HORÁCIO ANTONIO BRAGA FERNANDES DE OLIVEIRA

LOCALIZAÇÃO NO TEMPO E NO ESPAÇO EM REDES DE SENSORES SEM FIO

Belo Horizonte
Maio de 2008

HORÁCIO ANTONIO BRAGA FERNANDES DE OLIVEIRA
Orientador: Antonio Alfredo Ferreira Loureiro

LOCALIZAÇÃO NO TEMPO E NO ESPAÇO EM REDES DE SENSORES SEM FIO

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

HORÁCIO ANTONIO BRAGA FERNANDES DE OLIVEIRA

Belo Horizonte
Maio de 2008

HORÁCIO ANTONIO BRAGA FERNANDES DE OLIVEIRA
Advisor: Antonio Alfredo Ferreira Loureiro

LOCALIZATION IN TIME AND SPACE FOR WIRELESS SENSOR NETWORKS

Thesis presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

HORÁCIO ANTONIO BRAGA FERNANDES DE OLIVEIRA

Belo Horizonte
May 2008

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

Localização no Tempo e no Espaço em Redes de Sensores Sem Fio

HORÁCIO ANTONIO BRAGA FERNANDES DE OLIVEIRA

Tese defendida e aprovada pela banca examinadora constituída por:

Ph. D. Antonio Alfredo Ferreira Loureiro – Orientador
Universidade Federal de Minas Gerais

Ph. D. Eduardo Freire Nakamura – Co-orientador
FUCAPI

Ph. D. Alejandro C. Frery
Universidade Federal de Alagoas

Ph. D. Azzedine Boukerche
Universidade de Ottawa

Ph. D. Geraldo Robson Mateus
Universidade Federal de Minas Gerais

Ph. D. Wagner Meira Junior
Universidade Federal de Minas Gerais

Belo Horizonte, Maio de 2008

Aos meus pais.

Agradecimentos

Agradeço a Deus pela oportunidade de aperfeiçoamento tanto intelectual quanto moral. Pela presença constante em cada momento de minha vida. Pela família e pelos amigos.

Ao meu orientador, Prof. Loureiro, pela ajuda, incentivo, esclarecimento e, acima de tudo, exemplo. Ao meu co-orientador Nakamura pelo apoio, direcionamento, idéias e sugestões nos diversos projetos. À Linnyer pelo convite de participar do tBb e pelas diversas sugestões e acompanhamento.

Agradeço o pessoal de Manaus, que fizeram da UFMG a universidade mais "Manauara" do Brasil, com exceção da própria UFAM, claro. Gostaria de agradecer em especial os Professores Pio e Ruiter pela companhia, amizade e apoio desde minha chegada de Manaus. Agradeço também o Guillermo por dividir o aluguel comigo por mais de um ano.

Ao pessoal do Laboratório ATM pela companhia e ajuda diária. Pelo ambiente descontraído e, ao mesmo tempo, profissional que encontrei no laboratório.

Muito tenho a agradecer a meus pais que, mesmo distantes fisicamente, permaneceram próximos em pensamento através de suas orações, conversas e conselhos. À minha irmãzinha Liginha pelas mensagens de apoio e incentivo. Ao meu irmão David, por ter estado ao meu lado, me ajudando e incentivando, nestes últimos anos.

Agradeço à minha família de Além Paraíba (MG) e do Rio de Janeiro por terem me acolhido nas férias, ajudando a aliviar um pouco da imensa saudade e falta que sentia de minha família em Manaus.

Por fim, agradeço aos amigos de Manaus que, mesmo distantes, estiveram sempre presente através de suas mensagens e lembranças.

Resumo

Redes de Sensores Sem Fio (RSSF) são redes basicamente guiadas por eventos. Um evento pode ser definido como sendo composto por um critério causal e um critério espaço-temporal. No primeiro caso, o tipo do evento é identificado, enquanto que no segundo, a localização no tempo e no espaço em que o evento ocorreu é especificada. Um nó sensor é capaz de identificar o primeiro critério (tipo do evento) facilmente usando os seus diversos sensores. O critério espaço-temporal, entretanto, só pode ser identificado quando os nós sensores de uma RSSF possuem relógios sincronizados e são capazes de determinar suas posições físicas. Informações de tempo e espaço são necessárias também para uma série de algoritmos e protocolos em RSSFs como, por exemplo, em fusão de dados, rastreamento de objetos, contrução de mapas de energia, controle de densidade, etc. Logo, sincronização e localização em RSSFs são problemas importantes que precisam ser estudados e analisados.

Nesta tese, propõe-se que sincronização e localização em RSSFs são, na verdade, duas partes do mesmo problema: localizar os nós da rede no tempo-espaço. Nestas redes, nem informações de espaço sem tempo nem de tempo sem espaço podem ser consideradas informações completas. Além disso, as semelhanças entre os problemas de localização e sincronização sugerem que eles podem e devem ser estudados e analisados como um único problema: localização no tempo e no espaço. Sob este ponto de vista, tempo pode ser visto como uma dimensão do espaço, i.e., precisa-se resolver um problema de localização em 4-dimensões. Como resultado, ao se resolver ambos os problemas ao mesmo tempo, a utilização dos recursos da rede pode ser minimizada e ambos os problemas podem apresentar melhores resultados se comparado com os casos em que eles são resolvidos separadamente.

Três soluções diferentes para o problema de localização no tempo e no espaço são então propostas tendo-se em vista diferentes cenários em RSSFs. Tais soluções são os algoritmos Synapse, Lightness e Mobilis. Os algoritmos propostos não apenas aproveitam os recursos adicionais de hardware necessários para a localização dos nós para melhorar os resultados da sincronização como também aproveitam o custo maior de comunicação necessários para a sincronização de forma a melhorar a localização dos nós. Diversos experimentos são mostrados para avaliar a performance dos algoritmos propostos. Os resultados obtidos mostram que as soluções propostas são capazes de serem implementadas em RSSFs e também confirmam as vantagens em se resolver tanto a localização quanto a sincronização em um mesmo algoritmo.

Resumo Estendido

O documento desta tese foi originalmente redigido em inglês com o título "Localization in Time and Space for Wireless Sensor Networks". Para estar em conformidade com as normas da Universidade Federal de Minas Gerais, este resumo em português faz uma exposição abreviada de cada um dos capítulos que compões esta tese.

Resumo do Capítulo 1 - Introdução

Uma Rede de Sensores Sem Fio (RSSF) pode ser definida como uma rede composta por centenas ou milhares de pequenos nós sensores com limitações de recursos e que trocam mensagens entre si utilizando-se de comunicação sem fio de múltiplos saltos. RSSFs podem ser vistas como Redes Ad Hoc relativamente estáticas compostas basicamente por nós sensores. Estes sensores têm a capacidade de monitorar propriedades físicas em suas proximidades, como temperatura, umidade, pressão, luminosidade e movimento. Um nó sensor por si só tem uma capacidade bem limitada, mas quando combinada com centenas ou milhares de outros nós, a capacidade geral de monitoramento é drasticamente aumentada. Logo, nós sensores em uma RSSF podem cooperativamente executar tarefas desafiadores como monitorar precisamente as propriedades físicas de uma área extensa de interesse. Devido à sua ampla aplicação, este tipo de rede tem se tornado popular e abrangendo as mais diferentes áreas, como ambiental, médica, industrial e militar.

A principal tarefa de uma RSSF é a detecção de eventos, coleta de dados, e o relato de tais dados a uma estação de monitoramento (ver figura 1.1). Para se comunicar com a estação de monitoramento, um nó especial conhecido como nó sorvedouro é responsável por agrupar todos os dados coletados pelos nós sensores e enviar tais dados para a estação de monitoramento utilizando-se de uma comunicação mais poderosa (e.g., comunicação via satélites). Logo, uma RSSF é basicamente guiada por eventos que geram dados. Um evento por si só pode ser definido como composto por um critério causal e por um critério espaçotemporal:

1. *Critério Causal:* especifica o tipo de evento. No caso das aplicações em RSSFs, este critério causal pode ser fogo, movimento, mudanças na pressão ou na luminosidade.

2. *Critério Espaçotemporal:* especifica a localização no tempo e no espaço que um determinado evento ocorreu. Tempo pode ser representado em termos do *Coordinated Universal Time* (UTC) enquanto que espaço pode ser representado por latitude, longitude e altura.

Um nó sensor pode identificar o primeiro critério utilizando-se de seus próprios dispositivos. Entretanto, o critério espaçotemporal só pode ser identificado quando os nós sensores possuem relógios sincronizados e podem determinar sua localização física. Porém, nem a localização física nem a sincronização entre os nós estão disponíveis após o lançamento dos nós sensores na área de interesse. Logo, algoritmos e protocolos para localização e sincronização são problemas importantes a serem tratados em RSSFs.

Apesar de os problemas de localização e sincronização estarem fortemente relacionados entre si e ambos são necessários para a maioria das aplicações em RSSFs, tais problemas têm sido pesquisados basicamente como dois problemas completamente independentes, necessitando de soluções de algoritmos, protocolos e técnicas diferentes. Como resultado, soluções atuais para os problemas de localização e sincronização são totalmente independentes um dos outros e esta execução independente dos algoritmos resulta em uma menor eficiência tanto em termos de custo como de precisão. Como será mostrado neste trabalho, ao se resolver ambos os problemas conjuntamente, pode-se reduzir tanto o consumo de recursos como pode-se reduzir também os erros de localização e sincronização. Neste ponto de vista, tempo pode ser visto como uma outra dimensão do espaço. Como consequência, pretende-se solucionar um problema de posicionamento em 4D. Neste trabalho, este problema será tratado como *Localização no Tempo e no Espaço*.

Os principais objetivos deste trabalho são dois. Primeiro, pretende-se propor e investigar o problema de *Localização no Tempo e no Espaço* em RSSFs como a combinação dos problemas de localização e sincronização. O segundo objetivo é propor, desenvolver e avaliar a performance de diferentes tipos de algoritmos para localização no tempo e no espaço em RSSFs. Desta forma, as principais contribuições deste trabalho são a proposta e desenvolvimento de três novos algoritmos para localização no tempo e no espaço para RSSFs, denominados Synapse, Lightness e Mobilis, respectivamente.

Esta tese está dividida em oito capítulos. Na primeira parte deste trabalho, composta pelos capítulos 2 e 3, são apresentadas uma visão geral e uma de definição dos problemas de localização e sincronização em RSSFs. Em ambos os capítulos, os sistemas de localização e sincronização são divididos em diferentes componentes. Cada componente é estudado mostrando-se e analisando-se as principais técnicas e soluções aplicadas em cada um deles. Ao final dos dois capítulos, algumas observações finais são feitas a respeito dos principais desafios e pontos ainda em aberto que precisam ser estudados. Na segunda parte, composta pelo capítulo 4, o problemas de *Localização no Tempo e no Espaço* é estudado. Este capítulo começa mostrando-se a importância da localização e sincronização na maioria das aplicações em RSSFs e mostrando-se que localização e sincronização são na realidade duas partes do mesmo problema: localizar os nós sensores no tempo-espaço. É mostrado também que nem localização sem tempo e nem tempo sem localização podem ser consideradas informações completas na maioria dos casos. Em seguida, são apresentadas as semelhanças entre localização e sincronização que sugerem que ambos podem e devem ser tratados como um único problema. Por último, o problema de Localização no Tempo e no Espaço é formalmente definido e analisado. Na terceira parte desta tese, composta pelos capítulos 5, 6 e 7, são

explicados os algoritmos Synapse, Lightness e Mobilis para localização no tempo e no espaço, respectivamente. Em cada capítulo, a performance do algoritmo proposto é avaliada através de simulações. Finalmente, no capítulo 8, são apresentadas algumas conclusões e observações finais a respeito do presente trabalho e dos resultados alcançados.

Resumo do Capítulo 2 - Sistemas de Localização em Redes de Sensores Sem Fio

O problema da localização consiste em identificar a posição física (e.g., latitude, longitude, altitude) de um determinado objeto. Tal problema é bastante amplo e extenso, abrangendo áreas como robótica, redes ad hoc, redes de sensores sem fio, telefonia sem fio, militar, aviação e astronomia. Neste trabalho o problema de localização será abordado sob o ponto de vista das RSSFs.

A importância dos sistemas de localização decorre de diversos fatores, muitos deles restritos apenas às RSSFs. Alguns destes fatores incluem: identificar os dados coletados pelos nós, relacionar dados coletados, endereçamento dos nós, gerenciamento da rede e permitir o funcionamento dos algoritmos geográficos.

Uma das soluções mais simples para o problema da localização em uma RSSF é acoplar um receptor GPS em cada nó da rede. Uma das principais vantagens seria um erro de localização relativamente pequeno (2-15m, dependendo do receptor) e bastante preciso, uma vez que todos os nós teriam um erro similar. Entretanto, esta solução apresenta uma série de desvantagens como aumento do custo, falta de visada aos satélites, imprecisão, aumento do tamanho dos sensores e consumo de energia dentre outros.

Neste capítulo o problema de localização foi abordado sob o ponto de vista das redes de sensores sem fio. Os sistemas de localização foram divididos em três componentes: estimativa de distância/ângulo, cálculo de posição, e algoritmo de localização. A importância de se dividir os sistemas de localização em componentes, vem da necessidade de se reconhecer que o desempenho final do sistema de localização depende diretamente de cada um dos componentes que este se utiliza. Por exemplo, um sistema de localização poderia obter erros muito menores caso a técnica de TDoA fosse utilizado para estimar distâncias, ao invés do RSSI.

Seguindo no capítulo, cada um dos componentes foi estudado isoladamente e alguns métodos utilizados em cada um destes componentes foram também estudados e analisados. A principal conclusão que se pode verificar em cada um dos componentes é que o método de solução utilizado depende diretamente da aplicação e dos recursos disponíveis. Não há, portanto, uma solução única capaz de atender a todos os casos. A mesma conclusão pode ser obtida em relação aos sistemas de localização em geral. Não há uma solução única capaz de atender de forma ótima os diversos requisitos das redes de sensores nos mais variados cenários e aplicações. Há sim, soluções que obtém um bom desempenho em um determinado cenário e que necessitam de determinados recursos para funcionarem bem.

Resumo do Capítulo 3 - Sistemas de Sincronização em Redes de Sensores Sem Fio

O problema de sincronização consiste em sincronizar os relógios dos nós sensores com base em um tempo de referência ou no *Coordinated Universal Time* (UTC). Relógios sincronizados são muito importantes na maioria das aplicações em RSSFs já que estes são necessários em muitos protocolos, algoritmos e aplicações como fusão de dados, rastreamento de objetos, algoritmos de controle de densidade e ordenação de eventos.

Neste capítulo, o problema de sincronização é estudado sob o ponto de vista das RSSFs. São apresentadas uma visão geral e uma definição do problema de sincronização e de seus componentes. É mostrado os principais métodos usados pelos sistemas de sincronização para estimar o tempo de atraso de um pacote deixando um transmissor e chegando em um nó receptor. Esta estimativa de atraso é então utilizado para calcular o tempo do receptor com base no tempo do transmissor e no atraso do pacote, como mostrado na seção 3.3. Em seguida, é mostrado como as informação de atraso e tempos podem ser utilizadas para sincronizar os relógios de todos os nós da rede.

Ainda neste capítulo, os sistemas de sincronização são divididos em três componentes: estimativa de atraso, cálculo do tempo e algoritmo de sincronização. A importância de se dividir tais sistemas em componentes resulta da necessidade de se reconhecer que a performance final do sistema de sincronização dependerá diretamente de cada um destes componentes. Por exemplo, um sistema de sincronização poderia obter melhores resultados se a técnica de RTT (*Round-Trip Time*) fosse utilizada ao invés da técnica de estimativa de atrasos. O mesmo princípio se aplica aos outros componentes. Estes componentes podem ser vistos como subáreas do problema de sincronização que precisam ser estudadas separadamente.

Conforme esperado, em RSSFs, não á uma única, perfeita solução para o problema de sincronização que seja aplicável a todos os cenários. Neste capítulo são discutidos ainda alguns sistemas de sincronização propostos na literatura, cada qual enfatizando um cenário e/ou aplicação específicos. A necessidade de diferentes soluções para diferentes aplicação, bem como o grande número de aplicações em RSSFs, têm motivado fortemente o estudo e desenvolvimento de novas soluções para o problema de sincronização.

Resumo do Capítulo 4 - Localização no Tempo e no Espaço em Redes de Sensores Sem Fio

Como mencionado anteriormente e ilustrado na figura 1.1, RSSFs são comumente desenvolvidas para detectar eventos e coletar e entregar os dados sensoreados para as estações de monitoramento. Logo, uma RSSF é basicamente guiada por eventos.

Na definição de um evento citada anteriormente, dois eventos são o mesmo se, e somente se, eles possuem os mesmos critérios causal e espaçotemporal. A figura 4.1 ilustra quatro diferentes eventos em que pode-se facilmente identificar os critérios causais e espaçotemporais. Por exemplo, no primeiro evento, o `critério causal` é o `fogo` e o *critério espaçotemporal*

é `(1.72, 2.32, 0.53, 1203351004.1981)`. Eventos podem também ser ordenados e relacionados entre eles pelas leis de causalidade. Causalidade é uma relação direcional entre um evento (a causa) e outro evento (o efeito) que é consequência (resultado) do primeiro. No exemplo da figura 4.1, pode-se identificar que o evento 2 (e.g., movimento de animais) foi causado pelo (ou é efeito do) evento 1 (e.g., fogo na floresta).

A partir destes exemplos, pode-se facilmente perceber que em RSSFs baseadas em eventos, nem posição sem tempo nem tempo sem posição representam informações completas. Logo, é necessário tanto informações de posição quanto de tempo para poder se definir um evento completamente, confirme ilustrado na figura 4.4. Finalmente, além de serem necessários para se identificar o critério espaçotemporal dos evento, localização e sincronização são também necessários por uma série de outros protocolos, aplicações e algoritmos em RSSFs, a saber: fusão de dados, rastreamento de objetos, construção de mapas de energia, controle de densidade, etc.

Conforme mencionado anteriormente, as características do problema de sincronização são similares aos do problema de localização. Ao se notar com mais cuidado as soluções propostas para ambos os problemas, pode-se identificar uma série de similaridades entre eles. Por exemplo, os sistemas de localização, conforme mostrado no capítulo 2, podem ser divididos em componentes, cada qual responsável por solucionar uma parte do problema de localização, a saber: estimativa de distâncias, cálculo de posições, algoritmo de localização. Da mesma forma, os sistemas de sincronização podem também ser divididos em componentes semelhantes, conforme mostrado no capítulo 3, a saber: estimativa de atrasos, cálculo de tempos e algoritmo de sincronização. Logo, sistemas de localização e sincronização estão claramente relacionadas entre si, conforme mostrado na figura 4.5.

Desta forma, neste capítulo, é mostrada a importância da visão unificada da localização e sincronização em RSSFs. Nos próximos três capítulos da tese, novos algoritmos para localização no tempo e no espaço em RSSFs serão propostos e avaliados, a saber: o algoritmo Synapse, Lightness e Mobilis, respectivamente.

Resumo do Capítulo 5 - Um Algoritmo DV-Hop para Localização no Tempo-espaço em RSSFs

O Algoritmo Synapse (*Synchronization and Positioning for Sensor networks*) é um algoritmo de localização no tempo e no espaço desenvolvido para ser utilizado em RSSFs de pequeno a médio porte. O algoritmo Synapse é baseado no algoritmo de localização APS (*Ad Hoc Positioning System*). Este obedece o mesmo princípio do APS, resultando nas mesmas características de localização e utilização de recursos da rede, porém com o adicional de se poder localizar os nós sensores no tempo (i.e., sincronização). Este algoritmo se baseia no fato de que os nós beacons (nós equipados com GPS que ajudam a localizar os outros nós) já possuem relógios sincronizados, já que eles podem pegar tais informações de tempo do GPS. Relógios sincronizados localizados em diferentes partes da rede são capazes de estimar o tempo que um pacote de dados levou para deixar um beacon e chegar em outro. Baseado nesta informação,

pode-se calcular o *tempo médio de um salto*. Um nó normal pode sincronizar seu relógio considerando o tempo médio de um salto e também o número de saltos entre ele e os nós beacons. Resultados de simulações (apresentados na seção 5.3) mostram que o algoritmo Synapse é capaz de sincronizar os relógios dos nós com uma precisão de poucos microssegundos. Este algoritmo é explicado em detalhes no capítulo 5.

Como mostrado neste capítulo, quando implementado na camada MAC (*Media Access Control*), o algoritmo proposto sincroniza dos nós com uma precisão de microssegundos. O algoritmo também pode ser implementado na camada de aplicação, obtendo uma precisão de milissegundos, uma vez que nesta camada ele fica sujeito a imprecisões de acesso ao meio. O Algoritmo Synapse também mostrou-se tolerante a imprecisões do mundo real como os obtidos pelo GPS e pelos atrasos não determinísticos dentro dos nós sensores.

Algumas das vantagens do Synapse são os seguintes: primeiramente, ele funciona em redes esparsas; em segundo, sua parte de posicionamento é imune a imprecisões do RSSI (*Received Signal Strengh Indicator*); e, último, este algoritmo requer um pequeno número de nós beacons (ele pode funcionar com apenas três nós beacons na rede). Por outro lado, o principal problema do algoritmo Synapse é o seu custo de comunicação. Como um algoritmo do tipo DV-hop, a complexidade de comunicação deste algoritmo é determinada pelo número de beacons e pelo número de nós da rede, i.e., $O(nb)$, onde n é o número de nós e b o número de beacons. Como pode-se ver, é um custo de comunicação relativamente alto, motivo pela qual tal algoritmo é mais adaptado a redes de pequeno a médio porte.

Uma vez que a escalabilidade ainda é um problema no Algoritmo Synapse, para lidar com cenários maiores em RSSFs, no próximo capítulo é proposto o algoritmo Lightness, uma solução mais escalável e eficiente para o problema de localização no tempo e no espaço em RSSFs.

Resumo do Capítulo 6 - Um Algoritmo Leve para Localização no Tempo-espaço em RSSFs

Neste capítulo, é proposto um novo e mais leve algoritmo de localização no tempo e no espaço para RSSFs, que é chamado de *Lightness* (Lightweight Time and Space localization). O Algoritmo Lightness possui alguns aspectos chaves. Primeiramente, a parte de posicionamento do algoritmo é baseado no RPE (*Recursive Position Estimation*), um sistema de localização conhecido e escalável para redes e sensores que requer apenas 5% dos nós como sendo nós beacons equipados com GPS. Em segundo, este algoritmo assume que os nós beacons já possuem relógios sincronizados, uma vez que estes podem pegar suas informações de tempo a partir do GPS. Em terceiro, tanto as informações de localização como de sincronização são propagadas na rede ao mesmo tempo nos mesmos pacotes de dados com um custo total de comunicação em $O(n)$, i.e., cada nó envia apenas um único pacote de dados. Finalmente, é utilizada a técnica de *packet delay measurement technique* (explicada na seção 3.2.2) para sincronizar nós vizinhos utilizando-se apenas um único *broadcast*. Resultados de simulação mostram que o algoritmo proposto Lightness pode ser escalável a milhares de nós mantendo o

mesmo erro de sincronização de alguns poucos microssegundos ao mesmo tempo que é capaz de reduzir o erro de localização (quando comparado ao RPE original). Este algoritmo é explicado em detalhes no capítulo 6.

Uma desvantagem do algoritmo proposto é que ele não funciona em redes muito esparsas. Em redes normais ou de alta densidade, o aumento do número de vizinhos resulta em uma quantidade maior de referências para que um nó calcule sua posição e sincronize seu relógio. Quando um nó não possui pelo menos três vizinhos, este não calculará sua posição e não poderá encaminhar a recursão de localização e sincronização, o que fará com que o algoritmo termine antes que todos os nós tenham estimado suas informações de localização no tempo e no espaço.

Considerando estas características, pode-se dizer que o algoritmo proposto é mais aplicável a redes densas e de larga escala em cenários abertos (*outdoors scenarios*) onde há a possibilidade de se lançar pelo menos 5% dos nós como sendo nós beacons equipados com GPS.

Resumo do Capítulo 7 - Uma Abordagem com Beacons Móveis para Localização no Tempo-espaço em RSSFs

O Algoritmo Mobilis (*Mobile Beacon for Localization and Synchronization*) é uma nova abordagem para solucionar o problema de localização no tempo e no espaço em RSSFs que utiliza os serviços de um beacon móvel. Um beacon móvel é um nó que possui o conhecimento de seu tempo e posição a todo instante (e.g., equipado com GPS) e que tem a habilidade de se locomover na área de interessa da RSSF. Este beacon móvel pode ser uma pessoa, um veículo não-tripulado, um avião, um robô, etc. Tal tecnologia de beacon móvel tem sido aplicada com sucesso para resolver o problema de localização em algumas redes. Entretanto, esta é a primeira vez que tal recurso é utilizado para tratar do problema de sincronização e de localização no tempo e no espaço. O algoritmo Mobilis proposto pode sincronizar os nós utilizando a técnica de *packet delay measurement technique* (explicada na seção 3.2.2) ou a técnica de *round-trip time* (RTT – explicada na seção 3.2.1). No primeiro caso, a sincronização pode ser melhorada pelos pacotes extras necessários pela localização, enquanto que no segundo caso, a localização pode ser melhorada pelos pacotes extras necessários pela sincronização. Este algoritmo é explicado em detalhes no capítulo 7.

Em termos de aplicabilidade do algoritmo proposto, pode-se dizer que este algoritmo é mais aplicável para RSSFs externas ou submersas onde há a possibilidade de se lançar um nó beacon móvel. O algoritmo proposto não possui problemas de escalabilidade ou de densidade no que diz respeito aos resultados obtidos ou custo de comunicação. Os resultados obtidos, em termos de localização e sincronização foram os melhores em comparação com as outras soluções propostas neste trabalho. Tais resultados mostram que o algoritmo Mobilis é capaz de sincronizar os nós com um erro médio de apenas 2-4 μs e um erro de localização de 10% do raio de comunicação, o que torna tal algoritmo viável para a maioria das aplicações e protocolos em RSSFs.

O principal problema do algoritmo Mobilis proposto é que o mesmo requer o uso de um

beacon móvel capaz de se mover por toda a rede. Além disso, dependendo da velocidade do beacon móvel, alguns nós terão que esperar por um longo tempo até serem capazes de estimar suas informação de localização no tempo e no espaço. Entretanto, apesar destas desvantagens, o uso de um beacon móvel é ainda considerada como uma das únicas soluções disponíveis para cenários de RSSFs submersas.

Resumo do Capítulo 8 - Conclusões

Nesta tese, mostrou-se a importância de se combinar os problemas de localização e sincronização em um único problema: Localização no Tempo e no Espaço. Assim, o tempo pode ser observado como uma nova dimensão de forma que pretende-se agora resolver um problema de localização em quatro dimensões. O ponto principal é que, na maioria dos casos, uma posição sem informação de tempo ou uma informação de tempo sem a posição não podem ser consideradas informações completas para as aplicações. As semelhanças entre ambos os problemas também sugerem que eles podem e devem ser tratados como um único problema. Ao se fazer isso, recursos da rede podem ser economizados e, mais importante, ambos os problemas podem apresentar melhores resultados.

Devido à impossibilidade de se ter uma única solução para um determinado problema em RSSFs, nesta tese são propostas três algoritmos diferentes para localização no tempo e no espaço: o algoritmo Synapse, Lightness e Mobilis. Dependendo do cenário e das aplicações da rede, cada um dos algoritmos propostos tem as suas vantagens e desvantagens. Na tabela 8.1 são comparadas algumas das principais características dos algoritmos propostos em relação aos trabalhos encontrados na literatura.

Finalmente, as soluções propostas confirmam uma nova tendência em RSSFs: soluções integradas. Nas soluções integradas, diferentes problemas são solucionados em conjunto para prover soluções melhores e mais econômicas. Alguns trabalhos recentes também confirmam esta tendência em RSSFs.

Abstract

Wireless Sensor Network (WSN) based applications are usually event driven. An event by itself can be defined as being composed of a causal criterion and a spatiotemporal criterion. The first criterion specifies the type of event, while the second specifies the location in time and space where the event occurred. A sensor node is able to identify the first criterion easily by using its own sensing devices. The spatiotemporal criterion, on the other hand, can only be identified when the sensor nodes of a WSN have synchronized clocks and are able to determine their physical location. Time and space information is also required by a number of algorithms and protocols in WSNs such as information fusion, object tracking, energy maps, and density control. Thus, synchronization and positioning for WSN-based applications are challenging problems that need to be addressed.

In this thesis, we identify that *synchronization* and *positioning* in WSNs are actually two parts of the same problem, i.e., locating the nodes in a network in time-space. Neither *location without time* nor *time without location* is complete information in these networks. The similarities between the localization and synchronization problems also suggest that they can and should be addressed as a single problem. From our perspective, time can be seen as another dimension of space. As a consequence, we need to solve a 4D positioning problem. This approach allows us to save network resources and solve both problems more efficiently.

We propose three different solutions for the localization in time and space problem that are suitable for different scenarios in WSNs. These proposed solutions are called the Synapse, Lightness, and Mobilis algorithms. Our proposed algorithms not only take advantage of the additional hardware resources required by the positioning mechanism in order to improve the performance and scalability of synchronization, but also benefit from the additional communication required by the synchronization mechanism to decrease positioning errors. We also present an extensive set of experiments to evaluate the performance of our algorithms. Our results indicate that our proposed schemes are suitable for implementation in WSNs and also highlight the advantages of solving both localization and synchronization problems with a unified algorithm.

List of Publications

Publications done by the author during the doctorate:

Chapters

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. Localization Systems for Wireless Sensor Networks. Chapter in Book *Algorithms and Protocols for Wireless Sensor Networks* by Azzedine Boukerche. *Wiley&Sons*. Accepted to appear in 2008.

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. Secure Localization Systems: Protocols and Techniques in Wireless Sensor Networks. Chapter in Book *Algorithms and Protocols for Wireless Sensor Networks* by Azzedine Boukerche. *Wiley&Sons*. Accepted to appear in 2008.

Journals

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. Localization Systems for Wireless Sensor Networks. In *IEEE Wireless Communications* - Special Issue on Wireless Sensor Networks. Pages 6-12. Vol 14 (6). December 2007.

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. Using Voronoi Diagrams to Scale a Localization System in Wireless Sensor Networks. In *IEEE Wireless Communications*. Accepted to appear in 2008.

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. Secure Localization Algorithms for Wireless Sensor Networks. In *IEEE Communications Magazine*. May 2008. To Appear.

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. Vehicular Ad-hoc Networks - A New Challenge for Localization. In *Computer Communications - Elsevier*. Accepted to appear in 2008.

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. An Efficient Directed Localization Recursion Protocol for Wireless Sensor Networks. *IEEE Transactions on Computers*. Submitted. (2nd round of reviews)

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. An Efficient and Lightweight Algorithm for Time-Space Localization in Wireless Sensor Networks. *Special Issue of Performance Evaluation (Elsevier).* Submitted.

Papers

- Horacio A.B.F. Oliveira, Eduardo F. Nakamura, Antonio A.F. Loureiro, and Azzedine Boukerche. Localization in Time and Space for Sensor Networks. *AINA'07: Proceedings of the 21st IEEE International Conference on Advanced Information Networking and Applications.* Pages 539-546. Niagara Falls, Ontario, Canada. May 2007.

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. A Novel Lightweight Algorithm for Time-Space Localization in Wireless Sensor Networks. *MSWiM'07: Proceedings of the 10th ACM-IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems.* Pages 336-343. Chania, Crete Island, Greece. October 2007.

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura , and Antonio A.F. Loureiro. Localization in Time and Space for Wireless Sensor Networks: A Mobile Beacon Approach. In WoWMoM'08: Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks. Newport Beach, U.S.A. June 2008. To Appear.

- Azzedine Boukerche, Horacio A. B. F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. Enlightness: An Enhanced and Lightweight Algorithm for Time-Space Localization in Wireless Sensor Networks. *ISCC'08: Proceedings of the 13th IEEE Symposium on Computers and Communications.* Marrakech, Morocco. July 2008. To Appear.

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. A Novel Location-Free Greedy Forward Algorithm for Wireless Sensor Networks. *ICC'08: Proceedings of the 2008 IEEE International Conference on Communications.* Beijing, China. May, 2008. To Appear.

