UNIVERSIDADE FEDERAL FLUMINENSE

HEDER DORNELES SOARES

FTSP+: UMA ABORDAGEM INDEPENDENTE DE HARDWARE PARA SINCRONIZAÇÃO EM REDES DE SENSORES SEM FIO

NITERÓI

2016

UNIVERSIDADE FEDERAL FLUMINENSE

HEDER DORNELES SOARES

FTSP+: UMA ABORDAGEM INDEPENDENTE DE HARDWARE PARA SINCRONIZAÇÃO EM REDES DE SENSORES SEM FIO

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

Orientador:
CÉLIO VINICIUS NEVES DE ALBUQUERQUE

Co-orientador:
RAPHAEL PEREIRA DE OLIVEIRA GUERRA

NITERÓI

2016

HEDER DORNELES SOARES

FTSP+: UMA ABORDAGEM INDEPENDENTE DE HARDWARE PARA SINCRONIZAÇÃO EM REDES DE SENSORES SEM FIO

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

Aprovada em Março de 2016.

BANCA EXAMINADORA

Prof. Célio Vinicius Neves de Albuquerque - Orientador, UFF

Prof. Raphael Pereira de Oliveira Guerra, Coorientador, UFF

Prof. José Viterbo Filho, UFF

Prof. Alexandre Sztajnberg, UERJ

Niterói

2016

Dedico este trabalho aos meus pais Tim e Lia.

Agradecimentos

Agradeço primeiramente aos meus pais Euripedes Dorneles Costa (Tim) e Maria do Carmo Soares (Lia), pelo eterno orgulho, apoio, compreensão, ajuda, e, em especial, por todo carinho ao longo deste percurso. A minha irmã Ludmila Dorneles Soares, pelo carinho, compreensão.

Aos meus orientadores, Célio Albuquerque e Raphael Guerra, não somente pela orientação ou pelo suporte, mas também pelos ensinamentos e a dedicação ao meio acadêmico.

Aos amigos da *OptHouse*: Igor, João, Marcos, Matheus, Pablo, Edcarllos e Alan. Agradeço não somente pelo incrível acolhimento e aprendizado que adquiri durante todos esses anos, mas também pela convivência e pelos momentos de descontração.

Aos muitos amigos que fiz durante essa etapa: Diego, Eyder, Pedro, Priscilla, Rafael, Renatha, Uéverton, Gilberto, Glaubos, Igor, Gleice, Marques, Sávio, Ítalo, Leonardo, Jean, Alberto, Eduardo, Edhelmira, Paulo. Se me senti bem acolhido nesse lugar com certeza foi por causa de cada um de vocês.

Aos demais professores e funcionários do IC/UFF que direto ou indiretamente contribuíram para tornar possível a concretização deste trabalho.

A CAPES pelo apoio financeiro, sem o qual seria difícil o andamento dos estudos.

Muito obrigado a todos!

Resumo

Redes sensores sem fio são sistemas distribuídos compostos de nós com bateria acoplada e baixos recursos computacionais. Assim como em muitos sistemas distribuídos, algumas aplicações de RSSF requerem sincronização entre seus nós. Para as RSSF em particular, os algoritmos de sincronização precisam respeitar as limitações computacionais. O protocolo FTSP, bem conhecido por conseguir precisão de nanosegundos com baixa sobrecarga. Contudo, ele depende de MAC *Timestamp*, um recurso não disponível em todos os dispositivos. Neste trabalho, nós propomos uma versão do FTSP independente de MAC *timestamp*, que estende o seu funcionamento para *hardware* que não tenha MAC *timestamp* enquanto mantém a baixa carga e alta precisão da sincronização, nossos resultados indicam uma média de erro de sincronização de $3\mu s$ por salto, enquanto adiciona uma mensagem de correção a mais no processo.

Palavras-chave: RSSF, Sincronização de Relógios, FTSP, MAC *Timestamp*.

Abstract

Wireless sensors networks are distributed system composed of battery-powered nodes with low computational resources. Like in many distributed systems, some applications of WSN require time synchronization among their nodes. In the particular case of WSN, synchronization algorithms must respect the nodes computational constraints. The well known FTSP protocol is famous for achieving nanosecond precision with low overhead. However, it relies on MAC timestamp, a feature not available in all hardware. In this work, we propose MAC timestamp independent version in order to extend and adapt FTSP to work on hardware that do not have MAC timestamp while keeping the low overhead and high synchronization precision our results indicate an average synchronization error of $3\mu s$ per hop, while adding a corretion message.

Keywords: WSN, Time Synchronization, FTSP, MAC Timestamp.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ADC	:	*Analog to Digital Converter*;
BFS	:	*Breadth-First Search*;
CSMA/CA	:	*Carrier Sense Multiple Access with Collision Avoidance*;
EEPROM	:	*Electrically Erasable Programmable Read-Only Memory*;
FTSP	:	*Flooding Time Synchronization Protocol*;
GPS	:	*Global Positioning System*;
IEEE	:	Instituto de Engenheiros Eletricistas e Eletrônicos;
LTS	:	*Lightweight Tree-Based Synchronization*;
MAC	:	*Media Access Control*;
NTP	:	*Network Time Protocol*;
RBS	:	*Reference Broadcast Synchronization*;
RSSF	:	Rede de Sensores Sem Fio;
SFD	:	*Start Frame Delimiter*;
TDMA	:	*Time Division Multiple Access*;
TEP	:	*TinyOS Enhancement Proposals*;
TI	:	Tecnologia da Informação;
TPSN	:	*Timing-sync Protocol for Sensor Networks*;
UTC	:	*Universal Time Coordinated*;
WSN	:	*Wireless Sensor Network*;

Sumário

Capítulo 1

Introdução

As Redes de Sensores Sem Fio (RSSF) são compostas de pequenos dispositivos equipados com uma antena de comunicação sem fio, um ou mais tipos de sensores, e uma CPU de baixa capacidade de processamento de dados [2]. Estes dispositivos geralmente são chamados de *motes*. Devido ao alcance do sinal de rádio limitado e da restrição energética uma RSSF tem limitações e características únicas que a difere de redes de computadores e sistemas distribuídos tradicionais [1].

As RSSF tem várias aplicações em diversos campos. A implantação de sensores em aplicações militares foi sempre muito difundido, de modo que a introdução de *motes* era uma incorporação natural para o avanço dos sistemas já utilizados. Aplicações que foram aprimoradas com o uso de RSSF incluem o rastreamento de inimigos e alvos [59], monitoramento de veículos [49], sistema contra atirador [47] e sistemas de vigilância [18]. Monitoramento ambiental também provê oportunidades para aplicação de redes sensores sem fio. Nosso meio ambiente tem uma gama muito grande de informações que desempenham um papel importante em nossa qualidade de vida, como a qualidade do ar, água, som e radiação solar que estamos expostos todos os dias e que diretamente afetam nossa saúde [40, 4]. Com o interesse cada vez maior na computação de larga escala nos leva as preocupações com o consumo de energia elétrica em instalações de TI [7], e as RSSF têm um papel estratégico no monitoramento e controle destes ambientes [61].

Dessa forma, faz-se necessário que essas aplicações implementem mecanismos de sincronização de tempo com boa precisão e escalabilidade, tudo isso em conformidade com os seus baixos recursos computacionais e disponibilidade energética [61, 47, 59]. Métodos de sincronização tradicionais, como NTP [37], amplamente utilizado em servidores e clientes de rede não se aplicam em RSSF devido a fatores não deterministas relacionados com o acesso ao meio na transmissão sem fio dos sensores. O Flooding Time Synchronization

Protocol (FTSP) é o algoritmo de sincronização de relógio mais popular para RSSF [33], ele é tolerante a falhas, consegue alta precisão ($\sim 1,5\mu s$ por salto) utilizando *timestamps* em camadas baixas da pilha de rádio, usa a técnica de regressão linear para compensar o escorregamento do relógio usando poucas mensagens pela rede. Contudo, *timestamps* na camada MAC não é um recurso padronizado, e portanto, não interoperáveis entre diferentes hardware e protocolos da camada física. Há um esforço do Google para padronizar *timestamping* na camada MAC em RSSF [58], mas até agora não há muita conformidade.

Muitos protocolos de sincronização usam MAC *timestamp*: alguns têm menos precisão do que o FTSP, eles se concentram em outros problemas e fazem suposições mais restritivas [15, 56]; outros podem conseguir uma melhor precisão entre os nós distantes [39, 28, 50]. Elson et ai. propôs o Reference Broadcast Synchronization [12] (RBS) para eliminar a incerteza do remetente sem MAC *timestamp* removendo o *sender* do caminho crítico. A ideia é que um terceiro irá transmitir um pacote do tipo *beacon* para todos os receptores. O *beacon* não contém qualquer informação de tempo; em vez disso os receptores irão comparar seus relógios um ao outro para calcular o escorregamento de seus relógios. Tem $\sim 30\mu s$ erro por salto e é independente da MAC *timestamp*. Ranganathan e Nygard oferecem uma boa visão geral desses protocolos [44].

1.1 Sincronização em Redes Sensores Sem Fio

Em sistemas distribuídos como as redes de sensores sem fio, cada nó tem seu próprio relógio e sua própria percepção de tempo. No entanto, uma escala de tempo comum entre nós sensores é importante para identificar relações entre eventos que estejam sendo monitorados, para apoiar a eliminação de dados redundantes de sensores e para facilitar a operação da rede. Uma vez que cada nó em uma rede de sensores opera de forma independente e conta com o seu próprio relógio, as leituras do relógio de diferentes nós de sensores também será divergente. Além destas diferenças aleatórias, a diferença entre os relógios de um conjunto de sensores vai aumentar ainda mais devido às taxas de escorregamento dos seus osciladores. Portanto, a sincronização dos relógios é necessário para assegurar que os tempos de detecção dos eventos que estejam sendo monitorados possam ser comparados e conseguinte significância.

Neste contexto, as técnicas de sincronização de tempo para redes com fios receberam uma quantidade significativa de atenção, esses métodos não são apropriados para uso em RSSF sem modificação, devido aos desafios colocados pelos ambientes sensores sem

fio. Estes desafios incluem o tamanho das redes de sensores, a necessidade de auto-configuração e robustez, a mobilidade dos sensores além da necessidade primordial de conservação de energia [11].

Nas RSSF a eficiência energética é uma necessidade básica para seu funcionamento, em uma rede de larga escala não é possível fornecer uma fonte de energia para toda a rede, esses sensores geralmente são acoplados a uma bateria. Devido ao tamanho dos dispositivos a quantidade de energia que eles podem produzir ou armazenar é muito limitada [53].

Sincronização de tempo é um serviço necessário para muitas aplicações e serviços em sistemas distribuídos em geral. Numerosos protocolos de sincronização de tempo têm sido propostos para ambos os sistemas com e sem fio, por exemplo, o Network Time Protocol (NTP) [37] é uma abordagem de sincronização escalável, robusto e auto-configurável amplamente utilizado. Especialmente em combinação com o Sistema de Posicionamento Global (GPS), tem sido utilizado para alcançar a precisão da ordem de alguns micros-segundos. No entanto, abordagens como NTP não são adequados para RSSFs devido a ser ineficiente neste contexto pois necessita de maior quantidade de memória. Já o uso de GPS pode elevar o custo de implantação de um sistema de monitoramento, além de requerer vários minutos para sintonizar, GPS também necessita de um uso maior de energia.

1.1.1 Conceitos Básicos

Os relógios dos *motes* nas RSSF são baseados em osciladores de quartzo, o tempo é mensurado nas oscilações do cristal em conjunto a um contador, este contador tem seu valor decrementado até atingir zero, então ele reinicia novamente para seu valor padrão e o processo continua indeterminadamente, toda vez que o contador chega a zero é gerado uma interrupção e essa interrupção é chamada de *tick* ou *clock tick*, ela incrementa outro contador o do relógio a nível de *software* que é de onde vem tempo que utilizamos no sistema operacional. Com isso, esse relógio de *software* fornece para o *mote* um valor de tempo local (*local time*).

Supondo a existência de dois nós, cada um com o seu próprio relógio R_a e R_b, onde $R(t)$ simboliza o valor do relógio no tempo t, sendo t o tempo de referência. Se verificarmos $R_a(t)$ e $R_b(t)$ podemos encontrar uma diferença entre os tempos, essa diferença é chamada de *clock offset*. Para corrigir esta diferença é necessário realizar uma sincronização para que os tempos sejam iguais ou que a diferença seja tão pequena quanto possível. Podemos

definir o *clock rate* como a frequência que um relógio progride, assim a frequência do tempo t de um relógio R_a é $R_a^{'}(t)$, ainda temos o *clock skew* que é a diferença em termo de frequência entre dois relógios [53].

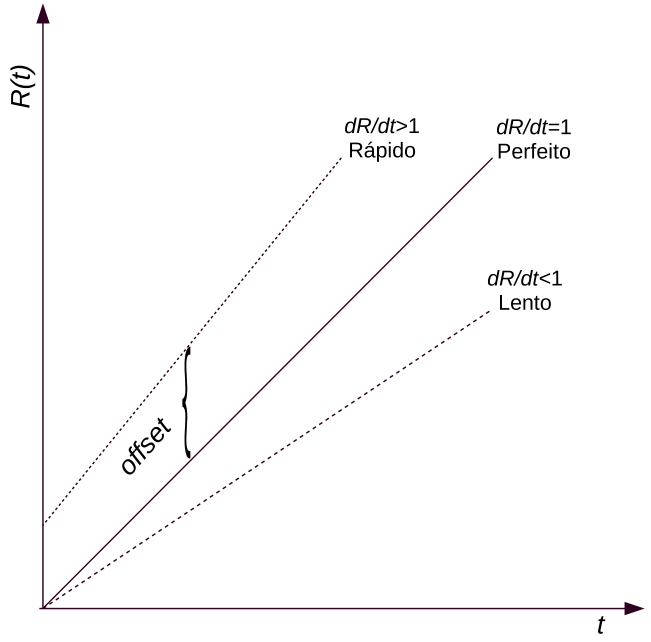


Figura 1.1: Comportamento do relógio em relação ao tempo de referência

Em um mundo perfeito $R_p(t) = t$ para qualquer instante de tempo t, o relógio perfeito deve ter a variação pelo tempo de referência iguais, como visto na Equação (1.1), onde os tempos não variam entre si. Vários fatores influenciam o *clock rate*, como por exemplo, temperatura e efeitos do ambiente, energia, limitação do hardware, impureza do cristal entre outros. Esta imprecisão do *clock* faz com que a taxa de alteração dos relógios os afaste do valor de referência, $dR/dt > 1$ resulta em um relógio rápido, já $dR/dt < 1$ apresenta um relógio lento, a Figura 1.1 ilustra esse efeito. Note que este comportamento é responsável pela instabilidade das leituras de tempo nos sensores, fazendo com que seja necessário realizar uma sincronização para corrigi-lo. Mesmo após a sincronização o relógio continuará se desviando, trazendo a necessidade realização de sincronizações periódicas.

$$\frac{dR}{dt} = 1 \tag{1.1}$$

A sincronização deve ocorrer de forma cíclica, pois os relógios sempre vão afastar-se dos valores de referência, mas o ajuste do tempo local deve ser realizado de forma gradativa e suave, ou seja, deve-se utilizar uma função para suavizar o declive do tempo local ao

invés de uma atribuição direta que poderia voltar ou adiantar várias unidades de tempo. Um dos problemas de efetuar uma mudança corrigindo o valor do tempo diretamente seria o relógio pulando para frente e para trás, ocasionando problemas como por exemplo, uma tarefa agendada poderia nunca ocorrer devido o tempo de sua execução nunca ter existido por causa do salto dos *ticks*, ou ainda o mesmo tempo poder ocorrer duas vezes em virtude de um salto para trás no relógio.

Podemos utilizar duas formas para classificarmos os tipos de sincronização: externo e interno. No método externo os relógios de todos os nós são sincronizados utilizando uma referência de tempo (*clock reference*) externa a rede, geralmente uma fonte de tempo UTC (*Coordinated Universal Time*), ou seja, o tempo do mundo real baseado em um relógio atômico. Por outro lado, na sincronização interna todos os nós são sincronizados sem a ajuda de uma referencia de tempo externa, o objetivo é minimizar a diferença entre os relógios locais dos sensores tendo assim uma noção consistente do tempo entre todos os nós da rede [26].

