Middleware para Redes de Sensores Sem-Fio: Projeto, Implementação e Avaliação de Consumo de Energia

Germano Guimarães[1], Eduardo Souto[1,3], Mardoqueu Vieira[2,4], Glauco Vasconcelos[2], Nelson Rosa[2], Carlos Ferraz[2], Judith Kelner[1]

[1]Grupo de Pesquisas em Redes e Telecomunicações do
[2]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Caixa Postal 50.740-540 – Recife – PE – Brasil

[3]Universidade Federal do Amazonas (UFAM)
Manaus – AM – Brasil

[4]Fundação Des. Paulo Feitoza (FPF)
Manaus – AM – Brasil

{ gfg, ejps, msv, gpv, nsr, cagf, jk }@cin.ufpe.br

Abstract. *The development of middleware for wireless sensor networks (WSNs) places new challenges to middleware developers due to low availability of resources (e.g., memory, processor, bandwidth and power) of sensor nodes. This paper presents the steps of designing, implementing and evaluating a middleware for Wireless Sensor Network (WSNs). The proposed middleware incorporates characteristics of traditional message-oriented middleware systems and an aggregation service that has been specially designed to save energy. In order to illustrate how these features are effective, we carry out an evaluation of the middleware regarding energy consumption.*

Resumo. *O desenvolvimento de middleware para Redes de Sensores sem Fio (RSSF) introduz novos desafios aos desenvolvedores devido à escassez e variabilidade de recursos (e.g., memória, processamento, largura de banda e bateria) nos sensores. Este artigo apresenta os principais pontos relacionados ao projeto, a implementação e a avaliação do consumo de energia em middleware para RSSF. Fundamental no projeto foram as implementações de um serviço de comunicação orientado a mensagem e um serviço de agregação. Na implementação, adotou-se um sistema operacional baseado em componentes e na avaliação do consumo de energia, uma ferramenta de avaliação de consumo de bateria.*

1 Introdução

A contínua miniaturização dos componentes de hardware e a evolução das tecnologias de comunicação sem fio têm estimulado o desenvolvimento e uso de Redes de Sensores Sem Fio (RSSF). Tipicamente, uma RSSF é composta por centenas ou milhares de nós sensores, dispositivos de baixo poder computacional, pouca capacidade de armazenamento de energia e equipados com um ou mais sensores. Estas redes têm sido usadas por diversos tipos de aplicações, tais como monitoramento de ambientes,

rastreamento de objetos, agricultura de precisão e sistemas militares [1], [2], [3]. Comum a estas aplicações é a necessidade contínua de coletar e integrar dados oriundos de um grande número de nós sensores.

O projeto, a implementação e a avaliação do consumo de energia por *middleware* para RSSF ainda encontram-se em um estágio inicial de desenvolvimento. Ao contrário de plataformas de *middlewares* tradicionais onde é crescente o número de serviços (e.g., serviços de infra-estrutura e de distribuição [25]), o projeto de *middleware* para RSSF ainda inclui um número reduzido de serviços como agregação de dados e gerenciamento de recursos. Em termos de implementação, as limitações de processamento e armazenamento, e o consumo de energia tornam um desafio à implementação real dos serviços mencionados. Por fim, a avaliação de *middleware* tipicamente inclui a avaliação do consumo de energia em detrimento a tradicional avaliação de desempenho.

Neste contexto, este artigo consolida publicações anteriores [4], [5], [6] e apresenta os passos do projeto, implementação e avaliação do consumo de energia de um *middleware* para RSSF. O projeto inclui essencialmente a definição dos serviços e do modelo de comunicação fornecidos pelo *middleware*. Na implementação são consideradas as abstrações fornecidas pela linguagem de programação adotada, as primitivas de comunicação disponibilizadas pelo SO e a API disponibilizada para construção de aplicações. Por fim, a avaliação do *middleware* inclui uma análise do consumo de energia. Estes passos foram adotados na construção de um novo *middleware* chamado Mires.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 apresenta os conceitos básicos de redes de sensores. A Seção 4 apresenta as etapas iniciais do projeto do *middleware*, onde são definidos a arquitetura, os serviços fornecidos pelo mesmo e as questões de implementação. Na seção seguinte, Seção 5, é apresentada a avaliação de consumo de energia e ocupação de memória do *middleware* implementado e finalmente a Seção 6 apresenta algumas conclusões e direções para trabalhos futuros.

