grande número de mensagens;
- Não serve para aplicações que requerem uma referência absoluta de tempo.

4.3 TPSN – Timing-Sync Protocol for Sensor Networks

Um exemplo de protocolo de sincronização Mestre-Escravo é o **TPSN** [Ganeriwal *et al*. 2003].Ele utiliza um único nó (nó raiz) para sincronizar toda a rede e é dividido em duas fases:

Fase 1 - Descoberta de nível;

Fase 2 - Sincronização.

A meta da primeira fase é criar uma topologia hierárquica na rede, onde cada nó é associado a um nível. O nível 0 (zero) é associado somente a um nó, o nó raiz. Na segunda fase um nó de nível *i* sincroniza com um nó de nível *i-1* e ao final dessa sequência, todos os nós estarão sincronizados com o nó raiz.

O TPSN assume ainda que certos requisitos são necessários: um protocolo de nível de enlace deve garantir que cada nó esteja ciente do conjunto de nós com os quais ele pode se comunicar diretamente, também denominado como o "*neighbour set*" (conjunto de vizinhos). Apesar de ser possível a existência

de enlaces unidirecionais, o TPSN emprega apenas enlaces com conectividade bidirecional para garantir a sincronização dentro de um conjunto de nós, sendo também possível criar árvores de dispersão (*spanning tree*) para evitar *loops*.

A abordagem deste protocolo também é a de minimizar as fontes de atraso no emissor. Mas de modo diferente do RBS. Ele usa uma técnica de *timestamp* na camada MAC, vista em maiores detalhes na subseção 4.4. Isto é, o TPSN registra o tempo de saída, gravando o valor no quadro quando ele está prestes a ser enviado, restando agora, apenas o tempo de propagação, da ordem de nano segundos, que pode ser ignorado para a precisão requerida. Ele aplica essa mesma técnica também no receptor, garantindo a eliminação desta componente em ambos os lados,origem e destino.

No TPSN o escorregamento de relógio não é tratado, gerando uma necessidade de sincronização frequente.

As vantagens desta abordagem são:

- A estrutura hierárquica provê escalabilidade;
- O protocolo provê uma sincronização global;
- Alcança uma referência global em toda a rede como o NTP, mas com um custo energético muito menor.

As desvantagens são:

- Falhas e mobilidade nos nós geram a necessidade de reestruturação da árvore, ocasionando o aumento do consumo de energia;
- A necessidade de enlaces bidirecionais impede o uso de *broadcast*, resultando em uma carga maior no meio sem fio;
- Necessita de hardware específico para seu *timestamp* na camada MAC.

4.4 FTSP – Flooding Time Synchronization Protocol

Como já mencionado em seções anteriores os atrasos não determinísticos envolvidos na transmissão e recepção dos quadros, devem ser

caracterizados, eliminados e/ou compensados. A proposta do FTSP [Maróti *et al.* 2004] é a de conseguir a sincronização de uma RSSF com erro da ordem de microssegundos em redes de múltiplos saltos, sujeitas a rápidas mudanças de topologia e falhas de enlaces, com uma combinação das seguintes ações:

- Eliminação dos erros relevantes encontrados nas diversas fases de envio e recepção;
- Uso do conceito de *timestamp* na camada MAC;
- Compensação do escorregamento de relógios através de regressão linear.

A decomposição e a caracterização do atraso primeiro em [Kopetz *and* Ochsenreiter 1987], e depois aprofundado em [Maróti *et al.* 2004], mostram que as diversas fases desde o momento em que a aplicação inicia a transmissão em um *mote "A"* até a recepção final em *"B"*, apresentam tempos aleatórios cujas fontes devem ser analisadas, pois experimentalmente sabemos que a ordem desses tempos pode ser bem maior em magnitude que a precisão requerida pela aplicação. Acompanhe a seguir a discriminação de cada fase.

Tempo até o envio: tempo desde a montagem do quadro até a requisição de envio a camada MAC, que depende do *overhead* do sistema operacional e da carga da CPU. Este é um fator aleatório da ordem de centenas de milissegundos.

Tempo de acesso ao meio: atraso devido ao tempo de espera para que o meio esteja livre, até o início da transmissão efetiva do quadro. Pode variar de milissegundos a segundos, dependendo do tráfego nas proximidades daquele nó. Sendo a parte menos determinística do processo.

Tempo de codificação: tempo que o rádio leva para codificar a mensagem transformando os bits em ondas eletromagnéticas. Esse tempo é determinístico e da ordem de centenas de microssegundos.

Tempo para transmissão: tempo que o *mote* leva para enviar todo o quadro. Depende do comprimento do quadro e da taxa alcançada pelo rádio.

Tabela 2.Caracterização das componentes de tempo do atraso de transmissão.

Tempo	Magnitude	Distribuição
Tempo até o envio	0 – 100ms	Não determinístico, depende da carga do processador.
Tempo de acesso ao meio	0 – 500ms	Não determinístico, depende da contenção no canal.
Tempo de transmissão e recepção	10 – 20ms	Determinístico, depende do comprimento da mensagem.
Tempo de propagação	< 1μs para distância de até 300m	Determinístico, depende da distância entre receptor e emissor.
Tempo para lidar com as interrupções	< 5μs na maioria dos casos, mas pode chegar a 30μs	Não determinístico, depende se as interrupções estão sendo desabilitadas.
Tempo de codificação decodificação	100 – 200μs, com variância < 2μs	Determinístico, depende do *chip set* do rádio.
Alinhamento de *Bytes*	0 – 400μs	Determinístico e pode ser calculado.

Tempo de propagação: tempo que o quadro leva para se propagar do transmissor ao receptor, é determinístico e depende da distância entre ambos. Da ordem de nanossegundos.

Tempo de alinhamento de byte: é o atraso devido à diferença de alinhamento de bytes entre transmissor e receptor. É determinístico e pode ser calculado no lado do receptor a partir do *bit* de *offset* e da taxa do rádio.

Tempo para recepção: intervalo de tempo que o receptor leva para receber toda a mensagem, é equivalente ao tempo para a transmissão. Note que esse tempo e o anterior se sobrepõem parcialmente.

Tempo de decodificação: tempo que o receptor decodifica a mensagem de ondas eletromagnéticas para bits. Termina quando o rádio interrompe o processador avisando que completou a recepção. Este tempo é na maioria

das vezes determinístico, mas flutuações na potência do sinal e sincronização dos *bits* podem introduzir algum *jitter*.

Tempo de tratamento das interrupções: É o tempo entre o rádio interromper o micro controlador e obter a resposta. Leva poucos microssegundos, mas se as interrupções estiverem desabilitadas, poderá ser bem maior.

Tempo para processar a mensagem: no receptor, tempo para tratar a mensagem recebida e notificar a aplicação.

É importante mencionar que a última geração de rádios usados em nós sensores adicionou mais recursos à interface rádio/processador sem, por isso, sobrecarregar o processador com o *overhead* de coordenar a transmissão. Entre esses melhoramentos, a interrupção SFD (*Start Frame Delimiter*) do CC2420 [Moss et al. 2007]possibilitou uma tomada de tempo mais precisa e granular, evento fundamental para o funcionamento desse protocolo.

O FTSP assim como o TPSN se utiliza da existência dessa interrupção em alguns modelos de motes para tornar seu método mais preciso. Além disso,usa regressão linear para gerência do escorregamento de relógios e elimina ou compensa as outras fontes de atraso conseguindo atingir uma precisão,segundo os autores, da ordem de 1µs. Na subseção 4.4.1 detalharemos melhor o uso desta interrupção.

4.4.1 Marca de tempo (*timestamp*)

O FTSP usa difusão para sincronizar os múltiplos receptores. Uma mensagem em broadcast contendo a marca de tempo do emissor é enviada e os receptores gravam seu tempo local no momento da recepção, assim, o mecanismo provê um ponto de referência para cada um dos receptores. Procuramos mostrar na Figura 8, o momento da adição do *timestamp*.

A colocação da marca de tempo nesse momento também reduz o *jitter* causado pelo tratamento das interrupções. O *timestamp* é feito em cada

limite de *byte* após os *bytes* de sincronização (*SYNC*), durante o envio e a recepção.

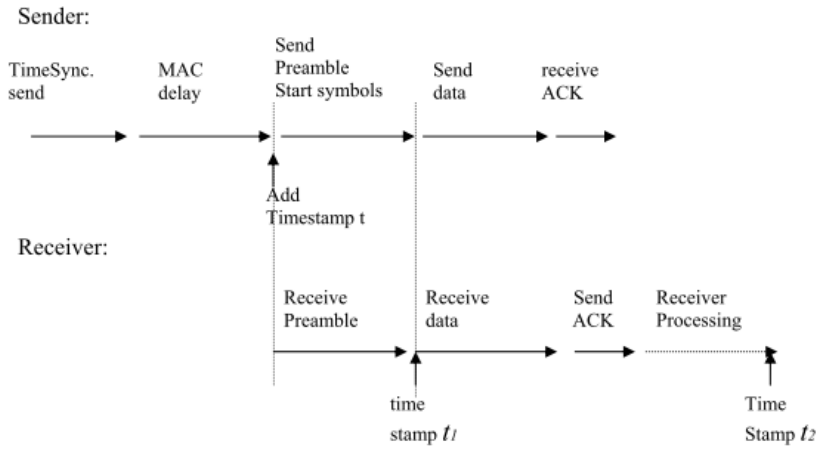

Figura 8. Momento de adição do *timestamp*. [Ping 2003]

4.4.2 Escorregamento de relógio

Como mencionado, se todos os relógios tivessem a mesma frequência, somente uma sincronização seria necessária para ajustar dois *motes*, mas os relógios dos nós, em redes de sensores, são muito imprecisos. Como visto na Figura9,o *offset* entre dois *motes* cresce linearmente com o tempo, sendo um problema a ser tratado.Abaixo descrevemos brevemente o método do FTSP para estimar o escorregamento local.