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. A Voronoi Approach for Scalable and Robust DV-Hop Localization System for Sensor Networks. *ICCCN'07: Proceedings of the 16th International Conference on Computer Communications and Networks.* Pages 497-502. Honolulu, Hawaii, USA. August 2007.

- Azzedine Boukerche, Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. Towards an Integrated Solution for Node Localization and Data Routing in Sensor Networks. *ISCC'07: Proceedings of the 12th IEEE Symposium on Computers and Communications.* Aveiro, Portugal. July 2007.

- Eduardo F. Nakamura, Luciana F. Pontello, Horacio A.B.F. Oliveira, and Antonio A.F. Loureiro. On Demand Role Assignment for Event-Detection in Sensor Networks. *ISCC'06: Proceedings of the 11th IEEE Symposium on Computers and Communications.* Pages 941-947. Pula-Cagliari, Sardinia, Italy. June 2006.

- Eduardo F. Nakamura, Horacio A.B.F. Oliveira, Luciana F. Pontello, and Antonio A.F. Loureiro. Atribuição Dinâmica de Papéis para Agregação de Dados em Redes de Sensores sem Fio. *SBRC'06: Proceedings of the 24o Simpósio Brasileiro de Redes de Computadores.* Curitiba, PR, Brazil. June 2006.

- Horacio A.B.F. Oliveira, Eduardo F. Nakamura, Antonio A.F. Loureiro, and Azzedine Boukerche. Error Analysis of Localization Systems in Sensor Networks. *GIS'05: Proceedings of the 13th ACM International Symposium on Advances in Geographic Information Systems.* Pages 71-78. Bremen, Germany. November 2005.

- Horacio A.B.F. Oliveira, Eduardo F. Nakamura, Antonio A.F. Loureiro, and Azzedine Boukerche. Directed Position Estimation: a Recursive Localization Approach for Wireless Sensor Networks. *IC3N'05: Proceedings of the 14th IEEE International Conference on Computer Communications and Networks.* Pages 557-562. San Diego, California, USA, October 2005.

Technical Reports

- Horacio A.B.F. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. Directed Position Estimation: a Recursive Localization Approach for Wireless Sensor Networks. Technical Report. RT.DCC 010/2005, Department of Computer Science, Federal University of Minas Gerais. Belo Horizonte, MG, Brazil, 2005.

Master's Thesis

- Horacio A.B.F. Oliveira. Um Algoritmo Recursivo de Localização para Redes de Sensores Sem Fio. Master Thesis. Department of Computer Science, Federal University of Minas Gerais. Belo Horizonte, MG, Brazil. August 2005.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

A Wireless Sensor Network (WSN) (Akyildiz et al., 2002; Boukerche, 2005; Nikoletseas, 2006; Arampatzis et al., 2005; Estrin et al., 2001; Ilyas and Mahgoub, 2004; Loureiro et al., 2003; Pottie and Kaiser, 2000) can be described as a network composed of hundreds or thousands of small, deployable, and resource constrained sensor nodes that exchange messages with each other using wireless and multihop communication. WSNs can be seen as relatively static Ad Hoc Networks (Boukerche, 2005; Elliott and Heile, 2000; Estrin et al., 2002; Jones et al., 2001; Gibson, 1999) that are composed mainly of sensor nodes. These sensors have the capability of monitoring physical properties in their vicinity, such as temperature, humidity, pressure, ambient light, and movement. A sensor node by itself has a limited sensing capability, but when it is combined with hundreds or thousands of other nodes the overall capability can be increased. Thus, sensor nodes in a WSN can cooperatively perform challenging tasks such as monitoring precisely the physical properties of an extensive area of interest. Due to its wide applicability, this type of network has become popular and covers many different domains such as the environmental, medical, industrial, and military fields.

The basic tasks of a WSN are event detection, data gathering, and reporting data to a monitoring station (see Figure 1.1). In order to communicate with the monitoring station, a special node known as the sink node is responsible for collecting all of the data gathered (or fused) by the sensor nodes and sending them to this monitoring station using a more powerful communication infrastructure (e.g., satellite links). Thus, a WSN is basically driven by events that generate data. An event by itself can be defined as being composed of a *causal criterion* and a *spatiotemporal criterion* (Davidson, 1980):

1. *Causal criterion:* specifies the type of event. In the case of the main applications of a WSN, this causal criterion could be fire, movement, changes in pressure, or changes in ambient light.

2. *Spatiotemporal criterion:* specifies the location in time and space where an event occurred. Time can be represented in terms of the Coordinated Universal Time (UTC)

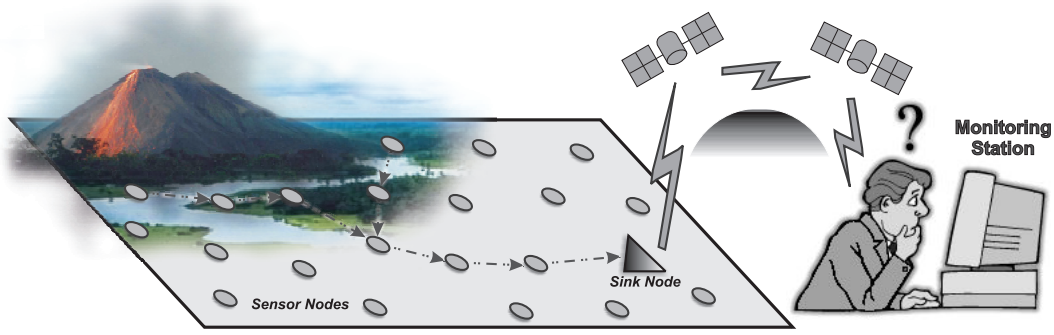


Figure 1.1: Example of a common scenario for Wireless Sensor Networks: sensor nodes detecting events from the environment, gathering data, and sending these data to a monitoring station.

while space can be represented by latitude, longitude, and altitude.

For instance, a typical event detected by a WSN application could be described as:

```
[Time: 1203351004.1981] --- "fire" detected at (1.72, 2.32, 0.53).
```

In this example, we can easily see the *causal criterion* of the detected event, which is the `fire`, and the *spatiotemporal criterion*, which is `(1.72, 2.32, 0.53, 1203351004.1981)`.

A sensor node in a WSN can identify the first criterion easily by using its own sensing devices and information-fusion techniques (Nakamura et al., 2005a, 2007). The *spatiotemporal criterion*, on the other hand, can only be identified when sensor nodes have synchronized clocks and can determine their physical location. However, neither positioning nor timing information is promptly available to sensor nodes after the nodes' deployment. First, the position of sensor nodes may not be engineered or predetermined. This allows for random deployment in inaccessible terrains and disaster relief operations (Akyildiz et al., 2002). Thus, a localization system (Bachrach and Taylor, 2005; Boukerche et al., 2007a; Chandrasekhar et al., 2006; Langendoen and Reijers, 2005) is required in order to provide the position information (e.g., latitude, longitude, and altitude) of the nodes. Second, the local clocks of the sensor nodes usually deviate gradually from the correct time due to external conditions such as temperature or battery voltage. Thus, a synchronization system (Sundararaman et al., 2005; Sivrikaya and Yener, 2004; Elson and Romer, 2003) is required to keep the clocks of the sensor nodes synchronized. Although the Global Positioning System (GPS) (Hofmann-Wellenho et al., 1997; Kaplan, 1996) could provide both timing and positioning information for the nodes, to equip all sensor nodes with a GPS receiver is not a good solution in WSNs because it increases their cost, size, and energy consumption (Doherty et al., 2001; Niculescu and Nath, 2001; Savvides et al., 2001). Thus, distributed multihop algorithms and protocols are usually the best solutions for solving the localization and synchronization problems in WSNs.

Although the localization and synchronization problems are strongly related to each other and both are required in almost all WSNs, these problems have been investigated basically

as two different problems requiring different solutions, algorithms, techniques, and protocols. For this reason, current solutions for synchronization and positioning in WSNs are completely independent from each other. As a result, the independent execution of these algorithms leads to lower efficiency in terms of both cost and accuracy. As we will show in this work, by solving these two problems jointly we can reduce both synchronization and position errors, as well as save energy and network resources. From our perspective, time can be seen as another dimension of space. As a consequence, we need to solve a 4D positioning problem. We refer to this problem as *Localization in Time and Space*.

1.2 Objectives

The main goals of this work are twofold. First, we aim to propose and investigate the *Localization in Time and Space* problem in WSNs as a combination of both the localization and synchronization problems. The second goal is to propose, design, and evaluate the performance of different types of time-space localization algorithms for WSNs.

To achieve these goals, several secondary objectives need to be accomplished. In the first case, i.e., in order to develop and define the proposed concept of *Localization in Time and Space*, the following goals need to be achieved:

1.1 *assess localization and synchronization problems in WSNs and the proposed independent solutions to these problems;*

1.2 *identify similarities between the localization and synchronization problems;*

1.3 *evaluate how localization techniques can be used to propose new synchronization algorithms or improve the existing ones;* and

Secondly, to propose different solutions for the *localization in time and space* problem and consider the different scenarios where a WSN can be applied, the following goals need to be achieved:

2.1 *propose a solution for the localization in time and space problem to be used in medium scale WSNs;*

2.2 *propose a solution for the localization in time and space problem to be used in large scale WSNs;*

2.3 *propose a solution for the localization in time and space problem to be used in WSNs with mobile beacons, such as underwater WSNs;* and

2.4 *analyze the performance of the proposed solutions.*

1.3 Main Contributions

The main contributions of this work are the design and development of three new time-space localization algorithms for WSNs, which we refer to as the Synapse, Lightness, and Mobilis algorithms, respectively:

- *The Synapse algorithm* (Synchronization and Positioning for Sensor networks) is a time-space localization algorithm designed for small and medium scale WSNs. The Synapse algorithm is based on the Ad Hoc Positioning System (APS) localization system (Niculescu and Nath, 2001). It obeys the same principles as APS, resulting in the same localization features and communication overhead, but adds the capability of locating nodes in time (i.e., synchronization). This algorithm relies on the fact that beacon nodes (GPS-equipped nodes that help in locating other nodes) already have synchronized clocks, since they receive their timing information from GPS. Synchronized clocks located in different parts of the network are able to estimate the time taken for a packet to leave one beacon node and arrive at another. Based on this information, we can compute the *average time of a hop*. A regular node synchronizes its clock by considering the average time of a hop and the number of hops between itself and each beacon node. Simulation results (presented in Section 5.3) show that our proposed Synapse algorithm can synchronize the nodes' clocks with a precision of a few microseconds. This algorithm is explained fully in Chapter 5.

- *The Lightness algorithm* (Lightweight Time and Space localization) has some key aspects. First, the positioning part is based on Recursive Position Estimation (RPE) (Albowicz et al., 2001), a well-known and scalable positioning system for sensor networks that requires only 5% of the nodes to be GPS-enabled beacon nodes. Second, this algorithm assumes that the beacon nodes already have synchronized clocks, since they can get their timing information from the GPS. Third, both timing and positioning information are propagated in the network with a total communication cost of $O(n)$. Finally, we use the packet delay measurement technique (explained in Section 3.2.2) to estimate the delay of a packet and synchronize neighbor nodes using only one broadcast. Simulation results show that our Lightness algorithm can scale to thousands of nodes while keeping a synchronization error of a few microseconds and decreasing the positioning error (compared to the original RPE algorithm). This algorithm is explained in Chapter 6.

- *The Mobilis algorithm* (Mobile Beacon for Localization and Synchronization) is a new approach to solving the time-space localization problem that uses a mobile beacon. A mobile beacon is a node that is aware of its time and position (e.g. equipped with a GPS receiver) and that has the ability to move around the sensor field. This beacon can be a human operator, an unmanned vehicle, an aircraft, or a robot. A mobile beacon has been successfully applied to solve the positioning problem. However, to the best of our

knowledge, ours is the first algorithm to address the use of a mobile beacon in synchronization and time-space localization problems. Our Mobilis algorithm can synchronize nodes by using the packet delay measurement technique (explained in Section 3.2.2) or the packet round-trip time technique (RTT – explained in Section 3.2.1). In the first case, synchronization can be improved by the extra packets required for location discovery, while in the second case localization can be improved by the extra packets required for synchronization. This algorithm is explained in Chapter 7.

1.4 Organization of the Thesis

This thesis is divided into eight chapters. The first part of this work, composed of Chapters 2 and 3, presents an overview and definition of the localization and synchronization problems in WSNs. In both chapters, the localization and synchronization systems are divided into three different components. Each component is studied by showing and analyzing the main techniques proposed and the solutions employed by each of them. At the end of both chapters, some final remarks are made regarding the main challenges and open issues that still need to be addressed.

In the second part, composed of Chapter 4, the *Localization in Time and Space* problem is addressed. This chapter begins by showing the importance of localization and synchronization for most WSN applications and showing that synchronization and localization are two parts of the same problem: localizing the nodes in time-space. We also show that neither location without time nor time without location represents complete information in WSN applications. Then, we present the similarities between the localization and synchronization problems, which suggest that they can and should be addressed as a single problem. Next, the *Localization in Time and Space* problem for WSNs is formally defined and analyzed. We also present some related work that suggests the unification of localization and synchronization in WSNs.

In the third part of this work, composed of Chapters 5, 6, and 7, we propose and explain the Synapse, Lightness, and Mobilis algorithms for localization in time and space, respectively. In each chapter, the performance of the proposed solution is evaluated through simulations.

Finally, in Chapter 8, we present some final remarks about the studied problems, their solutions, and the obtained results. We also present some possible extensions of our work.

Chapter 2

Localization Systems for Wireless Sensor Networks

2.1 Introduction

Despite the fact that the main goal of a WSN is to monitor an area of interest, several secondary objectives, or prerequisites, have to be achieved in order to reach the main objective (Figure 2.1). The definition of a localization system (Boukerche et al., 2007a,c,b; Bachrach and Taylor, 2005; Ji and Zha, 2004; Pathirana et al., 2005; Savarese et al., 2002; Shang et al., 2003; Ssu et al., 2005) between the sensor nodes is one of these prerequisites and as well as a fundamental issue for many applications in WSNs. Because sensor networks may be deployed in inaccessible terrains or disaster relief operations (Akyildiz et al., 2002), the position of sensor nodes may not be predetermined. Thus, a localization system is required in order to provide position information to the nodes.

The importance of localization information arises from several factors, many of which are related only to WSNs. These factors include the identification and correlation of the gathered data (Nakamura et al., 2005a,b, 2007), the addressing of the nodes (Heidemann et al., 2001), the management and query of nodes localized in a determined region (Navas and Imielinski, 1997), evaluation of node density and coverage (Xu et al., 2001), construction of energy maps (Mini et al., 2004; Machado et al., 2005), geographic routing (Xu et al., 2001), object tracking (Kumar et al., 2000), and other geographic algorithms. All of these factors make localization systems a key technology for the development and operation of WSNs.

One solution for the localization problem in WSNs is to equip each sensor node with a Global Positioning System (GPS) receiver (Hofmann-Wellenho et al., 1997; Kaplan, 1996). However, this solution has many disadvantages (Doherty et al., 2001; Niculescu and Nath, 2001; Savvides et al., 2001), such as the increase in the cost and size of the sensor nodes, the fact that GPS cannot be used when there is no visible satellite (e.g., indoor scenarios, underwater, severe climatic conditions, mars exploration), and the extra hardware consumes energy. Due to these disadvantages, the usage of GPS is usually limited to a small fraction of the nodes (e.g., beacon nodes – explained in the next section).

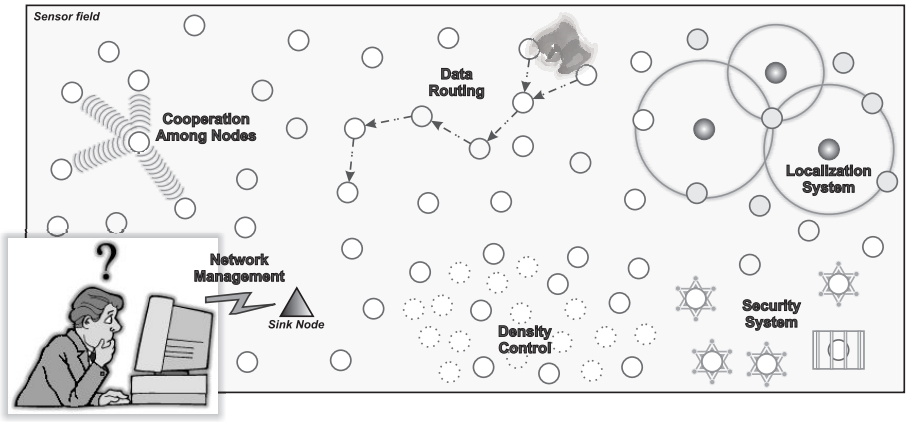


Figure 2.1: Several areas of the WSNs that work together in order to achieve one common goal: to monitor an area of interest.

In this chapter we will address the localization problem from the viewpoint of a WSN. In the next two sections, we briefly present an overview and definition of the localization systems for WSNs and their components. Section 2.2 shows the main methods used by localization systems to estimate distances and angles. In Section 2.3 we show the techniques that can be used by a node to compute its position, while in Section 2.4 we show how all of the estimated information about distances and positions can be manipulated in order to allow most or all of the nodes in a WSN to estimate their positions. Finally, in Section 2.5 we present our conclusions.

2.1.1 Problem Statement

A WSN can be composed of n nodes, with a communication range of r, and distributed in a two-dimensional squared sensor field $Q = [0, s] \times [0, s]$. For the sake of simplification, we consider symmetric communication links, i.e., for any two nodes u and v, u reaches v if and only if v reaches u and with the same signal strength. In this work we also consider *homogeneous WSNs*, i.e., networks in which all of the nodes have the same hardware specification (e.g., processor, memory, and energy capacity). The only exceptions are, as shown in this section, the beacon nodes that are equipped with a GPS receiver. Thus, we represent the network by the Euclidean graph $G = (V, E)$ with the following properties:

- $V = \{v_1, v_2, \ldots, v_n\}$ is the set of sensor nodes;
- $\langle i, j \rangle \in E$ iff v_i reaches v_j, i.e., the distance between v_i and v_j is less than r;
- $w(e) \leq r$ is the weight of edge $e = \langle i, j \rangle$, i.e., the distance between v_i and v_j.

Some terms can be used to designate the state of a node:

Definition 1 (Unknown Nodes – $\mathcal{U}$) *Also known as free or dumb nodes, this term refers to the nodes in the network that do not know their localization information. To allow these nodes to estimate their positions is the main goal of a localization system.*

Definition 2 (Settled Nodes – $\mathcal{S}$) *These nodes were initially unknown nodes that have managed to estimate their positions using the localization system. The number of settled nodes and the estimated position error of these nodes are the main parameters for determining the quality of a localization system.*

Definition 3 (Beacon Nodes – $\mathcal{B}$) *Also known as landmarks or anchors, these are the nodes that do not need a localization system in order to estimate their physical positions. Their localization is obtained by manual placement or by external means such as GPS. These nodes form the base of most localization systems for WSNs.*

The localization problem can then be defined as follows.

Definition 4 (Localization Problem) *Given a multihop network $G = (V, E)$ and a set of beacon nodes $\mathcal{B}$ and their positions (x_b, y_b), for all $b \in \mathcal{B}$, we want to find the position (x_u, y_u) of as many $u \in \mathcal{U}$ as possible, thus transforming these unknown nodes into settled nodes – $\mathcal{S}$.*

2.1.2 The Components of Localization Systems

Localization systems can be divided into three distinct components (see Figure 2.2):

1. *Distance/angle estimation:* this component is responsible for estimating information about the distances and/or angles between two nodes. This information will be used by the other components of the localization system.

2. *Position computation:* this component is responsible for computing a node's position based on available information concerning distances/angles and positions of reference nodes.

3. *Localization algorithm:* this is the main component of a localization system. It determines how the available information will be manipulated in order to allow most or all of the nodes in a WSN to estimate their positions.

Besides providing a didactic viewpoint, the importance of such a division into components comes, as we will see, from the need to recognize that the final performance of a localization system depends directly on each of these components. Also, each component has its own goal and methods of solution. They can thus be seen as subareas of the localization problem that need to be analyzed and studied separately. In the following sections, these components will be studied individually.

2.2 Distance/Angle Estimation

Distance/angle estimation consists in identifying the distance or angle between two nodes. Such estimates constitute an important component of localization systems, because they are used by both the position computation and localization algorithm components.

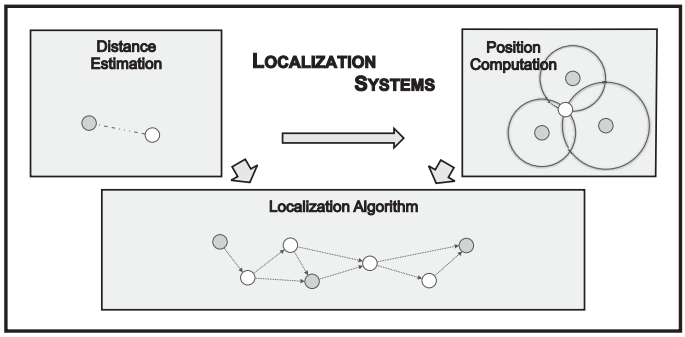


Figure 2.2: The division of localization systems into three distinct components: distance/angle estimation, position computation, and localization algorithm. The arrows indicate the information flow from one component to another.

Model	Description	Formula
Free Space (Friis, 1946)	Consider the ideal propagation condition without interferences or obstacles.	$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L}$
Two-Ray Ground (Rappaport, 1996)	Like the Free Space, but consider the possibility of signal reflexion in the ground.	$P_r(d) = \frac{P_t G_t G_r h_t^2 h_r^2}{d^4 L}$

Table 2.1: Two known radio propagation models. P_t and P_r are the transmitted and received signals power (strengths), G_t and G_r are the antenna gains of the transmitter and receiver, λ is the wavelength, d is the distance between the nodes, L is the system loss, and h_t and h_r are the heights of the transmit and receive antennas.

Different methods can be used to estimate such information. Some of these methods are very accurate but with higher costs (in terms of hardware, energy, and processor resources), while others are less accurate but already available on most sensor nodes.

In the following sections, some of the main methods used by localization systems to estimate distances/angles will be studied. These methods include RSSI, ToA/TDoA, AoA, and communication range.

2.2.1 Received Signal Strength Indicator – RSSI

RSSI can be used to estimate the distance between two nodes based on the strength of the signal received by another node. As depicted in Figure 2.3(a), a sender node sends a signal with a determined strength that fades as the signal propagates. The longer the distance to the receiver node, the weaker the signal strength when it arrives at that node.

The signal strength is commonly measured in dBm (Decibel in reference to one milliwatt) or in Watts. Theoretically, the signal strength is inversely proportional to the squared distance, and a known radio propagation model (Table 2.1) can be used to convert the signal strength into distance. However, in real-world environments, this indicator is highly influenced by noises, obstacles, and by the type of the antenna, all of which makes it hard to model mathematically the process. In these cases, it is common to make a system calibration (Whitehouse and Culler, 2002) in which the values of RSSI and distances are evaluated ahead of time in a controlled environment.

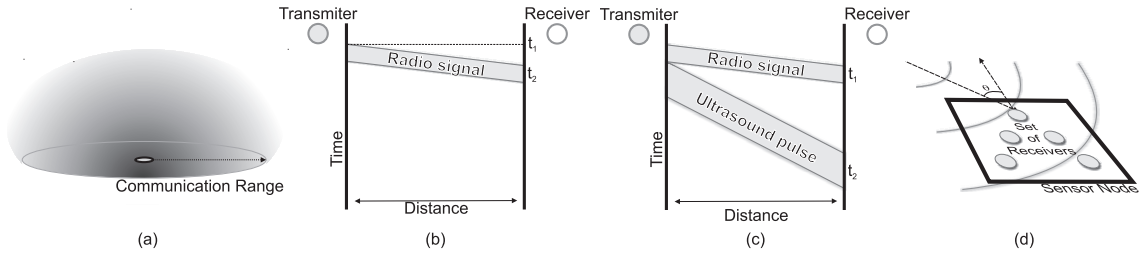


Figure 2.3: (a) Decrease in the signal strength; (b),(c) Methods of deriving the distance from the signal's arrival time; and (d) Angle of arrival of the signal.

This method, like the others, has both advantages and disadvantages. The main advantage is its low cost, because most receivers are capable of estimating the received signal strength. The disadvantage of this method is that it is very susceptible to noise and interference, which results in higher inaccuracies in the distance estimations. Some experiments (Savvides et al., 2001) show errors of 2 to 3 m in scenarios in which all nodes are placed in a plane field, 1.5 m from the ground, and with a communication range of 10 m.

Although RSSI shows plausible results in simulations and controlled experiments, its usage in real-world applications is still questionable (He et al., 2003). Still, considering its low cost, it is possible that a more sophisticated and precise use of the RSSI (e.g., with better transmitters) could become the most used technology of distance estimation from the cost/precision viewpoint (Bachrach and Taylor, 2005). However, this technology is not yet available.

2.2.2 Time [Difference] of Arrival – ToA/TDoA

A few different methods try to estimate the distance between two nodes using time based measures. The most simple and intuitive is Time of Arrival – ToA (Hofmann-Wellenho et al., 1997). In this case, the distance between two nodes is directly proportional to the time that the signal takes to propagate from one point to another. This way, if a signal is sent at time t_1 and reaches the receiver node at time t_2, the distance between the sender and receiver is $d = s_r(t_2 - t_1)$, where s_r is the propagation speed of the radio signal (speed of light), and t_1 and t_2 are the times at which the signal was sent and received, respectively (Figure 2.3(b)). This type of estimation requires precisely synchronized nodes, and the time at which the signal leaves the node must be in the packet that is sent.

The TDoA (Time Difference of Arrival) is based on (a) the difference between the times at which a single signal from a single node arrives at three or more nodes, or (b) the difference between the times at which multiple signals from a single node arrive at another node. The first case, which is more common in cellular networks, requires precisely synchronized receiver nodes (in this case, base stations). In the second case, which is more common and suitable for WSNs, the nodes must be equipped with extra hardware that is capable of sending two types of signals simultaneously. These signals must have different propagation speeds, like radio/ultrasound (Priyantha et al., 2001) or radio/acoustic (Whitehouse and Culler, 2002). Usually, the first signal is the packet itself, which propagates at the speed of

light ($\approx 300.000\,\mathrm{km/s}$), while the second signal is some kind of sound because of its slower propagation ($\approx 340\,\mathrm{m/s}$), which is six orders of magnitude slower than the first signal.

An example of TDoA that is suitable for WSNs is used by Savvides et al. (2001) and depicted in Figure 2.3(c), in which an ultrasound pulse is sent simultaneously with a radio signal. In this case, the nodes compute the difference in the arrival times of the two signals. The distance can now be computed using the following formula $d = (s_r - s_s) * (t_2 - t_1)$, where s_r and s_s are the propagation speed of the radio and ultrasound signals, and t_1 and t_2 are the arrival times of the radio and ultrasound signals, respectively.

The errors in the distance estimations obtained by TDoA are approximately centimeters. Experiments with ultrasound performed by Savvides et al. (2001) indicate errors of about 2 or 3 cm (smaller than the sensor node itself) in scenarios where nodes were separated by distances of 3 m. In Whitehouse (2002), acoustic sound was tested, and results showed errors of about 23 cm, with nodes at distances of 2 m.

Despite the lower errors, these systems have some disadvantages. The first is the need for extra hardware to send the second signal, which increases the cost of the node. The second disadvantage is the generally limited range of the second signal, which is normally between 3m and 10m with more powerful transmitters.

2.2.3 Angle/Direction of Arrival – AoA/DoA

The angle of arrival of the signal (Niculescu and Nath, 2003; Priyantha et al., 2001) can also be used by localization systems. This angle can be in relation to the node itself, to an electronic compass, or in relation to a second signal received by the node.

The estimation of the angle of arrival is usually done using directive antennas or an array of receivers – usually three or more – that are uniformly separated. In the last case, based on the time difference of arrival of the signal at each of the receivers, it becomes possible to estimate the angle of arrival of this signal (Figure 2.3(d)).

Experiments show that this method may have an inaccuracy of a few degrees (about 5^o in Priyantha et al. (2001)). The need for extra hardware is one of the disadvantages of this technique since it increases the cost and size of the sensor nodes. Also, this technique requires a minimum distance between the set of receivers in a sensor node, which also increases the final size of the sensor nodes in the WSN.

2.2.4 Communication Range

In some cases, the only information available to estimate a distance is the communication range of the sensor nodes. If a node receives a data packet from another node, then the distance between these nodes is between zero and the maximum communication range.

Usually, techniques that use this method of distance estimation do not require an accurate distance, but only an interval. To get only one distance (and not an interval), we can choose one point from the interval such as the middle point for instance. In this last case, the maximum error of this estimation will be one half the communication range.

Method	Precision	Maximum Distance	Extra Hardware	Challenges
RSSI	meters (2-4 m)	comm. range	none	variation of the Rssi, interferences
ToA	centimeters (2-3 cm)	comm. range	none	nodes synchronization
TDoA	centimeters (2-3 cm)	few meters (2,3-10 m)	ultrasound transmitter	maximum distance of work
AoA	some degrees (5^o)	comm. range	set of receivers	work on small sensor nodes
Comm. Range	half the comm. range	comm. range	none	-

Table 2.2: Comparison of the methods used to estimate distances/angles between two nodes. The chosen method depends on the application, scenario and available resources.

This method of distance estimation has the advantage of being the most simple and with the lowest cost. No extra hardware is required and neither extra computation is needed to estimate a distance. On the other hand, an error of half the communication range for each distance estimation is not viable for most localization systems. Consider, for instance, a communication range of 100 m. In this case, the error of this method can be about 50 m for each distance estimation.

2.2.5 Comments About the Distance/Angle Estimation

The choice of which method to use to estimate the distance between nodes in a localization system is an important factor that influences the final performance of the system. As will be shown in the next section, to estimate a position, a node generally uses at least three distance estimations, each of which has an associated error. On the other hand, if only the accuracy of these methods was important, we could just use TDoA since it has fewer errors. However, other factors including the size and cost (in terms of hardware, processor, and energy) of the nodes must also be taken into consideration. Thus, the method chosen for estimating distances will depend on the application requirements as well as available resources. Table 2.2 compares each one of the methods described in this section.

2.3 Position Computation

When a node has enough information about distances and/or angles and positions, it can compute its own position using one of the methods that will be studied in this section.

Several methods can be used to compute the position of a node. Such methods include trilateration, multilateration, triangulation, probabilistic approaches, bounding box, and the central position. The choice of which method to use also impacts the final performance of the localization system. Such a choice depends on the information available as well as the processor's limitations.

2.3.1 Trilateration and Multilateration

Trilateration is the most basic and intuitive method. This method computes a node's position via the intersection of three circles, as depicted in Figure 2.4(a). To estimate its position using trilateration, a node needs to know the positions of three reference nodes and its distance from each of these nodes. Distances can be estimated using one of the methods explained in the previous section.

The circles formed by the position of and distance to each of the references can be represented by the formula $(\hat{x} - x_i)^2 + (\hat{y} - y_i)^2 = d_i^2$, where $(\hat{x},\hat{y})$ is the position we want to compute, (x_i,y_i) is the position of the ith reference, and d_i is the distance of the ith reference node to the unknown node. In this case, we have three equations with two unknowns that can be solved, theoretically, into one solution.

In real-world applications, the distance estimation inaccuracies as well as the inaccurate position information of reference nodes make it difficult to compute a position. As depicted in Figure 2.4(b), the circles do not intersect at only one point, resulting in an infinite set of possible solutions.

Furthermore, when a larger number of reference points is available, we can use multilateration to compute the node's position. In this case, an overdetermined system of equations must be solved. Figure 2.4(c) depicts this case. Usually, overdetermined systems do not have a unique solution. When considering n reference points, as well as the error of the distance estimations, which makes $d_i = \hat{d}_i - \epsilon$, the system of equations becomes $(\hat{x}-x_i)^2+(\hat{y}-y_i)^2 = \hat{d}_i^2-\epsilon$, where ϵ is usually considered to be a Gaussian random variable with zero mean and finite variance. This system can be linearized, by subtracting the last equation, into $Ax \approx b$. This linear system can be solved easily using standard methods like the least squares approach.