Um conjunto de fatores podem influenciar o funcionamento dos sistemas de sensores sem fio, ocasionando alterações no seu desempenho. Os principais componentes que devem ser observados são listados abaixo:

Efeitos Ambientais: Pressão, temperatura, umidade são algumas das condições de ambiente que alteram o funcionamento normal dos osciladores. Muitas RSSF são criadas para implantação *outdoor*, nestes cenários as condições são bem diferentes das apresentadas em um ambiente estável como um laboratório [25]. Fenômenos climáticos como chuva, neblina e vento também tem influência na performance de sistemas de RSSF, podendo até mesmo impossibilitar seu funcionamento [41].

Limitações Energéticas: Economizar energia é o maior interesse em RSSF, o tamanho pequeno dos nós sensores limita o tamanho de sua bateria, bem como a quantidade de energia que ele pode produzir usando placa solar [14]. A transmissão de mensagens é a base dos algoritmos de sincronização, a energia utilizada para a transmissão de 1Kb a uma distância de 100 metros é aproximadamente de 3 joules, o mesmo necessário para o processador executar 3 milhões de instruções [43]. Transmitir é mais custoso energeticamente do que computar, assim os algoritmos de sincronização devem minimizar a necessidade de troca de mensagens para serem mais eficientes energeticamente.

Wireless e Mobilidade: A comunicação sem fio apresenta grandes desafios nas redes de sensores, como vimos, é gasto mais energia transmitindo dados do que processando, fenômenos climáticos como chuva, neblina degradam a performance da rede. A caracte-

rística de transmissão de dados por difusão nas RSSF requer a utilização de um protocolo MAC (Media Access Control) para ordenar o acesso ao meio, dependendo da densidade da rede, faixa de alcance dos nós (entre 20-100 metros) e o tráfego dos dispositivos podem incorrer em interferências na comunicação que levam a perdas de mensagens e conexão intermitente. Grande quantidade de redes sensores são móveis, os nós podem se mover, podem ocorrer falhas na comunicação, a bateria pode esgotar e o nó ficar inativo, causando constantes mudanças na topologia das redes. Por consequência destes problemas os protocolos de sincronização devem ser projetados para respeitar todas essas propriedades.

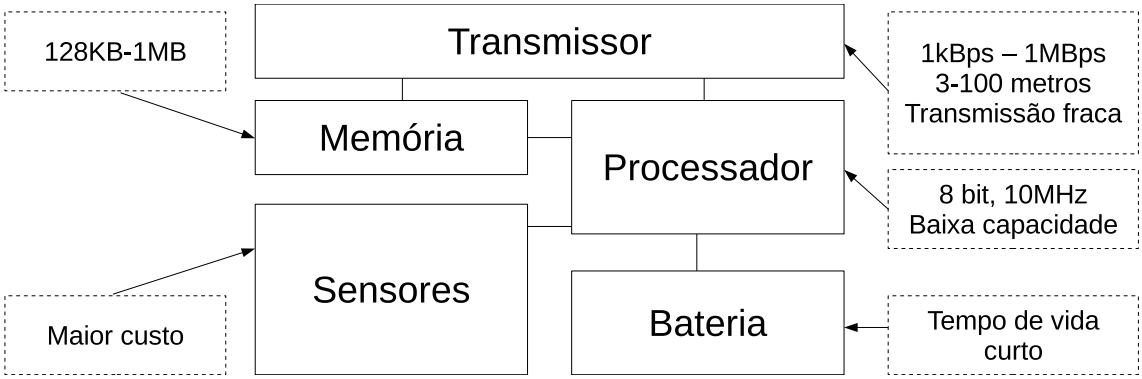


Figura 1.2: Componentes de um mote para RSSF [19]

Limitação de *Hardware*: O tamanho e o preço dos sensores são algumas das razões pelas quais seu *hardware* ser bastante limitado, aumentar o tamanho poderia elevar seu custo e consumo de energia, podendo tornar inviável as aplicações de redes sensores em larga escala [24]. A capacidade computacional destes dispositivos é similar aos computadores pessoais do início da década de 80, a Figura 1.2 ilustra os componentes típicos de um nó sensor, transmissor fraco sujeito a perdas, memória limitada, processador de baixa capacidade aliado a fonte de energia restrita. Os algoritmos para RSSF devem respeitar esse conjunto de desafios e limitações com o objetivo de minimizar o uso dos recursos e manter seus resultados funcionais.

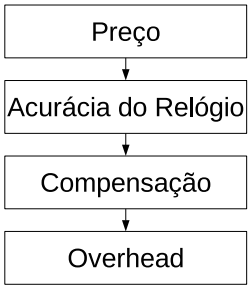


Figura 1.3: Tradeoff custo-beneficio [20]

Os avanços recentes nos sistemas integrados nos permitem desenvolver dispositivos

sensores com baixos recursos de energia, rádio e processamento. Considerando que uma rede de sensores consista de um bom número de sensores, o valor de um só nó afeta a implantação da rede toda. Existe uma tendência defendida por pesquisadores de que com a grande quantidade de sensores aplicados hoje em dia, a produção em larga escala faça com que seu valor possa chegar a ser menor que $1 dólar [8, 57] por unidade.

De maneira geral, os *motes* são fabricados com materiais de baixo custo e as vezes de menor qualidade, como por exemplo, um cristal de quartzo de baixa qualidade pode ser um dos fatores que tornam o relógios mais imprecisos. A Figura 1.3 apresenta o *tradeoff* entre o preço e a qualidade. Observe que o preço de produção do dispositivo determina a precisão do seu relógio, assim, quanto menos preciso o relógio é maior compensação será necessária para corrigir seu tempo, a compensação gera maior *overhead* de comunicação que é gasto no uso de comunicação/energia.

1.1.2 Fontes de Imprecisão em Comunicação

A troca de mensagem baseada em *timestamp* (marca de tempo) é a principal ferramenta dos protocolos de sincronização, porém o não determinismo causado pelas latências no processo de troca de mensagens interfere na precisão que pode ser obtida utilizando este método. Por exemplo, quando um nó envia uma mensagem com seu *timestamp* para outro, os diversos procedimentos pelo qual essa informação vai percorrer até o destino apresentam um custo de computação e tempo para ser realizada, assim quando se dá a recepção do dado no destino já se passou uma quantidade determinada tempo, o relógio do nó origem continuou rodando e o *timestamp* enviado está inconsistente com o valor atual da fonte. A latência deste procedimento pode ser decomposta em alguns componentes, descritos por Kopetz e Ochsenreiter [27] e classificados a seguir:

- Tempo no envio: É o tempo que o sistema leva para montar a mensagem e passar a requisição de transmissão para a camada de acesso ao meio. Fonte de atraso causado principalmente pelo sistema operacional e da carga de uso do CPU.
- Tempo de acesso ao meio: Ocorre devido a janela de contenção do acesso ao meio, é o tempo que o nó leva para ter acesso físico para transmissão esperando o canal estar desocupado para então para usá-lo. Redes sem fio compartilham o mesmo meio de comunicação necessitando de uma forma de coordenar quem pode usar o canal e assim evitar colisões. O protocolo MAC do 802.11 serve como base para as variações utilizadas em RSSF, estes buscam atingir maior eficiência no consumo de

energia, algumas opções destes protocolos são: BMAC (*Versatile low power media access*) [42], S-MAC (*Sensor-MAC*) [60], Box-MAC [38], PAMAS (*Power-Aware Multi-Access Protocol*) [48] e para uma especificação mais detalhada e atualizada desses protocolos podem ser obtidas em [54, 22].

- Tempo de transmissão: Tempo que leva para um nó transmitir uma mensagem para outro nó, varia dependendo do tamanho do pacote e da velocidade da banda.

- Tempo de propagação: Tempo que a mensagem leva para chegar no receptor desde o momento que deixou o emissor. Este tempo é muito pequeno e geralmente insignificante para a análise geral do atraso.

- Tempo de recebimento: É o tempo que o receptor leva para receber a mensagem, sendo igual ao da transmissão.

- Tempo de recepção: Similar ao tempo de envio, é o *delay* de receber a mensagem e encaminha-la para o nível da aplicação.

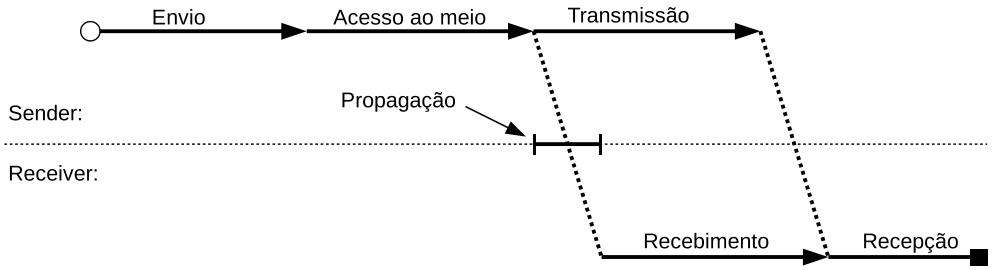


Figura 1.4: Decomposição das fontes de atraso na transmissão de uma mensagem

A Figura 1.4 ilustra as tomadas de tempo que compõem as fontes de atraso na troca de mensagem. As principais técnicas de sincronização concentram-se em eliminar esses *delays* ou ainda computar seus respectivos atrasos para usá-los mais a frente na forma de compensação. Parte desses trabalhos expandiram ainda mais a compreensão dos componentes de atraso [33, 21, 16]. Tais componentes estão listados abaixo:

- Tempo de tratamento de interrupções: O tempo que o microcontrolador demora para responder a solicitação do *chip* de rádio.

- Codificação e Decodificação: Codificação é o tempo que o transmissor tira para codificar os *bits* nas ondas de rádio. E decodificação o tempo que o receptor demora até transformar as ondas eletromagnéticas em *bits*.

- Alinhamento de Byte: Diferentes tipos de alinhamento entre o transmissor e receptor, causam a necessidade de reordenar os bytes. Alguns *chips* de rádio não são capazes de resolver o alinhamento de um fluxo de mensagem, o receptor deverá resolver o *offset* da mensagem.

Tempo	**Magnitude**	**Distibuição**
Enviar e Receber	$0 - 100\ ms$	Não determinístico, depende da carga do CPU
Acesso ao Meio	$10\ ms - 500\ ms$	Não determinístico, depende da contenção do canal
Transmissão e Recepção	$10\ ms - 20\ ms$	Determinístico, depende do tamanho da mensagem
Propagação	$< 1\mu s$ para distâncias acima de 300m	Determinístico, depende da distância
Tratar Interrupção	$< 5\mu s$ na maioria dos casos, mas pode chegar a 30 μs	Não determinístico, depende das interrupções
Codificar Decodificar	$100\ \mu s - 200\mu s$, $< 2\mu s$ de variância	Determinístico, depende o chip de rádio
Alinhamento de Bytes	$0 - 400\mu s$	Determinístico, pode ser calculado

Tabela 1.1: Fontes de atraso na transmissão de mensagens [33]

Alguns dos componentes de atraso são de origem não determinístico, outros podem ser calculados e estimados, a Tabela 1.1 sintetiza o tamanho e classificação dos componentes já comentados. Note que nem todos os transmissores são afetados por todas essas fontes de erro.

1.1.3 Mensagens de Sincronização

Os protocolos de sincronização são essencialmente construídos a partir de troca de mensagens entre os nós participantes da rede. O modelo mais simples de sincronização utiliza pelo menos uma mensagem, chamado de *pairwise synchronization* nele dois nós podem sincronizar apenas trocando uma mensagem, sendo que para sincronizar uma rede inteira repete-se a operação entre parelhos até que todos os pares estarem sincronizados, quando esse processo é concluído, têm-se o modelo *network-wide synchronization*. A seguir detalha-se os métodos de sincronização.

A sincronização unidirecional ou *one-way* é a forma mais elementar de sincronização *pairwise*, esse procedimento é ilustrado na Figura 1.5. Observe que o nó i no tempo t_1

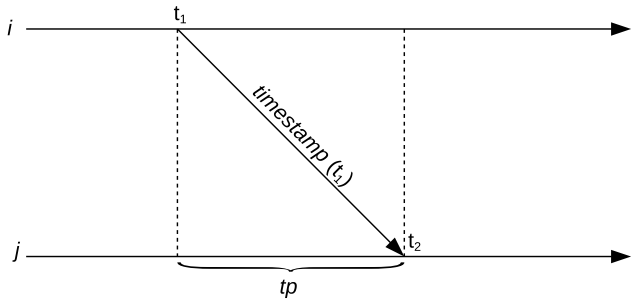


Figura 1.5: Unidirecional *pairwise synchronization*

envia seu *timestamp* para j, quando o nó j recebe a mensagem seu relógio local está marcando o tempo t_2, assim o valor de t_2 pode ser calculado como na Equação (1.2):

$$t_2 = t_1 + tp + \theta \tag{1.2}$$

$$t_j - t_i = \theta + tp \tag{1.3}$$

Onde tp é o tempo de propagação, este tempo é muito pequeno podendo ser ignorado. O *delay* de propagação para uma distância de 30 m é 10^{-7}s [25]. O θ é a diferença entre os tempos de i e j, ele indica o *offset* dos relógios. Com essas informações o nó j pode calcular o *offset* e ajustar seu relógio com base no tempo de i, de acordo com a Equação (1.3).

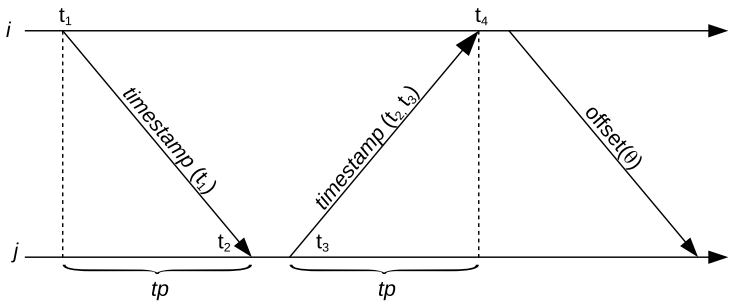


Figura 1.6: Bidirecional *pairwise synchronization*

A sincronização bidirecional, *two-way* ou ainda denominada emissor-receptor (*sender-receiver*) usa duas mensagens de sincronização. A Figura 1.6 apresenta o comportamento desta técnica. Note que primeiro o nó i envia seu tempo t_1 para j, este recebe a mensagem no tempo t_2 e registra seu tempo local. Assim, o tempo t_2 já pode ser calculado como na Equação (1.2). No próximo passo o nó j no tempo t_3 envia uma mensagem para i com os

timestamps t_2 e t_3, então no instante t_4 tanto o nó i como o j são capazes de calcular o *offset* entre ambos, porém o nó i é capaz de determinar mais precisamente tanto o *offset* quanto o tempo de propagação, como segue:

$$tp = \frac{[(t_2 - t_1) + (t_4 - t_3)]}{2} \tag{1.4}$$

$$\theta = \frac{[(t_2 - t_1) - (t_4 - t_3)]}{2} \tag{1.5}$$

O nó i com uma informação mais acurada, pode enviar uma nova mensagem para j com o valor do θ. A sincronização bidirecional tem a vantagem de moderar as incertezas na pilha de protocolos e no atraso de propagação usando troca de mensagens e a desvantagem de necessitar de um número adicional de troca de tráfego.

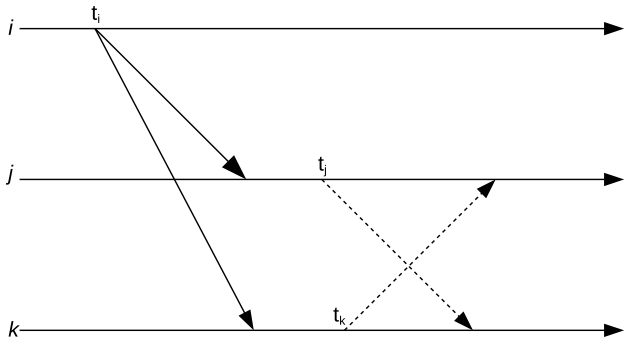


Figura 1.7: Sincronização *Receiver-receiver*

A sincronização receptor-receptor ou *receiver-receiver* utiliza das características inerentes ao *broadcast* na comunicação sem fio, onde a difusão de uma mensagem tem o mesmo tempo de recepção em cada um dos receptores, seu tempo de chegada varia muito pouco sendo sensível apenas aos atrasos de propagação e recebimento.