Consideremos o *mote* "*A"* como nó raiz e o *mote* "*B"* como nó buscando a sincronização dentro do alcance de "*A"*. "*A"* envia mensagens de sincronização para o *mote* "*B"* dentro de um período *T*. "*B"* então calcula seu *offset* e estima o escorregamento local usando as oito últimas entradas de uma tabela em uma de regressão linear.

Os parâmetros mais relevantes deste método são a precisão requerida e o intervalo de resincronização.

4.4.3 Sincronização multissalto

RSSFs costumam ter um raio não alcançável em apenas um salto. Para resolver esse problema, o protocolo cria o conceito de *reference points*, que servem para estabelecer uma referência de sincronismo. Estes pontos de referência contém um par de marcas de tempo global e local. Um nó que está dentro do alcance do nó raiz pode estabelecer esse par com o próprio nó raiz e os nós fora desse raio podem fazê-lo indiretamente pelo intermédio de outros nós já sincronizados, através das mensagens de sincronização mencionadas na subseção 4.4.2.

A mensagem de sincronização contém três campos:

- *Timestamp* – contém o tempo global do transmissor quando da difusão da mensagem;
- *RootId* – contém o *Id* do nó raiz;
- *SeqNum* – número de sequência que é incrementado pelo nó raiz quando uma rodada de sincronização se inicia.

Como as mensagens de sincronização não são enviadas periodicamente, devemos esperar, em uma rede densa, que muitas dessas mensagens de diferentes nós circulem em um curto espaço de tempo e que nem todas tenha um conteúdo relevante.Por exemplo: mensagens com S*eqNum* baixo. Para ajudar na filtragem das mensagens, cada nó mantém uma variável *highestSeqNum* e outra chamada *myRootID*. Assim, uma mensagem é recebida apenas se o conteúdo do campo *seqNum* é maior que *highestSeqNum* e quando *rootID=myRootID*. Isso garante que apenas a primeira mensagem recebida daquela rodada será usada.

A eleição do nó raiz: O FTSP difunde sua referência de tempo a partir de um único nó, mas redes de sensores são sujeitas a falhas e este nó pode perder conectividade. Para suprir esse problema o protocolo conta com um mecanismo de eleição, baseado no pré-requisito de que cada nó deve possuir um identificador único.

Quando um nó fica sem receber novas mensagens de sincronização por um período, cuja variável é chamada *ROOT_TIMEOUT*, ele se auto designa como nó raiz. Desta forma, depois de um período *ROOT_TIMEOUT* haverá pelo menos um nó raiz, podendo haver múltiplos. Quando então um nó recebe a mensagem de sincronização com o campo *rootID* menor que o seu *myRootID* ele atualiza o valor dessa variável para o menor. Este método garante que os nós com *rootID* mais altos vão seguidamente desistindo de sua situação e eventualmente ao final do processo restará somente um nó raiz.

Durante a etapa de eleição o protocolo não deve descuidar do sincronismo já alcançado. Deste modo, apenas os nós considerados já sincronizados, ou seja, aqueles que têm suficientes entradas em sua tabela de regressão transmitem suas mensagens de sincronização, pois, se um nó recebe um novo ponto de referência que esteja em desacordo com seu tempo global, ele limpa sua tabela de regressão.

Outro problema que deve ser resolvido pelo mecanismo é a introdução de um novo nó com *ID* inferior ao *rootID*. Nesse caso este novo nó se declarará como nó raiz e essa situação deve ser tratada. Para isso, o novo nó não começa transmitindo as mensagens de sincronização. Até que ele colete os pontos de referência, preencha sua tabela de regressão e calcule seu *offset* não se considerará sincronizado. Só então, após a fase descrita, ele iniciará a difusão das mensagens de sincronização, agora já auto designado como nó raiz.

As vantagens desse protocolo são:

- É aplicável a redes sujeitas a falhas ou mudanças de topologia;
- Possui rápida convergência;
- Trata o escorregamento de relógio.

As desvantagens são:

- Em redes multissaltos os erros são propagados exponencialmente [Lenzen et al. 2009]

- A falta de coordenação na transmissão das mensagens de sincronização causa colisão e perda de quadros.
- Necessita de hardware específico para seu *timestamp* na camada MAC.

4.5 PulseSync

Os autores deste protocolo argumentam que o limite teoricamente já estabelecido do escorregamento do relógio entre um par de nós é *Ω(D)*, sendo *D* o diâmetro da rede, é na realidade baseado em assunções que não descrevem o comportamento real da sincronização entre os relógios dos nós em uma RSSF. Argumentam ainda que ambos, o comportamento aleatório, tanto da variação do *jitter,* quanto do escorregamento dos relógios, não variam arbitrariamente, ao menos, não arbitrariamente rápido,como podemos verificar na figura 9.

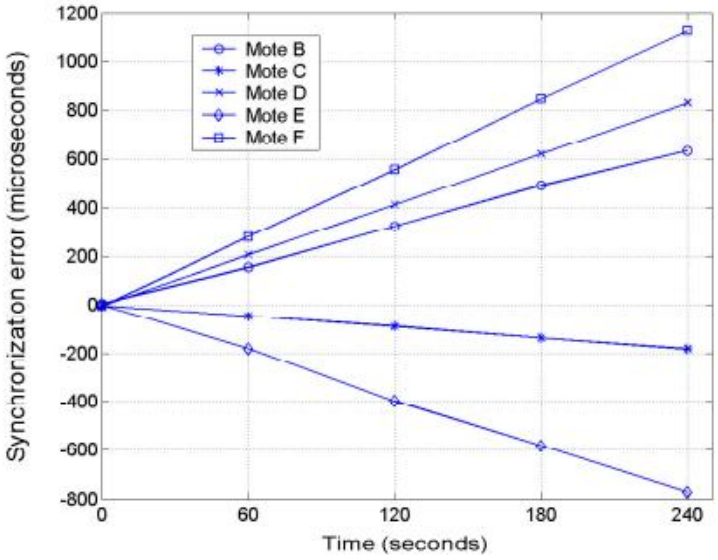

Figura 9. Escorregamento de relógio em relação a um *mote "A"*. [Lenzen et al. 2009]

No PulseSync [Lenzen *et al*. 2009], os autores promoveram uma análise do protocolo FTSP e afirmam ter constatado que este protocolo apresenta um erro de sincronização que cresce exponencialmente com diâmetro da rede (ver Figura 10). A proposta do PulseSync é corrigir essas deficiências, através

de um algoritmo assintoticamente ótimo. Ele funciona propagando um pulso com a referência de tempo, a partir de um nó raiz, o mais rapidamente possível através da rede, coordenando as transmissões dos nós. Assim, os autores afirmam alcançar com uma alta probabilidade, uma diferença entre os relógios como $\Omega(\sqrt{D})$.

Os resultados com *motes* reais mostram que em uma rede de 20 nós, a implementação de protótipo do PulseSync mostra-se cinco vezes mais precisa que o FTSP. Testes em ambientes simulados mostram que para redes maiores, o PulseSync oferece uma precisão que é várias ordens de magnitude melhor que o FTSP. No entanto, apesar desse mecanismo apresentar muitas vantagens em relação aos já mencionados, é ainda dependente do *timestamp* na camada MAC e essa facilidade não está presente universalmente em todos os sensores.

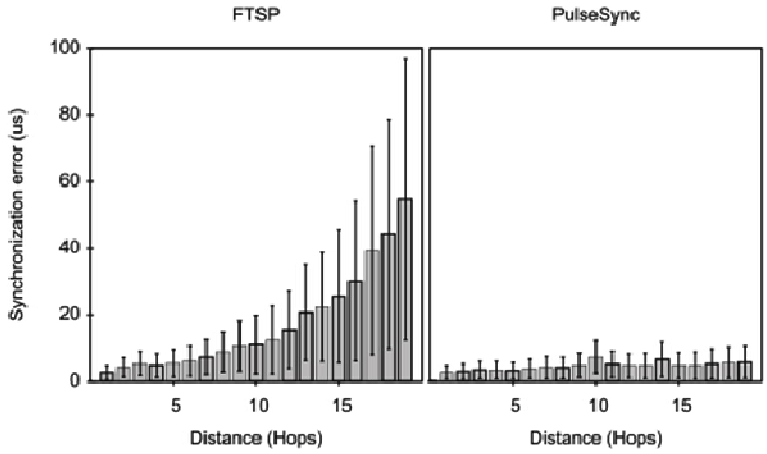

Figura 10. Comparação do erro de sincronização entre o FTSP e PulseSync em uma RSSF *multihop*. [Lenzen *et al.* 2009]

As vantagens desse protocolo são:

- Rápida convergência;
- Não propaga os erros de *offset* e escorregamento em redes *multihop*.

E a desvantagem:

- Necessita de hardware específico para seu *timestamp* na camada MAC.