The number of floating point operations needed to compute a position depends on the method used to solve the system of equations. In the case of the least square method, $(m + n/3)n^2$ floating point operations (where m is the number of unknowns and n is the number of equations) are required to estimate a position.

2.3.2 Bounding Box

The bounding box method, proposed by Simic and Sastry (2002), uses squares – instead of circles as in trilateration – to bound the possible positions of a node. An example of this method is depicted in Figure 2.4(d).

For each reference node i, a bounding box is defined as a square with its center at the position of this node (x_i,y_i), with sides of size $2d_i$ (where d is the estimated distance) and with coordinates $(x_i - d_i, y_i - d_i)$ and $(x_i + d_i, y_i + d_i)$. The intersection of all bounding boxes can be computed easily without any need for floating point operations, by taking the maximum of the low coordinates and the minimum of the high coordinates of all bounding boxes. This is the shaded rectangle in Figure 2.4(d). The final position of the unknown node is then computed as the center of the intersection of all bounding boxes.

Despite the final error of this method, which is greater than trilateration, computing

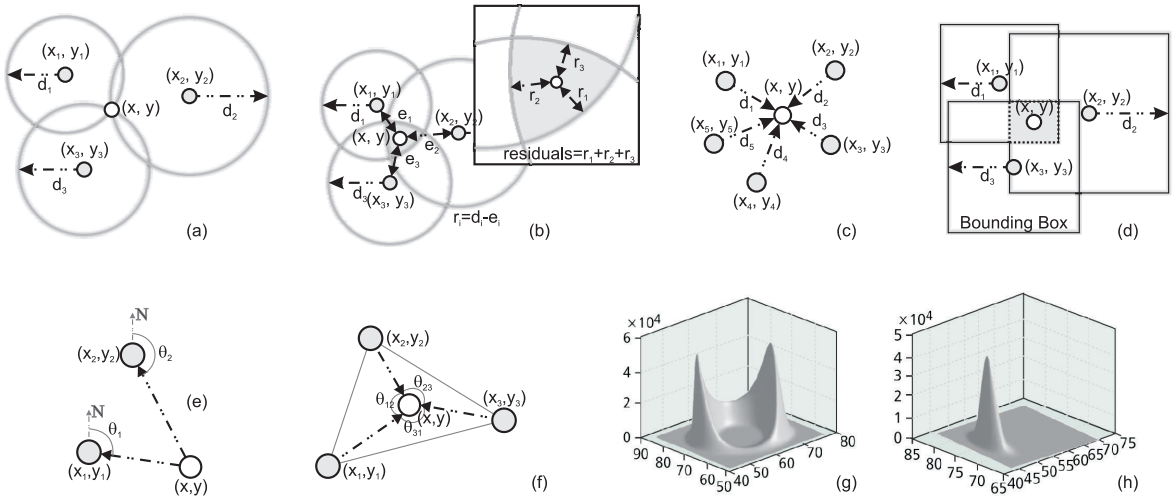


Figure 2.4: (a) Theoretical model of trilateration; (b) A more realistic model of the trilateration; (c) Multilateration; (d) Bounding Box; (e),(f) Triangulation; and (g),(h) Probabilistic approach (from Ramadurai and Sichitiu (2003)).

the intersection of squares uses fewer processor resources than computing the intersection of circles.

2.3.3 Triangulation

In triangulation (Niculescu and Nath, 2003; Priyantha et al., 2001), information about angles is used instead of distances. The position computation can be made remotely (Figure 2.4(e)) or by the node itself, the latter of which is more common in WSNs. In this last case, depicted in Figure 2.4(f), at least three reference nodes are required. The unknown node estimates its angle to each one of the three reference nodes and, based on these angles and on the positions of the reference nodes (which form a triangle), computes its own position using simple trigonometric relationships. This technique is similar to trilateration. In fact, based on the angles of arrival, it is possible to derive the distances to reference nodes (Niculescu and Nath, 2003).

2.3.4 Probabilistic Approaches

The uncertainty in the distance estimations has motivated the appearance of probabilistic approaches for computing a node's position. An example of a probabilistic approach is proposed by Ramadurai and Sichitiu (2003). In the cited work, the errors in distance estimations are modeled as Gaussian random variables. When an unknown node receives a packet from a reference node, it can be in any place around the reference node with equal probabilities. When another packet is received from another reference node, the unknown node computes its position again as depicted in Figure 2.4(g). In this last case, the node can be located in two different places with equal probability since with only two reference points it is not yet

possible to determine the exact location of the node. When new position information is received from other nodes, it becomes possible to identify the probable location of the unknown node, as depicted in Figure 2.4(h).

When an application requires a single position, the point with greater probability can be computed. The main drawbacks of this approach are the high computational cost and the space required to store the information. In Sichitiu and Ramadurai (2004), it is shown that considering the sample size as a grid of $d \times d$, the complexity of this method would be $O(3d^2)$. One possible application of this method consists in sending the gathered information to a central and more powerful node in order to compute the positions.

2.3.5 Central Position in Relation to the Reference Nodes

By the assumption that the most probable position of a node is the central point among all the reference nodes, we can compute the position of an unknown node without the need of estimating distances or angles, but only by using the communication range, as explained in Section 2.2.4.

In this case, the position of a node is computed by using the following equation (He et al., 2003):

$$(\hat{x}, \hat{y}) = \left(\frac{\sum_{i=1}^{n} x_i}{n}, \frac{\sum_{i=1}^{n} y_i}{n} \right) \tag{2.1}$$

where n is the number of reference nodes.

This is the simplest method in terms of computational resources and required information. Only $2n + 2$ float point operations (where n is the number of reference nodes) is required to compute a position. On the other hand, the obtained solutions are not accurate, mainly when the number of reference node is low.

2.3.6 Comments About the Position Computation

A number of other methods exist that aim to compute the position of a node. Location Fingerprinting is a method in which the signal characteristics obtained from a set of locations are catalogued and the position computation of a node consists in comparing its current signal characteristics with the ones that were catalogued previously. This technique is used by Bahl and Padmanabhan (2000) and other indoor localization systems, but the need to generate a signal signature database makes this technique unfeasible for most scenarios in WSNs. He et al., in the APIT algorithm (He et al., 2003), use triangles formed by three beacon nodes, and a node decides whether it is inside or outside these triangles by comparing its signal strength measurements with the measurements of its neighbors. The position of the node is computed by finding the centroid of the intersection of the beacon triangles that the node is within.

Other work concentrates all information about the distances between the nodes into a central node and use mathematical optimization techniques to compute the positions of the nodes. As an example, the work of Doherty et al. (2001) formulates the localization problem as a convex optimization problem based only on connectivity-induced constraints and uses a

Method	# Refs	Dist?	Angle?	Comp.	Challenges
Trilateration	3	yes	no	$O(1)$	susceptible to inacc. distances
Multilateration	$n \geq 3$	yes	no	$O(n^3)$	computational complexity
Triangulation	3	no	yes	$O(1)$	require extra hardware
Probabilistic	$n \geq 3$	yes	no	$O(3d^2)$ (d=*grid*)	comp. and space complexity
Bounding Box	$n \geq 2$	yes	no	$O(n)$	final position error
Central Position	$n \geq 1$	no	no	$O(n)$	final position error

Table 2.3: Comparison of the methods used to compute positions. Again, there is no ideal solution that works in all scenarios. The choice of what method to use will depend on the gathered information and on the available processor resources.

semi-definite program (SDP) to solve the problem. In addition, Shang and Ruml (2004) use a multidimensional scale (MDS) technique.

The choice of what method to use can also impact the final performance of the localization system. In the next section, some localization algorithms will be studied. Depending on the used localization algorithm, this error in the position computation can harm in greater or minor degree the localization system as a whole. In some algorithms, for example, the newly computed positions are used to assist the other unknown nodes to compute their positions. In this case, a small error on the position computation can result in a localization system with high errors.

The information of positions and distances gathered by a node and the available processor resources also restrict the choice of what method to used. Table 2.3 summarizes and compares the main characteristics of the position computation methods explained in this section.

2.4 Localization Algorithm

The localization algorithm is the main component of a localization system. This component determines how the information concerning distances and positions will be manipulated in order to allow most or all of the nodes in a WSN to estimate their positions.

Localization algorithms can be classified into a few categories: distributed (Albowicz et al., 2001; Hofmann-Wellenho et al., 1997; Niculescu and Nath, 2001) or centralized position computation (Doherty et al., 2001); with (Albowicz et al., 2001; Niculescu and Nath, 2001) or without an infrastructure (Hofmann-Wellenho et al., 1997; Priyantha et al., 2000, 2001); relative (Bulusu et al., 2000; Capkun et al., 2002) or absolute positioning (Albowicz et al., 2001; Hofmann-Wellenho et al., 1997); designed for indoor (Priyantha et al., 2000, 2001) or outdoor scenarios (Hofmann-Wellenho et al., 1997); and one hop (Hofmann-Wellenho et al., 1997; Priyantha et al., 2000) or multihop algorithms (Albowicz et al., 2001; Niculescu and Nath, 2001).

In the next sections, some proposed localization algorithms will be studied to show how this component differentiates from the other components. These algorithms are the Ad hoc

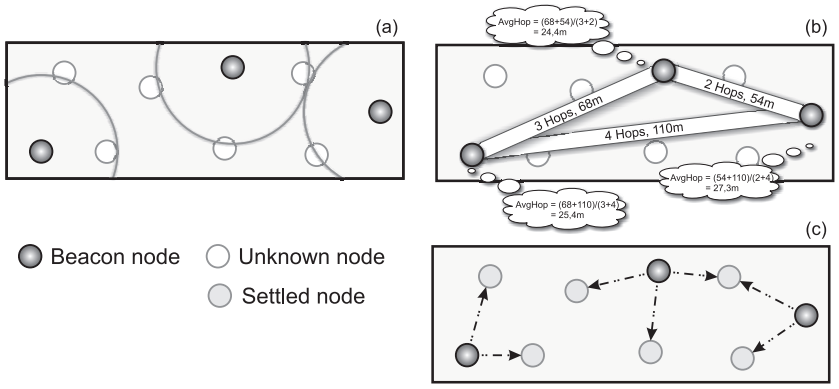


Figure 2.5: Example and phases of the APS – Dv-Hop: (a) initially, the beacon nodes broadcast their positions and (b) compute the average size of hop; (c) this last value is sent to the network and the nodes receiving it compute their positions.

Positioning System, Recursive Position Estimation, Directed Position Estimation, and Localization with a Mobile Beacon.

2.4.1 Ad Hoc Positioning System – APS

In APS (Niculescu and Nath, 2001), a reduced number of beacon nodes (e.g., three or more) is deployed with the unknown nodes. Then, each node estimates its distance to the beacon nodes in a multihop way. Once these distances are estimated, the nodes can compute their positions using trilateration. Three methods of hop by hop distance propagation are proposed: Dv-Hop, Dv-Distance, and Euclidean.

In Dv-Hop APS, the beacon nodes start the propagation of their position information (Figure 2.5(a)). Working as an extension of the distance vector algorithm, all nodes receive the position information of all beacon nodes as well as the number of hops to these beacons. When a beacon node receives the position information of the other beacon nodes, it has enough information to compute the average size of one hop based on its own position, on the position of the other beacon nodes, and also on the number of hops between them (Figure 2.5(b)). This last value is then flooded in a controlled way throughout the network as a correction factor. When an unknown node receives a correction, it is able to convert its distance to the beacon nodes from the number of hops to meters (Figure 2.5(c)). The complete Dv-Hop algorithm is shown on Algorithm 1. The complexity of exchanging message in this algorithm is driven by the total number of beacon and normal nodes, which is $O(n * m)$, where n is the number of nodes and m is the number of beacon nodes.

An advantage of APS is that its localization algorithm requires only a small number of beacon nodes in order to work. However, the way that distances are propagated, especially in Dv-Hop and Dv-Distance, as well as the way these distances are converted from hops to meters in Dv-Hop, result in erroneous position computation, which increases the final localization error of the system.

Algorithm 1 Ad Hoc Positioning System Dv-Hop localization algorithm

▷ **Variables:**
1: $positions_i = \emptyset$; {Set of beacon positions.}
2: $correction_i = -1$; {Correction computed/received by this node.}

▷ **Input:**
3: $msg_i = nil$.

Action:
4: **if** $n_i \in \mathcal{B}$ **then** {If this node is a beacon node.}
5: $(x_i, y_i) := getGpsPosition()$;
6: Send $beaconPos(i, x_i, y_i, 0)$ to all $n_j \in Neig_i$.
7: **end if**

▷ **Input:**
8: $msg_i = beaconPos(k, x_k, y_k, h_k)$.

Action:
9: **if** $k \notin \mathcal{R}_i$ **then** {If this node has not already received this packet.}
10: $\mathcal{R}_i := \mathcal{R}_i \cup \{k\}$;
11: $positions_i := positions_i \cup \{(x_k, y_k, h_k)\}$;
12: **if** $n_i \in \mathcal{B}$ **then** {If this is a beacon node, wait for more position packets}
13: [Re]Start $waitTimer$.
14: **end if**
15: Send $beaconPos(k, x_k, y_k, h_k + 1)$ to all $n_j \in Neig_i$.
16: **end if**

▷ **Input:**
17: $waitTimer$ timeout. {Executed only by the beacon nodes.}

Action:
18: $correction_i = \frac{\sum \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{\sum h_j}$ for all $(x_j, y_j, h_j) \in positions_i$;
19: Send $correction_i$ to all $n_j \in Neig_i$.

▷ **Input:**
20: $msg_i = correction_k$.

Action:
21: **if** $n_i \in \mathcal{U}$ **then** {If this node is an unknown node.}
22: $correction_i := correction_k$;
23: $h_j := h_j * correction_i$ for all $(x_j, y_j, h_j) \in positions_i$;
24: $(x_i, y_i) := positionComputation(positions_i)$; {Becomes a settled node.}
25: Send $correction_i$ to all $n_j \in Neig_i$.
26: **end if**

2.4.2 Recursive Position Estimation – RPE

In the RPE algorithm (Albowicz et al., 2001), nodes estimate their positions based on a set of initial beacon nodes (e.g., 5% of the nodes) using only local information. Localization information increases iteratively as newly settled nodes become reference nodes.

The RPE algorithm can be divided into four phases, as depicted in Figure 2.6. In the first phase, a node determines its reference nodes. In the second phase, the node estimates its distance to these reference nodes using, for example, RSSI. In the third phase, the node computes its position using trilateration (becoming a settled node). In the final phase, the node becomes a reference node by broadcasting its newly estimated position to its neighbors. When a node becomes a reference, it can assist other nodes in computing their positions as

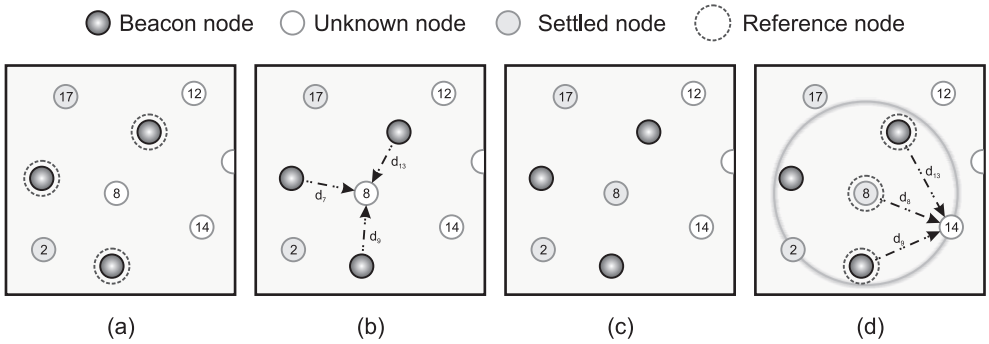


Figure 2.6: Example and phases of the RPE: (a) initially, the node chooses its reference nodes; then (b) it estimates its distance to each of the reference nodes; (c) computes its position using trilateration; and (d) broadcasts its newly estimated position to assist the other nodes.

well.

An advantage of this algorithm is that the number of reference nodes increases quickly, in such a way that the majority of the nodes can compute its position. However, this technique has the disadvantage of propagating the localization error. This means that the inaccurate position estimation of one node can be used by other nodes to estimate their positions, increasing this inaccuracy. Furthermore, a node must have at least three reference neighbors in order to compute its position. The complete RPE algorithm is shown in Algorithm 2 and its communication complexity is O(n), i.e., the same as that of flooding algorithm in which each node of the WSN sends only one packet.

2.4.3 Directed Position Estimation – DPE

By adding some restrictions to a recursive localization system as in RPE (see Section 2.4.2), we can make the localization recursion start from a single point (recursion origin) and follow a determined and known direction (Figure 2.7(a)). Once this behavior is guaranteed, it is possible to estimate a node's position using only two reference neighbors.

When a node has the position information of only two reference neighbors, a pair of possible points results from the system: one is the right position of the unknown node and the other is a wrong estimate (Figure 2.7(b)). Once the direction of the localization recursion is kept, it is easy to choose between the two possible solutions: the most distant point from the recursion origin is the right position of the unknown node. This is the base of the Directed Position Estimation – DPE (Oliveira et al., 2005a,b) localization algorithm.

The DPE algorithm is divided into four phases (see Figure 2.8). In the first phase, the recursion of such a system is started from a single point by the beacon structure (Figure 2.8(a)). In the second phase, a node determines its (two) reference nodes and estimates its distances to these nodes (Figure 2.8(b)). In the third phase, the node computes its position (Figure 2.8(c)) and then becomes a reference by sending this information to its neighbors (fourth phase - Figure 2.8(d)). This way, the recursion of such a system goes from the center to the edge of the WSN. The complete DPE algorithm is shown in Algorithm 3 and its communication com-

Algorithm 2 Recursive Position Estimation localization algorithm

▷ **Variables:**

1: $positions_i = \emptyset$ {Set of received positions.}
2: $references_i = \emptyset$ {Set of reference nodes.}

▷ **Input:**

3: $msg_i = nil$.

Action:

4: **if** $n_i \in \mathcal{B}$ **then** {If this node is a beacon node.}
5: $(x_i, y_i) := getGpsPosition()$;
6: Send $position(x_i, y_i, 0)$ to all $n_j \in Neig_i$.
7: **end if**

▷ **Input:**

8: $msg_i = position(x_k, y_k, r_k)$ such that $dist_k = distanceEstimation(msg_i)$.

Action:

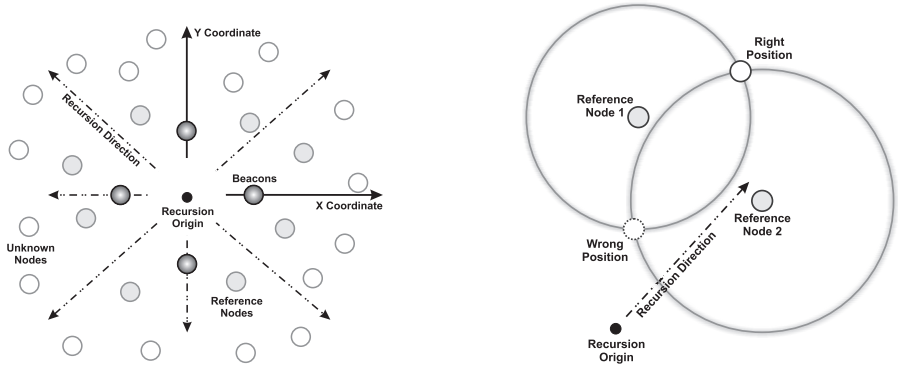
9: **if** $n_i \in \mathcal{U}$ **then** {If this node is an unknown node.}
10: $positions_i := positions_i \cup \{(x_k, y_k, r_k, dist_k)\}$;
11: [Re]Start $waitTimer$.
12: **end if**

▷ **Input:**

13: $waitTimer$ timeout.

Action:

14: **if** $size(positions_i) >= 3$ **then** {If there is enough positions.}
15: $references_i := chooseThreeBestPositions(positions_i)$
16: $(x_i, y_i, r_i) := positionComputation(references_i)$; {Becomes a settled node.}
17: Send $position(x_i, y_i, r_i)$ to all $n_j \in Neig_i$. {Becomes a reference node.}
18: **end if**



(a) (b)

Figure 2.7: (a) The DPE algorithm performing a directed localization recursion; and (b) a position estimate using only two reference neighbors. A pair of possible solutions results from the system. The right position of the node is the most distant point from the recursion origin.

plexity is the same as that of flooding.

This approach leads to a localization system that can work in a low density sensor network. In addition, the controlled way in which the recursion is made also leads to a system with fewer and more predictable errors. However, like the RPE, this technique has the disadvantage of propagating the localization error.

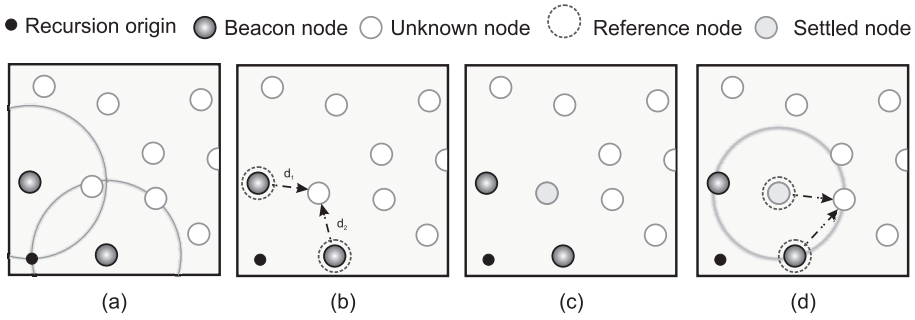


Figure 2.8: Example and phases of the DPE algorithm: (a) first, the beacon nodes start the recursion, (b) then a node determines its (two) reference nodes, (c) estimates its position, and (d) becomes a reference itself by broadcasting this information.

Algorithm 3 Directed Position Estimation localization algorithm

▷ **Variables:**
1: $positions_i = \emptyset$ {Set of received positions.}
2: $references_i = \emptyset$ {Set of reference nodes.}

▷ **Input:**
3: $msg_i = nil$.

Action:
4: **if** $n_i \in \mathcal{B}$ **then** {If this node is a beacon node.}
5: $(x_i, y_i) := getGpsPosition()$;
6: Send $position(x_i, y_i)$ to all $n_j \in Neig_i$.
7: **end if**

▷ **Input:**
8: $msg_i = position(x_k, y_k)$ such that $dist_k = distanceEstimation(msg_i)$.

Action:
9: **if** $n_i \in \mathcal{U}$ **then** {If this node is an unknown node.}
10: $positions_i := positions_i \cup \{(x_k, y_k, dist_k)\}$;
11: [Re]Start $waitTimer$.
12: **end if**

▷ **Input:**
13: $waitTimer$ timeout.

Action:
14: **if** $size(positions_i) >= 2$ **then** {If there is enough positions.}
15: $references_i := chooseTwoBestPositions(positions_i)$
16: $(x_i, y_i) := mostDistantFromOrigin(intersectCircles(references_i))$; {Becomes a settled node.}
17: Send $position(x_i, y_i)$ to all $n_j \in Neig_i$. {Becomes a reference node.}
18: **end if**

2.4.4 Localization with a Mobile Beacon – MBL

Some recent work (Sichitiu and Ramadurai, 2004) have proposed the use of mobile beacons to assist the nodes in the WSN with estimating their positions. A mobile beacon is a node that is aware of its position (e.g., equipped with a GPS receiver) and that has the ability to move around the sensor field. This beacon can be a human operator, an unmanned vehicle, an aircraft, or a robot.

The system operation in Sichitiu and Ramadurai (2004) is quite simple. Once the nodes

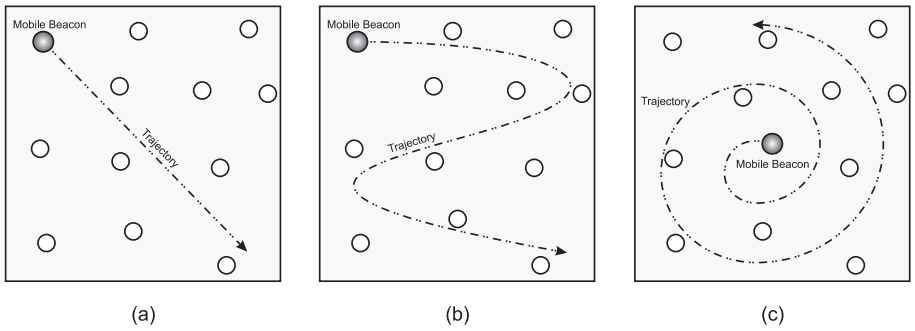


Figure 2.9: Operation and possible trajectories for the Localization with a Mobile Beacon: (a) the mobile beacon moving along the sensor field in a straight line; (b) a less rectilinear trajectory; and (c) a trajectory in spiral form.

are deployed, the mobile beacon travels through the sensor field broadcasting messages that contain its current coordinates. When a free node receives more than three messages from the mobile beacon, it computes its position using a probabilistic approach, based on the received coordinates and on the RSSI distance estimations. Figure 2.9 illustrates this scenario and three possible trajectories for the mobile beacon. Algorithm 4 details the functioning of the system. The communication cost for the WSN is null, since the nodes (except for the mobile beacon) do not need to send any packets.

An advantage of this algorithm is that position estimations are computed based on the same node (mobile beacon), thus keeping the mean localization error low and preventing its propagation. In addition, this algorithm avoids the use of nodes equipped with GPS, except for the mobile beacon. On the other hand, in this technique, a sensor node can estimate its position only when the mobile beacon passes near this node, which may take a long time depending on such factors as the size of the sensor field, the beacon's mobility capacity, and the node's trajectory. Still, the mobile beacon may never pass near some nodes, either because of the trajectory or due to a problem with the mobile beacon.

2.4.5 Comments About Localization Algorithms

Several localization algorithms focus on different aspects including errors, number of beacons, number of settled nodes, and GPS usage, among other things. Usually, these algorithms (Bulusu et al., 2000; Capkun et al., 2002) try to reduce or completely remove the need for GPS receivers on beacon nodes. Other algorithms take advantage of certain network features, such as beacons with high-powered transmitters (He et al., 2003), a directed localization recursion (Oliveira et al., 2005a) or a beacon infrastructure (Priyantha et al., 2000). The choice of which algorithm to use depends on the resources available, the scenario, the requirements of the application, and also on the mean localization error acceptable to the nodes. Table 2.4 compares the main characteristics the studied algorithms.

Algorithm 4 Mobile Beacon Localization algorithm

▷ **Variables:**
1: $positions_i = \emptyset$ {Set of position information.}

▷ **Input:**
2: $msg_i = nil$.

Action:
3: **if** $n_i \in \mathcal{U}$ **then** {If this node is a beacon node.}
4: StartWalking();
5: Start $posTimer$.
6: **end if**

▷ **Input:**
7: $posTimer$ timeout.

Action:
8: $(x_i, y_i) := getGpsPosition()$;
9: Send $position(x_i, y_i)$ to all $n_j \in Neig_i$.
10: Restart $posTimer$.

▷ **Input:**
11: $msg_i = position(x_k, y_k)$ such that $dist_k = distanceEstimation(msg_i)$.

Action:
12: $positions_i := positions_i \cup \{(x_k, y_k, dist_k)\}$;
13: **if** $size(positions_i) >= 3$ **then** {If there is enough references.}
14: $(x_i, y_i) := positionComputation(positions_i)$;
15: **end if**

Algorithm	Number of beacons	Position Computation	Infra-structured?	Positioning	Scenarios	Multi-hop?
APS	≥3	distrib.	no	absolute	outdoors	yes
RPE	5% nodes	distrib.	no	absolute	outdoors	yes
DPE	4	distrib.	no	relative	outdoors	yes
MBL	1 mobile	distrib.	no	absolute	outdoors	no

Table 2.4: Localization algorithms comparison. Some characteristics of the localization algorithms identify the possible scenarios that they can be applicable. The choice of what algorithm to use depends on the application requirements and on the available resources.

2.5 Summary

This chapter has addressed the localization problem from the viewpoint of a WSN. We divided localization systems into three components: Distance/Angle Estimation, Position Computation, and Localization Algorithm.

The importance of such a division into components results from the need to recognize that the final performance of localization systems depends directly on each of these components. For instance, a localization system should achieve better results if the TDoA method is used instead of RSSI to estimate distances. The same principle applies to the other components. These components can be seen as subareas of the localization problem that need to be studied separately.

A general rule in WSNs is that there is no single, perfect solution that is suitable for

every scenario. The same rule applies to the localization problem. This chapter has shown a number of proposed localization systems, each having an emphasis on a specific scenario and/or application. The necessity of having different solutions for different applications as well as the high number of possible applications of WSNs, has greatly motivated the study and proposals of new solutions to the localization problem.

Chapter 3

Synchronization Systems for Wireless Sensor Networks

3.1 Introduction

In the synchronization problem (Sundararaman et al., 2005; Sivrikaya and Yener, 2004; Romer et al., 2005; Romer, 2003, 2001; Elson, 2003; Elson et al., 2002; van Greunen and Rabaey, 2003; Sichitiu and Veerarittiphan, 2003; Shin et al., 2006; Sekhar et al., 2005), the nodes' local clocks must be synchronized based on a reference node or in Coordinated Universal Time (UTC). Synchronized clocks are very important for most WSN applications since they are required for a number of protocols, algorithms, and applications such as data fusion techniques (Nakamura et al., 2005a,b, 2007), object (or target) tracking algorithms (Kumar et al., 2000), energy maps construction (Mini et al., 2004; Machado et al., 2005), density control algorithms (Xu et al., 2001), and events ordering.

Two well-known solutions for this problem are the Network Time Protocol (NTP) (Mills, 2006) and the use of Global Positioning System (GPS) receivers (Hofmann-Wellenho et al., 1997; Kaplan, 1996). The first solution is not suitable for sensor networks, since it is not energy-efficient and does not consider the dynamics and unpredictability of these networks (Sivrikaya and Yener, 2004). Adding a GPS receiver to each node, as mentioned before, is not feasible in WSNs because nodes will be bigger and more expensive. Moreover, a GPS receiver results in additional energy consumption, and it cannot be used when there is no line-of-sight to the satellites (e.g., indoor scenarios, underwater, severe climatic conditions, Mars exploration).

In this chapter, we will address the synchronization problem in WSN. In the next two sections, we briefly present an overview and definition of the synchronization systems for WSNs and their components. Section 3.2 shows the main methods used by synchronization systems to estimate the time delay of a packet leaving a sender and arriving at a neighboring receiver node. This estimated delay is then used to compute the time of the receiver based on the time of the sender node, as shown in Section 3.3. In Section 3.4, we show how all the estimated information of delays and time can be manipulated to allow most or all of the nodes

in a WSN to synchronize their clocks. Finally, in Section 3.5, we present our conclusions.

3.1.1 Problem Statement

We extend the network model introduced in Section 2.1.1 to include time information. First, the actual time of the network, in which the nodes must be synchronized (e.g., UTC), is represented simply by t. The hardware clock of node i is defined as $t_i(t)$, since it is a monotonically non-decreasing function of t. Because no hardware clock is perfect, $t_i(t)$ has two components: $t_i(t) = d_i t + o_i$, where o_i is the offset, i.e., the difference between t and t_i at that instant, and d_i is the drift, i.e., how the local clock gradually deviates from t due to conditions like temperature or battery voltage.

Some terms can be used to designate the state of a node:

Definition 5 (Unsynchronized Nodes – $\mathcal{N}$) *A node whose clock is not synchronized with the reference. Synchronizing these nodes is the main goal of a synchronization system.*

Definition 6 (Synchronized Nodes – $\mathcal{C}$) *This node was initially unsynchronized, but has managed to synchronize its local clock by using a synchronization system. The number of synchronized nodes and the estimated synchronization error of these nodes are the main parameters for assessing the quality of a synchronization system. According to our model, a synchronized node would become instantly unsynchronized due to the clock's hardware inaccuracy. For the sake of simplification, we still consider this node as belonging to the group of synchronized nodes but with a corresponding synchronization error.*

Definition 7 (Root Nodes – $\mathcal{B}$) *These are the nodes that always have synchronized clocks (e.g., by using GPS). These nodes, also called master nodes, form the base of most synchronization systems for WSNs.*

The synchronization problem can be stated as follows.