No cenário descrito na Figura 1.7, têm-se um nó que envia *beacons* (não necessariamente contendo um *timestamp*) periodicamente, todos nós que recebem a mensagem registram o tempo de recebimento com o seu relógio local. Então, os *receivers* (j e k) trocam entre si a informação registrada, o que os permite calcularem a diferença entre o tempo de chegada dos *beacons*, ou seja, o *offset*. Esse tratamento pode remover duas das maiores fontes de atraso na comunicação sem fios: o atraso de envio e o tempo de acesso ao meio [6].

1.2 Definição do Problema

Alguns tipos de rádios tem embutido *chips* mais sofisticados capazes de coordenar a transmissão sem sobrecarga da CPU. Esses rádios tem a capacidade de acessar a fila de transmissão a qualquer momento, por exemplo, um dos *chips* mais comuns aplicados a RSSF o *Chipcon* CC2420 [5] que utiliza o padrão IEEE 802.15.4 tem essa característica, ele tem um recurso chamado SFD (Start of Frame Delimiter) que acessa a fila de transmissão e permite gerar interrupções durante o envio e recebimento dos dados, possibilitando o seu emprego em mensagens de sincronização mais refinadas realizadas na camada MAC.

A marca de tempo das mensagens na camada de acesso ao meio reduz o atraso relacionados as incertezas de comunicação, neste modelo é possível inserir o *timestamp* na mensagem no momento da transmissão e imediatamente no recebimento, com o *MAC timestamp* é possível eliminar as fontes de atraso na entrega da mensagem.

O FTSP é o estado da arte dos protocolos de sincronização *multi-hop* [50], e um dos mais populares [29, 20, 45] e vem disponível por padrão no sistema operacional TinyOS. Uma de suas principais características é o uso de timestamp na camada MAC. Se uma plataforma particular não suportar a mudança do conteúdo da mensagem depois de ocorrido o evento de sincronização, então o FTSP não pode ser implementado nessa plataforma. Assim o protocolo é fortemente dependente de recursos específicos no *chip* de rádio.

1.3 Motivação do Trabalho

Redes sensores são compostas de um grande número de nós. Assim, torna-se fundamental que o preço de um único *mote* não seja muito alto para não comprometer o valor da rede inteira, o que pode inviabilizar a implantação de um projeto. Neste sentido, os componentes do nó sensor podem se tornar cada vez mais simples ou menos especializados, como por exemplo o transmissor de rádio, este pode não contar com uma interface tão rica de recursos como MAC *timestamp* e acesso randômico ao FIFO durante a transmissão. Diante das restrições originais apresentadas pela implementação padrão do protocolo FTSP, no que tange a exigências específicas de *hardware*, além da imposição de modelos restritivos de sincronização inerente as RSSF, buscou-se uma abordagem que pudesse ser livre de requisitos específicos e simples de acoplar ao sistema. Uma abordagem independente de *hardware* para o FTSP poderia torná-lo apropriado para uma maior quantidade de dispositivos.

1.4 Contribuição da Dissertação

Baseado no problema descrito na Seção 1.2, este trabalho propõe a extensão do FTSP para funcionamento independente de *hardware*, o FTSP+. Ele fornece uma extensão para o funcionamento do FTSP em plataformas que não suportam a implementação do *timestamp* na camada MAC, visto que ainda é possível fornecer a funcionalidade de sincronização a nível de pacote, com um baixo custo de *overhead* de comunicação. A implementação do FTPS+ ainda etá de acordo com o padrão de sincronização formalizado pelo sistema TinyOS na sua TEP:133 [55].

1.5 Organização do Documento

Esta dissertação é composta por sete capítulos, os assuntos discutidos neste trabalho estão organizados conforme mostrado a seguir.

- **Capítulo 2:** Apresenta e descreve os principais protocolos de sincronização, detalhando seu funcionamento principalmente ao que tange os seus mecanismos de mensagens e correção de relógio, suas vantagens e desvantagens. No final traz uma classificação dos protocolos os agrupando de acordo com seus modelos de troca de mensagens.
- **Capítulo 3:** É realizado o detalhamento do protocolo FTSP, que serviu de base para a implementação da proposta apresentada nesta dissertação. Detalha-se ainda seus principais componentes com o foco de posteriormente mostrar as modificações.
- **Capítulo 4:** Apresenta o FTSP+, descreve o conceito da técnica que faz a estimativa de atraso que possibilita a substituição do MAC *timestamp*, por um a nível de aplicação.
- **Capítulo 5:** Apresentamos as ferramentas de software e os conceitos por traz do desenvolvimento do FTSP+. O sistema operacional TinyOS e a linguagem de programação nesC.
- **Capítulo 6:** Neste capítulo, descrevemos os experimentos realizados e os valores, métricas, ambientes e resultados.
- **Capítulo 7:** Conclusão acerca do trabalho baseado nos experimentos e resultados, trata do produto da dissertação e a aplicação da técnica em trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Existem diversos protocolos de sincronização de tempo disponíveis para redes sensores sem fio, a maior parte deles seguem as abordagens descritas no Capítulo 1. Neste capítulo é apresentado uma breve explicação de alguns protocolos proeminentes para sincronização de redes sensores sem fio.

2.1 RBS

Elson et al. [12] propuseram o *Reference Broadcast Synchronization* (RBS) em 2002, baseado no modelo bidirecional (*receiver-receiver*). Entre suas principais características têm-se a inovação quanto à utilização de mensagens de referência por difusão para a eliminação do não determinismo de comunicação relacionado ao tempo de envio e acesso ao meio, baseando no fato de que as mensagens de *broadcast* chegam nos receptores praticamente ao mesmo tempo.

O Algoritmo 2.1 exemplifica o funcionamento do RBS, usando um cenário com *receivers* onde um nó é determinado como referência. O nó eleito irá enviar por difusão uma mensagem de requisição para os nós ao seu alcance. Cada nó que recebe a requisição irá guardar seu tempo local. Após, os pares de receptores trocam seus *timestamps*, eles podem comparar seus tempos e calcular a diferença entre seus relógios (*offset*). O escorregamento do relógio será calculado através do método dos mínimos quadrados para encontrar uma estimativa da inclinação ou escorregamento do *clock* do outro nó.

Algoritmo 2.1: `RBS Passos`

1 Primeiro, o transmissor envia uma mensagem de referência por *broadcast*.
2 Cada *receiver* registra seu tempo de quando recebeu a mensagem de referência.
3 Pares de *receivers* trocam os seus registros de tempo de recebimento da mensagem.
4 O *offset* entre um par de *receivers* é a diferença entre seus relógios locais.

O RBS tem um alto nível de economia de energia, devido a forma com que o mesmo gerencia o processo de sincronização, atuando somente quando necessário. Este modelo de funcionamento é denominado sincronização *post-facto*, nele os nós correm seu relógio naturalmente e os tempos dos desvios dos seus relógios só serão calculados na ocorrência de um evento importante, assim o *timestamp* resultante é vinculado ao evento e não altera o relógio local. A correção do relógio é feita utilizando uma tabela com a informação dos valores de *offset* e *skew* de todos os pares ao seu alcance, e a cada mensagem recebida de outro *receiver* o valor é convertido para uma escala de tempo local.

A sincronização em cenários com multi-saltos é possível de ser implementada utilizando o RBS. Neste ambiente, apenas um nó como referência pode não ser o suficiente para cobrir todo o domínio de abrangência da rede, então múltiplos nós de referência podem ser utilizados, cada um com seu domínio de cobertura. Na verdade, a transmissão com vários saltos poderia acarretar um atraso na propagação de uma mensagem. Neste cenário, para tratar este problema, a sincronização de dois nós em dois domínios diferentes é realizado por um terceiro nó localizado na intersecção dos domínios.

Dentre as principais características do protocolo RBS, listamos as seguintes como vantagens:

- Remove as duas maiores fontes de atraso, tempo de envio e tempo de acesso.
- Sincronização *post-facto* realiza ajustes somente quando necessário, diminuindo o custo de energia.

Podemos destacar como desvantagens as que seguem:

- Nó que envia o *beacon* nunca é sincronizado. Impossibilita a utilização em aplicações que necessitem sincronização do nó raiz.
- Em redes de um salto com n nós, este protocolo precisa trocar $O(n^2)$ mensagens. Em caso de redes com grandes vizinhanças pode ter alto custo de computação.
- Alto número de troca de mensagens do protocolo pode aumentar o seu tempo de convergência.

- Não é escalável, com o aumento do número de nós o seu desempenho cai de forma significativa.

2.2 LTS

O protocolo *Lightweight Tree-Based Synchronization* (LTS) proposto por Van Greunen e Rabaey [56] apresenta duas abordagens, uma centralizada e outra distribuída ambas multi-saltos. O algoritmo segue o modelo emissor-receptor ilustrado anteriormente na Figura 1.6, com esquema de sincronização *pairwise* e faz uso de apenas 3 mensagens para sincronizar um par de nós.

A versão centralizada é uma extensão do exemplo simples de um salto. Nessa abordagem têm-se um nó de referência que é a raiz de uma árvore que comporta todos os elementos da RSSF. Nas instruções iniciais do algoritmo é realizado a construção de uma árvore geradora T que contém todos os nós, na qual a profundidade deve ser minimizada, visto que de cada nível da árvore introduz um salto na rede e cada salto resulta em acúmulo de erro das trocas de mensagens realizadas. Toda vez que o algoritmo é executado, a árvore é reconstruída. Uma vez construída a árvore, o nó de referência inicia o processo realizando sincronização *pairwise* com cada um dos seus nós filhos em T. Uma vez sincronizado os nós filhos, eles repetem o processo com seus subsequentes até que todos os nós estejam sincronizados. O tempo para o algoritmo convergir é proporcional a profundidade da sua árvore.

Na versão distribuída do LTS não realiza a construção da árvore geradora, uma vez que a sincronização não é mais responsabilidade apenas do nó de referência e sim dos próprios nós. Neste modelo existe um ou mais nós de referência, eles são requisitados pelos nós sensores a qualquer momento que esses precisem sincronizar bem como a frequência de ressincronização. Para os nós determinarem suas taxas de sincronização é preciso reunir os seguintes parâmetros: precisão necessária, número de saltos até o nó de referência, taxa de escorregamento do relógio e o tempo desde a última sincronização. Só então ele calcula sua taxa de sincronização. Desta forma, quando o nó determinar que precisa sincronizar fará solicitação de ressincronização ao nó de referência mais próximo.

Dentre as principais características do protocolo LTS, listamos as seguintes como vantagens:

- Usa poucos recursos computacionais, como memória e CPU.

- Suporta sincronização *post-facto*.
- Robusto a variação de canais, mudanças na topologia, tamanho e mobilidade da rede.
- Na versão distribuída, certos nós necessitam sincronizar menos frequentemente.

Os seguintes itens são as desvantagens:

- A precisão da técnica diminui de forma linear em relação a profundidade da árvore.
- Sensível a falhas do relógio ou informação errada da sub-rede.

2.3 TPSN

O TPSN (Timing-sync Protocol for Sensor Networks) introduzido por Ganeriwal et. al. em 2003 [16], usa o modelo *sender-receiver* e organiza a rede na estrutura de uma árvore, um único nó sincroniza toda a rede, para reduzir incertezas relativas a acesso ao meio TPSN utiliza *timestamp* na camada MAC. É dividido em duas fases descritas a seguir:

Fase de Descoberta de Nível: Esta fase é responsável por criar uma topologia hierárquica da rede, inicia no momento que a rede é ligada definindo um nó como *root* e atribuindo seu nível como 0. O nó *root* envia uma mensagem de descoberta chamada `level_discovery` que contém seu nível e identificador, todos os nós vizinhos que recebem este pacote usam ele para identificar seu nível, somando 1 ao nível do pacote recebido. Então estes vizinhos imediatos ao *root* reenviam a mensagem de descoberta com seu próprio identificador e nível, este processo repete-se até que eventualmente todos os nós tenham identificado seu próprio nível. Caso um nó não tenha identificado seu nível, seja por problema de erro na troca de mensagem ou por ter ingressado na rede após a fase de descoberta ter sido concluída, este nó pode enviar uma mensagem de `level_request` e seus vizinhos irão responder com seus respectivos níveis. Então ele atribui seu nível como sendo 1 a mais do que o menor nível dentre os recebidos. Uma falha importante que dever ser mencionada é o caso da reeleição do nó *root*, caso este venha a cair, nesta hipótese um dos nós do nível 1 é eleito e dá início a uma nova fase de descoberta.

Fase de Sincronização: Com a estrutura hierárquica da rede criada na fase anterior, temos o início da sincronização propriamente dita, onde o TPSN usa sincronização *pairwise* entre as arestas e funciona de forma similar ao LTS, iniciando a partir do *root* e avançando até os níveis mais externos da rede. Primeiramente o nó *root* envia uma

mensagem `time_sync`, quando o nó no nível 1 recebe a mensagem eles iniciam uma sincronização bidirecional, ao final ele será capaz de calcular seu escorregamento, *offset* e o tempo de propagação e assim ajustar o seu relógio. Os nós no nível 2 são capazes de escutar a comunicação realizada pelo nível 1, assim aguardam um tempo de *backoff* para iniciar o mesmo processo de sincronização, este tempo é necessário para que os nós terminem de ajustar seus relógios. Este processo continua nos níveis subsequentes até que toda estrutura esteja sincronizada.

O TPSN trouxe importantes contribuições para os protocolos de sincronização de RSSF, algumas de suas principais vantagens são listadas a seguir:

- O protocolo suporta sincronização *post-facto*, como forma de economia de energia.
- O TPSN foi desenvolvido para ser um protocolo multi-salto, assim teoricamente o tamanho da faixa de transmissão não causa problema.
- Tem incerteza no tempo de envio, ele tenta reduzir este indeterminismo usando *timestamp* na camada mac

A seguir alguns dos pontos fracos do protocolo:

- Autores não explicaram como o protocolo poderia suportar *sleep mode* sem quebrar a estrutura da árvore.
- Autores concluem que TPSN tem performance 2 vezes superior ao RBS, porém nos testes é apresentado o *offset* máximo do TPSN foi de 17 μs, maior que os 3 μs do RBS.
- Mobilidade pode causar a necessidade de reconstrução da árvore.
- Depende de *hardware* específico para o funcionamento do *timestamp* na camada MAC.
- O Tempo de sincronização aumenta com o com o crescimento do número de níveis.

2.4 PulseSync

Lenzen *et al.* em [28] introduziram o PulseSync, e expandiram em [29]. A ideia básica do PulseSync é distribuir a informação dos valores dos relógios da forma mais rápida possível

usando o menor número de mensagem. Na inicialização da rede os nós escutam o canal esperando por mensagens de sincronização, no em tando, passado um certo período de tempo se não houver nenhum recebimento ou se ele tem o ID menor o nó se declara como *root* e começa a enviar periodicamente mensagens que são chamadas de pulsos. Esses pulsos são mensagens de *broadcast* com as seguintes informações: *timestamp*, número de sequencia e o ID do *root*.

Depois de passar da fase inicial, onde é definido o *root*, os nós vizinhos a ele começarão a receber mensagens de sincronização. Os relógios dos nós são sincronizados através da rede utilizando *root* como nó de referência, uma vez que um nó adjacente tenha recebido uma mensagem de sincronização ele encaminha essa mensagem adiante.

Em uma topologia em que determinado nó tenha muitos vizinhos, ele deve receber mensagens de sincronização repetidas, então ele só irá utilizar e encaminhar a que chegou primeira, pois essa provavelmente tem o menor caminho até o *root*, logo sofreu menos com o atraso introduzido pelo percurso da mensagem. Como forma de manter o menor número de saltos o Pulsesync utiliza busca em largura (*Breadth-First Search* - BFS).