Na próxima seção é apresentada uma técnica simples de estimativa local (STELE) que elimina a necessidade de comunicação bidirecional e não requer o uso de hardware especializado.

Capítulo 5 – *STELE* – Uma Técnica Simples Para a Estimativa do Atraso Local em RSSF

5.1 Desafio

O nosso principal desafio no início desse trabalho foi o de conceber um mecanismo que

1. Fosse simples e de fácil implementação;
2. Não dependesse de características especiais desse ou daquele modelo de dispositivo;
3. Não exigisse muitas trocas de mensagens,
4. Não fosse custoso em termos de computação, memória ou uso do rádio
5. E atingisse um nível de sincronização da ordem de dezenas de microssegundos. O que para os relógios dos dispositivos disponíveis, com relógios de 32kHz é muito próximo do seu limite da precisão.

5.2 O mecanismo

Em uma troca de mensagens entre dois nós, um quadro enviado em um tempo T_0 alcançará um determinado vizinho em um tempo T_0+T_X, onde T_X é o atraso na transmissão. Tal atraso é uma variável aleatória que tipicamente apresenta grande variância. Se o quadro em questão contém informações de sincronismo é necessário, então, eliminar T_X. O nosso método se propõe a eliminá-lo estimando e compensando seu valor.

A técnica proposta se baseia no fato do transmissor ser capaz de determinar o momento em que a transmissão de um quadro é concluída, e que este tempo pode ser usado para estimar o momento em que o receptor receberá o quadro. Desse modo as componentes de tempo relativas ao transmissor são removidas, sobrando apenas o tempo de propagação, da ordem de nano segundos e as componentes de tempo envolvidas no processo de recepção e estas últimas apresentam valores e variâncias menores quando comparadas ao processo de transmissão, já que na recepção não há contenção [Elson *et al.* 2002].

A Figura 11ilustra o conceito da estimativa local do atraso sob o ponto de vista do nosso mecanismo, onde T_X consiste no atraso real de transmissão, que é o tempo desde que a aplicação no *mote 'A'* envia uma mensagem até a recepção final pelo *mote 'B'* e T_E é a estimativa desse atraso, calculado localmente, no nó emissor, ou seja, é o tempo entre o envio de uma mensagem pela aplicação do *mote 'A'* até o tempo em que o rádio deste *mote* notifique à CPU o seu envio. Em seguida explicaremos em detalhes o passo-a-passo do mecanismo:

1. Em um instantet_0 a aplicação que roda em '*A'* comanda o envio de um quadro para o mote '*B';*
2. Em um instante t_1 a CPU finalmente envia o quadro para o rádio;
3. No instante t_2 o quadro finalmente ganha o meio;
4. Em t_3 a notificação do envio é enviada de volta a CPU;
5. E em t_4 a aplicação computa esta notificação.

No instante t_4 a aplicação já possui condições de computar a estimativa T_E que é expresso por t_4-t_0.

Observe que todas as fases relevantes para a estimativa ocorrem em *"A"*. E essa estimativa será boa o suficiente quanto ela se aproxime do atraso real. E terminando podemos afirmar que a qualidade desta estimativa se baseia em quatro fatores:

- A forte correlação estatística, que pretendemos demonstrar, entre a grandeza T_E e o atraso T_X;
- A nossa estimativa será tão acurada quanto média da diferença T_X-T_E for próxima de zero e tão precisa quanto menor for sua variância;
- O tempo estimado T_E, relativo ao intervalo entre o envio da mensagem para o rádio e o momento real de seu envio, inclui a maior parte do valor esperado para o atraso real T_X;
- O tempo de recepção é proporcionalmente pequeno e também menos aleatório, podendo ser compensado pela estimativa, sem prejuízo para precisão almejada.

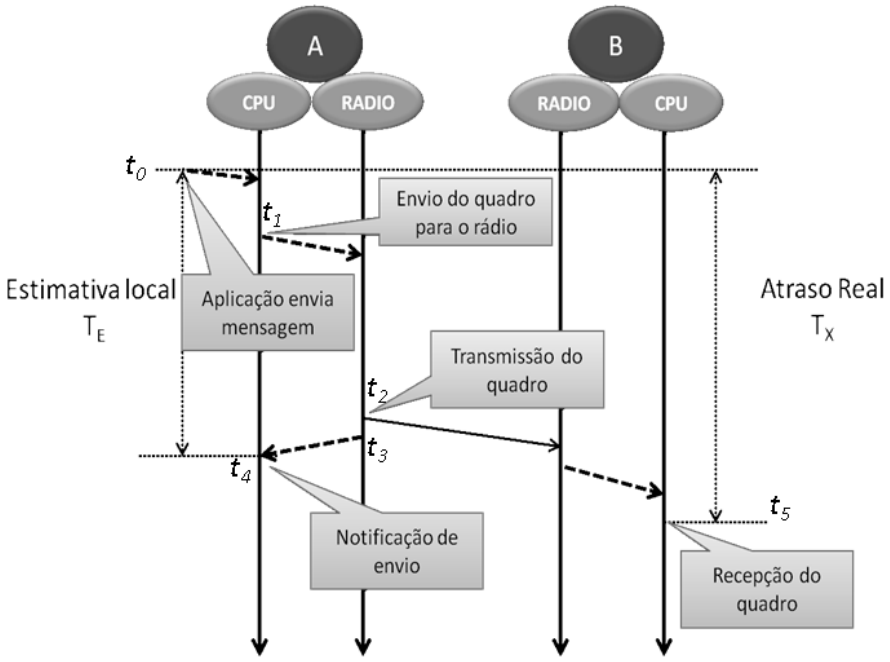

Figura 11. Diagrama de tempo da troca de mensagem do mecanismo proposto.

A notificação de envio, vista na Figura 11, iniciada em t_3é fundamental para esta técnica e pode ser facilmente acessada em uma série de dispositivos típicos das redes sem fio. Nos sensores que utilizam o sistema operacional TinyOS, por exemplo, ela é fornecida pelo evento *sendDone()*, ao passo que nas redes IEEE 802.11 existem outros mecanismos de notificação equivalentes.

Através desta notificação, o nó *'A'* poderá computar uma estimativa de quando *'B'* receberá sua mensagem e pode, em uma mensagem posterior, enviar uma correção para *'B'*.

O erro de estimativa, ε, será a diferença entre T_X e T_E e, claramente, ε será uma variável aleatória, tendo sua própria média e variância e a compensação do atraso será tão eficiente quanto à capacidade do modelo de relacionar estatisticamente o comportamento de T_E ao de T_X. Esse erro, como esperado apresenta variâncias diferentes para diferentes modelos de *motes*. Todavia, mesmo com essa dessemelhança em valores absolutos, se as suposições iniciais sobre os relacionamentos entre estimativa e atraso permanecerem válidas, nosso método também é válido e poderá ser aplicado.

Na subseção 5.3 apresentaremos com mais detalhes a montagem do experimento para teste e a avaliação do mecanismo e os resultados dos testes com ***motes* reais** exibidos na subseção 5.7 mostrarão com clareza a correlação prevista e o grau de eficácia da técnica proposta.

5.3 Avaliação do mecanismo

Em nosso experimento, para as tomadas das medidas do atraso de transmissão, utilizamos os *motes* de baixa potência citados e apresentados na subseção 2.2 e a tabela 3 assinala as principais diferenças entre os dois *motes*.

Tabela 3. Principais diferenças entre as especificações dos motes.

Modelo	Micro controlador	Rádio	Memória	Flash
Iris Mote	ATmega 1281	Atmel AT86RF230 802.15.4/ZigBee compliant radio	8 KB RAM	128 KB flash
MicaZ	ATmega 128	TI CC2420 802.15.4/ZigBee compliant radio	4 KB RAM	128 KB flash

As observações experimentais do atraso foram tomadas para avaliar o comportamento estatístico da latência na transmissão nos dois diferentes modelos de nós sensores e as aplicações para os testes experimentais dos

motes foram escritas em nesC, um dialeto da linguagem C otimizado para os baixos recursos de memória e CPU dos motes.

Com auxílio da Figura12, podemos visualizar as camadas de *software* que a mensagem atravessa até o envio. Nela, podemos observara camada que provê uma interface assíncrona para os sensores presentes em cada *mote* e uma pilha de componentes que implementa a rede. As camadas mais baixas recebem os dados *bit* a *bit*, e realizam o controle de fase e taxas. Lembrando que o rádio destes dispositivos opera em modo *half-duplex*.

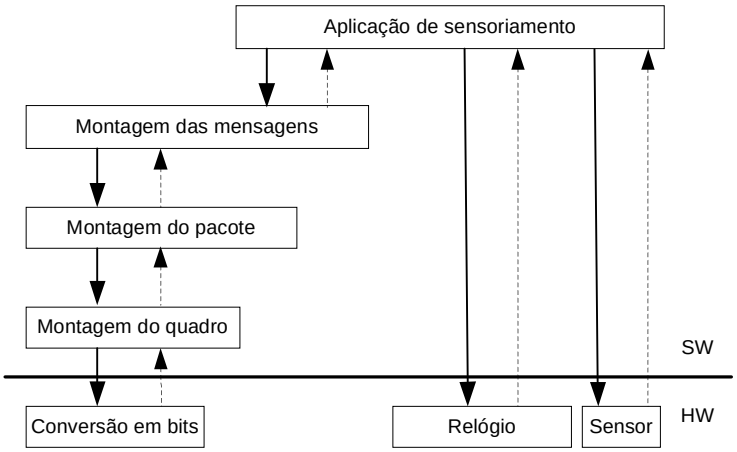

Figura 12. Componentes da aplicação do TinyOS.