2 Trabalhos Relacionados

Vários sistemas de *middleware* têm sido projetados para RSSFs. O Maté [12] é uma arquitetura para construção de aplicações específicas em uma máquina virtual que executa sobre o sistema operacional TinyOS [13]. O Maté fornece uma interface de programação simples para os nós sensores. Um outro *middleware*, o Impala [14] considera a própria aplicação para explorar técnicas de código móvel para mudar a funcionalidade do *middleware* que está executando em um nó sensor. A chave para a eficiência em energia do Impala está no fato de que as aplicações são projetadas de forma modular. Diferentemente do Impala e do Maté, o MiLAN (*Middleware Link Applications and Networks*) [9] tem uma arquitetura que pretende se ajustar sobre múltiplas redes físicas. O MiLAN atua como uma camada que permite um *plugin* de rede converter os comandos do MiLAN para os protocolo específicos da pilha de protocolos de rede. Conseqüentemente, o MiLAN pode continuamente adaptar-se às características específicas de qualquer rede. Finalmente, o Cougar [15] adota uma abordagem baseada em banco de dados onde as leituras dos sensores são consideradas

tabelas "virtuais" da base de dados relacional. Uma linguagem de consulta, denominada SQL-Like, é usada para emitir as tarefas na RSSF.

Existem vários outros projetos de *middleware* que, devido às limitações de espaço, não serão mencionados neste artigo. Contudo, a maioria deles está em um estágio inicial focando no desenvolvimento de algoritmos e componentes para RSSF, que mais tarde podem servir como uma base para o desenvolvimento de *middleware*. As primeiras experiências mostram que mesmo os simples protocolos e algoritmos podem exibir a complexidade surpreendente em grandes redes [11]. Apesar de tudo, há um uma grande distância a percorrer para o desenvolvimento de *middleware* bem sucedidos em RSSF, em termos de projeto e de implementação.

3 Conceitos Básicos

Uma rede de sensores consiste de um grande número nós sensores, dispositivos com poder computacional, comunicação sem fio e com capacidade de sensoriamento [1]. Estes nós são geralmente distribuídos em uma região de interesse. Cada nó sensor na região de interesse é responsável por extrair dados do ambiente (tais como, temperatura, umidade, pressão e luminosidade), processar e enviar estes dados através de um ou mais nós sorvedouros, que são responsáveis por transmitir os dados para o usuário final.

O projeto de aplicações para rede de sensores é altamente influenciado pela escassez de recursos (tais como, largura de banda, energia e memória), modelos de comunicação e com os requisitos da aplicação. Na transmissão sem fio a potência do sinal decai proporcionalmente com o quadrado da distância. Assim, se a comunicação entre nós sensores e o sorvedouro for através de um único salto será necessário ajustar a força do sinal para tornar possível a troca de mensagens na rede. Este procedimento irá causar um aumento no consumo de energia e conseqüentemente uma redução da vida da rede. Em determinados cenários uma solução seria adotar a comunicação *multi-hop*, onde alguns nós intermediários atuam como ponte entre o nó que originou a mensagem e o destinatário da mesma. Entretanto, numa RSSF densa, muitos nós freqüentemente detectam um mesmo fenômeno gerando redundância e transmissões de dados desnecessárias. Este fato torna mais crítico o consumo de energia. Uma técnica que ajuda a economizar energia é a agregação de dados [7], [8] cuja idéia é combinar os dados vindos de fontes diferentes e eliminar a redundância do dado transmitido, minimizando o número de transmissões e, portanto, economizando energia [16].

Requisitos específicos da aplicação também afetam os recursos da rede. Por exemplo, o consumo de energia para enviar mensagens para a rede tende a ser maior nas aplicações de monitoramento do que em sistemas de vigilância uma vez que a violação de um ambiente é um evento raro. Quando uma violação acontece, a aplicação deve garantir a entrega da notificação do evento no momento adequado. Numa aplicação de rastreamento, somente os nós sensores próximos ao evento observado precisam enviar informações para a rede.