Definition 8 (Synchronization Problem) *Given a multihop network, represented by a graph $G = (V, E)$, one or a set of root (or master) nodes $\mathcal{B}$, and their synchronized clocks, i.e., $t_b = t$ for all $b \in \mathcal{B}$, we want to find the offset and drift (o_i, d_i) of as many unsynchronized nodes $i \in \mathcal{N}$ as possible.*

3.1.2 The Components of Synchronization Systems

Synchronization systems can be divided into three distinct components (see Figure 3.1):

1. *Delay estimation:* this component is responsible for estimating the time delay in which a packet leaves a sender node and arrives at a destination node. This information will be used by the other components of the synchronization systems.

2. *Time computation:* this component is responsible for computing a node's correct time based on available information concerning estimated delays and timestamps received by

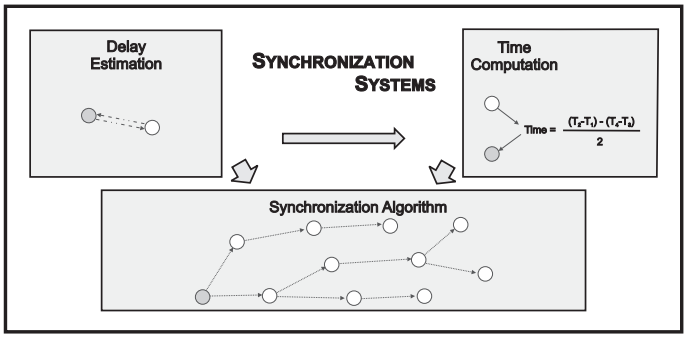


Figure 3.1: The division of the synchronization systems into three distinct components: delay estimation, time computation, and synchronization algorithm. The arrows indicate the information flow from one component to another.

synchronized nodes. The main techniques in this component are used to compute the clocks' offsets and drifts.

3. *Synchronization algorithm:* this is the main component of a synchronization system. It determines how the available information will be used to allow most or all of the nodes to synchronize their clocks.

Like the component divisions of a localization system, in addition to providing a didactic viewpoint, the importance of such a division of synchronization systems into components is that the final performance of a synchronization system depends directly on each of these components. Also, each component has its own goal and solutions. Thus, they can be seen as subareas of the synchronization problem that need to be analyzed and studied separately. In the following sections, these components will be analyzed individually.

3.2 Delay Estimation

Delay estimation consists in identifying the trip time delay that a packet takes to leave a sender node and arrive at the destination node. Such an estimation is an important component of the synchronization systems since it is used by the time computation component to compute the clocks' offsets and drifts.

Different methods can be used to estimate this information. Some of these methods are very accurate but have higher costs (e.g., higher communication costs), while others are less accurate but able to estimate the delay based on a single transmitted packet.

In the following sections, we will explain the most known delay estimation techniques in WSNs: the Round Trip Time (RTT) and the Delay Measurement.

3.2.1 Round Trip Time - RTT

In the Round Trip Time technique (RTT), the time in which a packet leaves a sender node, arrives at a receiver node, and is then echoed back to the sender is computed. Based on

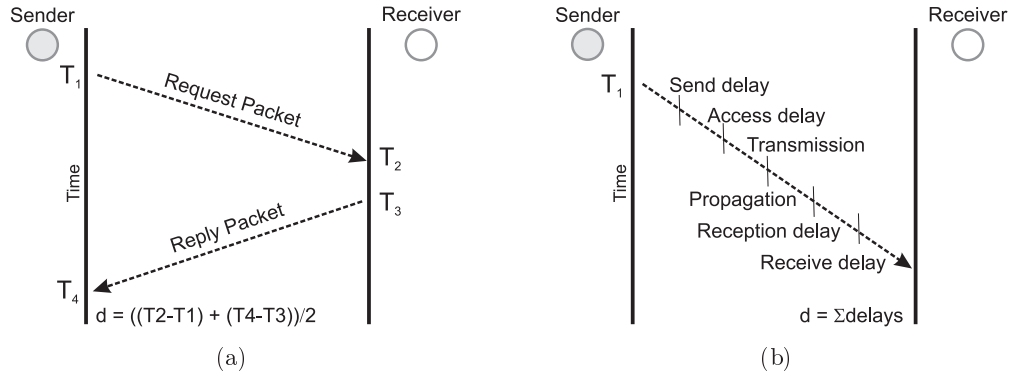


Figure 3.2: Delay estimation techniques: (a) the Round Trip Time (RTT) and (b) the Delay Measurement.

information from both local and remote clocks, the sender node can compute the delay of the packets.

As depicted in Figure 3.2(a), the sender node first sends a *request packet* to a receiver node. The request packet contains the local (unsynchronized) timestamp of the sender node (T_1). When the receiver node receives a request packet, it sends a *reply packet* to the sender node. This packet contains the original T_1 timestamp and the local timestamps of the receiver node when the request packet was received (T_2) and when the reply packet was sent (T_3). After receiving the reply packet, the sender node saves the time of arrival of this packet using its own local time (T_4). After that, the sender node is able to estimate the delay of a packet using the following equation:

$$delay = \frac{(T_2 - T_1) + (T_4 - T_3)}{2} \tag{3.1}$$

Basically, this equation computes the delay of a packet to be sent and replied, removes the processing time of the receiver node (i.e., the time needed to process the received request packet plus the time taken to create and send the reply packet), and divides the number by two, in order to get the average trip delay of a single packet. This technique can be used to estimate the trip delay of a packet over a single hop communication as well as over a multihop communication in which the request and reply packets travel through multiple nodes between the sender and the receiver. Also, several request packets can be sent to compute the average delay in order to get a more precise information.

This technique has been used by a number of WSN synchronization protocols (Ganeriwal et al., 2003; Mills, 2006).

3.2.2 Delay Measurement

In the delay measurement technique, all delays in the packet transfer time between the sender and the receiver are estimated and the final packet delay is the sum of all individual delays. The advantage of this technique is that the delay can be computed based only on a single packet, instead of requiring two packets as in the RTT technique.

Figure 3.3: The sources of delay in a packet exchange are divided between both the sender and receiver nodes.

This technique, depicted in Figure 3.2(b), has also been used in synchronization protocols for WSNs (Maroti et al., 2004; Ping, 2003). There are several non-deterministic delays that are variable and cannot be computed in the delay measurement. These non-deterministic delays increase the error in the delay estimation, which is propagated to the time computation component and affects the final synchronization error of the nodes.

Two versions of the delay measurement technique can be used. The first is implemented in the application layer, which, as shown in Figure 3.3, is exposed to a number of error sources, especially the medium access. In the second version, the delay measurement technique is implemented in the Media Access Control (MAC) layer. In this case, the packet is timestamped when it arrives at a receiver node and when it leaves a sender node (the time when the node has access to the medium). Consequently, the nodes can remove a number of delays (which could be sources of error) and precisely compute the delay of a packet. In this case, as we can see in Figure 3.3, the sources of non-deterministic delays are propagation and node interruptions, since the time of transmission and reception can be estimated based on the size of the packet.

This technique is used mostly to compute the packet delay between neighbor nodes. However, it could also be used in a multihop environment if the packet is timestamped while it is received and sent by each of the intermediate nodes.

3.2.3 Comments About the Delay Estimation

The technique used to estimate the packet's trip delay between nodes will basically depend on the required precision and the scenario in which it will be used. In order to compute the packet's trip delay between two nodes in a multihop communication, the RTT technique is usually the method of choice. The RTT technique also has a better performance compared to the delay measurement technique when both are implemented in the MAC layer. However, the communication cost of the delay measurement technique is half the cost of the RTT technique.

The delay measurement method's choice to estimate the packet's trip delay between nodes in a synchronization system is an important factor that influences the final performance of the system. Usually, as will be shown in the next section, to compute a time, a node basically uses the estimated delay. Thus if the estimate delay contains errors, it will affect the time computation process and, as a consequence, the performance of the whole synchronization system. Thus, the method chosen for estimating delays will depend on the application requirements as well as the resources available.

3.3 Time Computation

As mentioned in Section 3.1.1, the clock of a node i, denoted as $t_i(t)$, is a function of both *offset* and *drift*:

1. *Offset* is the difference between t and t_i, i.e., $o_i = t - t_i$. It basically shows the synchronization error of a node at a given instant t.

2. *Drift* defines how the local clock gradually deviates from t due to conditions like temperature or battery voltage. Different clocks have different drifts. The clock drift of a node can also change during its lifecycle.

Thus, t_i can be defined as a function of t as $t_i(t) = d_i t + o_i$, where o_i is the offset and d_i is the drift. In order to compute a node's synchronized time, it is important to have both offset and drift information. If only the offset is available, the nodes' clocks will become highly unsynchronized over a small period of time. When combined with the drift information, the time computation can be more precise over time. For instance, Maroti et al. (2004) show that the clocks of the Mica2 motes (Crossbow, 2004) can drift up to $40\,\mu s$ per second, which means that it would be necessary to continuously synchronize the nodes' clocks every second if we wanted to keep their clocks with a synchronization error in the micro-second range. However, if we know that a node's clock drifts $40\,\mu s$ per second, we can use this information to compute a more precise time based both on this drift information and on the last computed offset.

3.3.1 Offset Computation

Basically, after receiving a packet from a synchronized node containing a timestamp, and after estimating the packet's trip delay, a node i can compute its offset using the following equation:

$$o_i = timestampRecv_i - (timestampSent_k + delay_{i,k}) \tag{3.2}$$

where $timestampRecv_i$ is the local timestamp of the unsynchronized node i, $timestampSent_k$ is the timestamp of the synchronized node k, and $delay_{i,k}$ is the estimated delay of the synchronization packet. The precision of the computed offset is basically affected by the delay estimation. Thus, a number of delay estimations are usually performed in a very small period of time in order to compute several offsets. Once several offsets are available, a number of information-fusion techniques can be executed in order to obtain a more precise offset. These information-fusion techniques include the use of average, linear regression, etc. A weighted average can also be used if the estimated delays and timestamps come from different sources with different synchronization accuracies.

3.3.2 Drift Computation

Because no clock is perfect, it is important to estimate the clocks' drifts to compute a precise time. This is especially important in WSNs, in which the limited hardware results in more

inaccurate clocks and the limited energy available in the nodes makes it essential to decrease the number of synchronization procedures.

Drift is computed mostly accurately by analyzing a number of estimated offsets over time and observing how they slowly increase or decrease. This method is used by Maroti et al. (2004) in the Flooding Time Synchronization Protocol (FTSP – see Section 3.4.2), in which linear regression is used to estimate the drift.

3.3.3 Comments About the Time Computation

Most synchronization approaches in WSNs focus on the delay estimation or synchronization algorithm components. Few solutions take advantage of the available additional timing and delay estimations to propose more efficient time computation techniques. Time computation techniques also need to be studied in the presence of less accurate delay and timestamp information due to the limited hardware of the sensor nodes.

3.4 Synchronization Algorithm

The synchronization algorithm is the main component of a synchronization system. This component determines how the information concerning the estimated delays and computed times will be used so that the nodes in a WSN can synchronize their clocks.

In the next sections, several synchronization algorithms will be studied to show how this component differs from the others. These algorithms are the Timing-Sync Protocol for Sensor Networks (Ganeriwal et al., 2003), the Flooding Time Synchronization Protocol (Maroti et al., 2004), Delay Measurement Time Synchronization (Ping, 2003), and Reference-Broadcast Synchronization (Elson et al., 2002).

3.4.1 Timing-Sync Protocol for Sensor Networks – TPSN

The Timing-sync Protocol for Sensor Networks (TPSN) (Ganeriwal et al., 2003) is a synchronization system for WSNs that works by constructing a hierarchical structure in the network and performing the delay estimation and time computation between the nodes along the edges of the hierarchical structure. The TPSN synchronization algorithm works in two phases: the level discovery phase and the synchronization phase (see Figure 3.4). Both phases are initiated by a single special node called the *root node*. This node, usually the sink node equipped with a GPS receiver, is responsible for starting and coordinating the synchronization process.

In the *level discovery phase*, the root node starts the process by broadcasting a *level discovery* packet to its neighbors, as depicted in Figure 3.4(a). A level discovery packet always contains the level of the node that sent it. In the case of the root node, this level is 0. Neighbor nodes receiving the level discovery packet store the ID of the sender node as their parents and assign themselves the level specified in the level discovery packet incremented by 1. After a small random time (in order to avoid collisions), each of the root's neighbor nodes broadcasts another level discovery packet containing its level (Figure 3.4(b)). Once a node

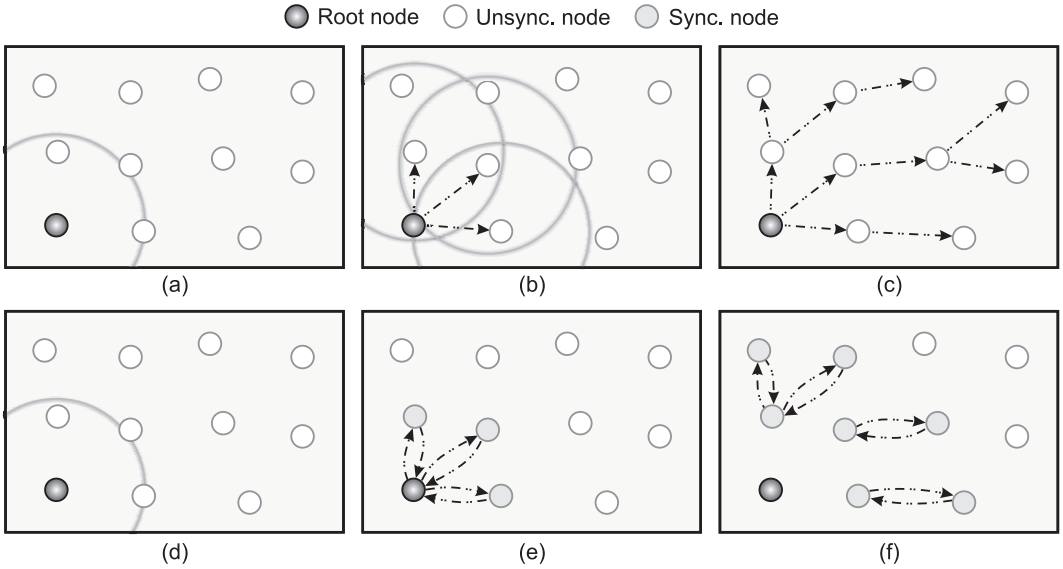


Figure 3.4: Example of the Timing-Sync Protocol for Sensor Networks (TPSN) algorithm: (a), (b), and (c) the level discovery phase; and (d), (e), and (f) the synchronization phase.

receives a level discovery packet, it discards further level-discovery packets in order to save communication costs and avoid loops. This process is repeated for every node until all nodes have established their parents and levels (Figure 3.4(c)).

At this point, the root node starts the *synchronization phase* by broadcasting a *time synchronization* packet, as depicted in Figure 3.4(d). The root's neighbor nodes (on level 1) will receive the time synchronization packet from the root node and, after a random time (again to avoid collisions), they will synchronize their clocks with the root node. This pair-wise synchronization is done by estimating the packet's trip delay using the RTT technique, as explained in Section 3.2.1 (Figure 3.4(e)), and then by computing the node's time based on the timestamp on the RTT reply packet sent by the root node. The nodes on level 2 will overhear the RTT request packet sent by (one or more) nodes on level 1 and, after a random timeout, will also synchronize their clocks based on the already synchronized clocks of their parent nodes on level 1 (Figure 3.4(f)). This process continues until all of the nodes in the network have synchronized clocks. A simplified version of the FTSP algorithm is shown in Algorithm 5.

One drawback of TPSN is that it does not handle dynamic topology changes well, which is very common in WSNs since these networks have node failures, communication failures, node mobility, and node state transitions due to the use of power management schemes. Also, no procedures are proposed for computing the nodes' clock drifts. Finally, in this kind of synchronization algorithm, synchronization errors are easily propagated throughout the network in such a way that the more distant a node is from the root node, the higher will be its synchronization error.

Algorithm 5 Timing-sync Protocol for Sensor Networks synchronization algorithm

▷ **Variables:**
$parent_i$; {Parent of the node.}
$level_i$; {Level of the node.}
$sentDisc_i := false$; {If the node has already sent a discovery packet.}
$syncTimer_i$; {Timer to wait to start the synchronization.}

▷ **Input:**
$msg_i = nil$.
Action:
if $n_i \in \mathcal{B}$ **then** {If this node is a root node.}
  $parent_i := i$; $level_i := 0$; $sentDisc_i := true$;
  Send $levelDiscoveryPkt(i, level_i)$ to all $n_j \in Neig_i$.
  Start $syncTimer_i$.
end if

▷ **Input:**
$msg_i = levelDiscoveryPkt(k, level_k)$.
Action:
if $sentDisc_i = false$ **then** {If this node did not sent any discovery packet.}
  $parent_i := k$; $level_i := level_k + 1$; $sentDisc_i := true$;
  Send $levelDiscoveryPkt(i, level_i)$ to all $n_j \in Neig_i$.
end if

▷ **Input:**
$syncTimer_i$ timeout.
Action:
if $n_i \in \mathcal{B}$ **then** {If this node is a root node.}
  Send $timeSynchronizationPkt()$ to all $n_j \in Neig_i$.
else if $n_i \in \mathcal{N}$ **then** {If this node is an unsynchronized node.}
  $T_i^1 := t_i$;
  Send $rttRequestPkt(i, parent_i, T_i^1)$ to all $n_j \in Neig_i$.
end if

▷ **Input:**
$msg_i = timeSynchronizationPkt()$ or
$msg_i = rttRequestPkt(k, parent_k, T_k^1)$ such that $parent_k != i$.
Action:
if $n_i \in \mathcal{N}$ **then** {If this node is unsynchronized.}
  Start $syncTimer_i$.
end if

▷ **Input:**
$msg_i = rttRequestPkt(k, parent_k, T_k^1)$ such that $parent_k = i$ and
such that $T_i^2 = t_i$.
Action:
if $n_i \in \mathcal{N}$ **then** {If this node is unsynchronized.}
  $T_i^3 := t_i$;
  Send $rttReplyPkt(i, T_k^1, T_i^2, T_i^3)$ to k.
end if

▷ **Input:**
$msg_i = rttReplyPkt(k, T_k^1, T_k^2, T_k^3)$ such that $T_i^4 = t_i$.
Action:
$delay = ((T_k^2 - T_k^1) + (T_i^4 - T_k^3))/2$; {Delay Estimation.}
$offset = (T_k^3 + delay) - T_i^4$. {Offset Computation.}

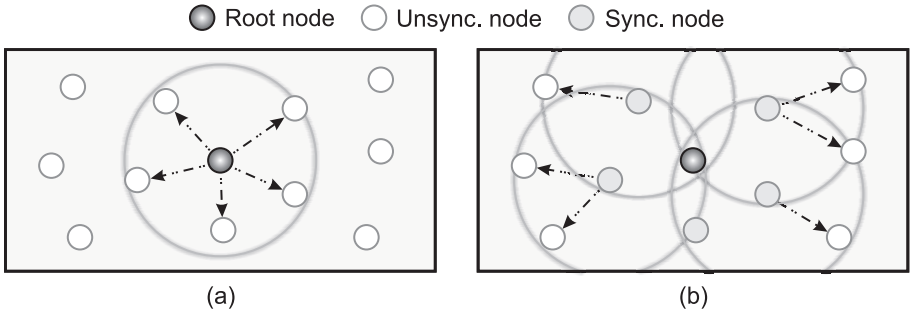


Figure 3.5: Example of the Flooding Time Synchronization Protocol (FTSP): (a) the root node initiating the synchronization flooding; and (b) the nodes propagating the synchronization flooding.

3.4.2 Flooding Time Synchronization Protocol – FTSP

As explained in Section 3.2.2, the Flooding Time Synchronization Protocol (FTSP) (Maroti et al., 2004) computes all delays in the packet transfer time to estimate the packet's trip delay between the sender and receiver nodes using only one broadcast. The multihop synchronization algorithm includes a leader election (root node) and a flooding-like algorithm to propagate the timing information. Multiple floodings can be used to compute the clock drift.

Unlike the TPSN algorithm (see Section 3.4.1), in FTSP, the root node is elected and dynamically reelected by the network. This election consists of a simple election process based on the unique IDs of the nodes. Once a node is elected as the root node, it starts sending synchronization floodings.

In a synchronization flooding, the root node sends a synchronization message containing its timestamp, its root ID, and a sequence number. Neighbor nodes receiving the synchronization message synchronize their clocks using the delay measurement technique (see Section 3.2.2) to compute the packet delay and the offset computation (see Section 3.3.1) to synchronize their clocks (Figure 3.5(a)). Once these nodes have synchronized their clocks, they propagate the synchronization flooding by broadcasting another synchronization message to their neighbors containing their newly synchronized timestamps, the root ID that initiated the synchronization flooding, and the sequence number set by the root node (Figure 3.5(b)). This process is repeated until all nodes have synchronized their clocks. A node receiving a synchronization message stores the root ID and the sequence number contained in the packet. If this node receives another synchronization message from the same root with a sequence number equal to or lesser than the last synchronization message received, it ignores the packet. This procedure reduces the communication cost and avoid loops. A simplified version of the FTSP algorithm (without the leader election process) is shown in Algorithm 6.

3.4.3 Delay Measurement Time Synchronization – DMTS

A very similar algorithm to FTSP is the Delay Measurement Time Synchronization (DMTS) (Ping, 2003). Just like in FTSP, DMTS has a root node that initiates the synchronization

Algorithm 6 The algorithmic part of the Flooding Time Synchronization Protocol

▷ **Variables:**
1: $timestamp_i$; {Timestamp of the node.}
2: $rootId_i = \text{MAX\_INT}$; {Id of the root node.}
3: $seqNumber_i = -1$; {Sequence number sent by the root node.}
4: $syncTimer_i$; {Time between synchronization floodings.}

▷ **Input:**
5: $msg_i = nil$.

Action:
6: **if** $n_i \in \mathcal{B}$ **then** {If this node is a root node.}
7: Start $syncTimer_i$.
8: **end if**

▷ **Input:**
9: $syncTimer_i$ timeout.

Action:
10: $timestamp_i := t_i$;
11: $rootId_i := i$;
12: $seqNumber_i := seqNumber_i + 1$;
13: Send $syncPkt(timestamp_i, rootId_i, seqNumber_i)$ to all $n_j \in Neig_i$.
14: Restart $syncTimer_i$.

▷ **Input:**
15: $msg_i = syncPkt(timestamp_k, rootId_k, seqNumber_k)$ such that $delay_i = delayEstimation(msg_i)$.

Action:
16: **if** ($rootId_k = rootId_i$ and $seqNumber_i < seqNumber_k$) or ($rootId_k < rootId_i$) **then** {New sync.?}
17: $rootId_i = rootId_k$;
18: $seqNumber_i = seqNumber_k$;
19: $t_i = timeComputation(msg_i, delay_i)$;
20: $timestamp_i = t_i$;
21: Send $syncPkt(timestamp_i, rootId_i, seqNumber_i)$ to all $n_j \in Neig_i$.
22: **end if**

flooding. DMTS also uses the delay measurement technique to compute the packet's trip delay and then computes the offset in order to synchronize the clocks.

The main difference between the FTSP and DMTS algorithms is in the flooding control process. While in the FTSP algorithm the root node sends a sequence number in order to help the nodes detect already broadcasted synchronization packets, in the DMTS algorithm the concept of *time source level* is used. In this case, the root node is of time source level 0. Nodes that synchronize their clocks with the root node have level 1 and so on such that nodes that synchronize their clocks with a level n node will have a level of $n+1$. In order to avoid sending duplicate synchronization packets, the nodes in the WSN will only re-broadcast packets that come from a node with a lower level, and only once for a given time period. Otherwise, they will ignore the packet. This technique guarantees that each node will broadcast a timing packet exactly once. The DMTS algorithm is shown in Algorithm 7.

Algorithm 7 Delay Measurement Time Synchronization algorithm

▷ **Variables:**
1: $timestamp_i$; {Timestamp of the node.}
2: $rootId_i$; {Id of the root node.}
3: $level_i$ = MAX_INT; {Time source level of the node.}
4: $sent_i = false$; {If the node already sent a sync. packet.}
5: $syncTimer_i$; {Time between synchronization floodings.}

▷ **Input:**
6: $msg_i = nil$.

Action:
7: **if** $n_i \in \mathcal{B}$ **then** {If this node is a root node.}
8: Start $syncTimer_i$.
9: **end if**

▷ **Input:**
10: $syncTimer_i$ timeout.

Action:
11: **if** $n_i \in \mathcal{B}$ **then** {If this node is a root node.}
12: $timestamp_i := t_i$;
13: $rootId_i := i$;
14: $level_i := 0$;
15: Send $syncPkt(timestamp_i, rootId_i, level_i)$ to all $n_j \in Neig_i$.
16: **else** {This node is not root.}
17: $sent_i = false$. {The node is free to forward another synchronization packet.}
18: **end if**
19: Restart $syncTimer_i$;

▷ **Input:**
20: $msg_i = syncPkt(timestamp_k, rootId_k, level_k)$ such that $delay_i = delayEstimation(msg_i)$.

Action:
21: **if** $(sent_i = false)$ and $((rootId_k = rootId_i$ and $level_i < level_k)$ or $(rootId_k != rootId_i))$ **then**
22: $rootId_i = rootId_k$;
23: $level_i = level_k + 1$;
24: $t_i = timeComputation(msg_i, delay_i)$;
25: $timestamp_i = t_i$;
26: Send $syncPkt(timestamp_i, rootId_i, level_i)$ to all $n_j \in Neig_i$.
27: $sent_i = true$;
28: Start $syncTimer_i$;
29: **end if**

3.4.4 Reference-Broadcast Synchronization – RBS

Reference-Broadcast Synchronization (RBS) (Elson et al., 2002), is a synchronization system in which neighbor nodes exchange packets to maintain relative offsets that are used to compare the local clock of a node with its neighbors' clocks. In this scheme, there is no global synchronization but rather a set of local synchronizations.

In this algorithm, as shown in Figure 3.6, a node broadcasts a *reference packet* to its neighbors (Figure 3.6(a)). Neighbor nodes receiving the reference packet will record their local timestamps indicating when the packet was received (Figure 3.6(b)). Afterward, the nodes will exchange the recorded information between them, as depicted in Figure 3.6(c). Since all nodes will receive the packet at almost the same time, after exchanging their timestamps each node will know the exact differences between their clocks and their neighbors' clocks. In the

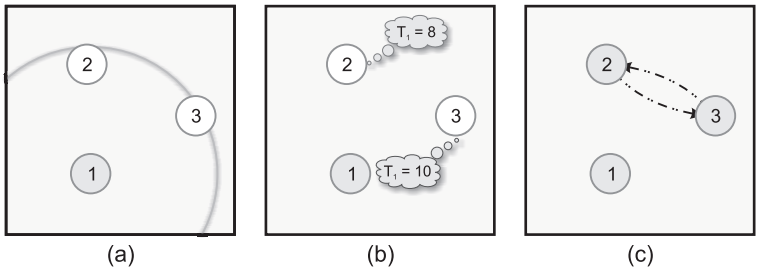


Figure 3.6: Example of the Reference-Broadcast Synchronization (RBS) algorithm: (a) a node sending a reference broadcast; (b) two other nodes recording their local timestamps upon the packet's arrival; and (c) the two nodes exchanging their timestamps.

Algorithm 8 The algorithmic part of Reference-Broadcast Synchronization.

▷ **Variables:**
1: $timestamps_i = \emptyset$; {Set of timestamps from neighbors.}

▷ **Input:**
2: $msg_i = nil$.

Action:
3: Send $referenceBroadcast()$ to all $n_j \in Neig_i$.

▷ **Input:**
4: $msg_i = referenceBroadcast()$ from node k such that $timestamp_i = t_i$.

Action:
5: $refNode_i := k$;
6: $timestamps_i := timestamps_i \cup \{(refNode_i, i, timestamp_i)\}$;
7: Send $sendTimestamp(refNode_i, timestamp_i)$ to all $n_j \in Neig_i$.

▷ **Input:**
8: $msg_i = timestamp(refNode_k, timestamp_k)$ from node k.

Action:
9: $timestamps_i := timestamps_i \cup \{(refNode_k, k, timestamp_k)\}$;

example depicted in Figure 3.6, both nodes 2 and 3 will be able to compute the offset between their clocks, which is ± 2.

The main advantage of the RBS algorithm is that since the reference packet is used by the receiver neighbors to compute the offset among themselves instead of synchronizing the clocks based on the sender, this technique eliminates all the non-deterministic delays associated with the sender side of the packet transmission. On the other hand, the exchange of timestamps between the nodes that receive the reference packet increases the communication cost of the technique, which limits its applicability to small and medium sized networks.

3.4.5 Comments About Synchronization Algorithms

Like a number of decisions in WSNs, choosing the best synchronization algorithm to use depends on the scenario, available resources, and requirements of the application, as well as on the mean synchronization error acceptable to the nodes. For static and less dynamic networks, algorithms based on a hierarchy (Ganeriwal et al., 2003; Ping, 2003) can be a good solution.

On the other hand, for more dynamic networks, an algorithm based on synchronization floodings (Maroti et al., 2004) can be more interesting.

3.5 Summary

This chapter has addressed the synchronization problem from the viewpoint of a WSN. We divided the synchronization systems into three components. These components of the synchronization systems are the Delay Estimation, the Time Computation, and the Synchronization Algorithm.

The importance of such a division into components results from the need to recognize that the final performance of synchronization systems depends directly on each of these components. For instance, a synchronization system should achieve better results if the RTT technique is used instead of the Delay Measurement to estimate delays. The same principle applies to the other components. These components can be seen as subareas of the synchronization problem that need to be studied separately.

As expected, in WSNs, there is no single, perfect synchronization solution that is suitable for every scenario. This chapter has discussed a number of proposed synchronization systems, each of which emphasizes a specific scenario and/or application. The necessity of having different solutions for different applications, as well as the high number of possible applications of WSNs, has greatly motivated the study and proposals of new solutions to the synchronization problem.

Chapter 4

Localization in Time and Space for Wireless Sensor Networks

4.1 Introduction

In this chapter, we define and explain our unified view of the localization and synchronization problems in WSNs, which we refer to as the *localization in time and space* problem. In the next section, we briefly define the term *event* as well as how two events can be considered different from each other. Section 4.3 shows the importance of time and space information when dealing with events in WSNs. Then, in Section 4.4, we show the similarities between the localization and synchronization problems by analyzing the proposed solutions for both problems. In Section 4.5, we extend the definitions of the localization and synchronization problems in order to combine both into our definition of the localization in time and space problem. Finally, in Section 4.6, we show some work that are related to this thesis when dealing with both time and space information in WSNs.

4.2 Definition of an Event

As mentioned in Chapter 1 and depicted in Figure 1.1, WSNs are commonly designed to detect events and gather and deliver sensor data to a monitoring station. Thus, WSNs are basically driven by events.

An event can be defined as "something that happens at a given place and time" or "a phenomenon located at a single point in space-time" (Fellbaum, 1998). However, multiple theories exist concerning the exact definition of an event and how to properly identify two different events. In this work, we follow the definition given by Davidson (1980), who states that an event is composed of both *causal* and *spatiotemporal* criteria:

1. *Causal criterion:* specifies the type of event. Considering the main applications of WSNs, this criterion could be fire, movement, and changes of pressure or ambient light. Regarding causal criteria, Davidson (1980) proposed that two events be considered the same if, and only if, they have the same cause and effect.

2. *Spatiotemporal criterion:* specifies the event's location in time and space. Time can be represented in terms of Coordinated Universal Time (UTC), while space can be represented by latitude, longitude, and altitude. In this criterion, two events are the same if, and only if, they occur at the same time and at the same location.

Thus, in this definition, two events are the same if, and only if, they have the same causal and spatiotemporal criteria. Figure 4.1 depicts four different events for which we can easily identify the causal and spatiotemporal criteria. For instance, in the first event, the *causal criterion* is the `fire` and the *spatiotemporal criterion* is `(1.72, 2.32, 0.53, 1203351004.1981)`.

```
1 ▷ [Time: 1203351004.1981] --- "fire" detected at (1.72, 2.32, 0.53)
2 ▷ [Time: 1203351006.2315] --- "movement" detected at (2.21, 2.67, 0.55)
3 ▷ [Time: 1203352163.1252] --- "movement" detected at (3.49, 0.74, 0.09)
4 ▷ [Time: 1203352399.1622] --- "no-fire" detected at (1.72, 2.32, 0.53)
```

Figure 4.1: Example of four different events that occurred in a WSN.