O nível de manipulação de pacotes, Figura 12, é responsável por enfileirar os pacotes recebidos e é onde reside o mecanismo de controle de acesso ao meio, principal fonte do não determinismo do atraso. O tipo de codificação utilizada nestes dispositivos é a Manchester (16 bits CRC) com taxa alcançando 250kbps [Woo, Alec and Culler, D. E. 2001].

Para a avaliação da eficácia do nosso mecanismo, montamos o cenário experimental mostrado na Figura 13 e descrito a seguir: três motes foram programados em nesC, sendo representados como *'0'* (zero), *'A'* e *'B'*, mais um dispositivo capturador de pacotes *'S'*. Todos dentro de um mesmo domínio de broadcast. A avaliação se inicia com o mote *'0'* enviando uma mensagem em difusão que alcança os motes *'A'* e *'B'*, respectivamente em T_{0A} e T_{0B}. Após a recepção destes quadros os sensores *'A'* e *'B'* passarão a usar, respectivamente, as marcas T_{0A} e T_{0B} como sua referência de tempo. Assumindo que o atraso entre estas marcas é menor que 30,5µs (1/32kHz),

podemos, a partir desse ponto, considerar os motes ‘A’ e ‘B’ sincronizados a menos desse erro < 30,5 μs.

Ao receber a mensagem de *‘0’*, o *mote ‘A’* dispara seu *clock* e a aplicação ordena o envio de uma mensagem em T_{1A}, que devido ao tempo de *back off*, entre outros fatores já descritos na Tabela 1, ganha o meio apenas em T_{2A}. Depois de algum tempo a CPU de *‘A’* é notificada do envio pelo evento *sendDone()* e em seguida envia a compensação T_E. Nesse ínterim *‘B’* finalmente recebe T_{1A}, no instante T_{1B}. *‘B’* em resposta a esse evento, envia uma mensagem em difusão contendo o valor de T_{1B}, finalizando um ciclo de tomada de tempo. A função de *‘S’* é capturar todos os quadros e servir como relógio que marca o tempo global do experimento. De modo, que o tempo T_X possa ser calculado *offline* para validação da técnica. Assim:

$$T_E = T_{2A} - T_{1A} \tag{5.1}$$

$$T_X = T_{1B} - T_{0B} - (T_{1A} - T_{0A}) \tag{5.2}$$

E o erro ε é calculado *off-line* durante o processamento dos dados e será:

$$\varepsilon = T_X - T_E \tag{5.3}$$

Consideramos o tempo de propagação, da ordem de nanossegundos desprezível para a precisão requerida. Um ponto importante nos resultados de nossos experimentos em ambos os modelos de *mote* é que $T_E < T_X$, isso nos permite prever que há ainda um componente que poderá ser usado para melhorar a precisão do ajuste.

Os *motes* usados têm um *clock* de 32kHz e todas as mensagens enviadas por ‘*0’*, *‘A’ e ‘B’* contém um campo com número de sequência, que serve como elemento de referência para as mensagens associadas a cada nó e a cada ciclo de cálculo de T_X.

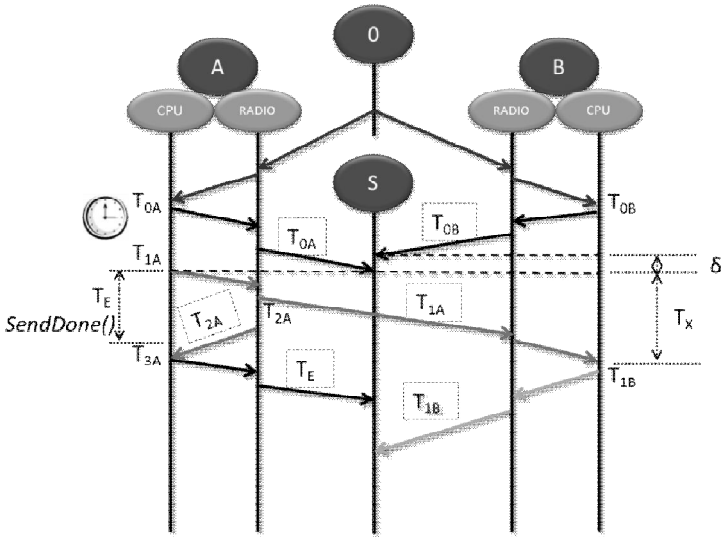

Figura 13. Diagrama de tempo detalhado do experimento.

5.4 Diferenças entre os modelos testados

Apesar das medidas do atraso de transmissão serem claramente não determinísticas,confirmamos essa suposição com testes de aleatoriedade realizados no ambiente estatístico *R,* que confirmam, com *p-value* > 0,5, que as amostras estudadas dos dois modelos de *motes* são randômicas e possuem diferenças estatísticas significativas entre si, suficientes, para querermos caracterizar suas distribuições separadamente.

Um sumário com os principais parâmetros medidos comparando ambos os modelos de *motes* é mostrado na Tabela 4 com intervalo de confiança de 0,95.

Testes não paramétricos de Wilcoxon com *p-value<<0,5* confirmaram a hipótese da significante diferença estatística das distribuições do atraso nos dois modelos de sensores, como pode ser inferido pelas distâncias entre os valores de um e outro modelo na Tabela 4. Ainda analisando a Tabela 4, iremos confirmar nossa suposição inicial na subseção 5.2. Que a estimativa será tanto melhor quanto menor for a variância do atraso e um dos fatos que apóia essa evidência é que a precisão no STELE alcançada nos *motes* Iris foi maior que nos resultados com os modelos MicaZ.

Tabela 4. Comparação dos parâmetros estatísticos dos atrasos T_X medidos nos dois modelos de sensores.

	MicaZ (ms)	Iris (ms)
Valor mínimo	2,01	1,56
Valor máximo	11,90	9,79
Intervalo (min-max)	9,89	8,23
Mediana	6,98	4,15
Média	6,92	4,64
Erro Padrão	0,39	0,11
Variância	7,45	4,12
Desvio padrão	2,73	2,03
Curtose	-1,20	-1,23
Assimetria	0,02	0,33
Coeficiente de Variação	0,39	0,44

Podemos também inferir pela sobreposição dos histogramas, na Figura 14, que o comportamento das distribuições ao longo do tempo é diferente e provaremos, na seção 5.7, que ainda assim, mesmo com essas claras diferenças de comportamento, a relação esperada, entre atraso e estimativa persiste com graus muito parecidos de correlação, mostrando que nosso método é aplicável para ambos os modelos.

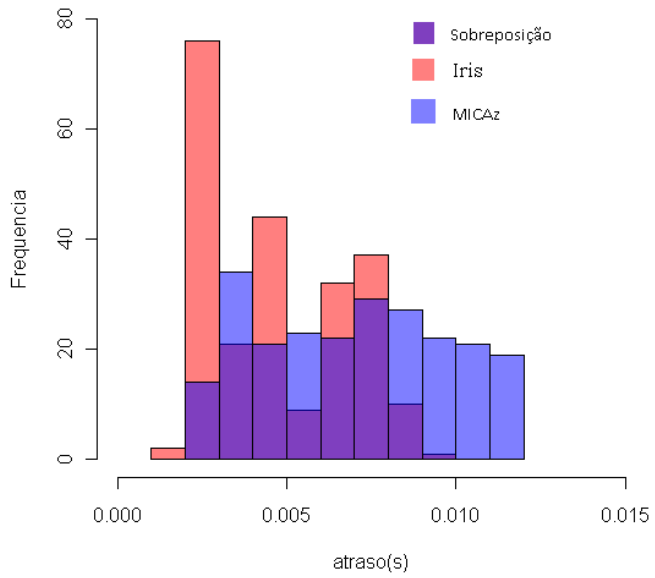

Figura 14. Sobreposição de histogramas dos atrasos reais.

5.5 O ambiente simulado

Alguns dos resultados apresentados até agora foram de experimentos com *motes* reais, no entanto iniciamos nossas tomadas de tempo e depuração dos códigos no simulador TOSSIM,que nos oferece diversas vantagens, mas também algumas desvantagens que descreveremos nesta seção.

Na simulação, podemos contar com uma referência global única de tempo e muito mais precisa que o relógio do *hardware* real, o que permite tomadas de tempo globais.Outra vantagem é que podemos modificar o nosso cenário de simulação que modela o ambiente real, apenas reconfigurando um arquivo de parâmetros, onde podemos modificar valores de topologia, ruído, propagação *etc.* Isso traz um benefício mas apresenta uma dificuldade, já que devemos ser cuidadosos para que o ambiente simulado expresse o mais fielmente possível o cenário real.O arquivo que contém as informações para a simulação do canal de rádio possui diversos parâmetros que podem ser configurados, entre eles :

- ***gain(src, dest)***: retorna o valor do ganho em dB entre origem e destino.
- ***setNoise(node, mean, variance)***: indica o nível de **ruído gaussiano** dada uma média e variância.
- ***initHigh****:* o limite máximo para o *back off* inicial – com valor *default* de 400 (adimensional).
- ***initLow***: o limite mínimo para o *back off* inicial com valor *default* de 20(adimensional).