As restrições de recursos combinadas com os requisitos específicos das aplicações tornam o desenvolvimento de aplicações para redes de sensores uma tarefa desafiadora. O projeto de um *middleware* para RSSF deve considerar estes tópicos a fim de esconder esta complexidade dos desenvolvedores de aplicações. Entretanto, devido à missão de suportar e otimizar uma ampla classe de aplicações, os "*tradeoffs"* entre o

grau de generalidade do *middleware* e o grau de especialização da aplicação devem ser explorados [10].

4 Projeto e Implementação de um *Middleware* para RSSF

A construção de qualquer modelo de *middleware* é fortemente influenciada pelos requisitos das aplicações e por questões relacionadas à infra-estrutura de comunicação (e.g., rede, SO) onde o *middleware* irá executar. Desta forma, a apresentação do Mires inicia com a definição dos serviços que devem ser providos pelo *middleware* para atender as necessidades das aplicações e para se adequar ao ambiente das RSSFs.

- Serviço de comunicação - aplicações para RSSF coletam e integram continuamente os dados gerados a partir de um numeroso e fisicamente disperso contingente de nós sensores. Nesse cenário, há um grande número de dispositivos trocando dados, enquanto algumas fontes de informação e sorvedouros podem não estar presentes na rede ao mesmo tempo. Conseqüentemente, a comunicação do tipo *request/response* (síncrona) nem sempre é adequada para satisfazer a tais requisitos, apesar de que, em certos cenários esse modelo é imprescindível. Primeiro, um cliente que solicitar atualizações instantâneas da informação necessitaria consultar continuamente os fornecedores de informação, o que conduziria à sobrecarga e ao congestionamento da rede. Segundo, como a energia é um recurso escasso, os pedidos desnecessários de informação devem ser evitados. Finalmente, como as partes nem sempre estão presentes, o cliente poderia ter que esperar um longo tempo por um servidor sem conectividade. A comunicação por passagem de mensagem (assíncrona) é mais adequada para o modelo de disseminação de informação (aplicações *data-driven*[22]) requerido pelas aplicações de rede de sensores [17], [18]. Neste tipo de comunicação, um *fornecedor* (*publisher*) da informação publica as mensagens que são enviadas a um ou mais *assinantes* (*subscribers*). Uma extensão a este modelo básico permite que mensagens sejam associadas a tópicos. Neste caso particular, os assinantes recebem somente as mensagens associadas com o tópico exato a que subscreveram. A comunicação assíncrona é a principal vantagem deste modelo de comunicação no contexto de ambientes *ad-hoc* e difuso como é o caso das RSSFs [19].
- Serviço de agregação - as aplicações de monitoramento requerem geralmente que os dados coletados nos nós sensores estejam agregados a fim reduzir o número das transmissões na rede. A agregação dos dados é a combinação dos dados de fontes diferentes usando funções tais como a supressão, o mínimo, o máximo e a média [22]. Uma implementação ingênua da agregação na rede de sensores deveria usar uma abordagem centralizada, baseada em servidor, onde todas as leituras do sensor são emitidas a uma estação base, que é responsável pela agregação dos dados [23].

Considerando a necessidade destes serviços, a arquitetura do Mires foi definida como mostrado na Figura 1. De acordo com esta figura, o Mires está posicionado acima do sistema operacional, encapsulando seus detalhes internos e fornecendo serviços de alto nível à aplicação do nó.

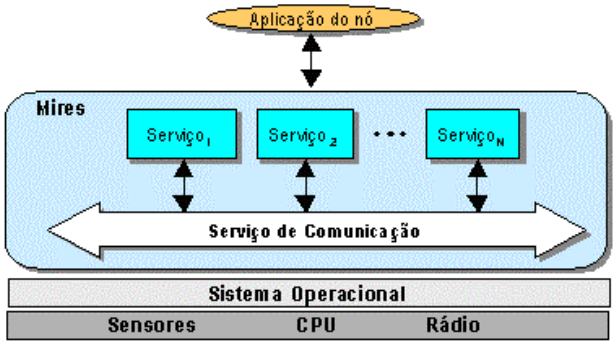

Figura 1. Arquitetura do Mires.