Como forma de diminuir os atrasos referentes a comunicação, o Pulsesync utiliza *timestamp* na camada MAC. Para transmitir rapidamente os pulsos através da rede, os nós começam a retransmitir as mensagens tão logo elas cheguem, o que pode causar colisões devido as interferências do meio sem fio. Existem algumas técnicas que trabalham em como melhorar a difusão de mensagens por inundação evitando a interferência [32, 52, 62], o Pulsesync não implementa essas soluções apenas define o intervalo de separação dos pulsos para evitar a colisões. Para correção do escorregamento do relógio é utilizado regressão linear para corrigir o declive do crescimento do relógio. O pseudocódigo código do nó *root* e do nó cliente é descrito pelos Algoritmos 2.2 e 2.3, respectivamente.

Algoritmo 2.2: `Rotina periódica sempre no intervalo do` *`Beacon`* `B`

```
Data: R(t) = relógio local
1 se Root então
2 |   Aguarda intervalo de sincronização
3 |   Envia(R(t), seqNum)
4 |   seqNum ← seqNum + 1
```

Algoritmo 2.3: Nó não *root*

```
1 se Não Root então
2     Recebe pulso
3     Armazena(pulso)
4     Delete(entradas velhas)
5     Aguarda backoff para reencaminhar pulso
6     Envia(R(t))
7     seqNum ← seqNum + 1
```

O PulseSync se apresenta como um protocolo estado da arte em sincronização de redes sensores, tem como principais vantagens os seguintes itens:

- Algoritmos assintoticamente ótimo, tem convergência rápida.
- Apresenta bom custo benefício entre escalabilidade e eficiência energética.
- Diminuiu a propagação de erro em redes multi-saltos.

Suas desvantagens são:

- Dependente de *hardware* específico para *timestamp* na camada MAC.
- Nó *root* é um ponto único de falha.

2.5 FTSP

O Flooding Time Synchronization Protocol (FTSP) [33] é um protocolo desenvolvido para ser uma solução de sincronização de RSSF que atenda as necessidades de escalabilidade, robustez a mudanças na topologia da rede relacionadas a falhas de nós, enlace e mobilidade. O FTSP combinou os progressos observados no RBS e TPSN, acarretando em um protocolo mais dinâmico, implementando melhorias no *timestamp* a nível de MAC além de prover correção de relógio.

Primeiro, no processo de inicialização da rede o nó com o menor ID é eleito como *root*, o FTSP cria uma estrutura *ad hoc* com o nó *root* propagando periodicamente informação de tempo. Em seguida, a partir do nó *root*, a mensagem de sincronização é transmitida para todos os seus vizinhos contendo o tempo do seu relógio local. Além disso, elimina parte das incertezas na transmissão fim-a-fim usando *timestamp* na camada MAC. No que tange ao nó receptor, são armazenados os tempos de chegada usando seu relógio local. Após armazenar uma quantidade predefinida de mensagens de sincronização, os nós usam

regressão linear para calcular o seu *offset* e o escorregamento do seu relógio com base nas mensagens do nó de referência. Assim, o nó pode mudar o declive do crescimento do seu relógio e mantê-lo atualizado.

De maneira geral, os nós recebem sincronização direto do nó *root*, porém nem todos estão no seu raio de alcance. De fato, a sincronização multi-saltos é realizada por outros nós que já tenham sido sincronizados, estes encaminham mensagens até todos estarem sincronizados. Em redes densas e com saltos, temos o problema de que determinado nó receba mensagens de sincronização repetidas, neste caso o FTSP incorpora na mensagem além do *timestamp*, um número de sequência e o ID do *root*, o que facilita o descarte das mensagens mais antigas.

Definimos algumas de suas vantagens de forma breve como:

- O algoritmo estado da arte em sincronização de relógios em redes sensores sem fio.
- Resiliente as mudanças da topologia da rede.
- Convergência rápida do tempo global.
- Regressão linear para tratar o escorregamento do relógio.
- Algoritmo simples de eleição de líder.

As suas desvantagens são:

- Exibe crescimento de erros baseado no tamanho de saltos da rede.
- Dependente de *hardware* específico para funcionamento do seu *timestamp*.

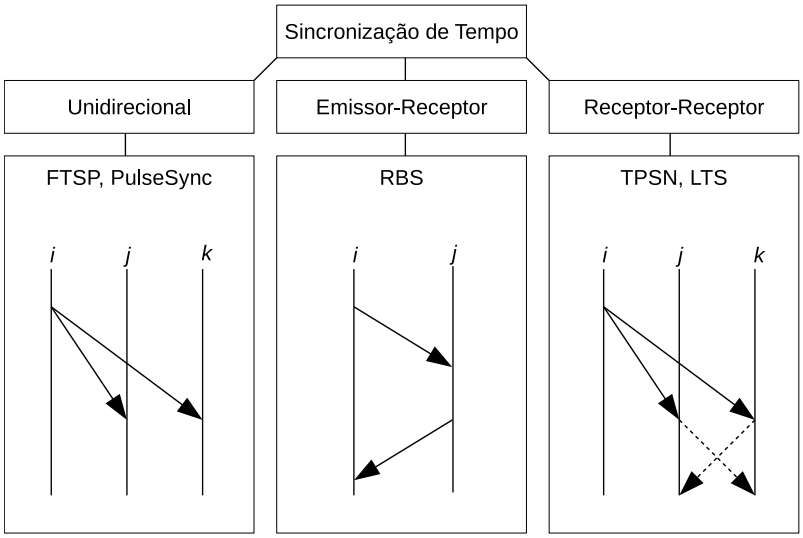


Figura 2.1: Classificação dos protocolos de sincronização

A Figura 2.1 segmenta os protocolos descritos neste capítulo baseado nos modelos de troca de mensagens descritos na Seção 1.1.3. No próximo capítulo têm-se a descrição mais detalhada do funcionamento do FTSP, que é a principal fonte de interesse deste trabalho.

Capítulo 3

Revisão detalhada do FTSP

Neste capítulo iremos descreve-se o funcionamento do FTSP, principalmente no que se refere aos esquemas de sincronização com o nó *root*, fase de eleição e reeleição do líder, envio periódico de *beacons*, marcação de tempo na camada de acesso ao meio e o uso de regressão linear para corrigir o escorregamento do relógio.

3.1 Flooding Time Synchronization Protocol

O FTSP surgiu em 2004 [33] tendo como principal meta efetuar a sincronização dos relógios de todos os participantes de uma rede, oferecendo boa performance, simplicidade e baixo custo computacional. Essas características visam atender as restrições que os componentes das RSSF possuem, onde todos os nós sofrem de erros ocasionados pela impureza do cristal, bem como, erros inerentes ao enlace de comunicação sem fio não confiável.

A partir do uso de uma única mensagem o FTSP sincroniza múltiplos receptores. Neste processo, a mensagem com a marca de tempo é gravada no instante em que está sendo enviada, que também é praticamente o mesmo momento em que esta sendo marcada como recebida pelo receptor. Este mecanismo de *timestamp* na camada MAC elimina a maior parte das fontes de erros na etapa de comunicação comentados na Seção 1.1.2 e é empregado em muitos outros protocolos de sincronização [15, 28]. O trabalho [13] trata do uso de regressão linear para a correção do escorregamento do relógio, que é o usado pelo FTSP para compensar o seu relógio e assim manter um alto nível de acurácia.

O nó *root* é o responsável por disseminar o tempo global pela rede, porém há redes em que os nós não estão ao alcance da faixa de cobertura do seu sinal. O FTSP resolve esse

problema, pois dispõe do recurso de rede multi-saltos, pelo qual constrói uma estrutura *ad hoc*. Nessa nova estrutura, os nós diretamente sincronizados com o *root* assim que ficam atualizados passam a sincronizar os nós subsequentes com sua informação de tempo global.

3.2 *Timestamp*

O FTSP usa *broadcast* para sincronizar seus nós, essa mensagem contém um *timestamp* com o valor estimado do tempo global no momento em que a transmissão é realizada. No instante do recebimento os nó obtém o valor do tempo do seu relógio local, assim com a diferença entre o tempo global e o local o nó pode estimar o seu *offset*.

Caso a leitura do tempo seja armazenada na etapa de envio ainda no nível de aplicação, vários componentes de atrasos serão acumulados até que a mensagem seja lida pelos receptores, a Seção 1.1.2 lista esses componentes de atraso e a Tabela 1.1 quantifica a magnitude dessas fontes de imprecisão. Esse procedimento é ilustrado na Figura 3.1. Note que no tempo t_0 é iniciado o processo de envio da mensagem de sincronização a partir do nó i, logo em seguida é criado o pacote que armazena o *timestamp* t_1 e comanda o envio para o sistema operacional, que por sua vez, encaminha a mensagem para a pilha de protocolos. O tempo de acesso ao meio é variável e depende da janela de contenção da rede. Quando i ganha o direito de enviar a mensagem, dá-se inicio a transmissão em t_2, em t_3 o nó j termina de receber a transmissão, mas somente algum tempo depois o nó j gera uma interrupção e o sistema operacional irá armazenar o *timestamp* da recepção em t_5. Neste momento j tem como calcular seu *offset*, porém esse valor não é representativo devido aos atrasos que as marcas de tempo sofreram em relação de seus valores ideais.

Visando corrigir esse problema, a leitura dos tempos pode ser efetuada em locais mais estratégicos. Vários transmissores tem *chips* capazes de modificar o conteúdo da mensagem depois da transmissão ter sido iniciada, *timestamps* na camada de acesso ao meio podem ser facilmente implementados nesses dispositivos para eliminar vários componentes de atraso.

Desta forma, quando é iniciado o processo de transmissão a mensagem vai para a fila de saída, porém, quando os primeiros *bits* são enviados o *timestamp* é executado. Em um espaço reservado no *payload* do pacote é inserido uma informação adicional de quanto tempo a mensagem levou desde a criação até o envio, procedimento este realizado pelo *chip* de rádio (uma interrupção SFD - *Start Frame Delimiter*). Isso permite que o receptor

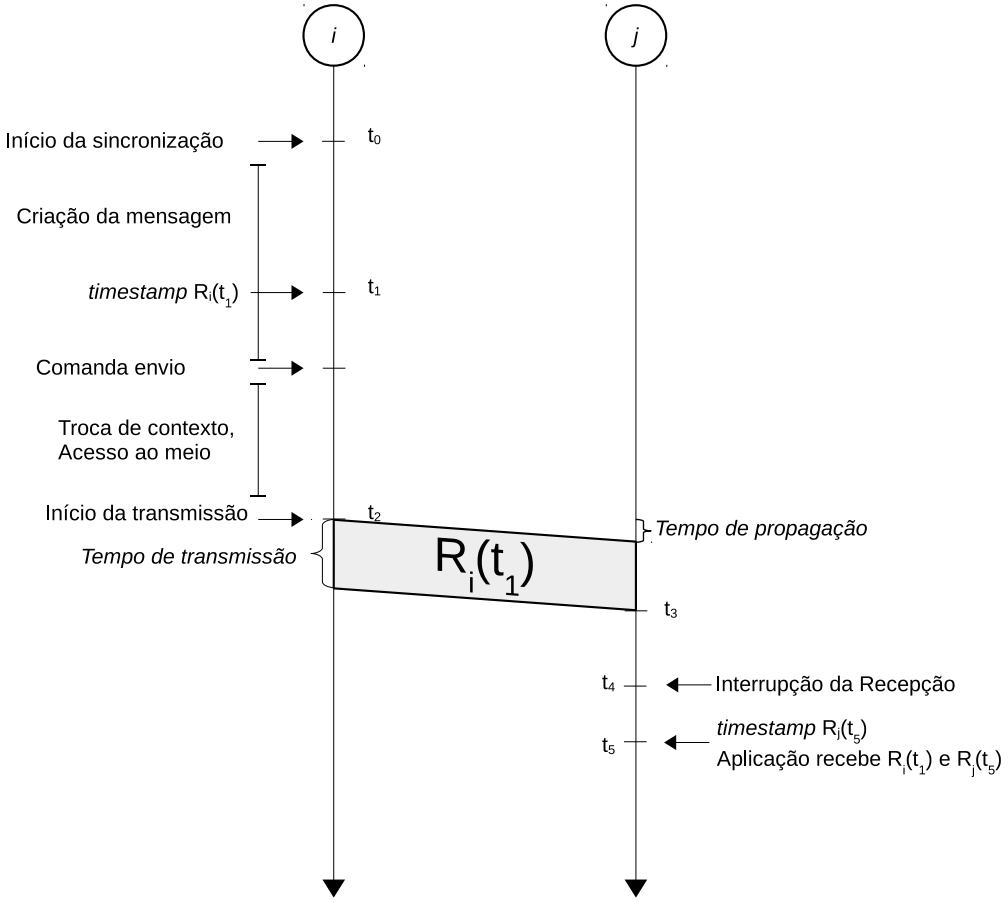


Figura 3.1: Procedimento de leitura do *timestamp* em nível de aplicação

calcule e elimine os atrasos relacionados ao envio. Do lado do receptor é possível usar MAC *timestamp* no momento do recebimento e também eliminar os atrasos do receptor.

Na Figura 3.2 têm-se a ilustração do procedimento de envio de mensagem de sincronização utilizando *timestamp* na camada MAC. O funcionamento do MAC *timestamp* segue basicamente esse diagrama, onde o nó i inicia o procedimento de envio mensagem de sincronização em t_0, logo em seguida cria-se a mensagem e comanda-se o envio. No início da transmissão é inserido na mensagem o tempo t_3, que é o mesmo instante do início do envio dos dados. Já no lado do receptor, depois do atraso de propagação e do tempo de transmissão o último *bit* chega, então uma interrupção é disparada no tempo t_5 junto com ela é registrada o *timestamp* t_5.

O formato da mensagem do FTSP segue o modelo da Figura 3.3. A mensagem começa com um preâmbulo, seguido do conjunto de *bytes* de SYNC, um campo de dados e por fim um campo de identificação de erros CRC. O PREAMBLE é usado para sincronizar a frequência dos rádios, o SYNC é utilizado para calcular o *bit offset* e com isso alinhar os *bytes* no receptor. Os *timestamps* são armazenados no limite de cada *byte* transmitido

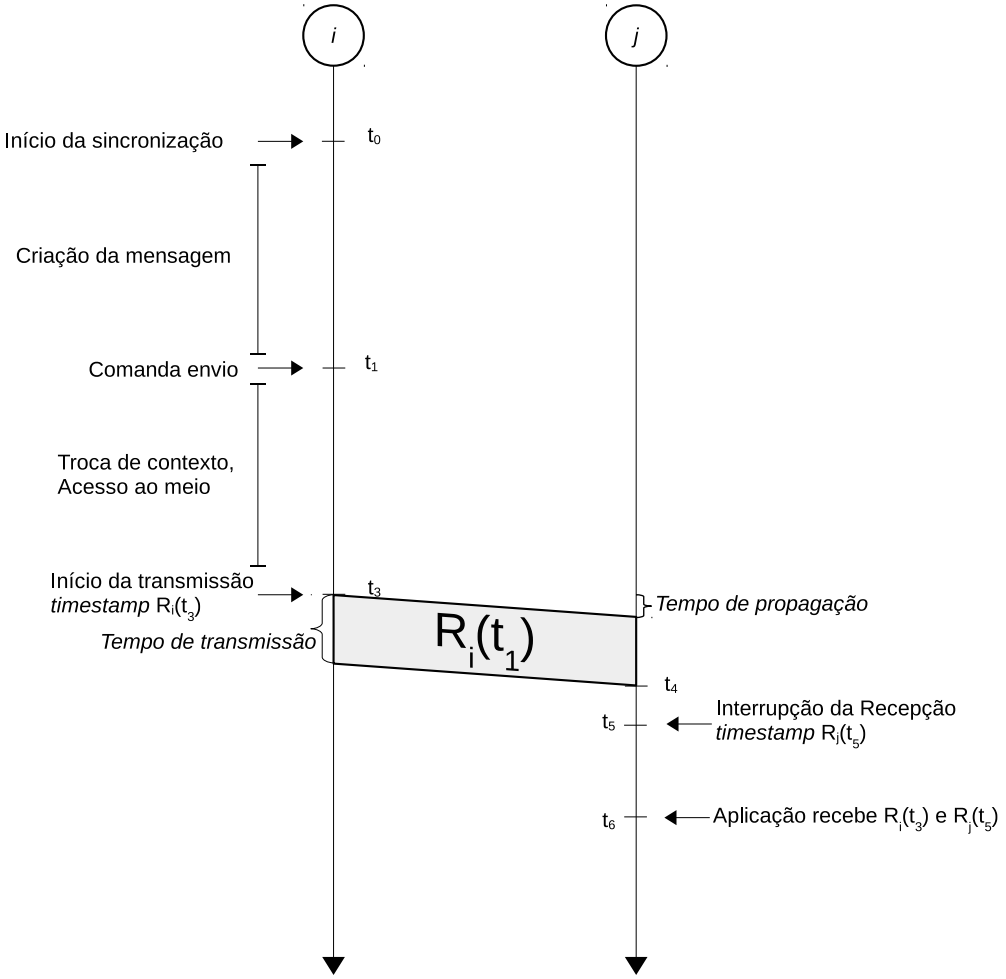


Figura 3.2: Procedimento de leitura do *timestamp* na camada MAC

depois de SYNC, tando no envio quanto no recebimento. Os tempos são normalizados subtraindo um múltiplo inteiro correspondente ao tempo de transmissão. Somente o tempo final é inserido no campo de dados da mensagem.