Os dois primeiros são configurados somente para o script de simulação feito em Python, os dois últimos devem ser modificados antes da compilação do código que será usado na simulação. Os valores usados para programação dos *motes* foram 400 e 20, para *gain* e *setNoise* respectivamente.Em seguida explicaremos como chegamos a esses valores para ***initHigh***: e ***initLow.***

O TOSSIM simula o comportamento do rádio do MicaZ, *Chipcon* 2420, que por sua vez usa o padrão IEEE 802.15.4 para definir suas camadas MAC e PHY. O nosso método usa o fato do atraso de contenção nesta camada, responder pela maior fatia de tempo na formação do atraso total, de modo que seu comportamento em ambiente simulado deverá espelhar ao máximo o

comportamento real, para que os resultados sejam o mais próximo possível da realidade. Assim, uma série de simulações trocando estes valores foram executadas para encontrar a melhor tupla ***initHigh*** e ***initLow*** (simulados),vistos na linha evidenciada com borda dupla na Tabela 5, de modo a ajustar não somente as médias e as variâncias que se assemelhassem ao ambiente real, mas também os valores mínimos e máximos do atraso, que podem ser comparados com a Tabela 5.

5.6 Resultados em ambiente simulado

Depois dos ajustes explicados na seção anterior, executamos as mesmas análises dos parâmetros estatísticos da seção 5.4 em ambiente simulado,tais como o teste de Wilcoxon e os valores da Tabela 6 confirmam que resultados em ambiente simulado para o modelo Micaz, equivalem ao atraso real encontrado experimentalmente. Nela exibimos uma comparação entre os parâmetros estatísticos do TOSSIM *versus* os resultados com ambos os modelos de *motes* reais,demonstrando que o TOSSIM modela com realismo a distribuição do atraso e pode ser empregado, desde que observadas as características reais do meio a ser simulado assim como o modelo de mote a ser comparado.Visualizamos essas similaridades através da sobreposição de histogramas apresentada na Figura 16.,Mas apesar das inúmeras vantagens encontradas no ambiente simulado para a fase inicial do trabalho, que era a caracterização do atraso, o TOSSIM não modela bem a diferença de tempo entre o envio da mensagem pela aplicação e o retorno do evento de notificação *sendDone()*. Essa discretização do comportamento é observado na Figura 15, pois na simulação, para cada valor do atraso T_X, há apenas dois valores possíveis para a estimativa T_E, ou é igual ao atraso ou é 1ms menor.

Tabela 5. Valores dos parâmetros e seus impactos na média e variância dos atrasos em ambiente simulado.

initHigh	*IniitLow*	Média (ms)	Variância
780	80	7,5	9,8
780	90	7,6	9,0
780	100	7,7	9,2
780	120	8,0	8,8
785	95	7,9	9,9
790	90	7,9	9,8
800	**120**	**7,9**	**9,5**
820	140,0	8,6	9,7
840	80	8,0	11,1
840	120	8,3	10,2
840	140	8,5	9,8

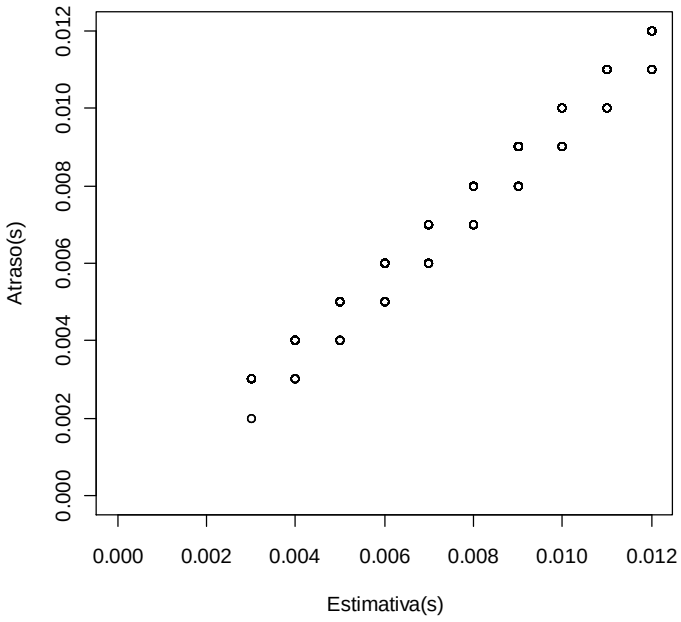

Figura 15. Dispersão do atraso *mote* MicaZ em ambiente simulado TOSSIM.

Para concluir, ao compararmos os resultados experimentais com resultados simulados, nos deparamos com valores de T_E simulados muitos distantes dos valores obtidos com motes reais, como observado na Tabela 6. Nosso trabalho confirma que o TOSSIM simula a distribuição do atraso de transmissão entre dois motes com bastante fidelidade, mas não se adequa para simular o funcionamento do nosso mecanismo como ferramenta de estimativa do atraso entre dois nós. Consideramos então que uma

contribuição colateral de nosso estudo foi a de demonstrar que ao usar o TOSSIM, em trabalhos futuros, deveremos aprimorar a modelagem do evento de “notificação de envio”, de maneira a tornar a simulação mais próxima ao real funcionamento dos motes.

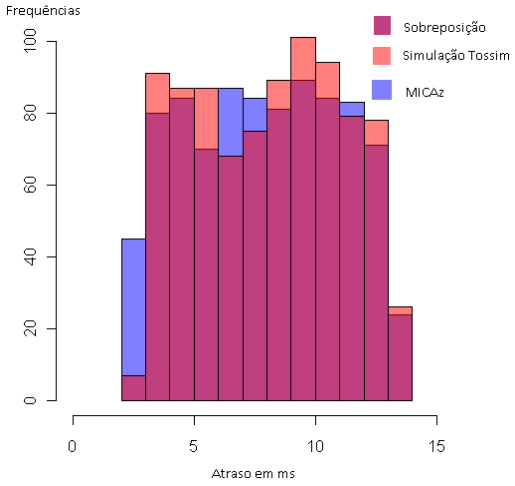

Figura 16. Sobreposição de histogramas dos atrasos no mote MicaZ e no TOSSIM.

Tabela 6. Comparações entre a estimativa T_E em *motes* reais e no TOSSIM

	MicaZ (ms)	TOSSIM (ms)	Iris(ms)
Valor mínimo	2,01	**0.0**	1,49
Valor máximo	11.47	**1.0**	9,70
Intervalo	9,46	**1.0**	8,21
Mediana	6,65	**1**	4,08
Média	6,69	**0,72**	4,53
Erro Padrão	0,13	**0,014**	0,11
Variância	7,5	**0,2**	4,08
Desvio padrão	2,74	**0.44**	2,02
Curtose	-1,22	**-1.01**	-1,24
Assimetria	0,03	**-0.99**	0,33

5.7 Resultados com motes reais

Depois de demonstrar a aleatoriedade das duas variáveis, a significativa diferença estatística entre elas e determinar que não há um padrão na distribuição do atraso entre os dois tipos de *motes*, mostraremos nosso resultado que depende da relação estatística que existe entre essas duas

grandezas e esse resultado será tão bom com quanto sua correlação for próxima de 1.

A associação de dados bivariados se dá quando existe uma ligação direta entre as variações destas VAs, isto é, quando uma grandeza varia a outra também o faz. Quando o crescimento de uma variável tende a acompanhar o aumento da outra diz-se que há uma correlação positiva.Ao contrário, se uma cresce e a outra diminui, define-se que há uma correlação negativa [Mauro *and* Andrade 2007]. Um dos parâmetros que mede estas correlações é chamado de ρ e também conhecido como covariância normalizada. Diz-se que a correlação é positiva quando $\rho>0$ e negativa, quando $\rho<0$. Este índice é adimensional e varia entre "+1" e "-1", tais valores, são seus extremos e indicam uma correlação perfeita entre as duas variáveis analisadas. Na Tabela 7, encontramos os tipos de correlação existentes.

Tabela 7. Tipos de correlações entre variáveis

ρ	**Correlação**
0,0	Nula
0,0 a 0,3	Fraca
0,3 a 0,6	Média
0,6 a 0,9	Forte
0,9 a 0,99	Fortíssima
1	Perfeita

Análises estatísticas sobre os dados obtidos experimentalmente, dispostos na Tabela 8, nos mostraram através do teste de Pearson, que a relação entre as variáveis T_E e T_X, é de $\boldsymbol{\rho=0.9986}$ com 95% de intervalo de confiança para o Micaz e $\boldsymbol{\rho=0.9995}$com o mesmo intervalo para o Iris, caracterizando assim essa associação como **"positiva fortíssima"**. Tal comportamento é claramente observado através dos gráficos de dispersão entre a Estimativa T_E e o Atraso T_X apresentados na figura 17.