O componente principal da arquitetura do Mires é o serviço de comunicação. Este serviço é responsável por: anunciar os tópicos fornecidos pela aplicação do nó; manter a lista dos tópicos assinados pela aplicação terminal; publicar as mensagens que contêm os dados relacionados aos tópicos anunciados e intermediar a comunicação entre os demais serviços do *middleware*.

Para evitar desperdício de energia, somente as mensagens referentes aos tópicos assinados são transmitidas. Outro componente, o serviço de agregação, é definido para cada nó sensor, permitindo que o nó conduza a redução de dados na rede. Por sua vez, este serviço interage com a camada de rede que é responsável por transmitir as mensagens geradas localmente ou proveniente de outros nós na rede para o nó sorvedouro. Finalmente, os serviços adicionais estão previstos na arquitetura do Mires como uma indicação da possibilidade de inclusão de novos serviços. A única restrição é a necessidade de conectá-lo diretamente no serviço de comunicação.

Definida a arquitetura, o próximo passo no desenvolvimento do Mires é a implementação destes serviços.

4.1 Implementação do Serviço de Comunicação

Antes de apresentar detalhes das implementações dos serviços, vale a pena apresentar alguns aspectos relevantes do ambiente de implementação onde os serviços fora construídos, o *TinyOS*. Este ambiente tem um modelo de programação baseado em componentes fornecido pela linguagem nesC [20], uma linguagem de alto nível para a construção de aplicações estruturadas em componentes. Os programas nesC podem ser vistos como um grafo de componentes, onde cada componente fornece e utiliza as interfaces externas, que são compostas por comandos e por eventos. Os comandos são os procedimentos implementados pelo fornecedor da interface. Quando um fornecedor da interface sinaliza um evento, este causa a execução de um procedimento (tratador do evento) executado no usuário da interface. A comunicação entre os nós da rede é baseada nos paradigma *active messages* [21]. De acordo com este paradigma, cada mensagem contém o ID de uma rotina de tratamento a ser invocada no nó alvo e um *payload* contendo os dados a serem passado como argumentos. Esta comunicação baseada em eventos e orientada a mensagens faz do *TinyOS* uma boa fundação para a construção de uma infra-estrutura de comunicação *publish/subscribe*.

Como mencionado anteriormente, o serviço de comunicação é o principal componente do Mires. A Figura 2 ilustra as conexões entre os componentes do serviço

de comunicação (*PublishSubscribe*) e os outros elementos do Mires. O componente *PublishSubscribe* fornece as interfaces *Advertise* e *Publish* para a aplicação do nó sensor, as interfaces *PublishState* e *Notifier* aos serviços adicionais. Estas duas últimas interfaces permitem a adição de novos serviços ao Mires.

A interface *PublishState* define o comando usado pelos demais serviços para publicar seus resultados de processamento. A interface *Notifier* define três eventos a que serviços adicionais devem fornecer implementações de rotinas de tratamento para serem notificadas pelo serviço *publish/subscribe* de ocorrências como a chegada de dados locais ou remotos.

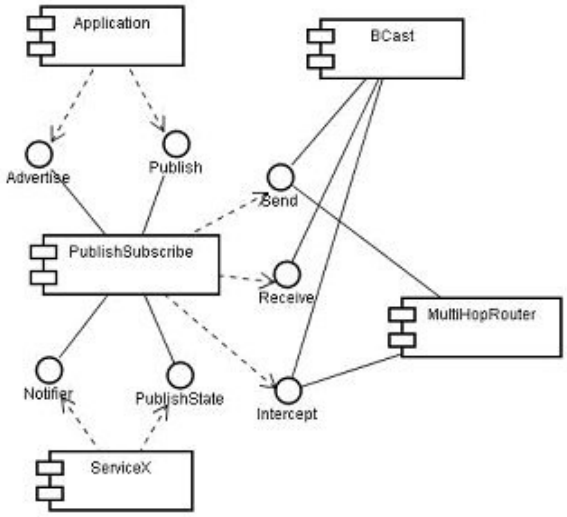

Figura 2. Diagrama de componentes do Serviço de Comunicação.