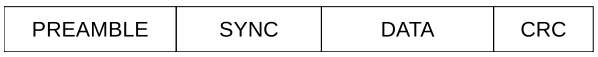


Figura 3.3: Formato da mensagem do FTSP

A forma como o *timestamp* é realizado depende do modelo do *chip* de rádio. Pode ser dividido em dois tipos, orientado a *byte* ou orientado pacotes. Nos *chips* orientados a *byte* como o CC1000 [23] (Mica2 e Mica2dot), geram interrupção por cada *byte* transmitido, essa interrupção armazena no metadado da mensagem o *timestamp* do momento em que o *byte* foi transmitido ou recebido. Ao final da transmissão é possível determinar um *timestamp* único, calculando a taxa de transmissão e a média dos *timestamps*. Os *chips* CC2420 [5] (MicaZ, TelosA, TelosB, TmoteSky) e RF230 (Iris) [10] são orientados a

pacote, nesses rádios ao invés de marcarem os tempos em cada *byte* transmitido, fazem apenas uma marcação para o pacote inteiro usando apenas uma única interrupção SFD.

3.3 Escorregamento do Relógio

Conforme apresentado na Seção 1.1.1, os sensores de baixo custo apresentam relógios muito imprecisos. Cristais de diferentes nós podem oscilar de forma ligeiramente diferente. Por exemplo, dado um *mote* Mica que tem em sua especificação de fábrica uma frequência de $7,3828\ MHz$, de fato pode ter $7,3827\ MHz$, enquanto um outro *mote* pode apresentar $7,3829\ MHz$, isso gera 20 *ticks* de erro por segundo. Este erro acumulado pode atrapalhar a precisão que se pretende oferecer em uma dada aplicação. Esse comportamento causa a necessidade de ressincronização em períodos mais curtos.

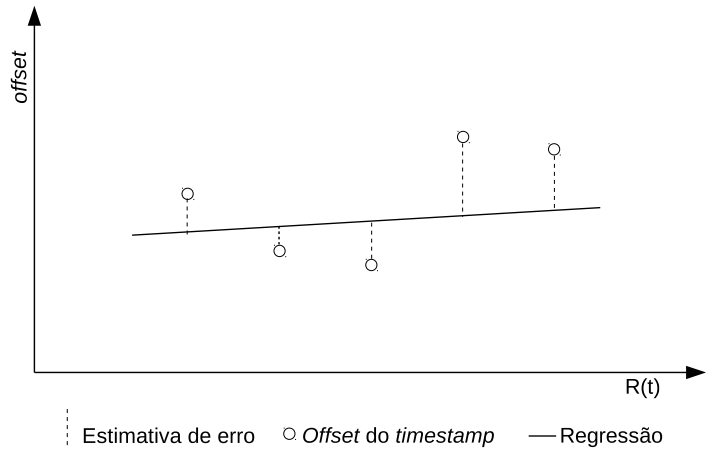


Figura 3.4: Regressão linear aplicada a *timestamps*

Regressão linear é o método mais utilizado para estimar o desvio do relógio de um nó para com o relógio da raiz [33]. No processo de sincronização um nó de referência ou *root*, envia seu uma mensagem com seu próprio tempo, os receptores usam esse tempo para iniciar seu relógio local. Então o *root* passa a enviar periodicamente seu *timestamp*, cada um dos receptores armazenam em uma tabela os tempos recebidos junto com o tempo de recebimento baseado no seu relógio local. A predição de erro é a diferença entre o tempo de referência do nó *root* e a estimativa de tempo do nó receptor.

De acordo com [3], usando regressão linear é possível predizer um padrão pelo calculo dos pares de *timestamps* recebidos, e utilizá-los para compensar os erros. Uma linha de regressão linear fornece a inclinação necessária para estimar o tempo do nó *root* no futuro, a Figura 3.4 ilustra esse processo. Assim, é possível ajustar de forma mais suave a taxa

de crescimento do relógio para algo mais próximo na noção global de tempo. No FTSP devido as limitações de memória dos dispositivos o tamanho padrão das tabelas é 8 pares de *timestamps*, os autores comprovaram em experimentos que a taxa de erro utilizando essa técnica é de $1.48\mu s$ em média, quando desligado o recurso o erro passa a ser crescente e acumula com o tempo.

3.4 Sincronização Multi-saltos

A sincronização multi-saltos do FTSP usa pontos de referência. Esses pontos são pares de *timestamps*, um com o tempo global e outro local, ambos referem-se ao mesmo instante de tempo. Eles são capturados no envio e recebimento de mensagens, como descrito na Seção 3.2, são transmitidos pelo nó *root* ou qualquer nó que esteja sincronizado. O nó *root* é eleito e reeleito dinamicamente, ele é o responsável por difundir periodicamente mensagens de sincronização, os nós ao seu alcance armazenam os pontos de referência. Nós que não estão ao alcance de transmissão do *root*, recebem através dos nós intermediários que já estejam sincronizados. Quando os nós têm pontos suficientes, eles são capazes de calcular seu respectivo *offset* e taxa de escorregamento do relógio. As principais questões sobre esse procedimento são: a mensagem de sincronização, tratamento da redundância de informação, eleição do *root*.

Mensagem de sincronização: A primeira questão do processo de sincronização é o conteúdo e o formato da mensagem de sincronização do FTSP, a seguir os campos.

- `rootID`: Este campo tem a informação do ID do *root*, referente ao nó que enviou a mensagem.
- `globalTime`: É a estimativa do tempo global do nó emissor no momento do envio da mensagem.
- `seqNum`: É um número que é incrementado toda vez que o *root* inicia uma nova rodada de sincronização. O `seqNum` é utilizado para que os nós sincronizados possam sabem quando uma mensagem é nova, assim conseguem diminuir a redundância de informação.

Informação redundante:

Problema da eleição do nó *root*:

```
1  event Radio.receive(TimeSyncMsg *msg){
2    if( msg->rootID < myRootID )
3        myRootID = msg->rootID;
4    else if( msg->rootID > myRootID
5      || msg->seqNum <= highestSeqNum )
6        return;
7    highestSeqNum = msg->seqNum;
8    if( myRootID < myID )
9        heartBeats = 0;
10   if( numEntries >= NUMENTRIES_LIMIT
11     && getError(msg) > TIME_ERROR_LIMIT )
12       clearRegressionTable();
13   else
14       addEntryAndEstimateDrift(msg);
15 }
```

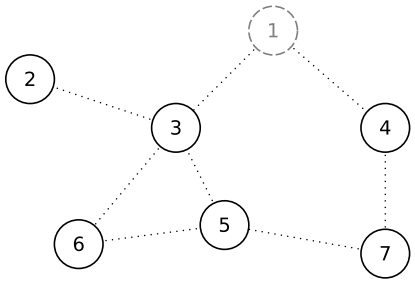
Figura 3.5: FTSP: Rotina do receptor [33]

A Figura 3.5 mostra a rotina de recebimento de mensagens de sincronização. As linhas 2 e 3 comparam o `rootID` da mensagem de sincronização recebida com a informação local que o nó tem sobre que é o *root* `myRootID`. Se a mensagem recebida tem o `rootID` menor, o nó assume `rootID` como seu root. As linhas de 4 a 7 ignoram as mensagens que tenham um `rootID` e `seqNum` menores, pois possuem informação irrelevante. Se `seqNum` é maior, o valor de `highestSeqNum` é atualizado. Linhas 8 e 9 fazem o nó *root* desistir de ser a *root* da rede quando existe um `rootID` menor que seu próprio ID. No caso de `rootID` ser maior que o verificado se o `seqNum` é maior ou igual ao valor do `highestSeqNum`, esta checagem previne informações redundantes por que a mensagem será apenas usada quando o `rootID` for menor ou igual a `myRootID` e o numero maior que o valor de `highestSeqNum`. Linhas 10 e 15 verificam se o tempo da mensagem está em discordância com a estimativa de tempo global mais recente, se é aplicável limpa a tabela de regressão, se não, acumula a mensagem para calcular a regressão linear e sincronizar. As linhas 10 a 15 armazenam mensagens de sincronização para calcular a regressão linear e realizar a sincronização.

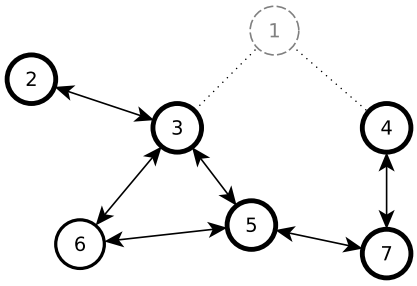
```
1  event Timer.fired() {
2    ++heartBeats;
3    if( myRootID != myID
4      && heartBeats >= ROOT_TIMEOUT )
5        myRootID = myID;
6    if( numEntries >= NUMENTRIES_LIMIT
7      || myRootID == myID ){
8        msg.rootID = myRootID;
9        msg.seqNum = highestSeqNum;
10       Radio.send(msg);
11   if( myRootID == myID )
12       ++highestSeqNum;
13   }
14 }
```

Figura 3.6: FTSP: Rotina do emissor [33]

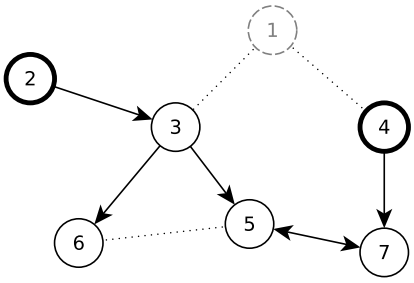
A Figura 3.6 mostra a rotina de envio de mensagem. Um nó decide tornar-se *root* por que está sem receber mensagens de sincronização por `ROOT_TIMEOUT` (linhas 3 a 5). Um nó envia mensagens de sincronização se ele é o *root* ou se está sincronizado (linhas 6 a 10). Se um nó é o *root*, ele também tem que incrementar seu *highestSeqNum*.



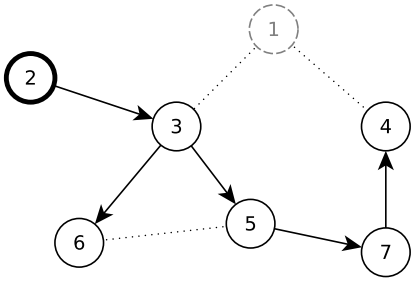
(a) Em um primeiro momento, todos os nós iniciam. As linhas pontilhadas denotam o alcance de transmissão do nó. O nó (1) permanece desligado.



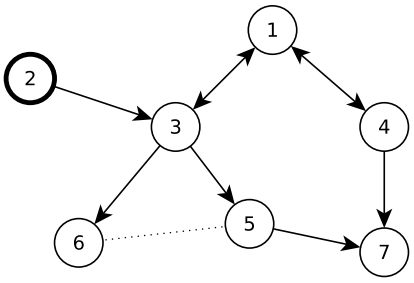
(b) Múltiplos *roots* na rede.



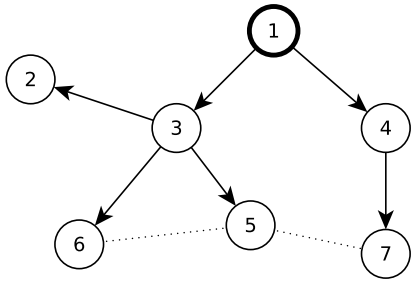
(c) Nós começam a receber mensagens com `rootID` menor, logo atualizam sua referência.



(d) A rede convergiu, e todos os nós ativos da rede estão sincronizados com o nó 2. Mensagens entre os nós 5-6 tem `seqNum` menor que `highestSeqNum` então são descartadas por redundância.



(e) Nó 1 é ativado, sendo o menor ID da rede vai tornar-se o *root*, porém precisa primeiro preencher sua tabela de regressão e sincronizar com o tempo global, após estar sincronizado inicia o envio de mensagens de sincronização



(f) Nó 1 declara-se *root*, passa a enviar suas mensagens de sincronização, após algumas rodadas a rede converge e todos os nós estão com `myRootID` = 1.

Figura 3.7: Eleição de lider

No próximo capítulo, iremos apresentar o FTSP+, que é uma alternativa ao FTSP sem necessidade do recurso de *timestamp* na camada MAC. Demostraremos os detalhes da técnica referente a estimativa do atraso no envio e o pseudocódigo da implementação deste recurso.

Capítulo 4

FTSP+

As principais técnicas para o cálculo e estimativa do atraso trabalham com *timestamp* no nível de acesso ao meio ou com a troca de mensagens, ambas são custosas, visto que mecanismos de marcação de tempo na camada MAC são dependentes de *hardware* específico e a troca de mensagens quebra com a premissa unidirecional do FTSP. Nós propusemos então o FTSP+ que substitui o recurso de MAC *timestamp* por um *timestamp* a nível de aplicação, baseado na ideia de que um nó é capaz de determinar no envio de uma mensagem quando ela termina de ser enviada, então é possível calcular os atrasos no processo de envio e com uma mensagem posterior efetuar a correção do *timestamp*. Esta método já foi previamente discutido por [51] que é uma técnica simples de estimativa de atraso local para RSSF.

4.1 Técnica

Em qualquer técnica de sincronização de relógios em sistemas distribuídos, os nós tem que dizer um ao outro o seu tempo local. A Figura 4.1 ilustra esse cenário. O nó emissor armazena o seu tempo local $t_1{}'$ na mensagem de sincronização no tempo t_1 e ordena o envio da mensagem. Devido as incertezas aleatórias do acesso ao meio, o emissor só recebe o acesso ao meio no tempo t_2 e começa a enviar a mensagem. O *tempo de acesso ao meio* é definido por $t_2 - t_1$. A mensagem propaga-se sobre o meio por um intervalo de tempo tp até que atinge o rádio do receptor no tempo t_3. O *tempo de propagação* é definido por tp. Devido as políticas de manipulação de interrupção e processamento do cabeçalho do pacote, o nó receptor marca o *timestamp* do recebimento no tempo t_4 com o seu tempo local $t_4{}''$. O *tempo de processamento* é dado por $t_4 - t_3$.