Events can also be ordered and related to each other by the laws of causality. Causality is a directional relationship between one event (the cause) and another event (the effect) which is the consequence (result) of the first (Flexner et al., 1998). In the example of Figure 4.1, we could identify that Event 2 (e.g., movement of animals) was caused by (or the effect of) Event 1 (e.g., fire in the forest).

4.3 Time and Position Estimation in WSN Protocols

Sensor nodes in a WSN are able to identify the first criterion (*causal criterion*) using their own sensing devices and information-fusion techniques. For instance, a sensor node can identify an event as `fire` if this event was detected by the sensor node using its smoke sensing device. On the other hand, the *spatiotemporal criterion* can only be identified when the sensor nodes of a WSN have synchronized clocks and are able to determine their physical location.

As mentioned in Chapter 1, most WSN applications envision that localization in both time and space is required. To better understand this issue, let us consider two scenarios in a WSN:

1. *Space without Time:* Consider a WSN scenario in which the nodes have been able to compute their positions but not their synchronized time. Now consider three events being detected by this WSN, as depicted in Figure 4.2. The following event log will be generated by the WSN and received by the monitoring station at almost the same time:

   ```
   1 ▷ "fire" detected at (1.72, 2.32, 0.53)
   2 ▷ "movement" detected at (2.21, 2.67, 0.55)
   3 ▷ "movement" detected at (3.49, 0.74, 0.09)
   ```

   Analyzing this log, we may conclude that Events 1 and 2 are related to each other due to their similar locations of occurrence. We can also infer that Event 3 is an independent

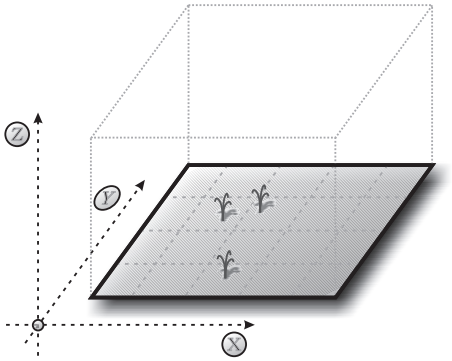


Figure 4.2: Three different events being detected in a WSN scenario in which the nodes have been able to compute their positions but not their synchronized time. In this Figure, 'Υ' represents an event occurring in the network.

and distant event. However, we cannot say for sure whether Event 2 (`movement`) was caused by Event 1 (`fire`), or if Event 1 (`fire`) was caused by Event 2 (`movement`). In the first case, we could think of people running from fire, while in the second case we could think of people causing the fire (security issues). This example shows the importance of event ordering (causality) in WSNs. However, although using the order of the packets' arrivals at the sink node to order events could be enough in some cases, it is hardly true in most WSN scenarios due to a number of reasons:

a) *Events may occur in bursts:* In a WSN, when no important events are being detected, sensor nodes will basically be sending periodic updates to the sink node. However, when an event occurs it is most likely that it will trigger a number of other events in a small period of time. For instance, a very `hot` and `dry` weather event could trigger a `fire` in a monitored forest which could trigger the `movement` of a number of animals in that place. These events may be detected at almost the same time and sent to the sink node by different nodes. A simple retransmission or a delay at one of the intermediate nodes could invert the order of the packets, giving the impression that a `movement` caused the `fire` that caused the `hot` and `dry` weather (i.e., an intentional fire).

b) *Large scale networks:* WSNs can be composed of hundreds or thousands of nodes distributed in a relatively large environment. Some sensor nodes will be near the sink node, while other nodes will be very far. In this case, the data from an event at a location near the sink node can arrive first in comparison to the data from an event detected in a location far from the sink node even though this second event occurred first.

c) *Data aggregation:* In order to save energy, sensor nodes may try to combine information sent by different neighbors and then send all of the received data aggregated into a single packet towards the sink node. This process could not only delay the forwarding mechanism but also make the order of the several aggregated events

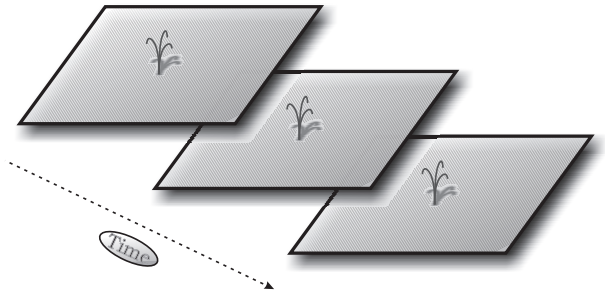


Figure 4.3: Three different events being detected in a WSN scenario in which nodes have been able to localize themselves in time but not in space.

meaningless.

d) *Sink communication:* Due to the use of long distant wireless communication, the sink node will also try to combine a number of different events into a single packet before sending it to the monitoring station.

In all of these cases, the order of the packets arriving at the sink node will probably be different from the order in which the events really occurred.

2. *Time without Space:* Now, try to think of a WSN in which the nodes have been able to localize themselves in time but not in space. Again, consider three events being detected by this WSN as depicted in Figure 4.3. The following event log will be generated by the WSN and received by the monitoring station:

```
1 ▷ [Time: 1203351004.1981] --- "fire" detected
2 ▷ [Time: 1203351006.2315] --- "movement" detected
3 ▷ [Time: 1203352163.1252] --- "movement" detected
```

Analyzing this log, we can see the order in which the events occurred. However, it is still impossible to compute the causality. Through the logs, we can say that Event 1 (`fire`) happened before Event 2 (`movement`). However, Events 1 and 2 could have occurred in totally different locations and could be completely independent from each other.

From these scenarios, we can easily see that in event-based WSNs neither location without time nor time without location represents complete information. Thus, we need both time and space information in order to define a detected event completely, as depicted in Figure 4.4.

Finally, besides being required to identify the spatiotemporal criterion of the events, the localization and synchronization of the sensor nodes are also required for a number of other applications, protocols, and algorithms:

- *Data fusion (or Information Fusion):* Data fusion can be defined simply as the combination of multiple sources to obtain improved information (e.g., cheaper, greater quality, greater relevance) (Nakamura et al., 2005a,b, 2007). In WSNs, data fusion techniques are usually employed to combine and reduce the data transmitted from the sensor nodes

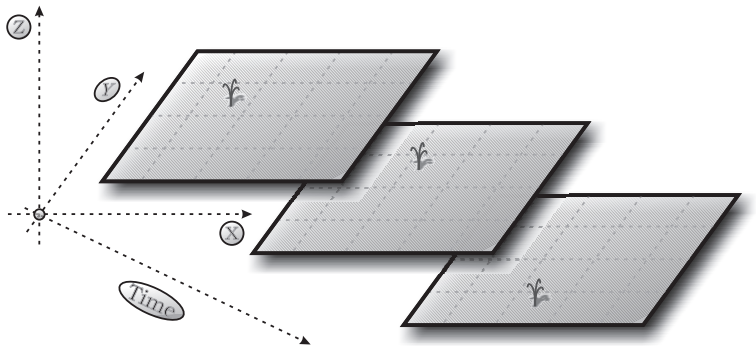


Figure 4.4: Three different events being detected in a WSN scenario in which the nodes have been able to compute both their positions and their synchronized time.

toward the sink node. However, in order to correctly fuse data from a single event detected by different nodes, the sensor nodes must have both synchronized clocks and knowledge of their physical positions. Figures 1.1 and 2.1 show examples of sensor nodes using data fusion techniques to combine the data detected by different sensor nodes.

- *Object tracking:* In object (or target) tracking (Kumar et al., 2000), the movement of several objects (e.g., humans, animals, vehicles) is monitored and/or predicted. This is considered one of the most popular applications in WSNs. However, in order to track correctly the movement of a target that is moving around several sensor nodes, these nodes need not only to know their positions but also to have precisely synchronized clocks. Otherwise, the tracked target might be considered as to be a different target by the unsynchronized nodes. Both position and timing information are also required to estimate the velocity of the targets.

- *Energy maps:* An energy map shows the amount of energy available in each part of the network (Mini et al., 2004; Machado et al., 2005). Position information about the nodes is obviously required in these protocols. Although proposed solutions for energy map construction in WSNs do not usually require synchronized clocks explicitly, most can take advantage of synchronized clocks in order to gather accurately the energy information of the nodes and generate the energy map of the network at a single point of the time.

- *Density control:* Density control algorithms (Xu et al., 2001) allow redundant nodes in a WSN to be turned off, which increases the network lifetime. In most cases, neighbor nodes exchange messages in order to coordinate a schedule of awake and sleep intervals. In order to coordinate this awake and sleep schedule effectively, the nodes' clocks need to be synchronized. Spatial localization information is also used by most of the solutions to improve performance, save energy, and guarantee sensing coverage.

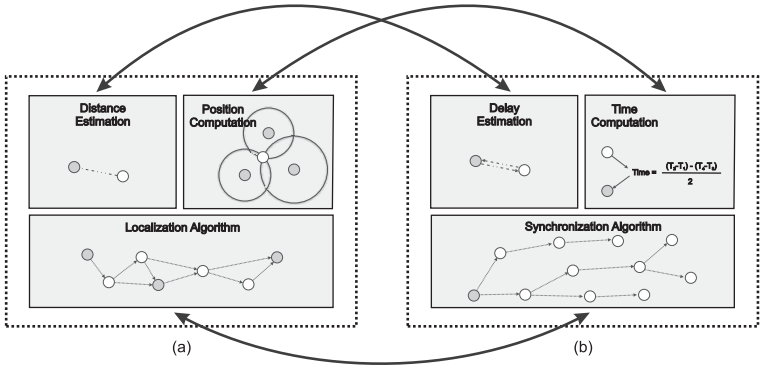


Figure 4.5: Relations between the components of the localization and synchronization systems.

4.4 Similarities in Positioning and Synchronization Systems

As mentioned before, the features of the synchronization problem are similar to those of the positioning problem in many aspects. If we look closer at the solutions to these problems, we can identify a number of similarities between them as well. For instance, positioning systems, as shown in Chapter 2, can be divided into components (Oliveira et al., 2005b), each of which is responsible for solving a single piece of the positioning problem: the *distance estimation*, the *position computation*, and the *localization algorithm* components. Synchronization systems can also be divided into components, as shown in Chapter 3, which are the *delay estimation*, the *time computation*, and the *synchronization algorithm* components. Thus, localization and synchronization systems are clearly related to each other, as shown in Figure 4.5. In the following sections, we will describe the similarities between each of these components.

4.4.1 Data Estimation and Gathering

In the first component of both localization and synchronization systems, the *distance and delay estimation* components, the nodes are basically gathering and exchanging information from and with their neighbors. In the case of localization systems, the distance and/or angle between nodes is estimated, while in synchronization systems, information about the packets' trip delay is estimated. In both cases, neighbor nodes need to exchange one or more messages. Thus, we can combine the information needed for both distance and delay estimation in the same packets in order to save energy and improve the estimations. Considering the main techniques for distance estimation (RSSI and TDoA – Section 2.2) and delay estimation (RTT and Delay Measurement – Section 3.2), we have four different options for combining these techniques:

1. *RSSI + Delay Measurement:* Both RSSI and Delay Measurement techniques require the exchange of a single packet between a sender and a receiver. As a result, the receiver node would be able to estimate the distance between itself and the sender node as well as the packet's trip delay. Since the RSSI technique has inaccuracies, it is possible for

the sender node to send multiple packets so that the receiver can make a better distance estimation. In this second case, the delay estimation could also be improved.

2. *RSSI + RTT:* The RTT technique requires the exchange of two packets for the sender (node that sends the first packet) to estimate the delay. In this case, the receiver node could also estimate the distance and send this information back to the sender node. Thus, the sender node would have the RTT and two distance estimations, which would result in a better distance estimation. If we need the receiver node to estimate the delay and the distance, an additional packet with time and distance estimations could be sent by the sender, resulting in an even better estimation, but with a total communication cost of three packets.

3. *TDoA + Delay Measurement:* TDoA uses a second, slower signal (e.g., ultrasound) to help estimate distances more accurately. We could take advantage of this extra hardware to improve the performance of the synchronization. When implemented at the MAC layer, the main source of errors in the delay measurement technique relies on the nodes' interruptions and packet processing. Also, the propagation delay is usually ignored. In this case, in order to estimate the packets' trip delay, the receiver node could prepare itself and wait for the ultrasound signal as soon as it receives the packet's signal containing the sender's timestamp. Since the receiver node will wait for the ultrasound, and there is no need for packet processing, it practically eliminates delays from interruptions and packet processing. Since the propagation of sound cannot be ignored, it could be computed based on the time difference of arrival between the packet's signal and the ultrasound. Thus, the ultrasound delay could be computed precisely which would allow the estimation of the packet's trip delay.

4. *TDoA + RTT:* The technique of the last option could be used here for each packet exchanged in the RTT technique, improving the performance of both distance and time estimation.

In this thesis, we focus on the investigation of the algorithmic part of time-space localization systems so they can be used in energy-limited large scale WSNs. Thus, although the techniques mentioned above can (and some will) be used to improve the performance of proposed time-space localization systems, in this work we will not evaluate or compare the performance of these techniques.

4.4.2 Information Computation

In the second component of localization and synchronization systems, the *position and time computation* components, nodes make use of mathematical analysis and techniques to compute their positions or their time using the gathered data. The mathematical techniques used for position computation (e.g., solving a system of equations, probabilistic approaches) can also be used for time computation so that memory and processor consumption can be minimized.

Since position computation requires three or more distance estimations, multiple delays can be estimated to improve the performance of the time computation (that usually requires only a single delay estimation).

4.4.3 Distributed Algorithm

Finally, in the third component of localization and synchronization systems, the *localization and synchronization algorithm* components, position and time are computed by the nodes based on a set of reference nodes: the beacon nodes. This component will be the main focus of the proposed solutions for time-space localization in this thesis. Like the other components, the algorithmic components of both localization and synchronization systems also share a number of similarities. For instance, these algorithms can be classified into a few categories:

- *distributed or centralized:* if the positions or time of the nodes can be computed in a distributed fashion by the sensor nodes (self localization), or by a single central node (e.g., a more powerful node, the sink node – remote positioning);
- *with or without infrastructure:* if there is no need for infrastructure (i.e., ad hoc random deployment in remote areas), or if there is a need to install an infrastructure in order to allow the operation of the localization algorithm (e.g., manual placement of beacon or root nodes);
- *relative or absolute data:* if the computed positions and time can be related to global coordinates (e.g., UTC, latitude, longitude) or related to a node or point of the network;
- *indoor or outdoor scenarios:* if the system is more appropriate for indoor or outdoor scenarios; and
- *single hop or multihop communication:* if all unknown nodes have direct communication with the beacon or root nodes, or if a multihop communication is needed.

In terms of performance evaluation, localization and synchronization algorithms also share a number of similarities. The following aspects can be used to evaluate the performance of the proposed algorithms:

- *mean error and consistence:* identifies the mean error of the computed positions and time and also whether this mean error is repeated in similar but different scenarios (consistence of the mean). This mean error limits the usage of the proposed systems to the applications in which this level of inaccuracies is acceptable.
- *communication cost:* identifies the algorithm's complexity in terms of the messages exchanged. Also identifies the cost of the localization or synchronization system to the sensor network.

- *number of settled/synchronized nodes:* defines the percentage of the network nodes that were able to compute their positions or time at the end of the algorithm. The ideal is that all nodes should be able to compute their positions or time; however, in many cases this is not possible.

- *number of beacon/root nodes:* identifies the number of beacon/root nodes required for the algorithm to work. Beacon/root nodes are usually more expensive than normal nodes, and their usage should be minimized.

Finally, certain network characteristics can affect the performance of both localization and synchronization algorithms. It is important to perform experiments for each proposed solution to evaluate their behavior when varying these characteristics, which include the following:

- *network density:* in high density networks we have shorter distances between the nodes, which results in smaller errors in distance estimations and also in the error of the localization system. In addition, the higher number of neighbors results in more information that can be used by an unknown/unsynchronized node to better compute its position/time.

- *network scale:* increasing the number of nodes (and keeping the network density, which increases the area of the sensor field) results in a higher number of hops between nodes. Usually, a higher number of hops results in more inaccurate position and time computation, increasing the mean error of the entire localization and synchronization system.

- *number of beacon nodes:* when deploying a higher number of beacon/root nodes in the network, the mean errors of both localization and synchronization decrease and the number of settled/synchronized nodes tends to increase.

- *GPS accuracy:* although considered by many work, GPS does not provide perfect localization or synchronization, especially in sensor networks where the sensor nodes have limited hardware. Since in most cases the beacon or root nodes use GPS to get their position and time information, the GPS accuracy will impact the final errors of the systems that depend on this service.

4.5 Localization in Time and Space Problem Statement

The need for both time and space information and the similarities between these two problems show the importance of combining them into a single problem: Localization in Time-Space. In doing so, we save energy and network resources, and also have the opportunity to improve time and position estimations in contrast to scenarios in which these problems are solved separately.

We extend the network model introduced in Sections 2.1.1 and 3.1.1 to combine both space and time information in the localization problem. Instead of defining new terms to designate the state of a node, in this work we use the terms already defined in Section 2.1.1

in the context of space localization, as well as the terms defined in Section 3.1.1 regarding time localization:

- *Unknown Nodes ($\mathcal{U}$) or Unsynchronized Nodes ($\mathcal{N}$)*;
- *Settled Nodes ($\mathcal{S}$) or Synchronized Nodes ($\mathcal{C}$)*; and
- *Beacon or Root Nodes ($\mathcal{B}$).*

The time-space localization problem can be stated as follows:

Definition 9 (Localization in Time and Space) *Given a multihop network, represented by a graph $G = (V, E)$, and a set of beacon/root nodes $\mathcal{B}$, their positions (x_b, y_b), and synchronized clocks $t_b = t$ for all $b \in \mathcal{B}$, we want to find the position (x_u, y_u) and time $t_u(t)$ for all unsynchronized and unknown nodes $u \in \mathcal{U}$.*

4.6 State of The Art

As shown in Chapter 2, a number of localization systems have been proposed for WSNs (Oliveira et al., 2005a,b; Boukerche et al., 2007a,c,b; Bachrach and Taylor, 2005; Albowicz et al., 2001; Bahl and Padmanabhan, 2000; Bulusu et al., 2000; Capkun et al., 2002; Chandrasekhar et al., 2006; He et al., 2003; Hofmann-Wellenho et al., 1997; Kaplan, 1996; Ji and Zha, 2004; Langendoen and Reijers, 2005; Pathirana et al., 2005; Priyantha et al., 2000; Savarese et al., 2002; Shang and Ruml, 2004; Shang et al., 2003; Sichitiu and Ramadurai, 2004; Ssu et al., 2005; Whitehouse, 2002; Whitehouse and Culler, 2002). Also, as shown in Chapter 3, a number of synchronization systems have been proposed for WSNs as well(Sundararaman et al., 2005; Sivrikaya and Yener, 2004; Romer et al., 2005; Romer, 2003, 2001; Elson and Romer, 2003; Elson, 2003; Elson et al., 2002; Elson and Estrin, 2001; Maroti et al., 2004; Ping, 2003; Ganeriwal et al., 2003; van Greunen and Rabaey, 2003; Sichitiu and Veerarittiphan, 2003; Shin et al., 2006; Sekhar et al., 2005; Mills, 2006). Although the localization and synchronization problems are strongly related to each other and required by almost all WSNs, these problems have been investigated basically as two different problems requiring different solutions, algorithms, techniques, and protocols. For this reason, current solutions for synchronization and positioning in WSNs are completely independent of each other.

The Global Positioning System – GPS (Kaplan, 1996; Hofmann-Wellenho et al., 1997) is a good example of a time-space localization system. GPS is composed of 24 satellites that operate in orbit around the earth. Each satellite circles the earth at a height of 20,200 km and makes two complete rotations every day. The orbits have been defined in such a way that each region of the earth can "see" at least four satellites in the sky.

A GPS receiver is a piece of equipment that is able to receive the information constantly being sent by the satellites. It uses this information to estimate its distance to at least four known satellites using the Time of Arrival (ToA) technique (as explained in Section 2.2.2),

and, finally, computes its position using trilateration (as explained in Section 2.3.1). Once these procedures have been executed, the receiver is able to know its latitude, longitude and altitude. GPS receivers have a localization error of ±10 to 30 m.

Regarding the synchronization part, GPS receivers are able to synchronize their clocks with a very high accuracy. Real world GPS receivers usually have a precision of nearly 200 ns (Elson and Estrin, 2001) or 1 μs in the case of the GPS module used by the MicaZ nodes (Crossbow, 2004). Since GPS receivers compute their distances to the satellites based on the Time of Arrival (ToA) of the signals, their clocks must be highly synchronized with the satellites' clocks. Also, each GPS satellite has an atomic clock which is able to maintain an accuracy of 10^{-9} seconds per day. Although GPS satellites use *GPS time* (which does not match the rotation of the Earth), they send to GPS receivers a correction that shows the difference between the *GPS time* and UTC.

One of the solutions for the localization in time and space problem in WSNs is to equip each sensor node with a GPS receiver. One of the main advantages would be the relatively small localization and synchronization error, since all nodes would have similar and correlated errors. However, this solution has many disadvantages (Bulusu et al., 2000; Doherty et al., 2001; Niculescu and Nath, 2001; Savvides et al., 2001), such as the increased cost and size of the sensor nodes, the fact that it cannot be used when no satellite is visible (e.g., indoor, underwater, and underground scenarios, severe climatic conditions, space exploration, dense foliage, electronic interference), and the fact that the extra hardware consumes energy. Due to these disadvantages, the usage of GPS is usually limited to a small fraction of the nodes (e.g., beacon nodes).

To the best of our knowledge, the work of Romer and Mattern (2005) and Romer (2005) are the first to address both the localization and synchronization problems in WSNs as related to each other. However, no integrated solution is proposed in their work. As we do in this thesis, Romer and Mattern (2005) and Romer (2005) proposed a four dimensional timespace vector space in which any point p could be specified by its coordinates (p_1, p_2, p_3, p_4) where (p_1, p_2, p_3) is the three-dimensional physical space and p_4 is the one-dimensional physical time. Then, under these assumptions, both the spatial and temporal distances between two timespace points p and q are defined as:

$$\textit{spatial distance} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2} \text{ ; and}$$
$$\textit{temporal distance} = |p_4 - q_4| \text{ .}$$

However, as mentioned earlier, in the work of Romer and Mattern (2005) and Romer (2005), no unified solution for the timespace localization problem is proposed. Their purpose is mainly to make this affinity explicit in order to better understand both the localization and synchronization domains.

The unified view of space and time has also been studied in other areas such as in physics (i.e., space-time continuum), general relativity, and mathematics. However, to the best of our knowledge, this work is the first to go further in treating the localization in time and space problem in WSNs as well as the first to propose unified solutions for the problem.

4.7 Summary

This chapter defined and showed the importance of our unified view of localization and synchronization in WSNs. In the next three chapters of this thesis, we will design and evaluate the performance of three different solutions for the localization in time and space problem to be used in different WSN scenarios. In each chapter, the proposed solutions are explained along with the complete proposed time-space localization algorithm. Then, the performance of the solutions is evaluated through simulations.

Chapter 5

A DV-Hop algorithm for Time-Space Localization in WSNs

5.1 Introduction

One of the contributions of this thesis is the design and performance evaluation of the Synapse algorithm (SYNchronization And Positioning for SEnsor networks), our first solution to the time-space localization problem in WSNs. The Synapse algorithm is based on the idea of the Ad Hoc Positioning System (APS – Niculescu and Nath, 2003). It obeys the same principles and communication pattern as the APS algorithm, resulting in the same localization features and communication overhead. However, our algorithm is also capable of locating the sensor nodes in time i.e., performing synchronization. Also, this synchronization is accomplished at the same time as the positioning.

The main idea of our Synapse algorithm relies on the fact that beacon nodes (GPS-equipped nodes that help in locating other nodes – see Section 2.1.1) already have synchronized clocks, since they receive their timing information from GPS. Synchronized clocks located in different parts of the network are able to estimate the times at which a packet leaves a beacon node and arrives at another beacon node through multihop communication. Based on both this timing information and the number of hops between the beacon nodes, we can compute the average time of a hop. A regular node synchronizes its clock by considering the average time of a hop and the number of hops between itself and every beacon node in the network. Simulation results show that our Synapse algorithm can synchronize the nodes' clocks with a precision of a few microseconds when using the MAC layer to timestamp the packets, or of a few milliseconds when we use only the application layer to timestamp the packets.

In this chapter, we will explain the details of our proposed Synapse algorithm as well as evaluate its performance. The remainder of this chapter is organized as follows. In the next section, we describe the functioning of our proposed scheme. In Section 5.3 we evaluate the performance of the proposed algorithm and, finally, in Section 5.4 we present our conclusions and discuss the pros and cons of our proposed algorithm when applied to WSNs.

5.2 Synapse - A Time-Space Localization System

In this section, we present our proposed time-space localization system: Synapse (SYNchronization And Positioning for SEnsor networks). This algorithm can be divided into two parts that operate at the same time, specifically the localization and synchronization components. The localization part is essentially the APS DV-Hop algorithm proposed by Niculescu and Nath (2003) and discussed in Section 2.4.1 (on page 17). The synchronization part, on the other hand, obeys the same principles as the localization algorithm, but is modified and extended to deal with time estimations. Then, these two parts are combined into a single algorithm, which is shown in Section 5.2.2.

5.2.1 The Synchronization Part of the Synapse Algorithm

The essential strategy of our Synapse algorithm is to use the already synchronized clocks of the beacon nodes (i.e., that are equipped with GPS) to compute the average time of a single hop. Then, unsynchronized nodes can synchronize their clocks based on the time at which the beacon nodes send their synchronization flooding throughout the network as well as on the number of hops to these beacon nodes.

To illustrate the Synapse algorithm, let us consider the example in Figure 5.1. In the first phase of the algorithm, all beacon nodes[1] send a synchronization flooding through the network containing their timestamps and their unique IDs (Figure 5.1(a)). For each synchronization flooding, every intermediate node that forwards a packet will store the beacon's ID, the packet's timestamp, the local timestamp when the packet arrives at the current node, and the number of hops to the beacon that started the flooding. To reduce congestion in the access medium, a node can wait for a random time and save this time in the packets before forwarding them (so that this random time does not affect the final synchronization).

When the beacon nodes receive the timing packets from the other beacon nodes' synchronization floodings, the second phase of the Synapse algorithm begins (Figure 5.1(b)). In this phase, every beacon node computes the average time of a hop:

$$avgHopTime_i = \frac{\sum_{b \in \mathcal{B}} (timeArrived_b^i - timeSent_b)}{\sum_{b \in \mathcal{B}} hops_b^i} \tag{5.1}$$

where $\mathcal{B}$ is the set of beacon nodes, $timeArrived_b^i$ is the time at which the packet from beacon b arrives at beacon i, $timeSent_b$ is the time at which beacon b started the flooding, and $hops_b^i$ is the number of hops between the beacons b and i.

In the third phase (Figure 5.1(c)), the beacon nodes send their estimated average hop times to the network in a controlled flooding, which is a flooding in which the intermediate nodes forward only the first packet received from the flooding received first. When an unsynchronized node receives the average hop time sent by the nearest beacon node, it synchronizes its clock based on this received average value and the information about hops and timestamps recorded

[1] The positioning part of the Synapse algorithm requires three or more beacon nodes to allow normal nodes to compute their position in 2D, as explained in Section 2.4.1.

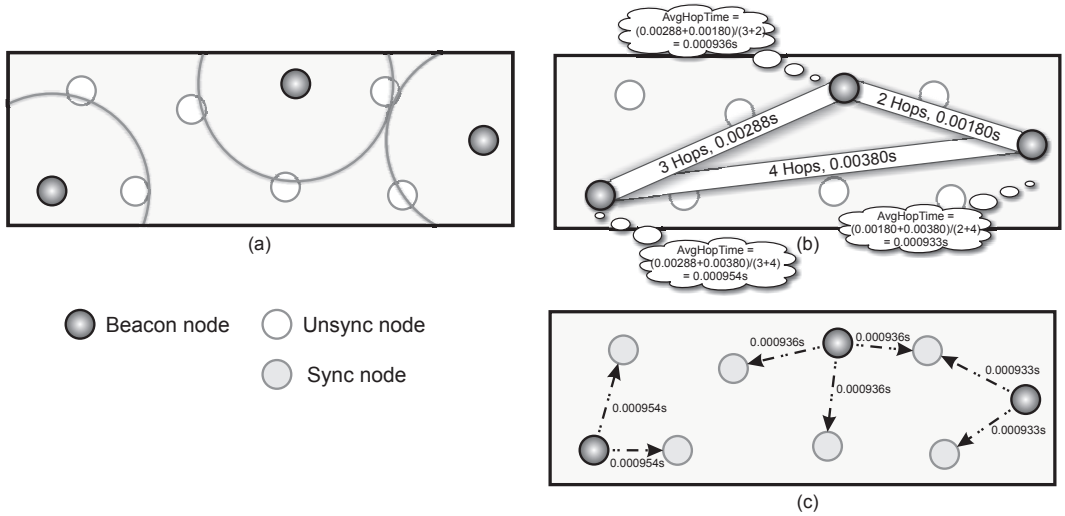


Figure 5.1: Example and phases of the Synapse synchronization part: beacon nodes (a) sending the synchronization floodings, (b) computing the average time of a hop, and (c) sending the computed average time of a hop to the unsynchronized nodes using a controlled flooding.

in the first phase of the algorithm. Thus, a node can estimate its own offset by computing

$$offset_i = \frac{1}{|\mathcal{B}|} \sum_{b \in \mathcal{B}} (localTm_b^i - (tmSent_b + avgHop \times numHops_b)) \tag{5.2}$$

where $localTm_b^i$ is the local time at which a packet from beacon b arrives at node i, $tmSent_b$ is the time at which beacon b started the flooding, $avgHop$ is the received average time of a hop, $numHops_b$ is the number of hops between the node i and beacon b, and $|\mathcal{B}|$ is the size of $\mathcal{B}$, i.e., the number of beacon nodes. Essentially, equation (5.2) computes the time at which the packet should arrive and subtracts this value from the local time at which the packet actually arrives at the node.

5.2.2 The Complete Synapse Algorithm

The complete Synapse algorithm is the combination of the localization and synchronization components. Since both components share the same principles as well as the same communication pattern, we need only to include both the position and timing information into the packets. Accordingly, in the first phase of the algorithm (lines 4–8 of Algorithm 9), the beacon nodes access and flood their position and time information that is then stored by the intermediate nodes (lines 9–17). In the second phase (lines 18–21), the beacon nodes estimate the average size and time of a hop, and send this information in a controlled flooding. In the third phase (lines 22–28), when the remaining nodes receive the average size and time of a hop, they estimate their positions and synchronize their clocks using the stored information.

Algorithm 9 The Complete Synapse Algorithm

▷ **Variables:**
1: $beaconsInfo_i \leftarrow \emptyset$; {Set of beacon positions and timing}
2: $avgHopSize_i \leftarrow -1$; {Average size of a hop computed/received}
3: $avgHopTime_i \leftarrow -1$; {Average time of a hop computed/received}

▷ **Input:**
4: $msg_i = nil$.

Action:
5: **if** $n_i \in \mathcal{B}$ **then** {If this node is a beacon node}
6: $(x_i, y_i, t_i) \leftarrow getGpsTimeAndPosition()$;
7: Send $beaconInfo(i, x_i, y_i, t_i, 0)$ to every $n_j \in Neig_i$.
8: **end if**

▷ **Input:**
9: $msg_i = beaconInfo(k, x_k, y_k, t_k, h_k)$ at time a_k.

Action:
10: **if** $k \notin \mathcal{R}_i$ **then** {If this node did not already receive this packet}
11: $\mathcal{R}_i \leftarrow \mathcal{R}_i \cup \{k\}$;
12: $beaconsInfo_i \leftarrow beaconsInfo_i \cup \{(x_k, y_k, t_k, a_k, h_k)\}$;
13: **if** $n_i \in \mathcal{B}$ **then** {If this is a beacon node, wait for more packets}
14: [Re]Start $waitTimer$.
15: **end if**
16: Send $beaconInfo(k, x_k, y_k, t_k, h_{k+1})$ to every $n_j \in Neig_i$.
17: **end if**

▷ **Input:**
18: $waitTimer$ timeout. {Executed only by the beacon nodes}

Action:
19: $avgHopSize_i \leftarrow \frac{\sum \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{\sum h_j} \forall j \in beaconsInfo_i$;
20: $avgHopTime_i \leftarrow \frac{\sum (a_j - t_j)}{\sum h_j} \forall (a_j, t_j, h_j) \in beaconsInfo_i$;
21: Send $correction(avgHopSize_i,\ avgHopTime_i)$ to all $n_j \in Neig_i$.