O componente *PublishSubscribe* usa três interfaces (*Send*, *Receive* e *Intercept*) que são implementadas pelos componentes de comunicação *Bcast* e *MultiHopRouter*. O componente *MultiHopRouter* é responsável por estabelecer a hierarquia de roteamento para o nó sorvedouro. O componente *Bcast* transmite a informação de configuração (e.g., tópicos assinados pela aplicação do usuário) através da rede.

4.2 Implementação do Serviço de Agregação

O serviço de agregação é definido para cada nó sensor, permitindo que o nó conduza a redução de dados na rede. Esta técnica é usada para reduzir o número de transmissões de mensagens, a latência e o consumo de energia em [8] e [24]. Para realizar esta exigência, o serviço de agregação pode ser configurado durante o processo de assinatura. O usuário pode configurar como os dados serão agregados (função de agregação) e os critérios de parada (política de agregação). Quando o critério de parada é satisfeito em um nó, este publica seu resultado local ao nó seguinte de acordo com o algoritmo de roteamento. O serviço de agregação executa três ações básicas:

- Agregar os dados originados do sensor local com os dados que vêm da rede;
- Controlar a associação entre os tópicos e suas funções de agregação. Isso é necessário porque os nós podem receber dados de diversos tópicos e cada tópico pode ser associado a uma função de agregação diferente; e

- Verificar a satisfação do critério da política de agregação e solicitar a publicação do estado associado com o tópico. Por exemplo, o critério tempo define um intervalo onde a função de agregação é aplicada em valores obtidos localmente ou gerados em outro nós. Quando o tempo acaba, o estado local será transmitido e reiniciado.

Uma função de agregação define como os dados são combinados. A Figura 3 mostra a estrutura do serviço de agregação proposta. O módulo *AggregationM* contém a implementação do serviço de agregação, tratando as notificações emitidas pelo serviço de publicação. As funções de agregação são executadas em módulos separados tais como AVG e SUM. O módulo *AggregateUse* realiza uma atividade de multiplexação, passando pedidos para o módulo de agregação correto de acordo com o seu identificador. Desta maneira, a flexibilidade para adicionar funções de agregação novas é garantida, requerendo apenas a criação de um módulo para a função nova e a associação entre a função e um identificador a um arquivo da configuração.

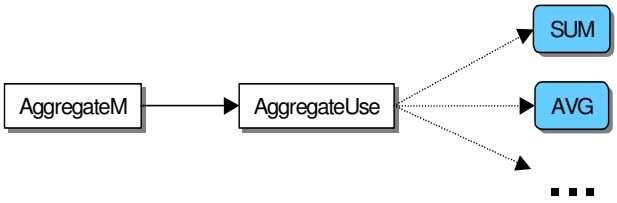

Figura 3. Estrutura do serviço de agregação.

O serviço de agregação acopla-se ao serviço de comunicação através do uso das interfaces *PublishState* e *Notifier* (ver Seção 4.1). Estas interfaces definem, respectivamente, os comandos para os demais serviços publicarem seus resultados e os eventos utilizados para a notificação da chegada de novos dados.

4.3 Implementação de Serviços Adicionais

Como mencionado anteriormente, os serviços adicionais podem ser facilmente incorporados ao Mires através das interfaces que definem eventos de notificação. Assim, o serviço de comunicação (componente *PublishSubscribe*) notifica os demais serviços (aqueles interessados na mensagem) sobre a mensagem que chega da rede ou é submetida pela aplicação local.