A imprecisão de sincronização acontece porque o nó de receptor pensa que no tempo

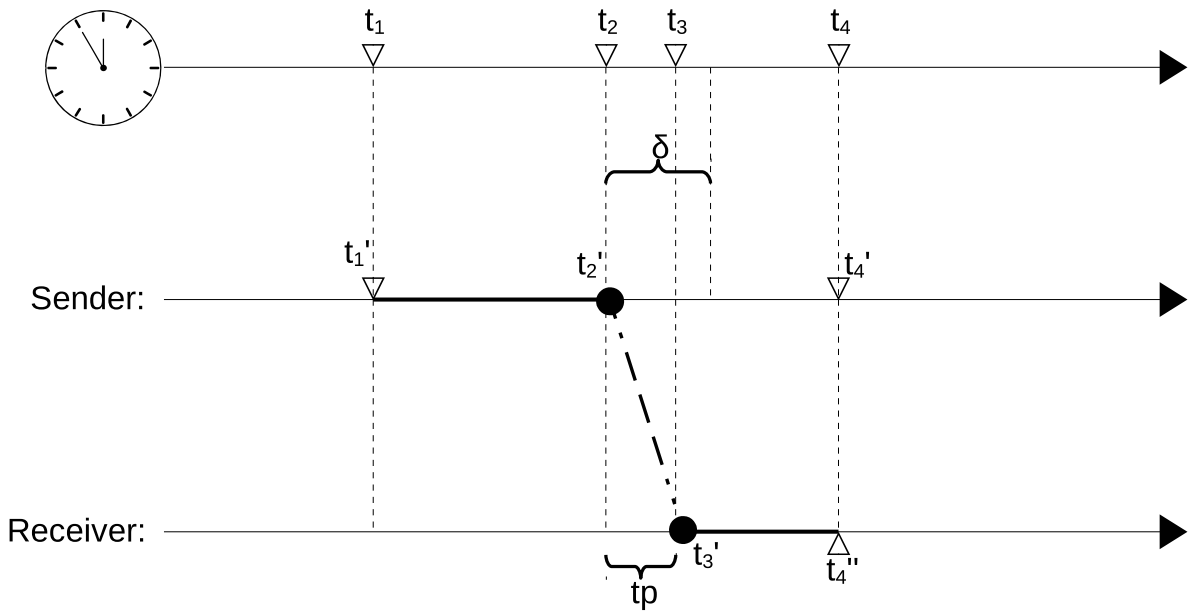


Figura 4.1: Syncronization steps.

t_4 o emissor tem o tempo $t_1{}'$ e o receptor tem o tempo $t_4{}''$. Como podemos ver na Figura 4.1, isso não é verdade. No tempo t_4, o nó emissor tem o tempo $t_4{}' = t_1{}' +$ *tempo de acesso ao meio* + *tempo de propagação* + *tempo processamento*. O *timestamp* na camada MAC faz *tempo de acesso ao meio* e *tempo de processamento* igual a zero. Desde que o *tempo de propagação* é desprezível ($\sim 1\mu s$) [33], algumas políticas de sincronização – incluindo o FTSP – podem atingir uma precisão muito boa. Contudo, sem o *timestamp* na camada MAC, esses tempo não são negligenciáveis e tem que ser calculados.

4.2 Modificação

O FTSP+ calcula o *tempo de acesso ao meio* usando uma manipulação de interrupção para marcar o tempo do momento que o nó obtém o acesso ao meio para enviar a mensagem de sincronização. Embora o acesso ao meio é concedido no tempo $t_2{}'$, *timestamp de acesso ao meio* é igual a $t_2{}' + \delta$, onde δ é o *overhead* para processar a manipulação da interrupção.

O emissor envia uma mensagem com o conteúdo $t_2{}' + \delta - t_1{}'$ então o receptor pode estimar $t_4{}'$. Vamos chamar esta estimativa de $\bar{t_4}{}'$. O receptor calcula $\bar{t_4}{}'$ as in Equação (4.1).

```
event Radio.receiveSyncMsg(SyncMsg *msg){
  if( msg->rootID < myRootID )
      myRootID = msg->rootID;
  else if( msg->rootID > myRootID
    || msg->seqNum <= highestSeqNum )
      return;
  highestSeqNum = msg->seqNum;
  if( myRootID < myID )
      heartBeats = 0;
  if( numEntries >= NUMENTRIES_LIMIT
    && getError(msg) > TIME_ERROR_LIMIT )
      clearRegressionTable();
  else if (MAC_Time==false){
      addToWaitCorrectionMsgList(msg);
  }else{
      addEntryAndEstimateDrift(msg);
  }
}

event Radio.receiveCorrectionMsg(
                              CorrectionMsg *msg){
  applyCorrection(msg);
  addEntryAndEstimateDrift(msg);
}
```

Figura 4.2: Algoritmo do receptor.

$$\begin{aligned} \bar{t4}' &= t1' - (t2' + \delta - t1') \\ \bar{t4}' &= t1' + \textit{tempo de acesso ao meio} + \delta \end{aligned} \tag{4.1}$$

A estimativa de erro é a diferença entre ${t_4}'$ e $\bar{t_4}'$, que é:

$$t4' - \bar{t4}' = \textit{tempo de propagação} + \textit{tempo de processamento} - \delta \tag{4.2}$$

Desde de que o *tempo de processamento* e o δ são latências do processamento da manipulação de interrupção. Nós investigamos seus valores em nosso experimento no Capítulo 6, eles tendem a anular-se mutuamente. Relembrando que o *tempo de propagação* é negligenciável.

Como podemos ver na Figura 4.2, o receptor FTSP+ tem rotinas para manipular dois

tipos diferentes de mensagens de recebimento: `Radio.receiveSyncMsg(SyncMsg *msg)` serve para as mensagens de sincronização e o `Radio.receiveCorrectionMsg(CorrectionMsg *msg)` é utilizado nas mensagens de correção. O *tempo de correção*, expressado na Equação (4.3), é a informação que é transmitida para o receptor aplicar como correção da mensagem anterior.

$$\textit{tempo de correção} = t2' - t1' + \delta \tag{4.3}$$

A rotina `Radio.receiveSyncMsg(SyncMsg *msg)` é diferente do receptor do FTSP somente nas linhas 13 e 14. Estas linha armazenam mensagens de sincronização em uma lista para aguardarem por suas respectivas mensagens de correção, caso MAC *timestamp* não esteja disponível.

Já a rotina `Radio.receiveCorrectionMsg(CorrectionMsg *msg)` recebe a mensagem com o valor da correção, e usa a função `applyCorrection(msg)` para aplicar a correção e remover da lista de espera as mensagens de sincronização que correspondem com a mensagem de correção recebida, corrige o *timestamp* e chama a função `addEntryAndEstimateDrift(msg)` que adiciona o valor já corrigido na tabela de regressão.

A Figura 4.3 apresenta a listagem do código da rotina do emissor do FTSP+, são duas funções `Timer.fired()` que periodicamente envia mensagens de sincronização (igual ao FTSP) e `sendDone(message_t *msg, error_t error)` que envia a mensagem de correção.

O `Timer.fired()` difere do FTSP somente pelas linhas 10 e 11, que coleta o *timestamp* no nível de aplicação.

A rotina `sendDone(message_t *msg, error_t error)` é a manipulação da interrupção de quando o acesso ao meio sem fio é garantido e assim coletar o tempo de acesso ao meio pelo cálculo do tempo que se passou entre o início do envio até ele ser concluído (`send/sendDone`). Ao final é enviado a mensagem de correção.

As linhas 21-22 mostram como o tempo de correção é calculado. O `seqNum` é utilizado para identificar qual mensagem será ajustada, `finalTime` e `initialTime` referem-se ao tempos t_2 e t_1 da Equação (4.3).

```
1  event Timer.fired() {
2    ++heartBeats;
3    if( myRootID != myID
4      && heartBeats >= ROOT_TIMEOUT )
5        myRootID = myID;
6    if( numEntries >= NUMENTRIES_LIMIT
7      || myRootID == myID ){
8        msg.rootID = myRootID;
9        msg.seqNum = highestSeqNum;
10       initialTime = call getLocalTime();
11       msg.timestamp = initialTime;
12       Radio.send(msg);
13   if( myRootID == myID )
14       ++highestSeqNum;
15   }
16 }
17
18
19 event sendDone(SyncMsg *msg, error_t error){
20  finalTime = call getLocalTime();
21  correctionMsg.correction = finalTime-initialTime;
22  correctionMsg.seqNum = msg.seqNum;
23  Radio.send(correctionMsg);
24 }
```

Figura 4.3: Algoritmo do emissor.

O próximo capítulo trata dos recursos de *software* e os requisitos dos sistemas para a implementação do FTSP+ na plataforma TinyOS.

Capítulo 5

Implementação

Este capítulo trata sobre o sistema operacional TinyOS a linguagem nesC e os dispositivos de *hardware*. Aspectos importantes de sua arquitetura para a implementação de um protocolo de sincronização.

5.1 Mote

O termo *mote* é utilizado para se referir ao dispositivo que é nó em uma rede de sensores sem fio, foi introduzido por pesquisadores da Universidade de Berkeley [9] no início da década de 90, assim, constantemente podemos encontrar referências como "Berkeley Mote" para designar estes dispositivos de rede, independente do fabricante.

O MicaZ é um *mote* disponível comercialmente e amplamente utilizado em pesquisas acadêmicas. Possui um processador de 4 MHz com 8 *bits*, memória de 128 kB. A diferença comparado a um computador com a capacidade similar do início da década de 1980, como o 8088, é o baixo consumo de energia que varia de 15 μA no modo de economia até 8 mA em execução. Outro componente significativo é o rádio, esses dispositivos possuem transmissores com frequência de 2.4 GHz e taxa de dados de 250 kbps, usam a estratégia de acesso ao meio CSMA/CA, conseguindo transferências de 40.000 bps.

A junção destas características, fornecem recursos que tornam o *mote* uma boa opção para diversas aplicações, a capacidade de computação o consumo de energia trazem a flexibilidade para implantação destes equipamentos em RSSF, a Tabela 5.1 contém as especificações dos *hardwares* utilizados.

Item	MicaZ	Iris
Processador	ATMega 128L 8MHz	ATmega1281 16MHz
Flash	128 kB	
Memória Dados	512 kB	
EEPROM	4 kB	
ADC	10 bit	
Chip de Rádio	CC2420	RF230
Frequência do Rádio	2.4 GHz	
Taxa de Transferência	250 kbps	
Voltagem de Operação	3,6 - 2,7	
Consumo de Energia	RX 19,7 mA TX 17,4 mA <15 μA (dormindo)	RX 16 mA TX 17 mA 8 μA (dormindo)

Tabela 5.1: Especificações de *hardware* dos *motes*

5.2 Sistema Operacional

O desenvolvimento do FTSP+ foi realizado utilizando o sistema operacional TinyOS, que é um SO bem difundido na área de redes sensores sem fio [31]. O TinyOS não é um sistema operacional convencional, em que se instala completamente no sensor, ele se apresenta como um arcabouço de um conjunto de componentes reutilizáveis que permitem o desenvolvimento de aplicações para sistemas embarcados em conjunto com o SO. Esses componentes são separados por funções características, no momento de construir a aplicação somente os componentes especificados serão integrados na aplicação final, mantendo o uso minimal de recursos.

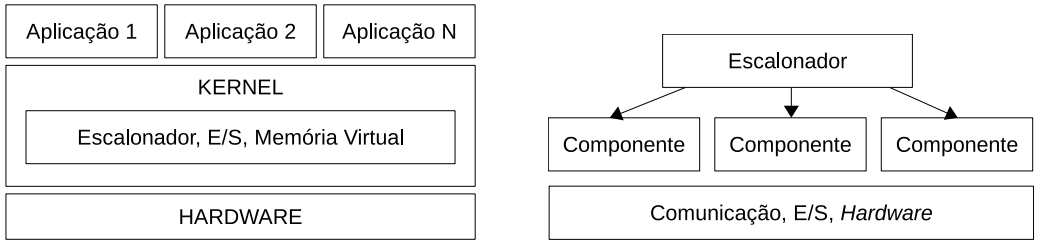
Com a limitação de recursos nos dispositivos sensores o TinyOS conta com pouco menos de 400 *bytes* de tamanho, é um sistema baseado em eventos, com suporte a concorrência, eventos assíncronos e comandos. Um programa em TinyOS possui a abstração dos componentes em um modelo de grafo, como vemos na Figura 5.2.

As aplicações e o próprio sistema operacional são escritos em nesC (*networked systems C*) [17] um dialeto da linguagem C, que dá o suporte a arquitetura de componentes e orientação a eventos, além de ser otimizada para reduzir o consumo de memória e ter primitivas que previnam problemas de baixo nível como condição de corrida. No desenvolvimento existe a separação da organização dos componentes e programação das interfaces, os componentes são conectados (*wired*) juntos em um arquivo de aplicação, já o código dos eventos e tarefas são feitos em um arquivo de construção.

Programas em nesC são construídos por itens definidos separadamente e então conectados de forma explicita para juntar todos em uma unidade [30]. A seguir definições

importantes sobre essas características:

- Componentes e interfaces: Um programa em nesC é um conjunto de componentes ligados entre si. O componente fornece interfaces que são responsáveis pela comunicação bidirecional entre os componentes, as interfaces fornecem comando e eventos.
- Implementação: A implementação pode ser dividida em duas partes no nesC, uma chamada *modules* e outra *configuration*. Em *modules* é implementado as interfaces, já em *configuration* é usado para juntar os componentes conectando suas interfaces.
- Modelo de concorrência: Define como os componentes interagem entre suas execuções. Temos dois tipos de tarefa, que são a tarefa (ou *task*) propriamente dita e eventos de dispositivos. Quando uma tarefa é enviada para execução ela roda até completar, não fazendo preempção. Os eventos de dispositivos são como as tarefas, só que são geradas como respostas a eventos.



(a) Arquitetura Tradicional (b) Arquitetura Tinyos

Figura 5.1: Comparação das arquiteturas

A arquitetura do TinyOS difere das arquiteturas clássicas, a Figura 5.1 ilustra a organização dos dois tipos. No cenário das RSSF sistemas operacionais tradicionais apresentam as seguintes restrições: grande necessidade de armazenamento e memória, maior consumo de energia, custo computacional das trocas de contexto e uso memória virtual entre outros. O TinyOS simplifica sua arquitetura para tornar-se mais específico para suas aplicações: não possui *kernel*, a manipulação do *hardware* é realizado diretamente, sem gerenciamento de processos tem apenas um processo de tempo real, não utiliza memória virtual, ao invés disso, tem um único espaço de endereço físico linear. A figura do escalonador representa uma grande mudança entre os paradigmas, o TinyOS coloca seu *scheduler* no topo da arquitetura, assim uma aplicação é o resultado da junção de seus componentes com o escalonador.

5.3 Diagramas e Componentes

O TinyOS foi desenvolvido seguindo um conjunto de diretrizes, chamado de TEP (TinyOS Enhancement Proposals) que orientam as modificações no núcleo do seu código, também é utilizado para guiar a criação de novas funcionalidades. Podemos citar as seguintes TEPs [55, 36, 46] com definições importantes para a implementação do FTSP+:

- TEP 102: Propõe a estrutura dos `Timers` (controladores do relógio) e suas propriedades de precisão, acurácia e tamanhos.
- TEP 132: Descreve o mecanismo de *timestamp* no nível de acesso ao meio. A funcionalidade que fornece o tempo de envio e recebimento de uma mensagem no processo de comunicação.
- TEP 133: Descreve o funcionamento do mecanismo das mensagens de sincronização de tempo, como os tempos são convertidos do tempo do emissor para o do receptor e como são tratados nas pilhas de protocolos.

Os *Timers* no TinyOS fornecem precisões listadas na Tabela 5.2, todas as precisões são binárias, ou seja, 1s contém 1024 milissegundos binários. A acurácia dependente de quão bem o dispositivo fornece de seu relógio, como vimos anteriormente os relógios são afetados pelas limitações de seu *hardware*. Assim um relógio de um *mote* que roda a 7.37MHz, tem valor real do relógio variando muito próximo desse valor. Os tamanhos são basicamente 8, 16, 32 e 64 *bits*, sendo 32 *bits* o tamanho indicado para representação dos tempos dos componentes.

Nome	TMilliC	T32kHz	TMicroC
Frequência	1,024 Hz	32,768 Hz	0.9216 MHz
Precisão	1024 ticks/s	32768 ticks/s	1048576 ticks/s
Periodo	976 ms	30.518 μs	1.084 μs

Tabela 5.2: Precisões dos *Timers* no TinyOS

O FTSP+ foi elaborado utilizando os *motes* Iris e Micaz [34, 35], e construído sobre a implementação já existente do FTSP. A Figura 5.2 apresenta os componentes utilizados pelo FTSP+, a alteração descrita na Seção 4.2 estão presentes no componente central da imagem o `TimeSyncP`.