▷ **Input:**
22: $msg_i = correction(avgHopSize_k,\ avgHopTime_k)$.

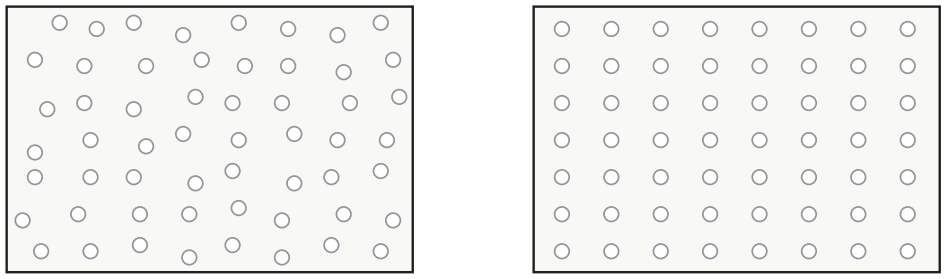
Action:
23: **if** $n_i \in \mathcal{U}$ **then** {If this node is an unknown node}
24: $h_j \leftarrow h_j \times avgHopsize_k$ for all $h_j \in beaconsInfo_i$;
25: $(x_i, y_i) \leftarrow positionComputation(beaconsInfo_i)$;
26: $offset_i = \frac{\sum (a_j - (t_j + avgHopTime_k \times h_j))}{|beaconsInfo_i|} \forall (a_j, t_j, h_j) \in beaconsInfo_i$;
27: Send $correction(avgHopSize_k, avgHopTime_k)$ to every $n_j \in Neig_i$.
28: **end if**

5.3 Performance Evaluation

In this section, we evaluate the synchronization part of our proposed scheme, Synapse. We will not evaluate the localization part of Synapse, since it is already evaluated in the original APS DV-Hop paper (Niculescu and Nath, 2003).

Parameter	*Value*
sensor field	$70{\times}70\,\mathrm{m}^2$
number of nodes	144 nodes (disturbed grid)
number of beacons	8 nodes (randomly chosen)
density	$0.03\,^{\mathrm{nodes}}/_{\mathrm{m}^2}$
communication range	15 m
bandwidth	250 kbps
GPS accuracy	$1\,\mu s$
non-deterministic errors	$5\,\mu s$

Table 5.1: Default scenario configuration for the simulations performed in order to evaluate the performance of the Synapse algorithm.



(a) Disturbed grid (b) Regular grid

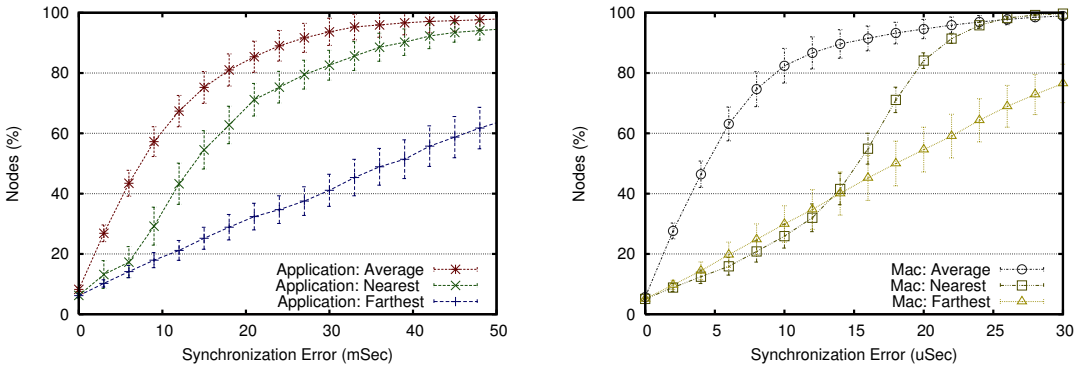
Figure 5.2: Deployment obeys a disturbed grid, avoiding large concentrations of nodes (a) and without forming a regular grid (b).

5.3.1 Methodology

The evaluation is performed through simulations using the NS-2 simulator (McCanne and Floyd, 2005). In all results, the curves represent the average values while the error bars represent the symmetric confidence intervals based on the Gaussian distribution for 95% of confidence from 33 different instances (seeds). The simulation parameters are based on the MicaZ sensor node (Crossbow, 2004), which uses the 802.15.4 standard. The communication range is fixed at 15 m for all nodes. The default parameters for the experiments are presented in Table 5.1.

Regarding the network topology, we assume that the node deployment obeys a disturbed grid (Figure 5.2(a)), that is, a grid in which the location of each node is disturbed by a random zero-mean Gaussian error with variance of 3 m. Therefore, nodes will tend to occupy uniformly the sensor field, avoiding large concentrations of nodes and without forming a regular grid (Figure 5.2(b)). We also avoided nodes and events outside the sensor field area.

Two versions of the Synapse Synchronization are evaluated. The first one is implemented in the application layer, which, as explained in Section 3.2.2 and shown in Figure 3.3 (on page 29), is exposed to a number of error sources, especially the access medium. This is not a good scenario for our scheme, since the Synapse algorithm uses the average time of a hop, which can become erroneous due to variations in the source of delays. In its second version, the Synapse algorithm is implemented in the MAC layer. In this case, the packet is timestamped when it arrives at a receiver node and when it leaves a sender node (right



(a) (b)

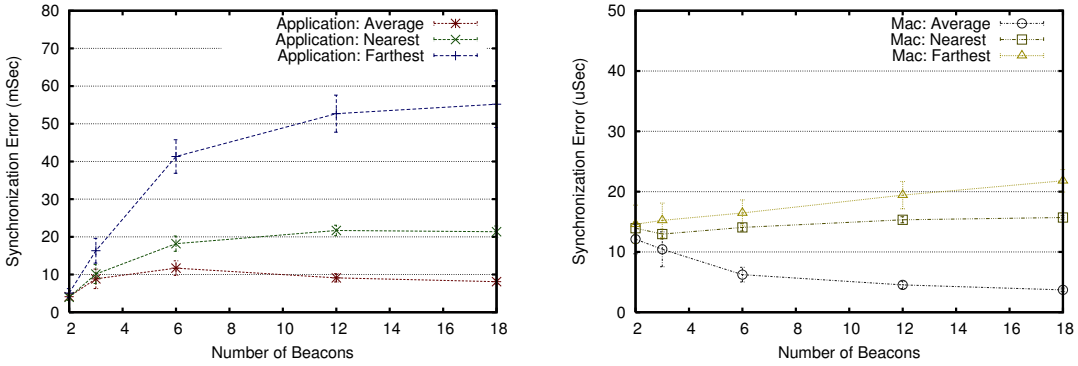
Figure 5.3: Cumulative error of the Synapse algorithm implemented in the (a) Application Layer and (b) MAC Layer.

before the node has access to the medium). The node receiving the packet can compute its exact delay based on the two local timestamps stored in the packet. Consequently, beacon nodes can remove such delays and compute precisely the average time of a hop. In this case, as we can see in Figure 3.3, the sources of non-deterministic delays are propagation and node interruption, since the time of transmission and reception can be estimated based on the size of the packet.

Our proposed scheme, as shown in Section 5.2.2, uses the average time of a hop and the average offset from all beacons. We also evaluate two additional variations of this scheme. In the first (called "Nearest"), we take the average time of a hop provided by the nearest beacon node (in addition to taking the average of all beacons). The offset of the nodes is also computed using only the nearest beacon node. The second variation (called "Farthest") follows the same principle, but uses the farthest beacon node in the computation. The reason for these extra experiments is to assess the behaviour of the proposed algorithm in such scenarios and also to show that the original proposal (called "Average") actually obtains the best results. Thus, the emphasis of our comments will be on the original proposal.

5.3.2 Error Analysis

The distribution of synchronization errors among the sensor nodes is depicted in Figures 5.3(a) – using the application layer – and 5.3(b) – using the MAC layer. The cumulative error identifies the percentage of nodes (y-axis) with an error smaller than a parametrized value (x-axis). A steep curve means that the majority of nodes has a small error. We can see that the best results are achieved by the original Synapse algorithm ("Average") in both layers. Also, we can see clearly how the proposed scheme can be affected by non-deterministic delays. While the MAC version of the algorithm results in errors of microseconds, the application-layer version results in greater errors of milliseconds. Finally, in the MAC version of Synapse, almost 80% of the nodes synchronized their clocks with an error smaller then $10\,\mu s$, which shows how the proposed algorithm can synchronize the nodes with a high degree of accuracy.



(a) (b)

Figure 5.4: Influence of the number of beacon nodes on the Synapse algorithm implemented in the (a) Application Layer and (b) MAC Layer.

5.3.3 Increasing the Number of Beacons

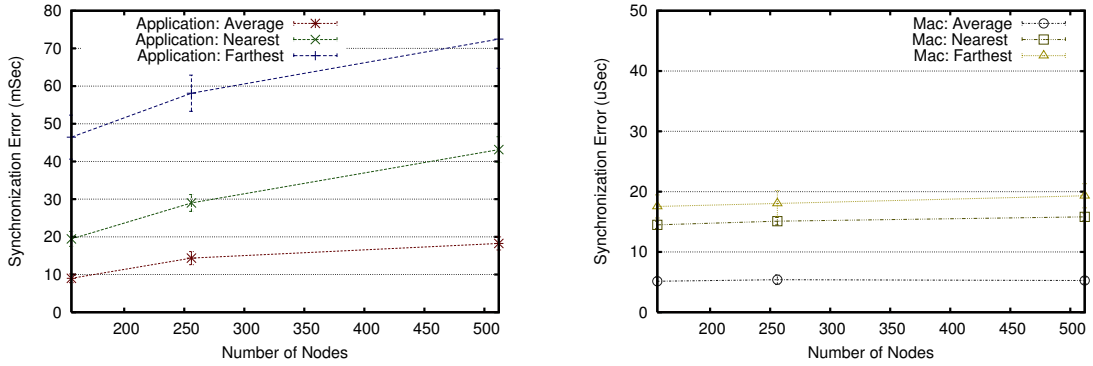
Figures 5.4(a) and 5.4(b) depict the average synchronization error of the nodes when the number of beacon nodes increases. Intuitively, we expect to decrease errors by increasing the number of beacon nodes since we use the average from all beacons in the estimations. This is exactly what happens when Synapse is implemented in the MAC layer. However, when the algorithm is implemented in the application layer, the synchronization error tends to increase. The reason is that the access medium is more intense in the first phase of the algorithm, since all beacon nodes start flooding the network. When the number of beacons increases, the communication medium gets congested, increasing the non-deterministic delays in this application version of Synapse. This also occurs in the MAC version, but the delays are known and can be removed when computing the time of a hop in the average version.

5.3.4 The Impact of the Network Scale

Scalability is evaluated by increasing the network size from 144 to 256 to 512 nodes with a constant mean density of $0.03\,^{\text{nodes}}/_{\text{m}^2}$. Thus, the sensor field is resized according to the number of sensor nodes. Figures 5.5(a) and 5.5(b) depict the average synchronization error of the nodes when the number of nodes increases. In the application layer, the greater the number of hops, the greater the non-deterministic delays and, consequently, the greater the synchronization error will be. However, in the MAC version, the synchronization error remains almost the same, which means that the proposed algorithm scales with the number of nodes without increasing the synchronization error.

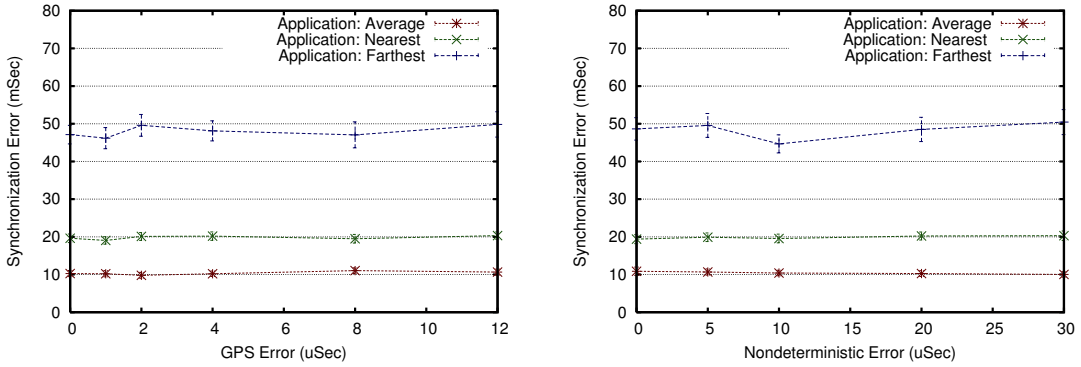
5.3.5 GPS and Non-deterministic Errors

In this section, we evaluate the proposed algorithm under some real world inaccuracies. Two aspects that can affect the performance of the algorithm are evaluated: GPS inaccuracy and non-deterministic errors. Real world GPS receivers usually have a precision of nearly 200 ns



(a) (b)

Figure 5.5: Influence of the number of nodes on the Synapse algorithm implemented in the (a) Application Layer and (b) MAC Layer.

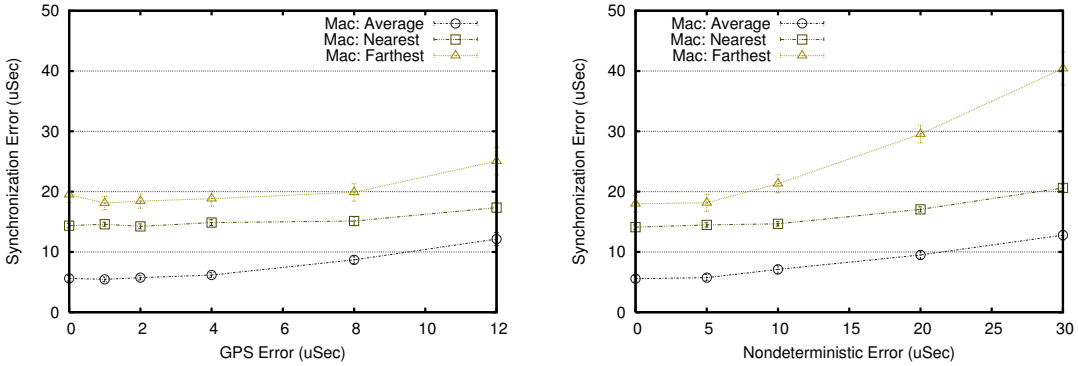


(a) (b)

Figure 5.6: Simulation results in the Application Layer: (a) Impact of the GPS error; and (b) Impact of the non-deterministic errors.

(Elson and Estrin, 2001) or 1 μs in the case of the GPS module used by MicaZ nodes (Crossbow, 2004). Still, to understand better how GPS affects the final synchronization error of the proposed algorithm, we increased the GPS accuracy from 0 to 12 μs. The results, depicted in Figures 5.6(a) and 5.7(a), show that in the application layer, GPS barely affects the algorithm at all, since delays in the access medium are the most evident sources of synchronization errors. On the other hand, in the MAC version of Synapse we can see a small increase in the synchronization error, however, this increase is much smaller than the GPS inaccuracy.

Finally, we evaluated the proposed algorithm under some non-deterministic delays. These non-deterministic delays include interruptions, system calls, context exchanges, and other inter-node processing delays that can vary from 5 to 30 μs (Maroti et al., 2004). Figures 5.6(b) and 5.7(b) depict the results of this experiment. Just like in the previous experiment, the introduced delay is not enough to change the synchronization error obtained by the application version of Synapse. However, when using the MAC version, the synchronization error increases more quickly, since this error is introduced in every hop of a packet. But even under a high



(a) (b)

Figure 5.7: Simulation results in the MAC Layer: (a) Impact of the GPS error; and (b) Impact of the non-deterministic errors.

non-deterministic error of 30 μs, the nodes still get an accurate time estimation, thus indicating that the proposed algorithm can be used in real world applications.

5.4 Summary

In this chapter, we proposed the Synapse (SYNchronization And Positioning for SEnsor networks) algorithm, a time-space localization algorithm for WSNs. The localization part of Synapse is similar to the APS DV-Hop algorithm. The novelty of this algorithm is the synchronization component, as well as the fact that two problems are solved together, thus saving resources. In this case, the main idea of our Synapse algorithm is to use the already synchronized clocks of the beacon nodes to compute the average time of a hop. Based on this concept, regular nodes can synchronize their clocks in relation to the beacon nodes.

When implemented in the MAC layer, our proposed algorithm synchronizes the clocks of the nodes with an accuracy of microseconds. The algorithm can also be implemented in the application layer, obtaining an accuracy of milliseconds. Synapse was also shown to tolerate the real world inaccuracies of GPS receivers and the non-deterministic delays inside the nodes.

Some of the advantages of the Synapse algorithm are the following: first, it is suitable for sparse networks; second, its positioning part is immune to RSSI inaccuracies; and third, it requires a small number of beacon nodes (it can work using only three beacons). On the other hand, the main drawback of the Synapse algorithm is its communication cost. As a DV-Hop algorithm, the communication complexity of this algorithm is determined by the total number of beacon and regular nodes, which is $O(nb)$, where n is the number of nodes in the network and b is the number of beacon nodes. Thus, this relatively high communication cost in the Synapse algorithm limits its usage to small to medium sized WSNs.

In terms of the applicability of our Synapse algorithm, we can say that the proposed algorithm is more suitable for sparse, small to medium sized WSNs in which there is the need for minimizing the number of deployed beacon nodes (due to hardware cost or inability to

deploy additional beacon nodes). The obtained results show that the proposed algorithm can synchronize the clocks of the sensor nodes with a good accuracy of a few microseconds, which is suitable for performing most tasks that require synchronized clocks such as data fusion, object or target tracking, events ordering, etc.

However, the positioning part of Synapse, which is based on the APS algorithm, results in relatively high and unpredictable localization errors of about 70% of the communication range. This localization error affects most protocols and applications that depends on localization to work properly (Oliveira et al., 2005b). On the other side, a number of work have been proposed to decrease this localization error of APS (Savarese et al., 2002; Niculescu and Nath, 2004; Cheng et al., 2005) and all of them can be used in the proposed Synapse algorithm. For instance, the Hop-Terrain algorithm, proposed by Savarese et al. (2002), has been designed to decrease the final localization error of APS by adding a refinement phase to the algorithm.

Since we still have the scalability issue of Synapse, in order to deal with the scenarios of larger WSNs, in the next chapter we propose the Lightness algorithm, a more scalable and efficient time-space localization system for use in large scale WSNs.

Chapter 6

A Lightweight Algorithm for Time-Space Localization in WSNs

6.1 Introduction

In order to deal with large scale WSNs, lightweight and scalable algorithms are required. With this fact in mind, we have proposed a new and lightweight time-space localization algorithm for WSNs which we refer to as the Lightness algorithm (Lightweight Time and Space localization).

Our proposed Lightness algorithm has some key aspects. First, the positioning component is based on Recursive Position Estimation (RPE – Albowicz et al., 2001), a well-known and scalable positioning system for sensor networks that requires only 5% of the nodes to be GPS-enabled beacon nodes. Second, this algorithm assumes that the beacon nodes already have synchronized clocks, since they can get their timing information from GPS. Third, both timing and positioning information are propagated in the network with only a $O(n)$ communication cost, being n the number of nodes. Finally, we use the packet delay measurement, shown in Section 3.2.2, to estimate the delay of a packet and synchronize neighbor nodes using only one broadcast. Simulation results show that our Lightness algorithm can scale to thousands of nodes while keeping a synchronization error of a few microseconds and decreasing the positioning error (compared to the original RPE algorithm).

In this chapter, we will explain the details of our proposed Lightness algorithm and evaluate its performance. The remainder of this chapter is organized as follows. In the next section we describe our proposed lightweight algorithm, the Lightness algorithm. The performance evaluation of the algorithm is shown in Section 6.3. Finally, Section 6.4 presents our conclusions and the pros and cons of this solution.

6.2 Lightness - A Novel Lightweight Algorithm for Time and Space Localization

In this section, we propose a new time-space localization algorithm: the Lightness algorithm (Lightweight Time and Space Localization). Just like our Synapse algorithm explained in Chapter 5, the Lightness algorithm can be divided into two parts that work at the same time, specifically the positioning and synchronization components. The positioning part is essentially the RPE algorithm proposed by Albowicz et al. (2001), which is discussed and explained in Section 2.4.2 (on page 18). The synchronization part obeys both the same principle and communication pattern as the positioning algorithm, but it is modified and extended to deal with time estimations. Then, these two parts are combined into a single algorithm, which is shown in Section 6.2.2.

6.2.1 The Synchronization Part of the Lightness Algorithm

The essential strategy of the Lightness algorithm synchronization component is to use the additional previously synchronized beacon nodes required by the positioning part to improve the synchronization performance. The positioning part requires 5% of the nodes to be GPS-enabled beacon nodes (Albowicz et al., 2001). Since these beacon nodes are equipped with GPS receivers, they are already synchronized with Coordinated Universal Time. Thus, the synchronization recursion[1] can start from different parts of the network at the same time. Also, the position computation in the localization part requires at least three reference nodes to compute the position of a node (e.g., using trilateration). Thus, we take advantage of this behavior in the synchronization part by taking the average time estimation from each reference node to improve the synchronization performance even further.

Consider the example in Figure 6.1. In the first phase of the algorithm, all beacon nodes broadcast a timing packet to their neighbors so that the beacon nodes can be used as reference nodes (Figure 6.1(a)). This timing packet contains the timestamp of the sender node i ($tmSt_i$), among other information. Neighbor nodes, such as node 8 in the example, measure the packet delay (explained in Section 3.2.2), which can provide the time of the packets' trip with very good precision, and store this delay estimation (d_i), the packet information, and the local time of the packet's arrival ($tmArr_i$), in a local set $times_i$ (Figure 6.1(b)).

When a node has the timing information of at least three reference neighbors (which is required by the positioning algorithm) it can compute its offset, which is the average of the individual offsets in $times_i$,

$$offset_i = \frac{\sum_{k \in times_i} [tmArr_k - (tmSt_k + d_k)]}{|times_i|}, \tag{6.1}$$

This node can then become a synchronized node as depicted in Figure 6.1(c).

In the fourth phase (Figure 6.1(d)), the recently synchronized node becomes a reference

[1] See Section 2.4.2, on page 18, for information about the localization recursion and position computation.

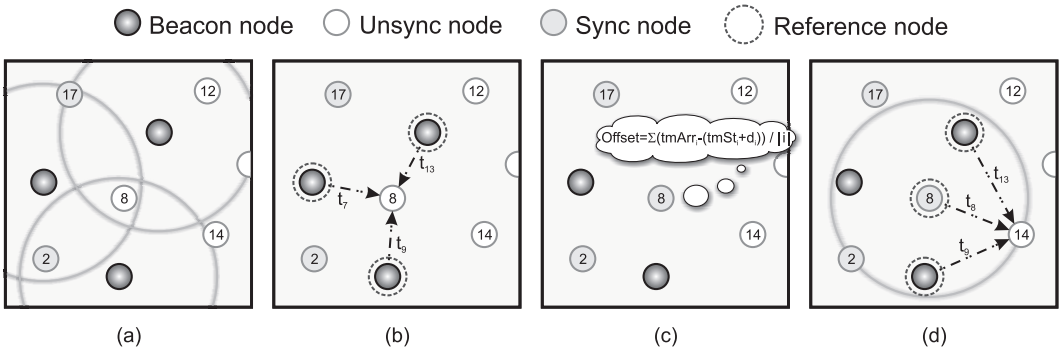


Figure 6.1: Example and phases of the Lightness synchronization component: (a) beacon nodes starting the synchronization recursion from different parts of the network; and (b) an unsynchronized node estimating the packets' delays, (c) computing its synchronized time, and (d) becoming a reference node.

node and sends a timing packet to its neighbors. This timing packet contains two values that are used to rank the references' information: the *number of hops to beacons* – h_i – and the *residual value* – rt_i:

- *Number of hops to beacons* – h_i*:* an estimation of the number of hops to the beacon nodes. Since the beacon nodes are spread through the network, this number of hops is an average of the number of hops of the node's reference neighbors computed by the following equation:

$$h_i = \frac{\sum_{k \in times_i} h_k}{|times_i|}, \tag{6.2}$$

  where h_k is the estimated number of hops of the k-th node in the $times_i$ set. The greater this number, the farther a node is from the beacon nodes. Beacon nodes have $h_i = 0$, and every time a node receives a packet from a neighbor it increases the number of hops (h_k) and stores it in its $times_i$ set.

- *Residual value* – rt_i*:* is the difference between the computed offset and the individual offsets in the $times_i$ set:

$$rt_i = \sum_{k \in times_i} (offset_i - offset_k), \tag{6.3}$$

  where $offset_k = tmArr_k - (tmSt_k + d_k)$. This value shows the sum of the difference between the computed offset and the individual offsets in $times_i$. The greater this number, the more inconsistent is the computed offset, since it means that the final offset was computed based on very different individual offsets.

The unsynchronized nodes sort the references by choosing the references closer to the beacons first (using the *hops to beacons* value) and using the *residual value* as a tiebreaker. As a result, the unsynchronized nodes are able to choosing for storage only the best references

saving, thus, memory. The decision to use the *hops to beacons* value for ranking the references was based on the simulation results (shown in Section 6.3.7).

Algorithm 10 The Complete Lightness Algorithm

▷ **Variables:**
1: $positions_i = \emptyset$ {Set of received positions}
2: $times_i = \emptyset$ {Set of received timestamps}
3: $x_i,\ y_i,\ rp_i$ {Position and residual of positioning}
4: $tmSt_i, rt_i, h_i$ {Timestamp, residual of sync., and hops}
5: $tmAr_i, offset_i$ {Time of a packet arrival, comp. offset}
6: $neigh_i$ {Set of neighbors nodes - broadcast}
7: $waitTimer_i$ {Timer to wait for more packets}

▷ **Functions:**
8: $getGpsPosition$ {Returns the nodes' GPS location}
9: $distanceEstimation$ {Pkts' travel distance estimation}
10: $positionComputation$ {Computes the nodes' position}
11: $delayMeasurement$ {Pkts' travel delay estimation}
12: $timeComputation$ {Computes the nodes' offset}

▷ **Input:**
13: $msg_i = nil$.

Action:
14: **if** $n_i \in \mathcal{B}$ **then** {If this node is a beacon node}
15: $(x_i, y_i) := getGpsPosition()$; $rp_i := 0.0$;
16: $tmSt_i := t_i$; {Gets the local time}
17: $rt_i := 0.0$; $hops_i := 0$;
18: Send $lightness(x_i, y_i, rp_i, tmSt_i, rt_i, h_i)$ to $neig_i$.
19: **end if**

▷ **Input:**
20: $msg_i = lightness(x_k, y_k, rp_k, tmSt_k, rt_k, h_k)$ such that
$dist_k = distanceEstimation(msg_i)$,
$delay_k = delayMeasurement(msg_i)$, and

Action:
21: **if** $n_i \in \mathcal{U}$ **then** {If it is an un(known/sync) node}
22: $positions_i := positions_i \cup \{(x_k, y_k, dist_k, rp_k)\}$;
23: $times_i := times_i \cup \{(tmSt_k, delay_k, t_i, rt_k, h_k + 1)\}$;
24: [Re]Start $waitTimer_i$.
25: **end if**

▷ **Input:**
26: $waitTimer_i$ timeout.

Action:
27: **if** $size(positions_i) >= 3$ **then** {Enough positions?}
28: $(x_i, y_i, rp_i) := positionComputation(positions_i)$; {Becomes a settled node}
29: $(offset_i, rt_i, h_i) := timeComputation(times_i)$; {Becomes a synchronized node}
30: Send $lightness(x_i, y_i, rp_i, tmSt_i, rt_i, h_i)$ to $neig_i$. {Becomes a reference node}
31: **end if**

6.2.2 The Lightness Algorithm

The complete Lightness algorithm is the combination of the positioning and synchronization components. Since both parts share the same principle and the same communication pattern, we need only to combine both the position and timing information into the packets.

Accordingly, in the first phase of the algorithm (lines 13–19 of Algorithm 10), the beacon nodes access and broadcast their position and time information that is stored by the neighbor nodes in addition to the estimated packet delay and distance information (lines 20–25). In the third phase (lines 28–31), the unsynchronized nodes estimate both their positions (as in the RPE algorithm) and their offsets (as in Equation 6.1). Finally, these newly estimated times and positions are broadcasted to the neighbors to assist these other nodes in becoming synchronized and settled nodes (lines 32 and 33).

6.3 Performance Evaluation

In this section, we evaluate the synchronization part of our proposed scheme, the Lightness algorithm. We will not evaluate the positioning part of Lightness, since it has already been evaluated in the original RPE paper (Albowicz et al., 2001).

6.3.1 Methodology

In order to evaluate the performance of our proposed protocol, we have carried out an extensive set of simulation experiments using the NS-2 simulator (McCanne and Floyd, 2005). In all of our results, the curves represent the average values, while the error bars represent the symmetric confidence intervals for 95% of confidence from 33 different instances (seeds). The simulation parameters are based on the MicaZ sensor node. The communication range is fixed in 15 m for all nodes. The default parameters for the experiments are presented in Table 6.1.

Regarding the network topology, we assume that the node deployment obeys a disturbed grid, that is, a grid in which the location of each node is disturbed by a random zero-mean Gaussian error. Therefore, nodes will tend to occupy the sensor field uniformly, avoiding large concentrations of nodes and without forming a regular grid.

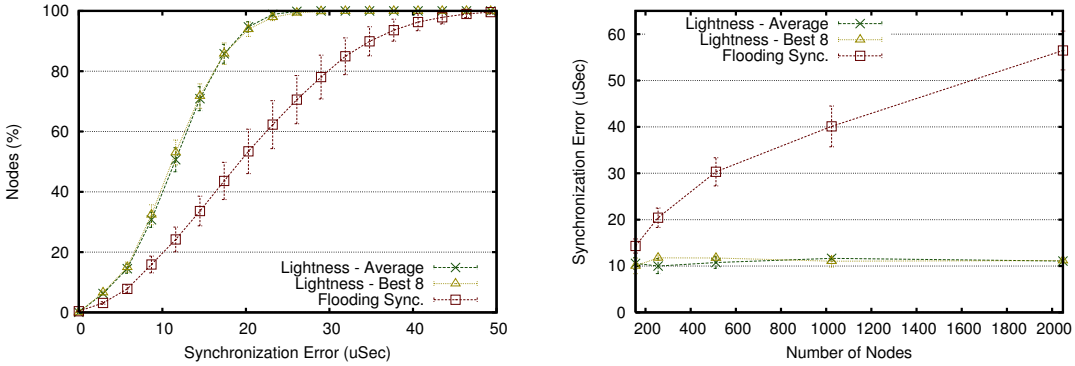
Two versions of the Lightness algorithm are evaluated. In the first version, offsets are computed using all of the available information, as explained in the last section (Average). The second version of the algorithm stores only the 8 best reference neighbors in $times_i$ (Best 8). In this case, the number of hops to beacons (h_i) is used to rank the references, and the residual value (rt_i) is used as a tiebreaker. We compare the Lightness algorithm with a flooding-like synchronization algorithm in which only one node is the time reference (root or master node). This type of algorithm is used by synchronization systems like the Flooding Time Synchronization Protocol (FTSP) (explained in Section 3.4.2, on page 34) and Delay Measurement Time Synchronization (DMTS) (explained in Section 3.4.3, on page 34).

6.3.2 Error Analysis

The distribution of synchronization errors among the sensor nodes is depicted in Figure 6.2(a). As mentioned before, the cumulative error identifies the percentage of nodes (y-axis) with an error smaller than a parametrized value (x-axis). Thus, a steep curve means that the majority of nodes has a small error. We can see that the best results are achieved by the Lightness

Parameter	*Value*
sensor field	$92 \times 92\,\mathrm{m}^2$
number of nodes	256 nodes (disturbed grid)
number of beacons	5% of nodes (randomly chosen)
density	$0.03\,^{\mathrm{nodes}}/_{\mathrm{m}^2}$
communication range	15 m
GPS accuracy	$1\,\mu s$
one hop sync accuracy	$4\,\mu s$ (Maroti et al., 2004; Ping, 2003)

Table 6.1: Default scenario configuration for the simulations performed to evaluate the performance of the Lightness algorithm.