Se um serviço estiver interessado na notificação de um determinado evento, executa somente uma rotina de tratamento específica a esse evento. Há três tipos de eventos de notificação: *topicArrival*, *stateArrival* e *topicSetupArrival*. O evento *topicArrival* sinaliza que a aplicação local submeteu os dados coletados dos sensores. Quando este evento ocorre, o serviço notificado decide primeiramente se os dados serão localmente processados (por exemplo, agregado) ou transmitidos, comunicando então sua decisão ao componente *PublishSubscribe* (serviço de comunicação). O evento *stateArrival* é muito similar ao anterior, exceto pelo fato de que os dados vêm da rede. Os dados da rede representam os resultados de processamento de um nó posicionado em um nível mais baixo da hierarquia de roteamento. Finalmente, o evento *topicSetupArrival* é a mensagem de assinatura transmitida da aplicação do usuário que contém uma lista de tópicos assinados (por exemplo, temperatura e umidade) e a

informação de configuração para os serviços (por exemplo, a política que estabelece o critério de parada como limite de tempo e número das amostras). Assim que o critério de parada para um tópico assinado for satisfeito, o serviço pode publicar os resultados processados localmente para a rede. O componente *PublishSubscribe* interage então com o componente na camada de rede (algoritmo de roteamento) para emitir uma mensagem que contém os resultados locais ao nó no nível seguinte da hierarquia. Este processo se repete até que uma mensagem de publicação alcance o nó sorvedouro, que é responsável por transmitir os dados coletados pela rede de sensores à aplicação do usuário.

4.4 Interação entre os Serviços

Os diagramas de seqüência apresentados na Figura 4 mostram as interações entre o componente *PublishSubscribe* e os outros componentes do Mires. A aplicação do nó anuncia ao componente *PublishSubscribe* sua capacidade de monitorar os dados relacionados a um determinado tópico (fase de anúncio). O componente *PublishSubscribe* encapsula esta informação em uma mensagem *advertiseMsg* e transmite-a rede através do componente *MultiHopRouter*. A interação na parte inferior da Figura 4 refere-se à ocorrência da chegada de uma *advertiseMsg* em um nó. Todas as mensagens são dirigidas ao nó sorvedouro, mas nos nós intermediários o componente *MultiHopRouter* sinaliza um evento *Intercept* sempre que recebe uma *advertiseMsg*. Em seguida, o componente *PublishSubscribe* extrai a informação dos tópicos anunciados da mensagem, atualiza sua estrutura interna de controle e retorna uma indicação ao *MultiHopRouter* que a mensagem deve ser enviada aos nós superiores na hierarquia da rede.

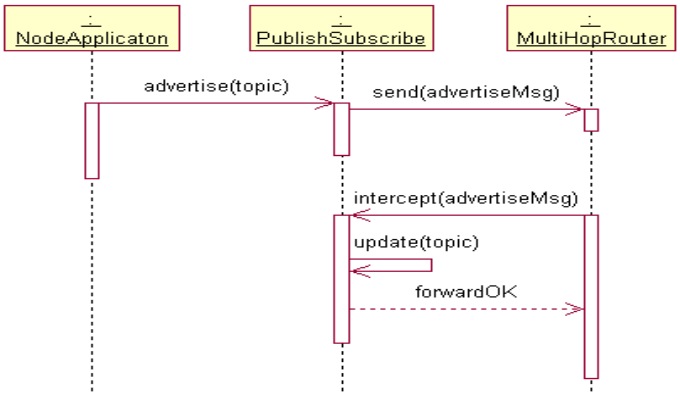

Figura 4. Fase anúncio de tópico.

A Figura 5 ilustra a interação da assinatura de tópicos que é iniciada após a fase de anúncio. A aplicação do usuário invoca o comando de envio do nó sorvedouro a fim transmitir os tópicos assinados para a rede. Em cada nó que recebe a *subscribeMsg*, o componente de *Bcast* sinaliza um evento *receive*. Então, o componente *PublishSubscribe* extrai a informação de configuração da mensagem e sinaliza o evento *topicSetupArrival* para notificar os componentes dos demais serviços.

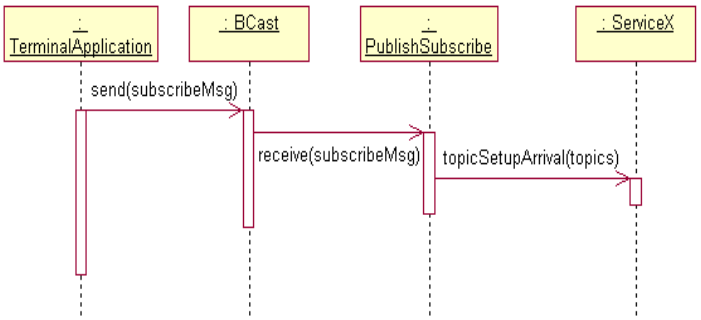

Figura 5. Fase de assinatura de tópico.