O `TimeSyncP` faz parte a implementação original do FTSP, porém foi o único componente modificado. O FTSP usa uma mensagem de sincronização com informações como

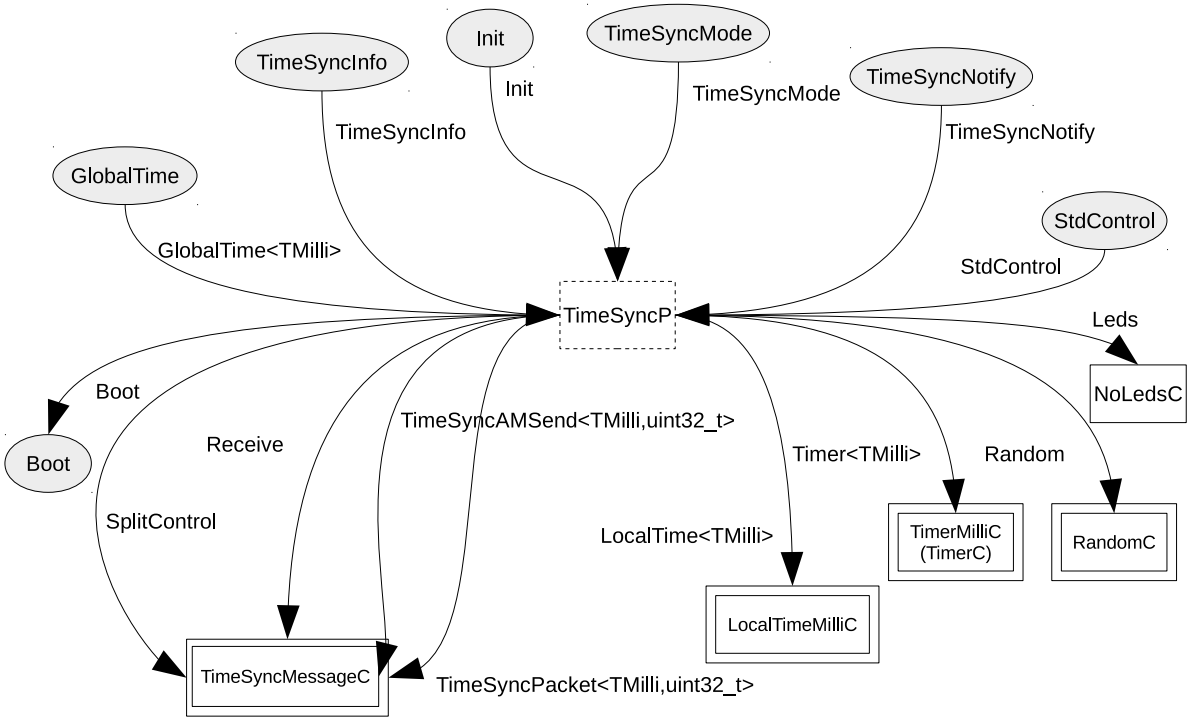


Figura 5.2: Diagrama de Componentes do FTSP+

o ID do nó raiz da rede, o ID do nó que está enviando a mensagem de sincronização, um número de sequência da mensagem e o seu *timestamp*. No FTSP+ é acrescentado mais uma mensagem na comunicação, que é a mensagem de correção. A estrutura da mensagem é listada a seguir:

```
typedef nx_struct TimeSyncMsg {
 nx_uint16_t rootID;
 nx_uint16_t nodeID;
 nx_uint8_t  seqNum;
 nx_uint32_t correction;
} TimeSyncMsg;
```

Figura 5.3: Formato da mensagem de sincronização

O mensagem conta com o `rootID`, se a mensagem tem *root* com ID maior que a informação do *root* local ela é descartada. O `nodeID` contém o ID de quem enviou a mensagem. O `seqNum` serve para identificar qual o *timestamp* a mensagem está corrigindo. O campo `correction` é o campo que traz a estimativa de erro do emissor e é com ele que será feita a correção dos *timestamp* anteriormente recebidos.

No documento da TEP 132 [36], é fornecido uma breve descrição sobre o padrão de *timestamps* oferecidos pela interface `PacketTimeStamp` que faz o acesso ao tempo de

recepção e envio de determinada mensagem.

A TEP 133 fornece a abstração para o mecanismo de sincronização de pares. Não provê um sincronização completa da redes, mas, com seus recursos é possível implementar um serviço de sincronização completo, exemplo o FTSP, pois os componentes e interfaces utilizados em sua implementação estão bem definidos em seu documento [55].

Outro aspecto importante da TEP 133 é o seu guia de implementação, que compreende duas abordagens, uma em que é possível mudar o *payload* da mensagem durante a transmissão usando a interrupção SFD dos rádios orientados a pacote. A segunda abordagem descreve a possibilidade da sincronização para plataformas em que não seja possível a alteração do conteúdo durante a transmissão. Podemos verificar que o FTSP atende a primeira abordagem, já a segunda não é atendida, porém com o FTSP+ ambas as abordagens são acolhidas.

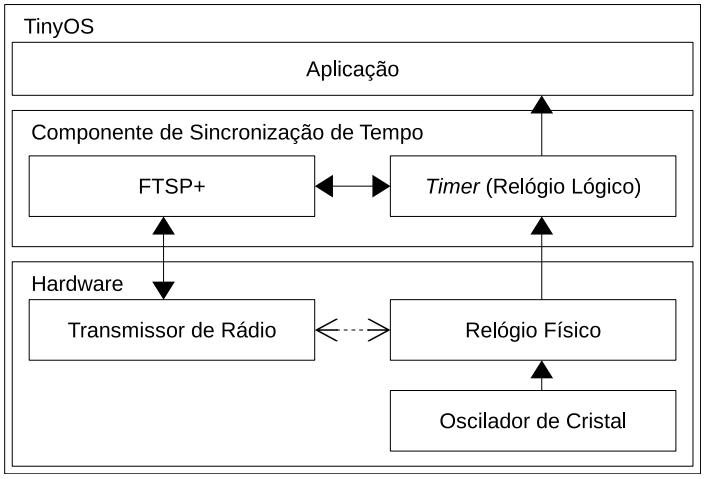


Figura 5.4: Arquitetura de *Software* do protocolo de sincronização

Ao final temos a seguinte arquitetura de sincronização resultante do FTSP+ no TinyOS, Figura 5.4. O oscilador de cristal incrementa o contador do relógio físico, que por sua vez alimenta o relógio lógico, o FTSP+ usa o relógio lógico (Componente `Timer`) para obter o valor do tempo local do nó, assim pode calcular seu desvio, estimar o tempo global e atualizar novamente o relógio, bem como, enviar e receber essas referências de tempo pelo transmissor de rádio. As setas da imagem denotam troca de dados entre os componentes, a seta tracejada refere-se ao mecanismo do FTSP de MAC *timestamp*.

O próximo capítulo traz experimento realizado com *motes* reais, e gráficos estatísticos dos resultados encontrados, os experimentos foram baseados nas técnicas descritas no trabalho.

Capítulo 6

Experimentos

Neste capítulo nos descrevemos nossos experimentos a respeito da precisão do FTSP+. Nós implementamos o FTSP+ sobre TinyOS 2.1.2 [31] e executamos os testes em *motes* Micaz [35].

Os *motes* Micaz suportam o *timestamp* na camada MAC, mas nós precisamos desta informação para medir a acurácia da nossas precisão. Para a sincronização, nó reescrevemos o recurso de MAC *timestamp* com o nosso *timestamp* na camada de aplicação.

Nós usamos *jiffys* como unidade de tempo por que esta é base de tempo do TinyOS e representa o tempo entre 2 *ticks* do relógio. Nossos experimentos são resultados de uma rede com os nós trocando mensagens de sincronização a cada 3 segundos durante 10 minutos.

	Média (μs)	**Desvio padrão**
tempo de processamento	0.87 ± 0.0095	0.33
δ	0.88 ± 0.0093	0.33
tempo de correção	9.01 ± 0.093	3.32
acesso ao meio	8.13 ± 0.093	3.31
$\mathbf{t4' - t\bar{4}'}$	-0.0047 ± 0.013	0.47

Tabela 6.1: Média e desvio padrão.

Nosso primeiro mediu a distribuição de probabilidade dos tempos de correção (relembrando da Seção 4.2 que é $t_2 - t_1 + \delta$). Como podemos ver na Figura 6.1, os tempos de correção variam na maioria de 5 até 13 *jiffys* com uma distribuição uniforme. Nós podemos ver na Figura 6.2, que mostra a função distribuição acumulada da correção dos tempos, que está em cerca de 80% dos casos de correção de tempo acima de 5 *jiffys*. Isto é um *overhead* significante e uma fonde de imprecisão que o FTSP+ é capaz de medir e

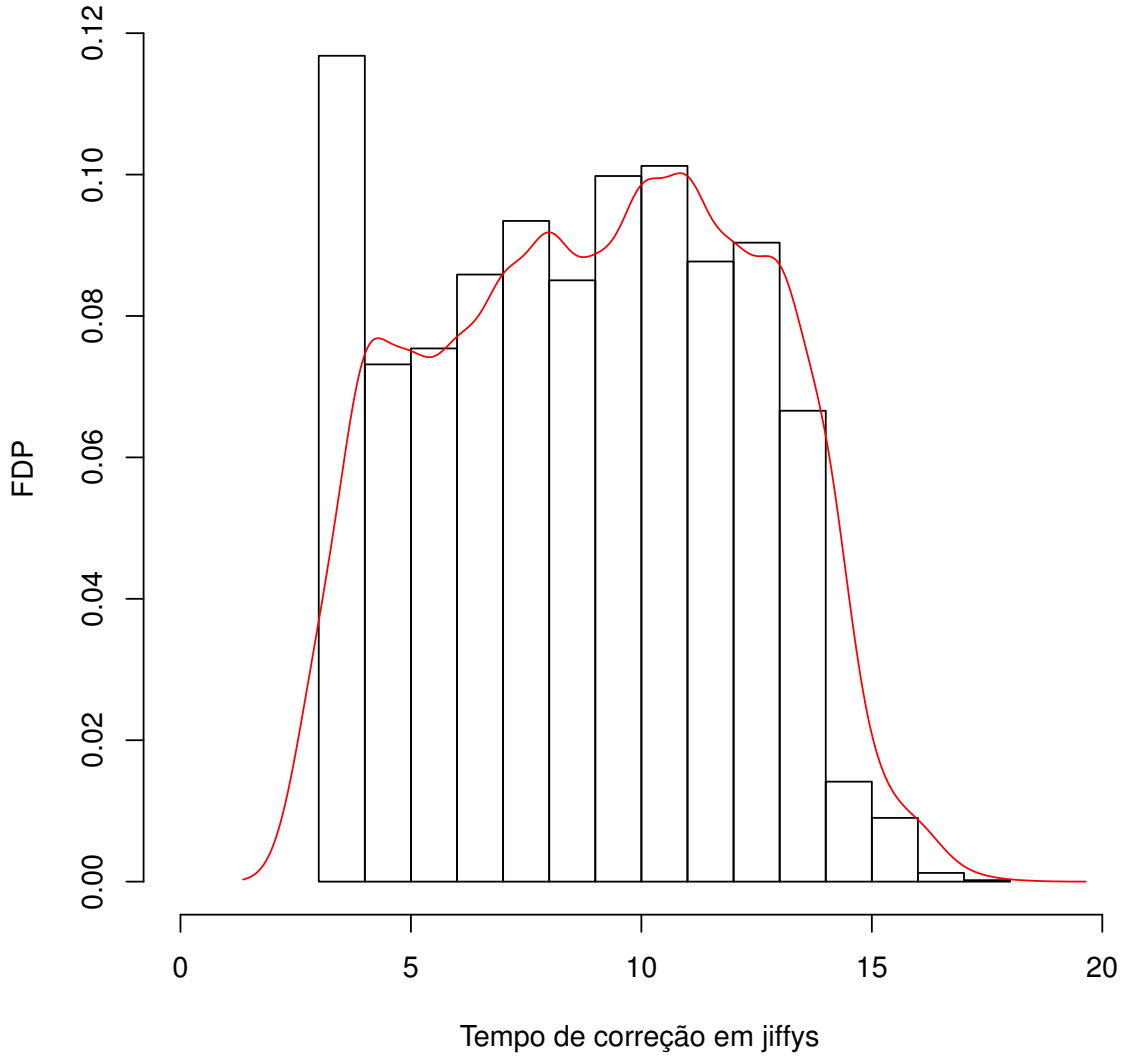


Figura 6.1: Histograma e F.D.P. dos tempos de correção.

fazer uma compensação. Para nosso cenário a média de correção de tempo de atraso é 9.01[1] *jiffys* como vemos na Tabela 6.1 onde podemos ver que temos a média com intervalo de confiança de 95% e desvio padrão.

Atraso em jiffys	0	1	2	3	4	5
Frequência	606	4271	0	1	1	1

Tabela 6.2: Frequência de atraso do receptor *tempo de processamento*.

Nosso segundo experimento mede a distribuição de probabilidade do δ e *tempo de processamento*. Relembrando da Seção 4.2 que eles são o tempo de processamento da

[1]Este valor é dependente de cenário enquanto depende da janela de contenção da rede.

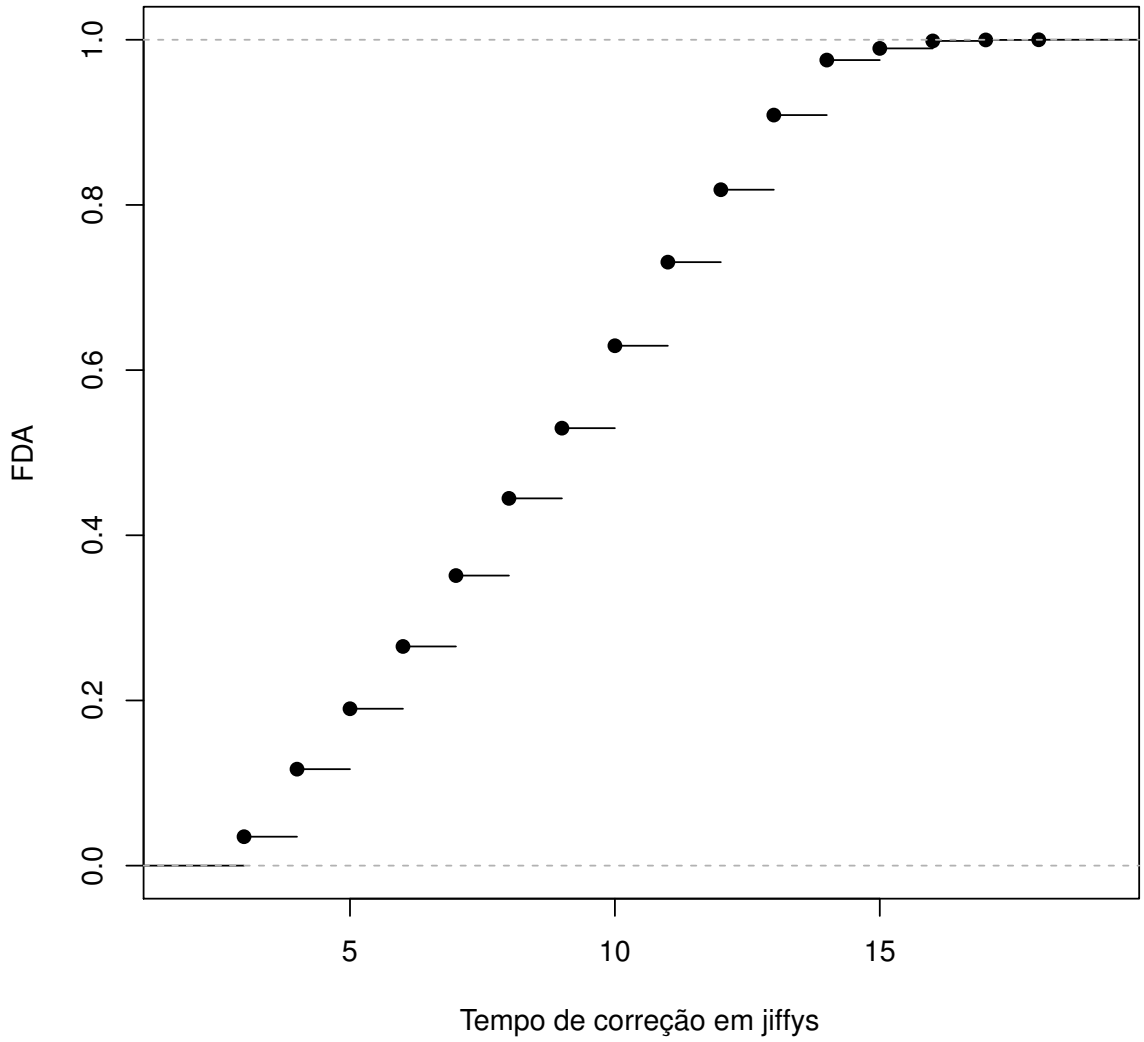


Figura 6.2: F.D.A. dos tempos de correção.