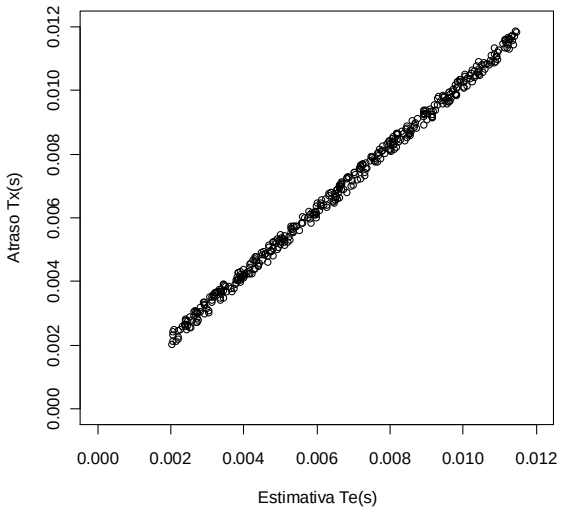

Figura 17a. Dispersão do atraso *mote* MicaZ

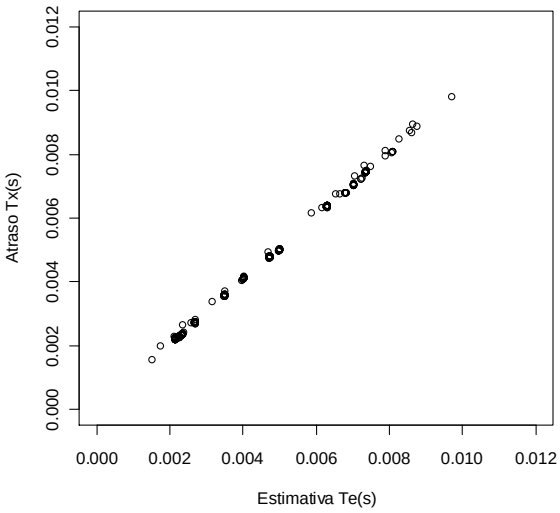

Figura 17b. Dispersão do atraso *mote* Iris

Devemos agora encontrar a função que, em termos médios, melhor se ajusta a essas curvas, que é claramente do tipo:

$$T_X = \alpha T_E + \beta \tag{5.4}$$

Onde α= coeficiente angular e β=coeficiente linear.

Através da aplicação de uma regressão linear sobre os dados obtidos experimentalmente, chegamos aos valores dos coeficientes α e β, apresentados em (5.4), que ajustam a melhor reta à curva de dispersão, que pode ser vista na Figura 18a.

$$T_X = 0{,}0003349 + 0{,}9981952 \; . \; T_E \tag{5.5}$$

Se procedermos da mesma forma para o Iris obteremos:

$$T_X = 0{,}00005011 + 1{,}003 \; . \; T_E \tag{5.6}$$

Essas equações mostram as proximidades de seus coeficientes angulares entre si e com a unidade, de modo que é possível perceber sua quase sobreposição, que confirma a eficácia do método para ambos os modelos.

O experimento, detalhado na subseção 5.3 e explicitado através da Figura 13, foi executado diversas vezes, nos fornecendo mais de 3000 tomadas de

tempo do atraso T_X e da estimativa T_E, com o mesmo cenário, para cada modelo analisado.

A Tabela 8 nos mostra um sumário desses resultados. E esses dados confirmam nossa hipótese da correlação entre T_E e T_X próxima de 1.Com o nosso método foi possível alcançar uma estimativa do atraso com erro médio da ordem de 240 ±30,5μs nos modelos Micaz e de 80 ±30,5μs no Iris. Esse fator de 30,5μs foi introduzido pela imprecisão inicial do experimento que se deve ao fato de nosso estudo começar com os nós sensores tendo seus relógios fora de sincronia e uma sincronização inicial deve ser alcançada. Para isso utilizamos um método semelhante ao RBS, em que um *mote* programado especificamente para este fim, envia um quadro de referência. Os motes *'A'* e *'B'* ao receber este primeiro quadro passam então a contar seus tempos locais.

Tabela 8. Comparação entre atraso e estimativa nos modelos Iris e MicaZ (IC=95%)

	Atraso MicaZ (ms)	Estimativa MicaZ(ms)	Atraso Iris (ms)	Estimativa Iris (ms)
Valor mínimo	2,01	2,01	1,56	1,49
Valor máximo	11,90	11,47	9,79	9,70
Intervalo	9,89	9,46	8,23	8,21
Mediana	6,98	6,65	4,15	4,08
Média	6,92	6,69	4,64	4,53
Erro Padrão	0,13	0,13	0,11	0,11
Variância	7,45	7,50	4,12	4,08
Desvio padrão	2,73	2,74	2,03	2,02
Curtose	-1,20	-1,22	-1,23	-1,24
Assimetria	0,02	0,03	0,33	0,33
Coeficiente de Variação	0,39	0,40	0,44	0,44

A limitação de frequência do relógio desses nós sensores além de introduzir o erro inicial também é responsável pelo comportamento discreto do erro ε apresentado nas Figuras 18a e 18b. Nelas observamos também a distribuição do erro ao longo dos atrasos.

Todos os testes foram realizados não em apenas um tipo de mote, mas em dois dos modelos mais utilizados em RSSF e em virtude da similaridade de

resultados obtidos em ambos os modelos não temos razão para supor que as relações entre o erro ε, a estimativa T_E e o atraso T_X não obedeçam de maneira similar a estes comportamentos em dispositivos de outros fabricante.

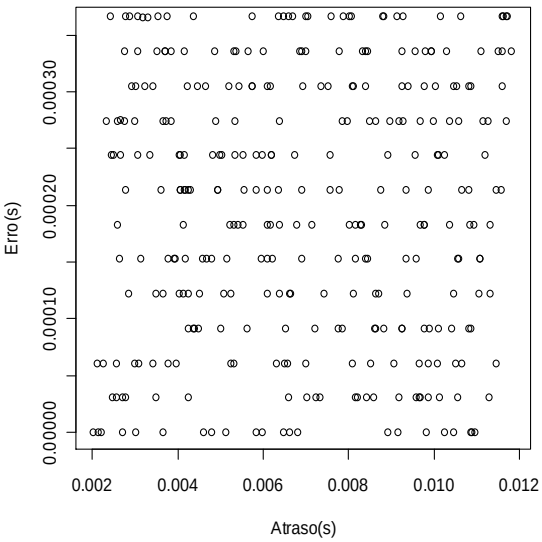

Figura 18a. Dispersão do erro no *mote* MicaZ

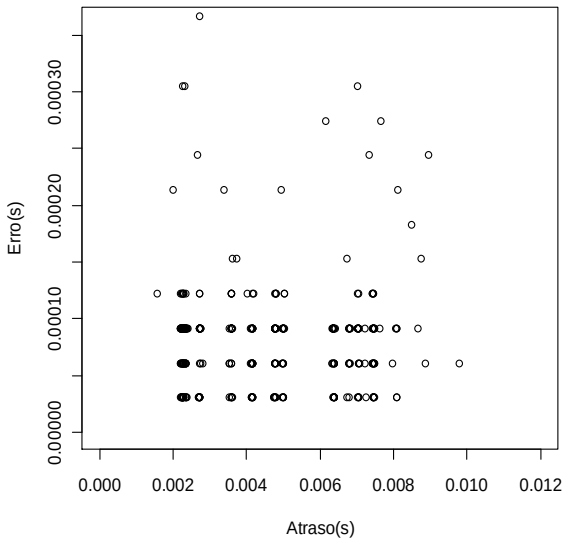

Figura 18b. Dispersão do erro no *mote* Iris

5.8 Exemplo de uso do STELE como técnica de sincronização

Descreveremos nesta subseção um exemplo de como uma técnica de sincronização poderia funcionar utilizando o STELE como mecanismo de estimativa do atraso. Na Figura 19 vemos um cenário onde um mote *'B'*, é sincronizado com o mote *'A'* com o envio de apenas duas mensagens. Para maior clareza do cenário estamos considerando desprezíveis as componentes do atraso devido ao tempo de propagação e de recepção em *'B'*.

Dois *motes 'A'* e *'B'* foram ligados e estão com seus relógios defasados de por exemplo 4s. O mecanismo então inicia sua tentativa de sincronização em um instante *t* = 2:02 com *'A'* enviando um quadro para *'B'* contendo este tempo *t* = 2:02, mas devido a contenção e outros fatores já explicados na subseção 4.4, o quadro ganha o meio somente em 2:04, chegando em

imediatamente em *'B'*, no seu tempo local 2:00. *'B'* então armazena este tempo de referência. Em 2:05 a CPU de *'A'* recebe a notificação do rádio que o primeiro quadro ganhou o meio somente em 2:04. Assim nossa aplicação computa o Tempo T_E = 2:04-2:02 = 2:02, portanto poderá enviar um quadro para *'B'* compensando o indeterminismo da fase de envio. Esse quadro contendo T_E sai de *'A'* em 2:07, chegando em *'B'* em 2:06, tempo local de *'B'*. Nesse momento *'B'* recupera o tempo 2:02, armazenado, computa também o tempo local desde a chegada da primeira mensagem, 2:06-2:00=0:06 e executa a soma 2:02+2:06+0:02, cujos fatores são respectivamente:

- Tempo local da *'A'* no envio de seu primeiro quadro de sincronização;
- Tempo local de *'B'* na recepção do segundo quadro contendo a compensação;
- E finalmente o valor da compensação.

Após essa soma, *'B'* finalmente ajusta o seu relógio local para 2:10 e agora se considera já sincronizado com o nó *'A'*.