A Figura 6 mostra como os dados monitorados são publicados e processados pela rede. A aplicação do nó periodicamente coleta leituras dos sensores e invoca o comando *publish* do componente *PublishSubscribe*. Se outros serviços estiverem em operação, o componente *PublishSubscribe* notifica-os com o evento *topicArrival*. Na interação seguinte, o serviço invoca o comando do *publishState* ao componente *PublishSubscribe* passando seus resultados de processamento. Então, o serviço *publish/subscribe* encapsula o estado local dentro da mensagem *publishMsg* e a envia através da rede usando o componente *MultiHopRouter*. Finalmente, a terceira interação ocorre quando uma mensagem *publishMsg* chega em um nó. O *MultiHopRouter* sinaliza um evento *intercept* que contém a mensagem *publishMsg*. O componente *PublishSubscribe* extrai o estado remoto da mensagem e notifica os outros serviços sinalizando o evento *stateArrival*. Assim, espera-se que o serviço notificado integre o estado remoto ao estado local.

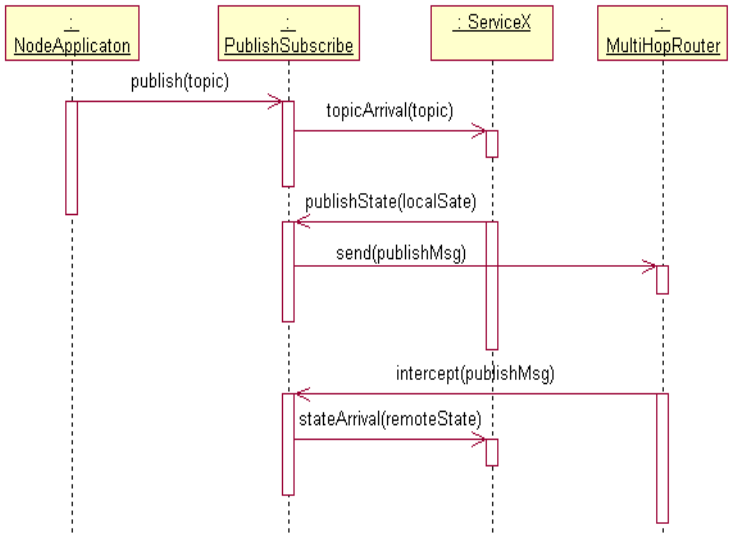

Figura 6. Fase de publicação.

5 Avaliação do Mires

Nesta seção definimos os passos para a avaliação do Mires. Inicialmente, apresentamos um cenário de utilização do *middleware* e a aplicação exemplo usada na avaliação. Em seguida, apresentamos experimentos para avaliação do consumo de energia e da ocupação da memória. Com a avaliação do consumo de energia, a intenção é apresentar

o consumo de energia adicional introduzido pelo Mires ao sistema. Em termos de ocupação de memória, o objetivo é apresentar o impacto no consumo de memória (RAM e ROM) introduzido pelo uso da camada de *middleware*.

5.1 Cenário de Avaliação

O cenário de avaliação representado na Figura 7 consiste de um ambiente de monitoração cúbico com as seguintes características: cada sala é formada por um grupo de nós sensores que podem monitorar variáveis tais como a temperatura, a umidade, o som e a luminosidade.

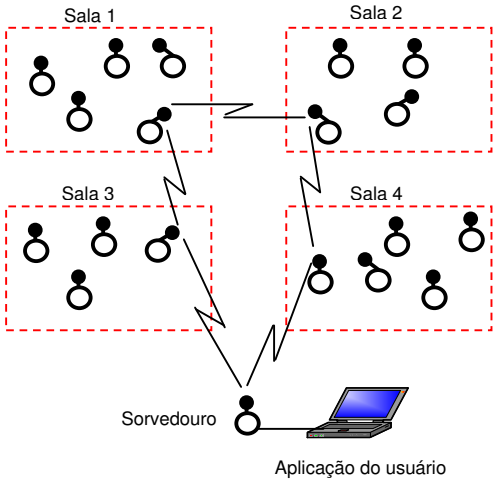

Figura 7. Nós sensores distribuídos em salas.