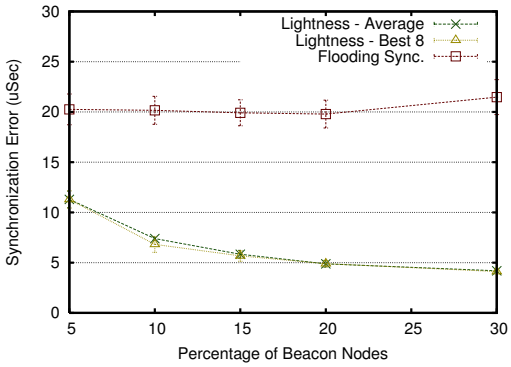
(a) (b)

Figure 6.2: Simulation results for the Lightness algorithm: (a) Cumulative error and (b) Impact of the network scale.

algorithms, for which 50% of nodes have a synchronization error smaller than 12 μs and almost all nodes have a synchronization error smaller than 20 μs. Also, we can see that by using only the 8 best references, we can reduce the memory usage without increasing the synchronization error.

6.3.3 The Impact of the Network Scale

In order to investigate how scalable our algorithm is, we increased the size of our network from 144 to 256, 512, 1024, and to 2048 nodes with a constant density of $0.03\,^{\mathrm{nodes}}/_{\mathrm{m}^2}$ while also keeping the constant beacon concentration (5%) required by the positioning algorithm. Thus, both the sensor field and the number of beacons are resized according to the number of sensor nodes. Figure 6.2(b) depicts the average synchronization error of the nodes when the number of nodes increases. In the flooding-like synchronization, the one hop inaccuracy is propagated and increases every hop. However, in the Lightness algorithm, the increased number of beacon nodes spread throughout the network and the average computed offset can maintain the synchronization error at about 11 μs for any number of nodes, which shows how the proposed algorithm can scale with the number of nodes without increasing the synchronization error. It is important to note that our Lightness algorithm also scales in terms of message exchange since its communication cost is only one flooding, i.e., $O(n)$, being n the number of nodes.



(a)

Figure 6.3: The impact of the beacons scale on the Lightness algorithm.

6.3.4 Increasing the Number of Beacons

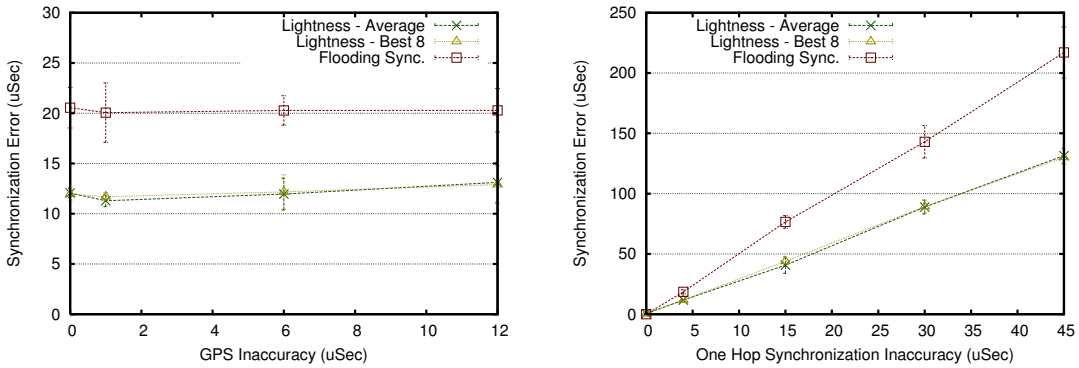
Figure 6.3(a) depicts the average synchronization error of the nodes when the number of beacon nodes increases. As we can see, by increasing the number of beacons, we can reduce the synchronization error. Particularly, when we increase the number of beacons from 5% to 10%, the synchronization error decreases greatly (nearly 50%), and it continues decreasing (at a lesser rate) with the increasing number of beacon nodes. This behavior is not possible in a flooding-like algorithm because it accepts only one node as the synchronization reference. It is important to note that increasing the number of beacons also reduces the positioning error (Oliveira et al., 2005a,b; Oliveira, 2005). The main reason for this enhanced performance in our algorithm is that the greater the number of beacon nodes, the lesser the number of hops in the algorithm's recursion and, thus, the lesser will be the error propagation.

6.3.5 GPS Inaccuracy

Unlike the other synchronization algorithms, our Lightness algorithm can be influenced by the GPS inaccuracy. Real world GPS receivers usually have an accuracy of nearly 200 ns (Elson and Estrin, 2001) or 1 μs in the case of the GPS module used by the MicaZ nodes. Thus, to understand how GPS affects the final synchronization error of the proposed algorithm, we have increased the GPS inaccuracy from 0 to 12 μs. The results, depicted in Figure 6.4(a), show that this influence is not critical. In the worst case, when we have a GPS inaccuracy of 12 μs, the synchronization error is smaller than 2 μs, which can be explained by the small concentration of beacon nodes (5%) and the average computed offset.

6.3.6 One Hop Synchronization Inaccuracy

The one hop synchronization accuracy tells us how well a pair of nodes can synchronize to each other. Several one hop synchronization techniques have been proposed in literature, including round-trip time (explained in Section 3.2.1), reference broadcasts (Section 3.4.4), and delay measurement (Section 3.2.2). Delay measurement is the technique used by our Lightness



(a) (b)

Figure 6.4: Impact of real world inaccuracies in the Lightness algorithm: (a) Impact of the GPS inaccuracy and (b) Impact of one hop synchronization inaccuracy.

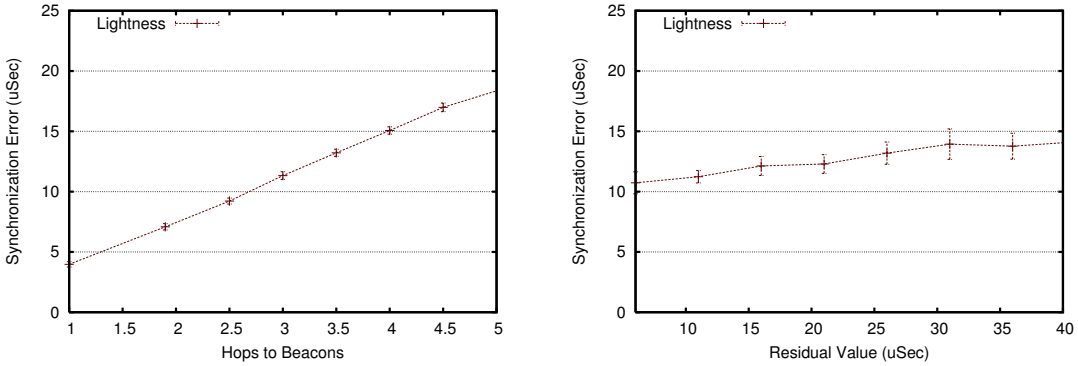
algorithm since it requires only one packet to be exchanged in order to work. It has a precision of about 2-4 μs in Maroti et al. (2004) and 2-32 μs in Ping (2003). To evaluate the effects of the one hop synchronization inaccuracy, we have increased this inaccuracy from 0 to 45 μs. The results, depicted in Figure 6.4(b), show that both Lightness and the flooding synchronization are influenced by this inaccuracy. However, compared to the flooding synchronization, the errors in the Lightness algorithm are smaller and less affected by this inaccuracy.

6.3.7 Deducing a Node's Synchronization Error

As previously discussed in Section 6.2.1, in the Lightness algorithm we compute the *hops to beacons* value (h_k) and *residual* value (rt_k) so that these values can be used by the nodes to rank their individual offsets. However, these values can also be used to deduce a node's synchronization error, as shown in Figures 6.5(a) and 6.5(b). From these figures, we can see that the *hops to beacons* value is more reliable in a certain range (1-4 hops), more consistent, and has a smaller confidence interval, all of which makes it a good indicator of a node's synchronization error.

6.3.8 Improving the Position Computation

In synchronization systems, the clock drift of a node is computed based on several synchronizations to estimate how the current hardware clock differentiates from the reference clock (see Section 3.3.2). In this case, the synchronization algorithm must be executed a number of times. When this drift computation is required, we can take advantage of the extra communication to improve the initial position estimation of the nodes. Figure 6.6 depicts the positioning error as several synchronizations are executed. We tested three different position computations. In the first case (Total Avg), we save each individual position computation in a set and take the average of all computed positions. In the second case (Best 3 Avg) we only take the average of the three best positions computed (sorted by the residual value - rp_i), thus decreasing the memory usage. The third case is the same, but uses the eight



(a) (b)

Figure 6.5: Deducing a node's synchronization error in the Lightness algorithm: (a) Hops to beacons as an error indication and (b) Residual value as an error indication.

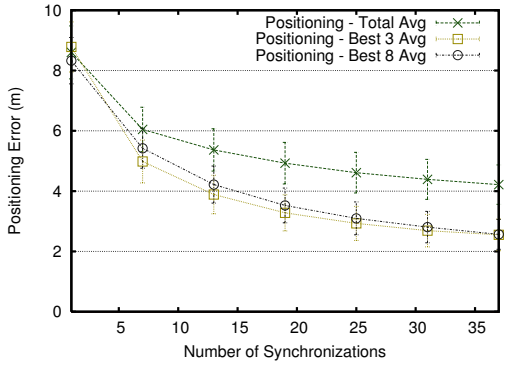


Figure 6.6: Improving the position computation of the Lightness algorithm: Positioning error in the drift computation.

best computed positions. As we can see in Figure 6.6, in some cases the positioning error decreases to about one third of the initial position. Using only the best estimations actually results in better position estimations, as the worse estimations are discarded from the final position computation. Though the difference is not significative at the 95% level, the graph suggests that using the Best 3 Avg strategy is constantly more accurate that the Best 8 Avg one. Being the former also more efficient in terms of memory usage, it seems as the best of the considered options.

6.4 Summary

In this chapter, we have proposed a new and lightweight time-space localization algorithm for wireless sensor networks, which we refer to as the *Lightness* algorithm (Lightweight Time and Space localization). The positioning part of Lightness is similar to the RPE algorithm. The novelty of this solution is the synchronization part of the proposed algorithm, as well as in the fact that two problems are solved together, thus saving resources and resulting in better performance for both problems. In particular, the main idea of Lightness is to use the already

synchronized clocks of the beacon nodes, which are spread throughout the sensor field, to start the synchronization process at the same time. In this solution, we also showed that the extra hardware required for position estimation can be used to better synchronize nodes, and that the extra communication used in synchronization can result in more accurate position estimations.

The proposed algorithm can scale to thousands of nodes, synchronizing their clocks with a precision of nearly 10 μs in any case. In terms of the communication cost, the proposed algorithm is also very scalable resulting in a communication cost of only $O(n)$, being n the number of nodes, which means that each node will send only one packet in order to localize itself in both time and space. The increase in the beacon nodes from 5% to 10% decreases the synchronization error by half, which also improves the positioning (Oliveira et al., 2005a,b; Oliveira, 2005). The GPS inaccuracy is not a problem and the proposed algorithm is less affected by the one hop synchronization inaccuracy. Finally, we show that the positioning error can be decreased by taking advantage of the synchronization drift computation.

One drawback of the proposed solution is that it cannot work in low density networks. In normal or high dense networks, the increase in density results in more neighbors for the nodes and, potentially, more reference points for estimating the node's position and time. On the other hand, in low density networks, the number of settled nodes (nodes that have estimated their positions) and synchronized nodes decreases since the nodes do not have enough reference neighbors to estimate their own positions (and thus they do not become references). When a node does not have at least three neighbors, it will not compute its position and then it will not be able to forward the localization and synchronization recursion, which will make the algorithm stop before all nodes have estimated their time-space information.

Considering these characteristics of the Lightness algorithm, we can say that this proposed algorithm is more applicable for large scale, dense, and outdoor WSNs in which there is the possibility of deploying at least 5% of the nodes equipped with a GPS receiver (to execute the role of beacon nodes). The results obtained in the synchronization part are slightly higher than that of the Synapse algorithm. However, the Lightness algorithm still synchronizes the clock of the nodes in a WSN with a few microseconds of error, which is suitable for performing most tasks that require synchronized clocks such as data fusion, object or target tracking, events ordering, etc.

The positioning part of the Lightness algorithm, which is based on the RPE algorithm, results in medium localization errors of about 50% of the communication range. This localization error can still affect most protocols and applications that depends on localization to work properly (Oliveira et al., 2005b). However, some techniques, such as a directed recursion (Oliveira et al., 2005a), can be implemented to decrease these localization errors and make them somewhat predictable.

Our previously proposed time-space localization algorithm, the Synapse algorithm, is able to work in low density networks. However, both the Synapse and Lightness algorithms require the entire network to be visible to GPS satellites in order to work, since the beacon nodes are randomly deployed in the network. In a number of WSN scenarios this requirement is not

possible. For instance, in underwater WSNs, submerged nodes do not have access to GPS information. Also, in other constrained scenarios, special GPS-equipped beacon nodes could simply be unavailable due to hardware limitations (e.g., cost, size, and energy consumption of all sensor nodes). For these cases, we have designed another time-space localization system that we refer to as the Mobilis algorithm. This algorithm will be the topic of the next chapter.

Chapter 7

A Mobile Beacon Approach for Time-Space Localization in WSNs

7.1 Introduction

In our previous Synapse and Lightness algorithms, we proposed the first two algorithms for time-space localization in WSNs. In this chapter, we take a step further and propose a new approach to solving this problem: using a mobile beacon. A mobile beacon is a node that is aware of its time and position (e.g. equipped with a GPS receiver) and that has the ability to move around the sensor field. This beacon can be a human operator, an unmanned vehicle, an aircraft, or a robot.

A mobile beacon has been successfully applied to solve the positioning problem; however, to the best of our knowledge, the current work is the first to address the use of a mobile beacon in synchronization and time-space localization problems. The use of a mobile beacon allows the algorithms to deal with cases in which it is not possible to deploy GPS-equipped nodes in the network.

The proposed time-space localization algorithm, which we refer to as the Mobilis (Mobile Beacon for Localization and Synchronization) algorithm, can synchronize nodes by using the packet delay measurement (explained in Section 3.2.2) or the packet round-trip time (RTT – explained in Section 3.2.1). In the first case, synchronization can be improved by the extra packets required for location discovery (trilateration), while in the second case, location information can be improved by the extra packets required for synchronization (RTT delay estimation).

In this chapter, we will explain the details of our proposed Mobilis algorithm and evaluate its performance. The remainder of this chapter is organized as follows. Section 7.2 describes our proposed Mobilis algorithm, which is then evaluated in Section 7.3. At the end of this chapter, in Section 7.4, we present our conclusions and the pros and cons of the Mobilis algorithm.

7.2 Mobilis – A Mobile Beacon Approach for Localization in Time and Space

In this section, we propose a new approach for time-space localization: the Mobilis algorithm (Mobile Beacon Localization and Synchronization). As with our previous algorithms, the Mobilis algorithm can also be divided into two parts that work at the same time: the positioning and synchronization parts. The positioning part is based on the Mobile Beacon Localization, an algorithm proposed by Sichitiu and Ramadurai (2004) and discussed in Section 2.4.4. The synchronization part obeys the same principle as the positioning part, but is extended to deal with time estimations. Then, these two parts are combined into a single algorithm. We are proposing two schemes for the Mobilis algorithm. In the first scheme we use the delay measurement technique to synchronize nodes, while in the second scheme we use the round-trip delay time technique.

7.2.1 First Scheme: Using Delay Measurement

In this scheme, shown in Algorithm 11, once the nodes are deployed, the mobile beacon travels constantly moving in a non self-intersecting trajectory through the sensor field broadcasting messages that contain its current coordinates and timestamp (lines 15–20 of Algorithm 11). When an unsynchronized node receives a packet from the mobile beacon, it can estimate the packet travel delay time using the Delay Measurement technique and, based on the timestamp stored in the packet, the node can also estimate its own offset (lines 21–23). Also, when an unknown node receives at least three messages from the mobile beacon, it can estimate its position based on the received coordinates and on the RSSI distance estimates (lines 24–27). Since the nodes will require at least three packets for positioning, synchronization can be improved by computing the average offset of all packets. An advantage of this scheme is that the communication cost of localizing and synchronizing regular nodes is null, since these nodes do not need to send any packets.

7.2.2 Second Scheme: Using RTT

In the second scheme of our proposed algorithm, we will use the RTT technique to estimate the packets' delay. As in the first scheme, depicted in Figure 7.1, the mobile beacon travels through the sensor field broadcasting messages that contain its current coordinates and timestamp. However, in this scheme, the mobile beacon will wait for replies to its advertisement from the regular nodes so the round-trip delay time can be computed. This scheme, shown in Algorithm 12, uses a variation of the RTT technique (depicted in Figure 7.2) and can be divided into four phases:

1. When moving through the sensor field, the mobile beacon stops and sends an *Advertisement Packet* containing its position and timestamp (T_1) — Figure 7.1(a) and lines 15–19 of Algorithm 12.

Algorithm 11 First scheme of the Mobilis algorithm

▷ **Variables:**
1: $positions_i = \emptyset$ {Set of position information.}
2: $times_i = \emptyset$ {Set of received timestamps.}
3: $advTimer_i$ {Timer to send advertisement packets.}
4: $x_i,\ y_i,\ t_i$ {Position and Time information.}

▷ **Functions:**
5: $getGpsPosTime$ {Returns the nodes' GPS info.}
6: $distanceEstimation$ {Pkts' travel distance estimation}
7: $positionComputation$ {Computes the nodes' position}
8: $delayMeasurement$ {Pkts' travel delay estimation}
9: $timeComputation$ {Computes the nodes' offset}

▷ **Input:**
10: $msg_i = nil$.

Action:
11: **if** $n_i \in \mathcal{B}$ **then** {If this node is a beacon node.}
12: StartMoving();
13: Start $advTimer_i$.
14: **end if**

▷ **Input:**
15: $advTimer_i$ timeout.

Action:
16: StopMoving();
17: $(x_i, y_i, t_i) := getGpsPosTime()$;
18: Send $position(x_i, y_i, t_i)$ to all $n_j \in Neig_i$.
19: StartMoving();
20: Restart $advTimer_i$.

▷ **Input:**
21: $msg_i = position(x_k, y_k, t_k)$ such that
$dist_k = distanceEstimation(msg_i)$, and
$delay_k = delayMeasurement(msg_i)$.

Action:
22: $positions_i := positions_i \cup \{(x_k, y_k, dist_k)\}$;
23: $times_i := times_i \cup \{(t_k, delay_k, t_i)\}$;
24: **if** $size(positions_i) >= 3$ **then** {Enough references?}
25: $(x_i, y_i) := positionComputation(positions_i)$;
26: **end if**
27: $t_i = timeComputation(times_i)$;

2. Every unknown node that receives the Advertisement Packet stores the packet arrival time (T_2) using its local clock, waits for a random small time (to avoid collisions), and sends a RTT *Request Packet* containing its local timestamp (T_3) — Figure 7.1(b) and lines 20–22 of Algorithm 12.

3. For every Request Packet received, the mobile beacon stores the packet arrival time (T_4) and then, after a timeout, sends a single *Reply Packet* containing all of the stored times (with corresponding node ID) and its current timestamp (T_5) — Figure 7.1(c) and lines 23–28 of Algorithm 12.

4. Upon receiving a Reply Packet, a regular node stores the arrival time (T_6) so that it

Algorithm 12 Second scheme of the Mobilis algorithm

▷ **Variables:**
1: $positions_i = \emptyset$ {Set of position information.}
2: $times_i = \emptyset$ {Set of received timestamps.}
3: $advTimer_i$ {Timer to send advertisement packets.}
4: $waitTimer_i$ {Timer to wait for request packets.}
5: $x_i,\ y_i,\ t_i$ {Position and Time information.}

▷ **Input:** {Only the Mobile Beacon}
6: $msg_i = nil$.

Action:
7: **if** $n_i \in \mathcal{B}$ **then** {If this node is a beacon node.}
8: StartMoving();
9: Start $advTimer_i$.
10: **end if**

▷ **Input:** {Only the Mobile Beacon}
11: $advTimer_i$ timeout.

Action:
12: StopMoving();
13: $(x_i, y_i, t_i^1) := getGpsPosTime()$;
14: Send $advPkt(x_i, y_i, t_i^1)$ to all $n_j \in Neig_i$.
15: Start $waitTimer_i$.

▷ **Input:** {Regular Nodes}
16: $msg_i = advPkt(x_k, y_k, t_k^1)$ such that $t_k^2 = t_i$ and
$dist_k = distanceEstimation(msg_i)$

Action:
17: $positions_i := positions_i \cup \{(x_k, y_k, dist_k)\}$;
18: Send $rqstPkt(t_i)$ to Mobile Beacon.

▷ **Input:** {Only the Mobile Beacon}
19: $msg_i = rqstPkt(t_k^3)$ such that $t_k^4 = t_i$

Action:
20: $times_i := times_i \cup \{(k, t_k^3, t_k^4)\}$;

▷ **Input:** {Only the Mobile Beacon}
21: $waitTimer_i$ timeout.

Action:
22: Send $replyPkt(times_i, t_i)$ to all $n_j \in Neig_i$.
23: StartMoving();
24: Start $advTimer_i$.

▷ **Input:** {Regular Nodes}
25: $msg_i = replyPkt(times_k, t_k^5)$ such that $t_k^6 = t_i$

Action:
26: $times_i := times_i \cup \{(t_k^1, t_k^2, t_k^3, t_k^4, t_k^5, t_k^6)\}$;
27: **if** $size(positions_i) >= 3$ **then** {Enough references?}
28: $(x_i, y_i) := positionComputation(positions_i)$;
29: **end if**
30: $t_i = timeComputation(times_i)$;

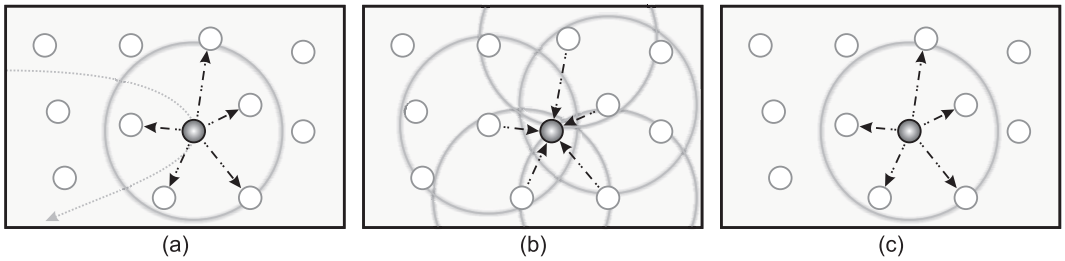


Figure 7.1: Example and phases of the second scheme of the Mobilis Algorithm: (a) beacon node sending an advertisement packet; (b) unsynchronized nodes sending a request packet; and (c) the beacon node sending a reply packet.

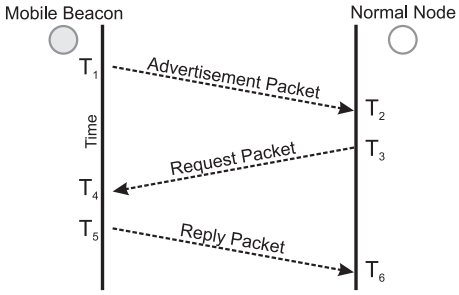


Figure 7.2: Mobilis variation of the RTT technique.

will have the sending and arrival time of all three exchanged packets. It then computes the packet delay (lines 29–34 of Algorithm 12) as follows:

$$d_1 = \frac{(T_4 - T_1) - (T_3 - T_2)}{2}, \tag{7.1}$$

$$d_2 = \frac{(T_6 - T_3) - (T_5 - T_4)}{2}, \tag{7.2}$$

$$d = \frac{d_1 + d_2}{2}. \tag{7.3}$$

As we can see, this algorithm is a slightly different and more accurate version of the RTT technique, since we have the time information of an additional packet. For the positioning component, after receiving (and replying to) three advertisements from the beacon, the nodes will be able to estimate their positions. However, in this case, the nodes will have three different distance estimations for each advertisement (one estimation for the Advertisement Packet, another for the Request, and another for the Reply). The nodes can take the average of these estimations. Since distance estimations are the most common source of errors for the position estimation, this technique will result in a better performance of the positioning part of the Mobilis algorithm. Since the delay measurement technique used in the first scheme of the Mobilis algorithm is implemented in the MAC layer, the version of RTT used in this second scheme is also implemented in the MAC layer (packets are timestamped immediately before being sent and immediately after being received).

Parameter	*Value*
sensor field	$92 \times 92\,\mathrm{m}^2$
number of nodes	256 nodes (disturbed grid)
density	$0.03\,^{\mathrm{nodes}}/_{\mathrm{m}^2}$
communication range	15 m
number of beacons	1 (mobile beacon)
beacon velocity	$20\,^{\mathrm{m}}/_{\mathrm{s}}$
advertisement time	0.5 s
GPS sync. inaccuracy	$1\,\mu s$
one hop sync inaccuracy	$5\,\mu s$ (Maroti et al., 2004; Ping, 2003)
RSSI inaccuracy	20% of communication range

Table 7.1: Default scenario configuration for the simulations performed to evaluate the performance of the Mobilis algorithm.

This scheme has other advantages as well. Since the mobile beacon receives a response from the nodes, the mobile beacon can itself compute the position of the nodes. It is also possible for the mobile beacon to create a map of the network containing the position of all nodes. The presence of a mobile beacon and the knowledge of its position by the nodes can also be used to calibrate the presence and movement sensors in these nodes. Lastly, the mobile beacon can also be used in this second scheme to provide a routing structure for the network. As we can see, with only one response packet being sent from the nodes to the mobile beacon a number of other systems could be improved, which is a positive aspect of this scheme.

7.3 Performance Evaluation

In this section, we will show the results obtained by our performance evaluation of the synchronization part of our current proposed scheme, the Mobilis algorithm.

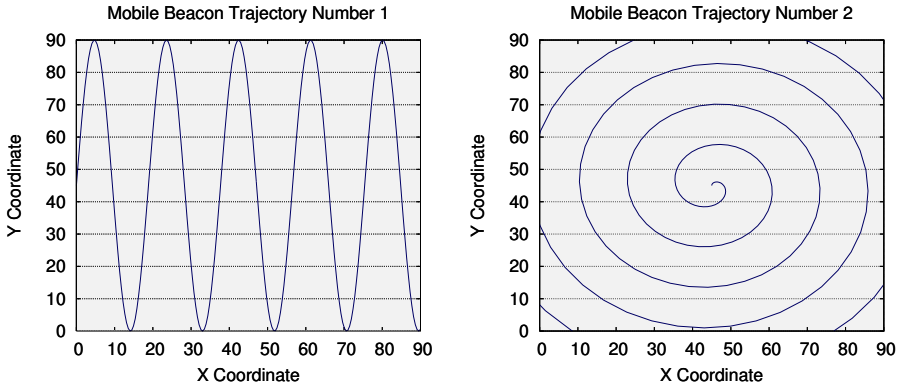
7.3.1 Methodology

In order to evaluate the performance of our proposed protocol, we have carried out an extensive set of simulation experiments using the NS-2 simulator (McCanne and Floyd, 2005). In all results, the curves represent the average values, while the error bars represent the confidence intervals for 95% of confidence from 33 different instances (seeds). As with the other simulations, the current simulation parameters are based on the MicaZ sensor node. The communication range is fixed at 15m for all nodes. To simulate the RSSI inaccuracy, each range sample is disturbed by a Gaussian distribution with the actual distance as the mean and 20% of this distance as the standard deviation. The default parameters for the experiments are presented in Table 7.1.

Regarding the network topology, as in the other scenarios we assume that the node deployment obeys a disturbed grid, that is, a grid in which the location of each node is disturbed by a random zero-mean Gaussian error. Therefore, nodes will tend to occupy the sensor field uniformly, avoiding large concentrations of nodes and without forming a regular grid.

Both schemes of the Mobilis algorithm are evaluated. In the first scheme the delay mea-

surement is used to synchronize clocks, while in the second scheme the RTT technique is used. Also, we evaluate two different trajectories for the mobile beacon (Figure 7.3): a simple sinusoidal path (Figure 7.3(a)); and a spiral path (Figure 7.3(b)). In our simulations of both trajectories, the mobile beacon is traveling with a velocity of 20 $^{m}/_{s}$. In the second scheme, as shown in Algorithm 12, the mobile beacon stops and wait for request packets from nearby nodes. Thus, in both schemes, the velocity of the beacon does not interfere on the final results.



(a) (b)

Figure 7.3: Evaluated trajectories for mobile beacon: (a) a simple sinusoidal path; and (b) a spiral path.

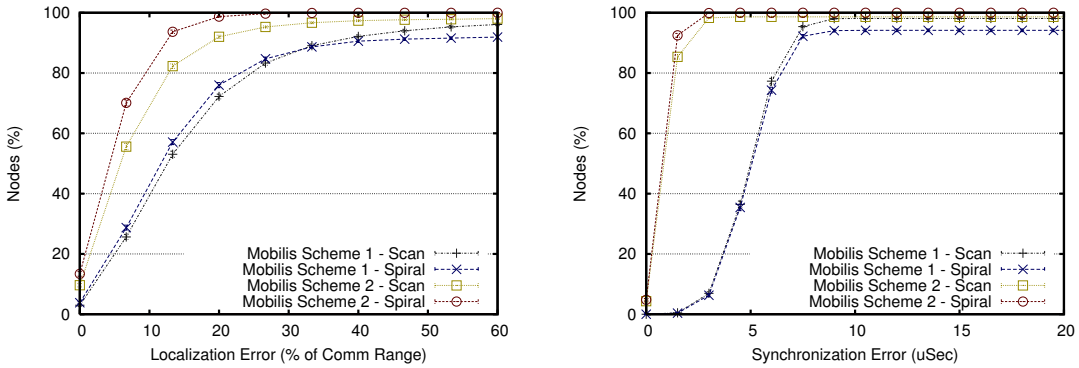
7.3.2 Error Analysis

7.3.2.1 Positioning Error

The distribution of position errors among the sensor nodes is depicted in Figure 7.4(a). As mentioned before, the cumulative error identifies the percentage of nodes (y-axis) with a positioning error smaller than a parametrized value (x-axis). A steep curve means that the majority of nodes has a small error. We can see that the best results are achieved by the second scheme of the Mobilis algorithm since this scheme has a more reliable distance estimation (average from all packets required by RTT). This graph also shows that spiral trajectories result in better positioning, since these trajectories are less rectilinear.

7.3.2.2 Synchronization Error

The distribution of synchronization errors among the sensor nodes is depicted in Figure 7.4(b). In this case, the cumulative error identifies the percentage of nodes (y-axis) with a synchronization error smaller than a parametrized value (x-axis). Again, a steep curve means that the majority of nodes has a small error. We can see that the best results are achieved by the second Mobilis scheme since it uses the RTT technique implemented in the MAC layer, which is more accurate than the delay measurement. In this second scheme, 90% of the nodes have a synchronization error smaller than 2 μs and almost all nodes have a synchronization error smaller than 4 μs.



(a) (b)

Figure 7.4: Error analysis of the Mobilis algorithm: (a) Positioning cumulative error; (b) Synchronization cumulative error.