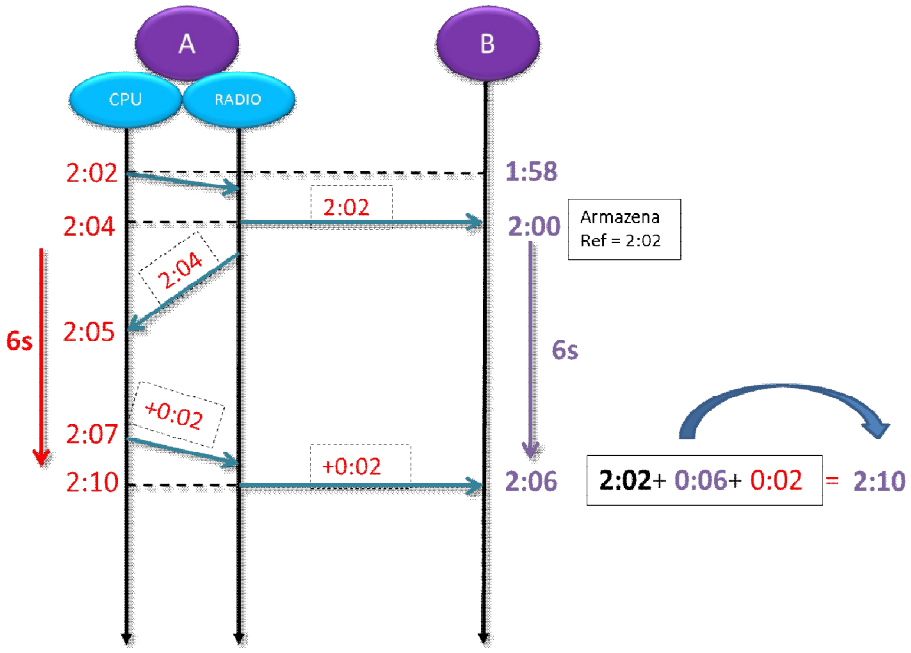

Figura 19.Exemplo de uso do STELE com técnica de sincronização

Note que nesse exemplo de uso não consideramos o tempo de propagação e recepção e computação em *'B'*, sendo esses tempos justamente as componentes formadoras de ε.

Capítulo 6– Consumo de energia

6.1 Consumo de energia em ambiente simulado

Para avaliar o consumo de energia dos *motes* durante o processo de sincronização, montamos dois cenários em ambiente simulado usando o Avrora [Titzer *et al*. 2005].O primeiro cenário de simulação contém uma rede de sensores com cinco nós em um mesmo domínio de difusão e o segundo consiste de uma rede contendo vinte nós em condições similares de transmissão e duas técnicas de sincronização foram usadas para comparação o STELE e o FTSP.

As medidas tomadas foram feitas usando apenas os monitores de rádio e CPU entre todos os oferecidos pelo Avrora, refletindo assim em nossos resultados, somente o consumo de energia em uso de CPU e em transmissão de quadros.O tempo de cada simulação foi de trezentos segundos de simulação, usando como intervalo de sincronização (*flag* necessário a implementação do FTSP): 1, 5,10, 30 e 60 segundos.Foram feitas 10 tomadas para cada intervalo e os dados médios destas tomadas estão apresentados nas Tabelas 9 e 10, com intervalo de confiança de 0,95. Os rádios em ambos os cenários permaneceram em modo *idle* sempre que não estavam transmitindo, jamais sendo colocados em modo *sleep* ou desligados.

Nessa simulação, ao compararmos nosso mecanismo com o FTSP, descrito na subseção 4.4, usamos uma versão modificada do TestFTSP distribuído no framework do TinyOS 2.1.1.Essas modificações visaram apenas remover o envio das mensagens de relatório que não são necessárias em um ambiente de sincronização para medição de consumo de energia.

A Tabela 9 tem como referência os dados da rede com cinco nós,enquanto a Tabela 10 mostra os dados obtidos a partir da simulação de uma rede com vinte nós.

Tabela 9. Gasto de energia referente à CPU e ao Rádio com 5 nós

Consumo em CPU(mJ)			Consumo do rádio em transmissão (mJ)		
Intervalo de Sincronização(s)	STELE	FTSP	Intervalo de Sincronização (s)	STELE	FTSP
1	178,757	471,0052	1	31,424	90,111
5	138,908	173,8109	5	6,222	13,387
10	133,891	148,5399	10	3,058	6,003
30	130,556	135,1897	30	0,949	1,622
60	129,725	131,340	60	0,422	0,433

Tabela 10. Gasto de energia referente à CPU e ao Rádio com 20 nós.

Consumo em CPU(mJ)			Consumo do rádio em transmissão (mJ)		
Intervalo de sincronização(s)	STELE	FTSP	Intervalo de sincronização(s)	STELE	FTSP
1	662,549	3939,134	1	31,424	368,017
5	551,303	837,314	5	6,222	124,432
10	532,961	714,532	10	3,058	22,509
30	520,745	563,847	30	0,949	5,111
60	517,693	527,330	60	0,422	1,544

6.2 Análise da simulação do consumo de energia

Os gráficos da Figura 20 mostram a evolução do consumo de energia em função do aumento do intervalo de sincronização, para ambos os mecanismos, com cinco e com vinte nós e a análise desses dados sugere, embora nosso mecanismo não seja um protocolo de sincronização completo como o FTSP, que com o aumento do número de nós o nosso método é menos custoso como gasto de energia no uso de CPU e que também é mais econômico em uso do rádio para o envio de quadros, basta acompanhar as duas Tabelas 9 e 10 na coluna "**Consumo do rádio em transmissão**" e observar que com o aumento do número de nós o consumo permanece constante dentro dos seus respectivos intervalos de sincronização.

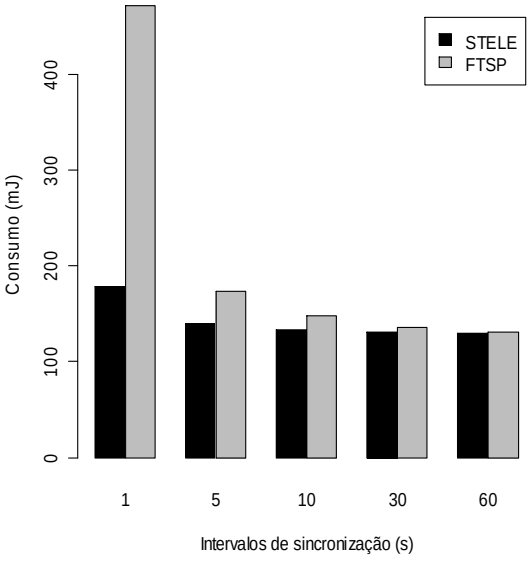

Figura 20a. Consumo de energia no uso de CPU.Simulação com cinco nós.

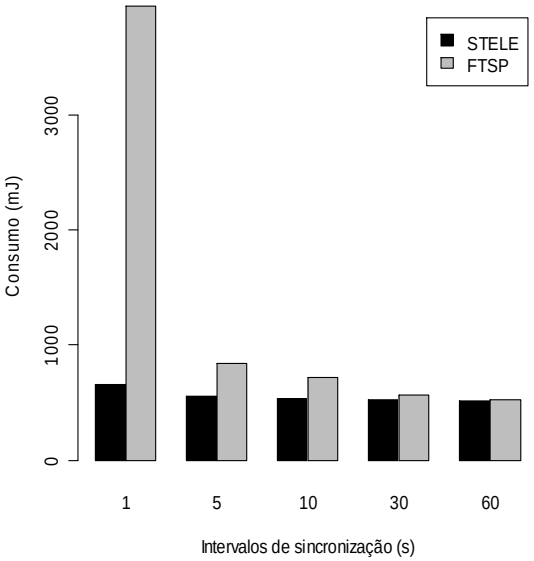

Figura 20b. Consumo de energia no uso de CPU.Simulação com vinte nós.

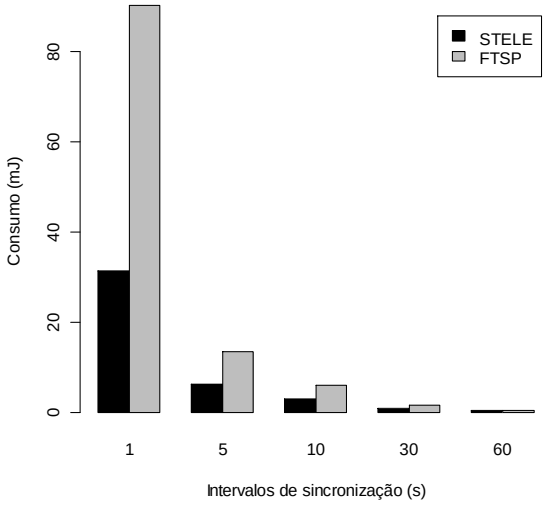

Figura 20c. Consumo de energia no uso do rádio. Simulação com cinco nós.

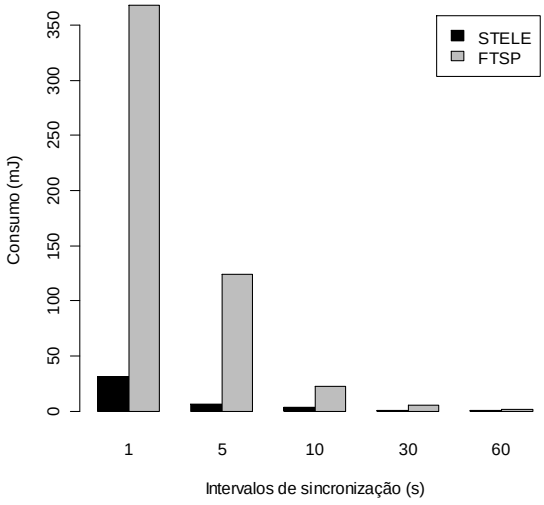

Figura 20d. Consumo de energia no uso de rádio. Simulação com vinte nós.