Atraso em jiffys	0	1	2	3	4	5
Frequência	581	4297	0	0	1	1

Tabela 6.3: Frequência de atraso para o δ.

interrupção do rádio no emissor e no receptor. Nó podemos ver na Tabela 6.2 e 6.3 que seus valores são na maioria na faixa entre 0 e 1 *jiffy*, com casos extremamente raros que em nossos experimentos não chegam a 5 *jiffys*. Suas médias são 0.87 e 0.88 *jiffys*, que podem ser consideradas iguais. Portanto, estes atrasos, que o FTSP+ não é capaz de calcular e compensar, são muito pequenas e não comprometem significativamente a precisão da sincronização.

Nós calculamos uma estimativa de erros e comparamos nossos resultados com outros protocolos (alguns usam MAC *timestamp*, outros não) e sumarizamos os resultados na Tabela 6.4. Sabendo que o MAC *timestamp* tem uma variação inerente de 1 *jiffy* de precisão e o FTSP tem um erro de $1.5\mu s$ por salto, nós utilizamos uma simples regra de três para estimar o erro de sincronização do FTSP+ por salto. O FTSP e o FTSP+ são diferentes apenas quanto a técnica de estimar o *timestamp*, e como podemos ver na Tabela 6.1 o *timestamp* do FTSP+ está em média de 0.0047 *jiffys* maior que o MAC *timestamp* ($t_4' - \bar{t_4'}$). Portanto, a nossa regra de três é dada na Equação (6.1).

$$\frac{1.5\mu s}{\text{FTSP+ sync error}} = \frac{1 jiffys}{(0.0047 + 1) jiffys} \tag{6.1}$$

Nós também estimamos o que poderia ser o erro do FTSP sem o MAC *timestamp* (nós o chamamos na tabela de mFTSP), no caso em que *timestamping* é em média 9 *jiffys* maior que o MAC *timestamp* (*tempo de acesso ao meio* + *tempo de processamento* = $8.13 + 0,87 = 9$). A Equação (6.2) mostra que a regra de três para esta estimativa. Como podemos ver, o RBS tem um erro de sincronização aproximadamente 10× maior que o FTSP+.

$$\frac{1.5\mu s}{\text{mFTSP+ sync error}} = \frac{1 jiffys}{(9 + 1) jiffys} \tag{6.2}$$

MAC Timestamping		**App Timestamping**		
FTSP	**PulseSync**	**FTSP+**	**mFTSP**	**RBS**
$1.5\mu s$ [44]	$1.5\mu s$ [44]	$1.508\mu s$ [2]	$15\mu s$ [1] [2]	$29\mu s$ [44]

Tabela 6.4: Média de erro de sincronização por salto.

No TinyOS 1.x, onde o FTSP foi primeiramente implementado e testado [33], *jiffys* estão por padrão na resolução de microsegundos. Um *jiffy* no TinyOS 2.1.2 é de aproximadamente $1ms$, mas existe uma opção de compilação com *TMilli* que permite uma resolução em microsegundos.

[2] Valores estimados baseados na média de erros em *jiffys*.

Capítulo 7

Conclusão e Trabalhos Futuros

Neste trabalho apresentamos uma versão modificada do *Flooding Time Synchronization Protocol* que trabalha sem o *timestamp* na camada de acesso ao meio. Para manter a acurácia alta usamos interrupções de rádio para medir o instante que os nós recebem o acesso ao meio. O emissor usa uma mensagem de correção, além da mensagem de sincronização, como forma de compensação do atraso relativo ao acesso ao meio.

Em nossos experimentos executamos testes em *motes* Micaz rodando TinyOS 2.1.2 para medir a estimativa do tempo de correção, tempo de processamento e acesso ao meio. Nós mostramos que o tempo de acesso ao meio é a principal fonte erro de sincronização, quantificamos os valores em um ambiente de testes real. Também mostramos que o tempo de processamento é muito baixo, na média de 0.87 *jiffys*. Os resultados deste trabalho foram submetidos ao Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'16).

Existem trabalhos recentes, que procuram melhorar o desempenho do FTSP [45], outros estão criando alternativas que implementam funcionalidades que são o ponto de desvantagem do FTSP [50, 28]. No trabalho [29], os autores demonstram resultados melhores que o FTSP nos testes que realizaram, porém todos esses mecanismos fazem uso de *timestamp* na camada de acesso ao meio. Assim, a técnica desenvolvida no FTSP+, pode servir como trabalho futuro a inclusão do recurso também a esses novos protocolos de sincronização.

Referências

[1] Arampatzis, T.; Lygeros, J.; Manesis, S. A survey of applications of wireless sensors and wireless sensor networks. In *IEEE International Symposium on Intelligent Control, Mediterrean Conference on Control and Automation* (2005), IEEE, pp. 719–724.

[2] Baronti, P.; Pillai, P.; Chook, V. W. C.; Chessa, S.; Gotta, A.; Hu, Y. F. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. *Computer Communications* (2007), 1655–1695.

[3] Capriglione, D.; Casinelli, D.; Ferrigno, L. Analysis of quantities influencing the performance of time synchronization based on linear regression in low cost wsns. *Measurement 77* (2016), 105–116.

[4] Cardell-Oliver, R.; Smettem, K.; Kranz, M.; Mayer, K. Field testing a wireless sensor network for reactive environmental monitoring [soil moisture measurement]. In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. Proceedings of the 2004* (Dec 2004), pp. 7–12.

[5] Chipcon, S. C. 2.4 ghz ieee 802.15. 4. *ZigBee-ready RF Transceiver 1* (2003).

[6] Cho, H.; Kim, J.; Baek, Y. Enhanced precision time synchronization for wireless sensor networks. *Sensors 11*, 8 (2011), 7625–7643.

[7] Chong, F.; Heck, M.; Ranganathan, P.; Saleh, A.; Wassel, H. Data center energy efficiency:improving energy efficiency in data centers beyond technology scaling. *Design Test, IEEE 31*, 1 (Feb 2014), 93–104.

[8] Cuevas, Á.; Cuevas, R.; Urueñ, M.; Larrabeiti, D. A proposal for zigbee clusters interconnection based on zigbee extension devices. In *Wireless Sensor and Actor Networks*. Springer, 2007, pp. 227–238.

[9] Culler, D.; Hill, J.; Horton, M.; Pister, K.; Szewczyk, R.; Wood, A. Mica: The commercialization of microsensor motes. *Sensor Technology and Design, April* (2002).

[10] Datasheet, I. Crossbow technology, inc.; san jose, ca. *CA, USA* (2006).

[11] Elson, J.; Estrin, D. Time synchronization for wireless sensor networks. In *Parallel and Distributed Processing Symposium., Proceedings 15th International* (April 2001), pp. 1965–1970.

[12] Elson, J.; Girod, L.; Estrin, D. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Operating Systems Review 36*, SI (2002), 147–163.

[13] Elson, J.; Girod, L.; Estrin, D. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review 36*, SI (2002), 147–163.

[14] Elson, J.; Römer, K. Wireless sensor networks: A new regime for time synchronization. *SIGCOMM Computer Communication Review 33*, 1 (Jan. 2003), 149–154.

[15] Ganeriwal, S.; Kumar, R.; Srivastava, M. B. Timing-sync protocol for sensor networks. In *1st International Conference on Embedded Networked Sensor Systems* (2003), ACM, pp. 138–149.

[16] Ganeriwal, S.; Kumar, R.; Srivastava, M. B. Timing-sync protocol for sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems* (2003), ACM, pp. 138–149.

[17] Gay, D.; Levis, P.; Von Behren, R.; Welsh, M.; Brewer, E.; Culler, D. The nesc language: A holistic approach to networked embedded systems. In *Acm Sigplan Notices* (2003), vol. 38, ACM, pp. 1–11.

[18] Gui, C.; Mohapatra, P. Power conservation and quality of surveillance in target tracking sensor networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, MobiCom '04, ACM, pp. 129–143.

[19] Hill, J.; Horton, M.; Kling, R.; Krishnamurthy, L. The platforms enabling wireless sensor networks. *Communications of the ACM 47*, 6 (2004), 41–46.

[20] Holtkamp, H. Decentralized synchronization for wireless sensor networks. *arXiv preprint arXiv:1304.0646* (2013).

[21] Horauer, M.; Schossmaier, K.; Schmid, U.; Höller, R.; Kerö, N. Psynutc-evaluation of a high-precision time synchronization prototype system for ethernet lans. Tech. rep., DTIC Document, 2002.

[22] Huang, P.; Xiao, L.; Soltani, S.; Mutka, M. W.; Xi, N. The evolution of mac protocols in wireless sensor networks: A survey. *Communications Surveys & Tutorials, IEEE 15*, 1 (2013), 101–120.

[23] Instruments, T. Cc1000 single chip very low power rf transceiver. http://www.ti.com/lit/ds/symlink/cc1000.pdf.

[24] Kahn, J. M.; Katz, R. H.; Pister, K. S. J. Next century challenges: Mobile networking for "smart dust". In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking* (New York, NY, USA, 1999), MobiCom '99, ACM, pp. 271–278.

[25] Karl, H.; Willig, A. *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, 2007.

[26] Kopetz, H. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[27] Kopetz, H.; Ochsenreiter, W. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers C-36*, 8 (Aug 1987), 933–940.

[28] Lenzen, C.; Sommer, P.; Wattenhofer, R. Optimal clock synchronization in networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems* (2009), ACM, pp. 225–238.

[29] Lenzen, C.; Sommer, P.; Wattenhofer, R. Pulsesync: An efficient and scalable clock synchronization protocol. *IEEE/ACM Transactions on Networking (TON) 23*, 3 (2015), 717–727.

[30] Levis, P.; Gay, D. *TinyOS programming*. Cambridge University Press, 2009.

[31] Levis, P.; Madden, S.; Polastre, J.; Szewczyk, R.; Whitehouse, K.; Woo, A.; Gay, D.; Hill, J.; Welsh, M.; Brewer, E., et al. Tinyos: An operating system for sensor networks. *Ambient intelligence 35* (2004).

[32] Levis, P. A.; Patel, N.; Culler, D.; Shenker, S. *Trickle: A self regulating algorithm for code propagation and maintenance in wireless sensor networks*. Computer Science Division, University of California, 2003.

[33] Maróti, M.; Kusy, B.; Simon, G.; Lédeczi, Á. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems* (2004), ACM, pp. 39–49.

[34] MEMSIC. Iris datasheet. http://www.memsic.com.

[35] MEMSIC. Micaz datasheet. http://www.memsic.com.

[36] Miklos Maroti, J. S. Tinyos extension proposal (tep) 132: Packet timestamping, 2010. "Online; acessado em 01/02/2016".

[37] Mills, D. Internet time synchronization: the network time protocol. *Communications, IEEE Transactions on 39*, 10 (Oct 1991), 1482–1493.

[38] Moss, D.; Levis, P. Box-macs: Exploiting physical and link layer boundaries in low-power networking. *Computer Systems Laboratory Stanford University* (2008), 116–119.

[39] Nazemi Gelyan, S.; Eghbali, A.; Roustapoor, L.; Yahyavi Firouz Abadi, S.; Dehghan, M. Sltp: Scalable lightweight time synchronization protocol for wireless sensor network. In *Mobile Ad-Hoc and Sensor Networks*, vol. 4864. Springer Berlin, 2007, pp. 536–547.

[40] Oliveira, L. M.; Rodrigues, J. J. Wireless sensor networks: a survey on environmental monitoring. *Journal of communications 6*, 2 (2011), 143–151.

[41] Otero, J.; Yalamanchili, P.; Braun, H.-W. High performance wireless networking and weather. *High Performance Wireless Research and Education Network* (2001).

[42] Polastre, J.; Hill, J.; Culler, D. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems* (2004), ACM, pp. 95–107.

[43] Pottie, G. J.; Kaiser, W. J. Wireless integrated network sensors. *Communications of the ACM 43*, 5 (2000), 51–58.

[44] Ranganathan, P.; Nygard, K. Time synchronization in wireless sensor networks: a survey. *International journal of UbiComp (IJU) 1*, 2 (2010), 92–102.

[45] Shannon, J.; Melvin, H.; Ruzzelli, A. G. Dynamic flooding time synchronisation protocol for wsns. In *Global Communications Conference (GLOBECOM), 2012 IEEE* (2012), IEEE, pp. 365–371.

[46] Sharp, C.; Turon, M.; Gay, D. Tinyos extension proposal (tep) 102: Timers, 2010. "Online; acessado em 01/02/2016".

[47] Simon, G.; Maróti, M.; Lédeczi, Á.; Balogh, G.; Kusy, B.; Nádas, A.; Pap, G.; Sallai, J.; Frampton, K. Sensor network-based countersniper system. In *Proceedings of the 2nd international conference on Embedded networked sensor systems* (2004), ACM, pp. 1–12.

[48] Singh, S.; Raghavendra, C. S. Pamas-power aware multi-access protocol with signalling for ad hoc networks. *ACM SIGCOMM Computer Communication Review 28*, 3 (1998), 5–26.

[49] Sinopoli, B.; Sharp, C.; Schenato, L.; Schaffert, S.; Sastry, S. S. Distributed control applications within sensor networks. *Proceedings of the IEEE 91*, 8 (2003), 1235–1246.

[50] Sommer, P.; Wattenhofer, R. Gradient clock synchronization in wireless sensor networks. In *International Conference on Information Processing in Sensor Networks* (April 2009), pp. 37–48.

[51] Sousa, C.; Carrano, R. C.; Magalhaes, L.; Albuquerque, C. V. Stele: A simple technique for local delay estimation in wsn. In *Computers and Communication (ISCC), 2014 IEEE Symposium on* (2014), IEEE, pp. 1–6.

[52] Stann, F.; Heidemann, J.; Shroff, R.; Murtaza, M. Z. Rbp: robust broadcast propagation in wireless networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems* (2006), ACM, pp. 85–98.

[53] Sundararaman, B.; Buy, U.; Kshemkalyani, A. D. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks 3*, 3 (2005), 281–323.

[54] Suriyachai, P.; Roedig, U.; Scott, A. A survey of mac protocols for mission-critical applications in wireless sensor networks. *Communications Surveys & Tutorials, IEEE 14*, 2 (2012), 240–264.

[55] TinyOS. Tinyos extension proposal (tep) 133: Packet-level time synchronization. "Online; acessado em 01/02/2016".

[56] van Greunen, J.; Rabaey, J. Lightweight time synchronization for sensor networks. In *Proceedings of the 2Nd ACM International Conference on Wireless Sensor Networks and Applications* (2003), ACM, pp. 11–19.

[57] Wang, L. *Topology-Based Routing for Xmesh in Dense Wireless Sensor Networks*. ProQuest, 2007.

[58] Wang, S.; Ahn, T. Mac layer timestamping approach for emerging wireless sensor platform and communication architecture, Dezembro 2009. US Patent App. 12/213,286.

[59] Yang, H.; Sikdar, B. A protocol for tracking mobile targets using sensor networks. In *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on* (2003), IEEE, pp. 71–81.

[60] Ye, W.; Heidemann, J.; Estrin, D. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2002), vol. 3, IEEE, pp. 1567–1576.

[61] Zanatta, G.; Bottari, G. D.; Guerra, R.; Leite, J. C. B. Building a WSN infrastructure with COTS components for the thermal monitoring of datacenters. In *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014* (2014), pp. 1443–1448.

[62] Zhu, T.; Zhong, Z.; He, T.; Zhang, Z.-L. Exploring link correlation for efficient flooding in wireless sensor networks. In *NSDI* (2010), vol. 10, pp. 1–15.

APÊNDICE A – Configuração Experimento Flocklab

A configuração de um teste no Flocklab é realizada utilizando um arquivo XML, ele define tudo que é necessário fara um simples experimento no *testbed*. O XML determina quando um teste deve iniciar e parar, qual plataforma e sistema operacional será usado, o que será monitorado e como será a recuperação dos resultados.

A.1 Estrutura de configuração de um teste