7.3.3 Impact of GPS Inaccuracy

7.3.3.1 Positioning Inaccuracy

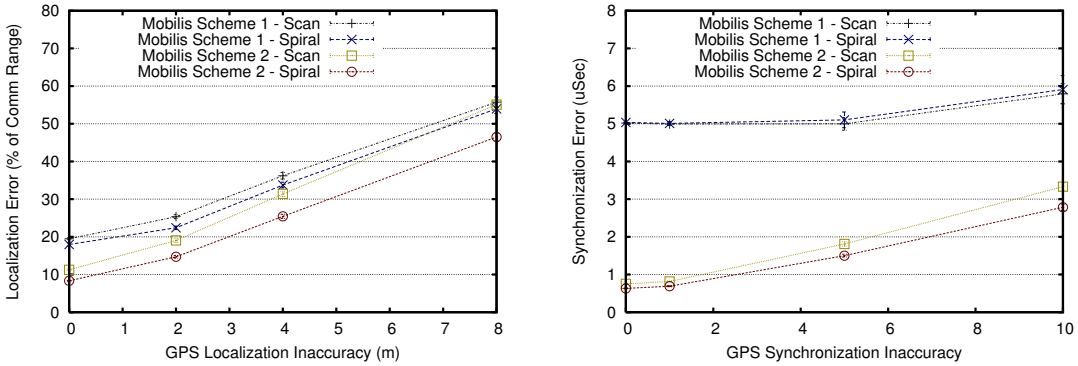
GPS receivers do not provide perfect positioning. In this section, we evaluate the proposed Mobilis algorithm by introducing errors in the computed position of the mobile beacon. As we can see in Figure 7.5(a), the Mobilis algorithm can be affected by this GPS inaccuracy. However, unlike the results in a number of other localization systems (Albowicz et al., 2001; Oliveira et al., 2005a), these errors do not propagate to the other nodes since all sensor nodes compute their positions based only on the information provided by the mobile beacon.

7.3.3.2 Synchronization Inaccuracy

Unlike other synchronization algorithms, the Mobilis algorithm can be influenced by the GPS inaccuracy. Commercial GPS receivers usually have an accuracy of nearly 200 ns (Elson and Estrin, 2001) or 1 μs in the case of the GPS module used by the MicaZ nodes. Thus, to understand how GPS affects the final synchronization error of the proposed algorithm, we have increased the GPS synchronization inaccuracy from 0 to 10 μs. The results, depicted in Figure 7.5(b), show that this influence is not critical. In the worst case, when we have a GPS inaccuracy of 10 μs, the synchronization error is smaller than 3 μs in the second scheme of Mobilis. We do not evaluate the impact of the GPS Positioning inaccuracy in the positioning part of Mobilis because nearby GPS computed positions have correlated errors. Thus, the positioning error of the nodes always increases in the same way as the GPS positioning error.

7.3.4 Impact of RSSI Inaccuracy

Distance estimations using RSSI measurements are not accurate. Depending on the environment, such an inaccuracy may lead to significative errors in the estimated positions. We evaluate this impact by adding some noise to the real distances. This noise follows a Gaussian distribution with the actual distance as the mean and a percentage of this distance as



(a) (b)

Figure 7.5: Impact of GPS inaccuracy on the resulted (a) localization errors and (b) synchronization errors.

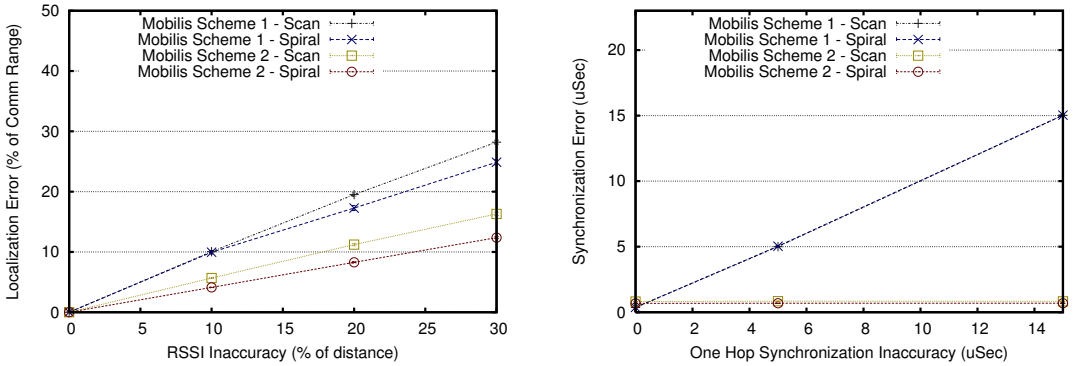
the standard deviation. Figure 7.6(a) compares the increase of the standard deviation of the Gaussian distribution (used to simulate the noise) from 0 to 30% of the actual distance for both Mobilis algorithms. This graph shows that the second scheme of the Mobilis algorithm, i.e. spiral, is more robust to RSSI inaccuracies, resulting in errors of almost 12% of the communication range when the RSSI inaccuracy is about 30%. In this case, we can see clearly how the positioning part takes advantage of the synchronization part to improve its performance, which shows the importance of solving both problems jointly.

7.3.5 Impact of One Hop Synchronization Inaccuracy

The one hop synchronization accuracy tells us how well a pair of nodes can synchronize to each other using the delay measurement technique. In this technique, there is a number of non-deterministic delays that is variable and cannot be computed in the delay measurement. This increases the synchronization error of this technique, resulting in a precision of about 2-4 μs in Maroti et al. (2004) and 2-32 μs in Ping (2003). To evaluate the effects of this one hop synchronization inaccuracy, we have increased this inaccuracy from 0 to 15 μs. The results, depicted in Figure 7.6(b), show that these non-deterministic delays can affect the synchronization of the first scheme of Mobilis. Since the RTT technique does not have this problem, the second scheme of Mobilis is not affected.

7.4 Summary

In this chapter, we designed and evaluated the performance of a new solution for the time-space localization problem: the *Mobilis* algorithm (Mobile Beacon for Localization and Synchronization). By using a mobile beacon, all sensor nodes were able to localize themselves both in time and space. We proposed two schemes for the Mobilis algorithm. In the first scheme, only the beacon node sends packets and all regular nodes are able to synchronize and compute their positions with a zero communication cost algorithm. In the second scheme, a



(a) (b)

Figure 7.6: Impact of real world inaccuracies on the Mobilis algorithm: (a) Impact of the RSSI inaccuracy on the localization part; and (b) Impact of the one hop synchronization inaccuracy on the synchronization part.

modified version of the RTT technique is used to estimate the packets' delay and synchronize the nodes. In this last case, we also take advantage of the additional packet exchange to improve the performance of the positioning part. In the best option, the proposed algorithm shows the importance of combining both positioning and synchronization into a single unified problem of localization in time and space. By doing so, the proposed algorithm manages to improve synchronization in the first scheme and localization in the second scheme, in contrast to the scenarios in which these problems are solved separately. In both cases, communication and processing resources can be reduced, thus saving energy and network resources.

In terms of the applicability of our Mobilis algorithm, we can say that the proposed algorithm is more suitable for outdoor or underwater WSNs where there is the possibility of deploying a mobile beacon node. The proposed algorithm does not have any scalability nor density issues in terms of the obtained results and communication overhead. Both synchronization and localization errors are the lowest in comparison to the other proposed solutions. Our performance evaluation shows an average synchronization error of only 2-4 μs and also a localization error of about 10% of the node's communication range, which makes the Mobilis algorithm suitable for most protocols and applications that require timing and/or positioning information to function properly.

The main drawback of our proposed Mobilis algorithm is that it requires the use of a beacon node capable of moving throughout the network. Also, depending on the velocity of the mobile beacon some nodes will have to wait for a long time before they become able to localize themselves. However, aside from these disadvantages, the use of mobile beacons is still considered the only solution available for some scenarios such as underwater WSNs (Chandrasekhar et al., 2006). Thus, the proposed solution can be used to localize nodes in time and space in a number of scenarios in which the current proposed solutions are unable to work.

Chapter 8

Conclusions

8.1 Final Remarks and Summary of Contributions

Wireless Sensor Networks are basically guided by events. An event by itself can be defined as being composed of a causal criterion and a spatiotemporal criterion. The first criterion specifies the type of event, while the second specifies the location in time and space where the event occurred. A sensor node is able to identify the first criterion easily by using its own sensing devices. The spatiotemporal criterion, on the other hand, can only be identified when the sensor nodes of a WSN have synchronized clocks and are able to determine their physical location. Time and space information is also required by a number of algorithms and protocols in WSNs such as data fusion, object tracking, energy maps, and density control. Thus, synchronization and positioning for WSN-based applications are challenging problems that need to be addressed.

In this thesis, we propose that synchronization and positioning in WSNs are actually two parts of the same problem: localizing the nodes in a network in both time and space. Neither location without time nor time without location is a complete set of information in these networks. The similarities between these problems also suggest that they can and should be addressed as a single problem. From our perspective, time can be seen as another dimension of space. In consequence, we need to solve a 4D positioning problem. By treating these problems in this way, network resources can be conserved and both problems can show better performance.

The wide range of applicability of WSNs as well as the different scenarios in which their deployment is envisioned result in a very well-known characteristic of these networks: there is no single solution for a problem in WSNs. As a result, different solutions are proposed for the same problem in WSNs. Each of these solutions is designed to target a specific scenario such as dense or sparse networks; small, medium, or large scale networks; static, low or highly mobile networks; underwater, underground, indoor, or outdoor networks; etc. Thus, in WSNs, the choice of which protocols and algorithms to use to provide the network infrastructure (e.g., access media, routing, synchronization, and localization) will depend on both the scenario in which the nodes will be deployed and the envisioned network applications.

Due to this inability to produce a single solution to a problem in WSNs, in this thesis we propose three different solutions for the localization in time and space problem: the Synapse, Lightness, and Mobilis algorithms.

- *The Synapse Algorithm:* The Synapse algorithm (SYNchronization And Positioning for SEnsor networks – shown in Chapter 5) is a time-space localization algorithm designed for use in small to medium sized WSNs. The Synapse algorithm is based on the APS (Ad Hoc Positioning System) localization system (Niculescu and Nath, 2001). It obeys the same principles as the APS, resulting in the same localization features and communication overhead, but with the additional capability of locating nodes in time (i.e., synchronization). The main idea of this algorithm relies on the fact that beacon nodes (GPS-equipped nodes that help to locate other nodes) already have synchronized clocks, since they get their timing information from the GPS. Synchronized clocks located in different parts of the network are able to estimate the times at which a packet leaves a beacon node and arrives at another beacon node. Based on this information, we can compute the average time of a hop. A regular node synchronizes its clock by considering the average time of a hop and the number of hops between itself and each beacon node. Our simulation results show that the Synapse algorithm can synchronize the nodes' clocks with a precision of a few microseconds.

- *The Lightness Algorithm:* The Lightness algorithm (Lightweight Time and Space localization – shown in Chapter 6) has some key aspects. First, the positioning component is based on Recursive Position Estimation (RPE) (Albowicz et al., 2001), a well-known and scalable positioning system for sensor networks that requires only 5% of the nodes to be GPS-enabled beacon nodes. Second, this algorithm assumes that beacon nodes already have synchronized clocks, since they can get their timing information from GPS. Third, both timing and positioning information are propagated in the network with an $O(n)$ communication cost. At last, we use the packet delay measurement (Maroti et al., 2004; Ping, 2003) to estimate the delay of a packet and synchronize neighbor nodes using only one broadcast. Simulation results show that the Lightness algorithm can scale to thousands of nodes, while keeping a synchronization error of a few microseconds and decreasing the positioning error (compared to the original RPE algorithm).

- *The Mobilis Algorithm:* The Mobilis algorithm (Mobile Beacon for Localization and Synchronization – shown in Chapter 7) is a step further in the propose of a new approach for solving the time-space localization problem: the use of a mobile beacon. A mobile beacon is a node that is aware of its time and position (e.g. equipped with a GPS receiver) and that has the ability to move around the sensor field. This beacon can be a human operator, an unmanned vehicle, an aircraft, or a robot. A mobile beacon has been successfully applied to solve the positioning problem; however, to the best of our knowledge, this is the first algorithm to address the use of a mobile beacon in synchronization and time-space localization problems. The proposed time-space localization

Algorithm	Number of beacons	Provides Positioning?	Provides Synchronization?	Absolute or Relative?	Infrastructured?	Multihop?	Comm. Complexity
RPE	5% nodes	yes	no	Absolute	no	yes	$O(n)$
DPE	4	yes	no	Both	no	yes	$O(n)$
APS	≥ 3	yes	no	Absolute	no	yes	$O(nb)$
MBL	1 mobile	yes	no	Absolute	no	yes	*none*
FTSP	1	no	yes	Relative	no	yes	$O(l+n)$
DMTS	1	no	yes	Relative	no	yes	$O(l+n)$
GPS	24 sat.	yes		Absolute	yes	no	—
Synapse	≥ 3	yes		Absolute	no	yes	$O(nb)$
Lightness	5% nodes	yes		Absolute	no	yes	$O(n)$
Mobilis	1 mobile	yes		Absolute	no	yes	*none* or $O(n)$

Table 8.1: Comparison of algorithms. In terms of communication complexity, $O(l)$ refers to the complexity of the leader election and $O(b)$ is the number of beacon nodes.

algorithm, which we refer to as the Mobilis algorithm, can synchronize nodes by using the packet delay measurement (Maroti et al., 2004; Ping, 2003) or the packet round-trip time (RTT). In the first case, synchronization can be improved by the extra packets required for location discovery, while in the second, localization can be improved by the extra packets required for synchronization.

Depending on the WSN scenario and applications envisioned, each of our proposed algorithms has its pros and cons. In Table 8.1, we compare some of the main characteristics of our proposed algorithms with respect to other known work found in the literature.

Finally, our proposed solutions confirm a new trend in WSNs: integrated solutions. In integrated solutions, multiple problems are solved jointly to provide cheaper and better solutions. Recent work confirms this new trend of algorithms in WSNs. For instance, Nakamura et al. (2005b) proposed a solution in which the routing problem is solved with a data fusion system in an on demand role assignment algorithm. Boukerche et al. (2007b) proposed an algorithm that solves both localization and routing problems at the same time. In all of these cases, the proposed integrated algorithms were able to save resources in terms of communication cost, memory consumption, and processor usage when compared to the solutions in which the problems are solved separately.

8.2 Directions for Future Research

The results obtained in this thesis are very promising. The solutions proposed in the current work usually take advantage of both the additional hardware required for localization to improve the performance of synchronization and the additional communication required for synchronization in order to improve the performance of localization. In the end, we have a time-space localization system with improved results for both localization and synchronization with a communication cost lower or, in the worst case, equal to the case where localization and synchronization are solved by different and independent solutions. However, during the

progress of this thesis work, we also noticed some issues that can be carried out in future work and projects.

In the *Synapse algorithm*, future studies will include its implementation in a testbed composed of MicaZ nodes. We also plan to use the RSSI (Received Signal Strength Indicator – available in the nodes and also used by the localization systems) to estimate the propagation time of the packets in order to decrease even further the non-deterministic delays and, consequently, the synchronization errors as well. This last experiment could be also carried out using the TDoA technique as explained in Section 4.4.1. In this last case, the available information of both distance (computed throw TDoA) and signal strength could also be used to detect obstacles between two nodes, which could be useful for other applications such as robot guidance.

In the *Lightness algorithm*, as mentioned, the positioning part of the proposed system does not work well in sparse networks since the positioning recursion of such a system can stop or not start at all, both of which will stop the synchronization. Another possible solution is to make the synchronization recursion continue even if the localization recursion stops due to lack of reference nodes. In this last case, synchronization would not be affected. In the synchronization part of the Lightness algorithm, we can investigate using better one hop synchronization techniques such as the RTT. In this case, the increased communication cost of the synchronization could be used to reduce the positioning error. Finally, a study should be done on how to remove the GPS dependency of the Lightness algorithm, since it limits the algorithm's usability to outdoor scenarios.

In the *Mobilis algorithm*, the trajectory of the mobile beacon needs to be planned carefully. If there are parts of the network that are not covered by the mobile beacon's trajectory, the nodes in that area will not be able to compute their positions and synchronize their clocks. One solution to this problem is to combine the Mobilis algorithm with another kind of algorithm such as the recursive or DV-Hop algorithms used in our previous solutions. In this case, any nodes not covered by the trajectory of the mobile beacon could estimate their positions based on the positions and times of the nodes that have been able to estimate their positions and synchronize their clocks using the packets sent by the mobile beacon. Another solution is to make the nodes keep track of the routes to the current location of the mobile beacon and make the unknown and unsynchronized nodes send a trajectory request to the mobile beacon after some time. In this case, the mobile beacon could be headed toward the first intermediate node with a known position in the trajectory request. Finally, the trajectory of the beacon node could be dynamically traced as a function of the network information such as nodes' density and energy maps.

Appendix A

Glossary of Terms

AHLoS *Ad-hoc Localization System*
AoA *Angle of Arrival*
APS *Ad hoc Positioning System*

DMTS *Delay Measurement Time Synchronization*
DoA *Direction of Arrival*
DPE *Directed Position Estimation*

FTSP *Flooding Time Synchronization Protocol*

GAF *Geographical Adaptive Fidelity*
GEAR *Geographical and Energy Aware Routing*
GPS *Global Positioning System*

MAC *Media Access Control*
MBL *Mobile Beacon Localization*

NS *Network Simulator*
NTP *Network Time Protocol*

RBS *Reference-Broadcast Synchronization*
RPE *Recursive Position Estimation*
RSSI *Received Signal Strength Indicator*
RTT *Round Trip Time*

TDoA *Time Difference of Arrival*
ToA *Time of Arrival*
ToF *Time of Flight*
TPSN *Timing-Sync Protocol for Sensor Networks*

UTC *Coordinated Universal Time*

WSN *Wireless Sensor Network*

Bibliography

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cyirci, E. (2002). Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422.

Albowicz, J., Chen, A., and Zhang, L. (2001). Recursive position estimation in sensor networks. In *ICNP '01: Proceedings of the 9th International Conference on Network Protocols*, pages 35–41, Washington, DC, USA. IEEE Computer Society.

Arampatzis, T., Lygeros, J., and Manesis, S. (2005). A survey of applications of wireless sensors and wireless sensor networks. In *Med'05: Proceedings of the 13th Mediterranean Control Conference*, pages 719–724, Limassol, Cyprus. IEEE.

Bachrach, J. and Taylor, C. (2005). Localization in sensor networks. In Stojmenovic, I., editor, *Handbook of Sensor Networks: Algorithms and Architectures*. John Wiley & Sons.

Bahl, P. and Padmanabhan, V. N. (2000). RADAR: An in-building RF-based user location and tracking system. In *INFOCOM 2000. Proceedings of the 9th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 775–784, Tel Aviv, Israel. IEEE.

Boukerche, A. (2005). *Handbook of Algorithms for Wireless Networking and Mobile Computing*. Chapman & Hall/CRC.

Boukerche, A., Oliveira, H. A. B. F., Nakamura, E. F., and Loureiro, A. A. (2007a). Localization systems for wireless sensor networks. *IEEE Wireless Communications - Special Issue on Wireless Sensor Networks*, 14(6):6–12.

Boukerche, A., Oliveira, H. A. B. F., Nakamura, E. F., and Loureiro, A. A. (2007b). Towards an integrated solution for node localization and data routing in sensor networks. In *ISCC'07: Proceedings of the 12th IEEE Symposium on Computers and Communications*, pages 449–454, Aveiro, Portugal.

Boukerche, A., Oliveira, H. A. B. F., Nakamura, E. F., and Loureiro, A. A. (2007c). A Voronoi approach for scalable and robust DV-hop localization system for sensor networks. In *ICCCN'07: Proceedings of the 16th International Conference on Computer Communications and Networks*, pages 497–502, Turtle Bay Resort, Honolulu, Hawaii, USA.

Bulusu, N., Heidemann, J., and Estrin, D. (2000). GPS-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34.

Capkun, S., Hamdi, M., and Hubaux, J.-P. (2002). GPS-free positioning in mobile ad hoc networks. *Cluster Computing*, 5(2):157–167.

Chandrasekhar, V., Seah, W. K., Choo, Y. S., and Ee, H. V. (2006). Localization in underwater sensor networks: Survey and challenges. In *WUWNet '06: Proceedings of the 1st ACM international workshop on Underwater networks*, pages 33–40, New York, NY, USA. ACM Press.

Cheng, K.-Y., Tam, V., and Lui, K.-S. (2005). Improving APS with anchor selection in anisotropic sensor networks. In *ICAS-ICNS'05: Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*, page 49, Washington, DC, USA. IEEE Computer Society.

Crossbow (2004). Mica2 - wireless measurement system. [Online] Available: http://www.xbow.com.

Davidson, D. (1980). *Essays on Actions and Events*. Clarendon, Oxford.

Doherty, L., Pister, K. S., and Ghaoui, L. E. (2001). Convex position estimation in wireless sensor networks. *INFOCOM'01: Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies.*, 3:1655–1663.

Elliott, C. and Heile, B. (2000). Self-organizing, self-healing wireless networks. In *PWC'00: Proceedings of IEEE International Conferenceon Personal Wireless Communications*, pages 355–362.

Elson, J. (2003). *Time Synchronization in Wireless Sensor Networks*. PhD thesis, Department Computer Sciences, University of California.

Elson, J. and Estrin, D. (2001). Time synchronization for wireless sensor networks. In *IPDPS'01: Proceedings of the 15th International Parallel & Distributed Processing Symposium*, pages 1965–1970, Washington, DC, USA. IEEE Computer Society.

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

Elson, J. and Romer, K. (2003). Wireless sensor networks: a new regime for time synchronization. *SIGCOMM Computer Communications Review*, 33(1):149–154.

Estrin, D., Culler, D., Pister, K., and Sukhatme, G. (2002). Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69.

Estrin, D., Girod, L., Pottie, G., and Srivastava, M. (2001). Instrumenting the world with wireless sensor networks. In *ICASSP'01: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 2033–2036, Salt Lake City, Utah.

Fellbaum, C. (1998). *Wordnet: An Electronic Lexical Database*. Bradford Books, 01 edition.

Flexner, S. B., Staff, R. H., Hauck, L. C., and House, R. (1998). *Random House Unabridged Dictionary*. Random House Value Pub, 2nd edition.

Friis, H. T. (1946). A note on a simple transmission formula. In *Proc. IRE*, page 34.

Ganeriwal, S., Kumar, R., and Srivastava, M. B. (2003). Timing-sync protocol for sensor networks. In *SenSys'03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 138–149, New York, NY, USA. ACM Press.

Gibson, J. D., editor (1999). *The Mobile Communication Handbook*. IEEE Press.

He, T., Huang, C., Blum, B. M., Stankovic, J. A., and Abdelzaher, T. (2003). Range-free localization schemes for large scale sensor networks. In *MobiCom'03: Proceedings of the 9th International Conference on Mobile Computing and Networking*, pages 81–95, New York, NY, USA. ACM Press.

Heidemann, J., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D., and Ganesan, D. (2001). Building efficient wireless sensor networks with low-level naming. In *SOSP'01: Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 146–159, Banff, Alberta, Canada. ACM Press.

Hofmann-Wellenho, B., Lichtenegger, H., and J.Collins (1997). *Global Positioning System: Theory and Practice*. Springer-Verlag, 4th edition.

Ilyas, M. and Mahgoub, I. (2004). *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, chapter 20. CRC Press LLC.

Ji, X. and Zha, H. (2004). Sensor positioning in wireless ad-hoc sensor networks using multidimensional scaling. *InfoCom'04: Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, 4:2652–2661.

Jones, C. E., Sivalingam, K. M., Agrawal, P., and Chen, J. C. (2001). A survey of energy efficient network protocols for wireless networks. *Wireless Networks*, 7(4):343–358.

Kaplan, E. D. (1996). *Understanding GPS: Principles and Applications*. Artech House.

Kumar, S., Alaettinglu, C., and Estrin, D. (2000). Scalable object-tracking through unattended techniques (scout). In *ICNP'00: Proceedings of the 2000 International Conference on Network Protocols*, page 253, Washington, DC, USA. IEEE Computer Society.

Langendoen, K. and Reijers, N. (2005). Distributed localization algorithms. In Zurawski, R., editor, *Embedded Systems Handbook*. CRC press.

Loureiro, A. A. F., Nogueira, J. M. S., Ruiz, L. B., Mini, R. A. F., Nakamura, E. F., and Figueiredo, M. S. (2003). Redes de sensores sem fio. *SBRC'03: Proceedings of the 21st Brazilian Symposium on Computer Networks*, pages 179–226. Belo Horizonte, MG, Brasil.

Machado, M. D. V., Goussevskaia, O., de Freitas Mini, R. A., Loureiro, A. A. F., Mateus, G. R., and Nogueira, J. M. S. (2005). Data dissemination using the energy map. In *WONS'05: Proceedings of the 2nd Conference on Wireless on Demand Network Systems and Services*, pages 139–148, St. Moritz, Switzerland.

Maroti, M., Kusy, B., Simon, G., and Ledeczi, A. (2004). The flooding time synchronization protocol. In *SenSys'04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 39–49, Baltimore, MD, USA. ACM Press.

McCanne, S. and Floyd, S. (2005). NS network simulator. [Online] Available: http://www.isi.edu/nsnam/ns/.

Mills, D. L. (2006). Network time protocol version 4 reference and implementation guide. Technical Report 06-06-1, Department of Electrical and Computer Engineering, University of Delaware.

Mini, R. A. F., Loureiro, A. A. F., and Nath, B. (2004). The distinctive design characteristic of a wireless sensor network: the energy map. *Computer Communications*, 27(10):935–945.

Nakamura, E. F., Figueiredo, C. M., and Loureiro, A. A. (2005a). Information fusion algorithms for wireless sensor networks. In Boukerche, A., editor, *Algorithms and Protocols for Wireless and Mobile Networks*, pages 841–864. Chapman & Hall/CRC.

Nakamura, E. F., Figueiredo, C. M., and Loureiro, A. A. (2005b). Information fusion for data dissemination in self-organizing wireless sensor networks. In *ICN'05: Proceedings of the 4th International Conference on Networking*, volume 3420 of *Lecture Notes in Computer Science*, pages 585–593, Reunion Island, France. Springer-Verlag.

Nakamura, E. F., Loureiro, A. A. F., and Frery, A. C. (2007). Information fusion for wireless sensor networks: Methods, models, and classifications. *ACM Computing Surveys*, 39(3):9/1–9/55.

Navas, J. C. and Imielinski, T. (1997). Geocast – geographic addressing and routing. In *MobiCom'97: Proceedings of the 3rd Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 66–76, New York, NY, USA. ACM.

Niculescu, D. and Nath, B. (2001). Ad hoc positioning system (APS). *GlobeCom'01: Proceedings of the Global Telecommunications Conference*, 5:2926–2931.

Niculescu, D. and Nath, B. (2003). Ad hoc positioning system (APS) using AoA. *InfoCom'03: Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, 3:1734–1743.

Niculescu, D. and Nath, B. (2004). Error characteristics of ad hoc positioning systems (APS). In *MobiHoc'04: 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 20–30, New York, NY, USA. ACM Press.

Nikoletseas, S. (2006). Algorithms for wireless sensor networks: Design, analysis and experimental evaluation. In *WEA'06: Proceedings of the 5th International Workshop in Experimental Algorithms*, pages 1–12, Cala Galdana, Menorca, Spain.

Oliveira, H. A. B. F. (2005). Um algoritmo recursivo de localização para redes de sensores sem fio. Master's thesis, Department of Computer Science, Federal University of Minas Gerais.

Oliveira, H. A. B. F., Nakamura, E. F., Loureiro, A. A., and Boukerche, A. (2005a). Directed position estimation: A recursive localization approach for wireless sensor networks. In *IC3N'05: Proceedings of the 14th IEEE International Conference on Computer Communications and Networks*, pages 557–562, San Diego, USA.

Oliveira, H. A. B. F., Nakamura, E. F., Loureiro, A. A., and Boukerche, A. (2005b). Error analysis of localization systems in sensor networks. In *GIS'05: 13th ACM International Symposium on Geographic Information Systems*, pages 71–78, Bremen, Germany.

Pathirana, P., Bulusu, N., Jha, S., and Savkin, A. (2005). Node localization using mobile robots in delay-tolerant sensor networks. *IEEE Trans. on Mobile Comput.*, 4(4).

Ping, S. (2003). Delay measurement time synchronization for wireless sensor networks. Technical Report IRB-TR-03-013, Intel Research.

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

Priyantha, N. B., Chakraborty, A., and Balakrishnan, H. (2000). The cricket location-support system. In *MOBICOM'00: Proceedings of the 6th ACM Conference on Mobile Computing and Networking*, pages 32–43, Boston, MA, USA.

Priyantha, N. B., Miu, A. K., Balakrishnan, H., and Teller, S. (2001). The cricket compass for context-aware mobile applications. In *MobiCom'01: Proceedings of the 7th ACM Annual International Conference on Mobile Computing and Networking*, pages 1–14, Rome, Italy. ACM.

Ramadurai, V. and Sichitiu, M. L. (2003). Localization in wireless sensor networks: A probabilistic approach. In *ICWN'03: Proceedings of the International Conference on Wireless Networks*, pages 275–281, Las Vegas, NV, USA.

Rappaport, T. (1996). *Wireless Communications, Principles and Pratice.* Prentice Hall.

Romer, K. (2001). Time synchronization in ad hoc networks. In *MobiHoc'01: Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing*, pages 173–182, New York, NY, USA. ACM Press.

Romer, K. (2003). Temporal message ordering in wireless sensor networks. *MED-HOC Net'03: Proceedings of the 2nd Mediterranean Workshop on Ad-Hoc Networks*, pages 131–142.

Romer, K. (2005). *Time Synchronization and Localization in Sensor Networks.* PhD thesis, ETH Zurich, Zurich, Switzerland.

Romer, K., Blum, P., and Meier, L. (2005). Time synchronization and calibration in wireless sensor networks. In Stojmenovic, I., editor, *Handbook of Sensor Networks: Algorithms and Architectures*, pages 199–237. John Wiley & Sons.

Romer, K. and Mattern, F. (2005). Towards a unified view on space and time in sensor networks. *Computer Communications*, 28(13):1484–1497.

Savarese, C., Rabaey, J. M., and Langendoen, K. (2002). Robust positioning algorithms for distributed ad-hoc wireless sensor networks. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 317–327, Berkeley, CA, USA. USENIX Association.

Savvides, A., Han, C.-C., and Strivastava, M. B. (2001). Dynamic fine-grained localization in ad-hoc networks of sensors. In *MobiCom'01: Proceedings of the 7th ACM/IEEE International Conference on Mobile Computing and Networking*, pages 166–179, Rome, Italy. ACM.

Sekhar, A., Manoj, B. S., and Murthy, C. S. R. (2005). A novel solution for time synchronization in wireless ad hoc and sensor networks. In *HiPC'05: Proceedings of the 12th International Conference on High Performance Computing*, pages 333–342, Goa, India.

Shang, Y. and Ruml, W. (2004). Improved MDS-based localization. *InfoCom'04: Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, 4:2640–2651.

Shang, Y., Ruml, W., Zhang, Y., and Fromherz, M. P. (2003). Localization from mere connectivity. In *MobiHoc'03: Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*, pages 201–212, Annapolis, Maryland, USA. ACM.

Shin, K.-Y., Lee, K., Kim, H., Mah, P. S., Park, S., Lim, C. D., and Kim, H.-N. (2006). A flexible, high-precise time synchronization for multi-hop sensor networks. In *ISORC'06: Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pages 169–173, Washington, DC, USA. IEEE Computer Society.

Sichitiu, M. and Veerarittiphan, C. (2003). Simple, accurate time synchronization for wireless sensor networks. *WCNC'03: Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 1266–1273.

Sichitiu, M. L. and Ramadurai, V. (2004). Localization of wireless sensor networks with a mobile beacon. In *MASS'04: Proceedings of the 1st IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pages 174–183, Florida, USA.

Simic, S. and Sastry, S. (2002). Distributed localization in wireless ad hoc networks. Technical Report UCB/ERL M02/26, UC Berkeley.

Sivrikaya, F. and Yener, B. (2004). Time synchronization in sensor networks: a survey. *IEEE Network*, 18(4):45–50.

Ssu, K.-F., Ou, C.-H., and Jiau, H. (2005). Localization with mobile anchor points in wireless sensor networks. *Vehicular Technology, IEEE Transactions on*, 54(3):1187–1197.

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

van Greunen, J. and Rabaey, J. (2003). Lightweight time synchronization for sensor networks. In *WSNA '03: Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*, pages 11–19, New York, NY, USA. ACM Press.

Whitehouse, K. (2002). The design of Calamari: an ad-hoc localization system for sensor networks. Master's thesis, University of California at Berkeley.

Whitehouse, K. and Culler, D. (2002). Calibration as parameter estimation in sensor networks. In *WSNA'02: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 59–67, New York, NY, USA. ACM Press.

Xu, Y., Heidemann, J. S., and Estrin, D. (2001). Geography-informed energy conservation for ad hoc routing. In *MobiCom'01: Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pages 70–84, Rome, Italy. ACM.