Dados do FTSP mostram que apesar do consumo de energia cair com o aumento do intervalo de sincronização para a ordem de poucos miliJoules, o FTSP difunde suas mensagens de sincronização em momentos aleatórios dentro deste intervalo. Isto exige que os rádios permaneçam ligados

aguardando estes quadros. Essa situação dificulta a abordagem do *duty cycling* para economia de energia no uso desse protocolo. Já o STELE se adequa bem a essa solução. O que garantiria uma economia expressiva, se levarmos em consideração que **apenas** o consumo de energia do rádio é da ordem de dezenas de joules.

Capítulo 7–Considerações finais

7.1 Cenários de uso

Como cenários de uso, a nossa técnica pode ser usada para sincronização em duas fases: na primeira fase o tempo do nó Mestre é enviado e numa segunda fase apenas o *offset* com a estimativa do atraso é transmitido. Outro cenário de uso seria a previsão de *jitter*: neste caso, o nó mestre poderia aprender, durante os quadros iniciais, a variação do atraso na rede, levando em consideração sempre a condição do enlace local e nos quadros posteriores compensaria essa variação, atrasando ou adiantando as transmissões em relação ao dado aprendido.

7.3 Conclusões e trabalhos futuros

Alcançar a sincronização é um fator crucial para a construção real de aplicações que rodem sobre redes de sensores e o nível de precisão requerido depende da necessidade de cada aplicação. A contribuição de nosso trabalho foi a de apresentar um mecanismo simples, pouco custoso em energia e que não requer enlaces bidirecionais, fugindo da abordagem tradicional que exige muitas trocas de quadros entre servidores e clientes. Essas vantagens somadas ao fato que sua implementação é independente de características especiais de hardware confere ao mecanismo STELE a sua universalidade como solução de sincronização. Isto é, este mecanismo pode ser usado em quaisquer dos modelos de *motes* existentes. Outra contribuição de nosso trabalho foi a de demonstrar que a modelagem do hardware pelo TOSSIM, no que toca a notificação do envio pode ser melhorada para refletir com maior precisão o funcionamento de motes reais.

Esta técnica, também por enviar os quadros em tempos determinados, se adequa bem a soluções que use *duty cycling* e se ajusta perfeitamente a aplicações que permitam o uso de sincronização *post facto*, onde os nós estão geralmente fora de sincronismo, e somente após a ocorrência de um evento externo, um nó inicia a sincronização com seus vizinhos.

Em trabalhos futuros pretendemos verificar a escalabilidade do mecanismo em redes multissalto sujeitas a diferentes níveis de interferência e em cenários de mobilidade.

Referências

Akyildiz, I. F. and Vuran, M. C. (2010). Wireless Sensor Networks. Chichester, UK: John Wiley & Sons, Ltd. p. 243–262

Alizai, M. H., Landsiedel, O., Wehrle, K. and Group, D. S. (2008). Modeling Execution Time and Energy Consumption in Sensor Node Simulation. PIK Journal Special Issue on Energy Aware Systems, v. 32, n. 2.

Anastasi, G., Conti, M., Di Francesco, M. and Passarella, A. (2006). An adaptive and low-latency power management protocol for wireless sensor networks. MobiWAC 2006 Proceedings of the 2006 ACM International Workshop on Mobility Management and Wireless Access, v. 2006, p. 67–74.

Committee, T. (2008). IEEE Std 1588-2008, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. Society. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4579760.

Crossbow (2005a). Crossbow MicaZ mote specification. http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf, [acessado em 8 de agosto de 2012].

Crossbow (2005b). Crossbow MIB520 interface board specification. http://www.willow.co.uk/html/mib520-_usb_gateway.html, [accessed on Aug 8].

Crossbow (2007). Crossbow Iris mote specification. http://www.dinesgroup.org/projects/images/pdf_files/iris_datasheet.pdf, [acessado em 8 de agosto de 2012].

Elson, J., Girod, L. and Estrin, D. (2002). Fine-grained network time synchronization using reference broadcasts. ACM SIGOPS Operating Systems Review, v. 36, n. SI, p. 147–160.

Estrin, D., Govindan, R., Heidemann, J. and Kumar, S. (1999). Next Century Challenges: Scalable Coordination in Sensor Networks. In Proceedings of the 5th annual ACMIEEE international conference on Mobile computing and networking. . ACM. http://portal.acm.org/citation.cfm?id=313556.

Fraboulet, A., Chelius, G. and Fleury, E. (2007). Worldsens: Development and Prototyping Tools for Application Specific Wireless Sensors Networks.

2007 6th International Symposium on Information Processing in Sensor Networks, p. 176–185.

Ganeriwal, S., Kumar, R. and Srivastava, M. B. (2003). Timing-sync protocol for sensor networks. Proceedings of the first international conference on Embedded networked sensor systems SenSys 03, p. 138–149.

Haas, C., Wilke, J. and Stöhr, V. (2012). Realistic Simulation of Energy Consumption in Wireless Sensor Networks. [G. Pietro Picco & W. Heinzelman, Eds.]In Lecture Notes in Computer Science Wireless Sensor Networks. . Springer Berlin Heidelberg. http://www.springerlink.com/content/hvk72j4335531531/.

Kopetz, H. and Ochsenreiter, W. (1987). Clock Synchronization in Distributed Real-Time Systems. IEEE Transactions on Computers, v. C-36, n. 8, p. 933–940.

Lenzen, C., Sommer, P. and Wattenhofer, R. (2009). Optimal clock synchronization in networks. Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems SenSys 09, p. 225–237.

Levis, P., Lee, N., Welsh, Matt and Culler, D. (2003). TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In Proceedings of the 1st international conference on Embedded networked sensor systems. . ACM. http://portal.acm.org/citation.cfm?id=958491.958506.

Levis, P., Madden, S., Polastre, J., et al. (2005). TinyOS: An Operating System for Sensor Networks. Ambient Intelligence, v. 35, p. 115–148.

Man, L. A. N. and Committee, S. (2007). IEEE Std 802.15.4aTM-2007, IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—LANs and MANs—Specific Requirements—Part 15.4: Wireless MAC and PHY Specifications for LR-WPANs—Amendment 1: Add Alternate PHYs. IEEE. v. 2007p. 203

Maróti, M., Kusy, B., Simon, G. and Lédeczi, Á. (2004). The flooding time synchronization protocol. In Proceedings of the 2nd international conference on Embedded networked sensor systems SenSys 04. ACM Press. http://portal.acm.org/citation.cfm?doid=1031495.1031501.

Mauro, N. and Andrade, É. J. P. (2007). Hidrologia Estatística. 1. ed. von Sperling;Soares, Ernesto José Márcio Henriques.

Mills, D. L. (1985). Network Time Protocol (NTP). Network. IETF. http://www.ietf.org/rfc/rfc958.txt.

Moss, D., Hui, J., Levis, P. and Choi, J. Il (2007). TEP 126 CC2420 Radio Stack. Architecture, p. 1–11.

Osterlind, F., Dunkels, A., Eriksson, J., Finne, N. and Voigt, T. (2006). Cross-Level Sensor Network Simulation with COOJA. 2006, p. 641–648.

Ping, S. (2003). Delay Measurement Time Synchronization for Wireless Sensor Networks. Intel Research, IRB TR-03-013, p. 12.

Python Software Foundation (2000). Python Programming Language. http://www.python.org/, [acessado em 7 de abril de 2012].

R Development Core Team (2007). R: A language and environment for statistical computing. ISBN 3-900051-07-0. [Austria, Ed.]R Foundation for Statistical Computing Vienna Austria. R Foundation for Statistical Computing. http://www.r-project.org.

Ranganathan, P. and Nygard, K. (2010). Time synchronization in wireless sensor networks: A survey. Proceedings of the IEEE SoutheastCon 2010 SoutheastCon, v. 1, n. 2, p. 92–102.

Sousa, C., Carrano, R. C., Magalhães, L. C. S. and CVN Albuquerque. (2013). STELE - Uma Técnica Simples Para Estimativa Local do Atraso de Transmissão em RSSF. In proccedings of the CSBC/SEMISH - XL Seminário Integrado de Software e Hardware, No prelo.

Sundani, H. (2011). Wireless Sensor Network Simulators A Survey and Comparisons. Computer Networks, v. 2, n. 2, p. 249–265.

The Network Simulator, NS–2 (1982). www.isi.edu/nsnam/ns/, acessado em 05 de abril de 2012].

Titzer, B. L., Lee, D. K. and Palsberg, J. (2005). Avrora: scalable sensor network simulation with precise timing. . 2005, p. 477–482.

Woo, Alec and Culler, D. (2002). Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks. Citeseer, n. 02-0013, p. 1–11.

Woo, Alec and Culler, D. E. (2001). A transmission control scheme for media access in sensor networks. Proceedings of the 7th annual international conference on Mobile computing and networking MobiCom 01, p. 221–235.

Yick, J., Mukherjee, B. and Ghosal, D. (2008). Wireless sensor network survey. Computer Networks, v. 52, n. 12, p. 2292–2330.