Os sensores são agrupados nas salas formando *clusters*. Cada sala tem um nó responsável pela comunicação com o nó sorvedouro, chamado *cluster head*. Cada nó dentro de um conjunto integra a informação dos nós abaixo dele na hierarquia de roteamento por meio de alguma técnica de agregação e relata os resultados acima na hierarquia até o *cluster head*. Um nó sorvedouro recebe as tarefas de monitoramento da aplicação externa e as distribui pela rede. Assim, o fluxo das tarefas vai do nó sorvedouro aos nós sensores, enquanto o fluxo de dados segue em sentido oposto. A aplicação externa coletará estes dados, lidando com eles da forma desejada.

5.2 Aplicação de Monitoramento de Ambientes

A Figura 8 mostra uma aplicação do usuário e sua interação com a arquitetura do *middleware*. A execução do MIRES pode ser dividida nas seguintes fases: estabelecimento, anúncio, assinatura e publicação da rede.

Após a distribuição dos nós sensores, mensagens de configuração começam a ser trocadas com o objetivo de estabelecer rotas para o nó sorvedouro. Este processo é controlado pelo componente de roteamento que sinaliza a conclusão do estabelecimento da rede à aplicação do nó sensor.

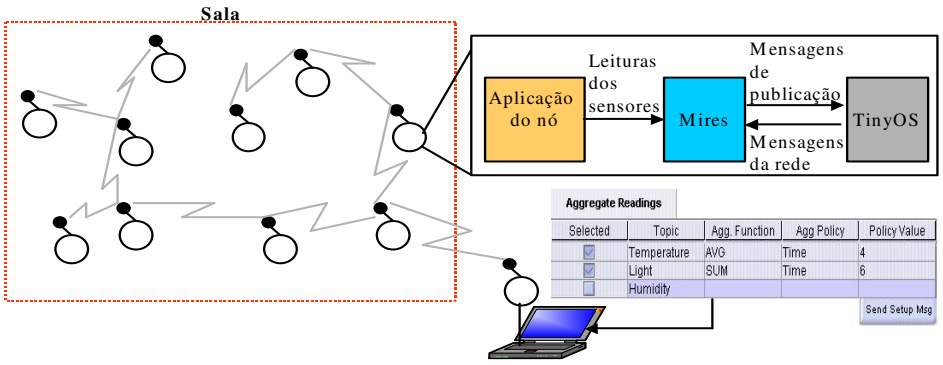

Figura 8. Aplicação exemplo.

Quando o estabelecimento da árvore de roteamento é concluído, a fase de anúncio está iniciada. Nesta fase, a aplicação do nó sensor informa o *middleware* que publicará dados de um tópico específico (por exemplo, temperatura). É responsabilidade do Mires informar os outros nós sensores na árvore de roteamento. Assim, a aplicação não necessita preocupar-se sobre o procedimento da difusão do tópico na rede.

Uma vez que os tópicos anunciados chegam ao nó sorvedouro, são passados à aplicação do usuário. Neste estudo de caso, uma interface gráfica foi construída para permitir que o usuário selecione um ou mais tópicos de interesse, especificando políticas e funções de agregação desejadas. Um *screenshot* desta aplicação é mostrado na Figura 8, que demonstra uma configuração onde o usuário está interessado em receber dados sobre a temperatura a cada quatro minutos e luminosidade a cada seis minutos. A aplicação do usuário emite uma mensagem subscrever ao nó sorvedouro e, então, o Mires é responsável pela disseminação das configurações do usuário aos outros nós na rede. Este procedimento corresponde à fase de assinatura.

Após a fase de assinatura, a aplicação do nó começa publicar os dados monitorados, competindo ao Mires a responsabilidade de notificar o serviço de agregação (descrito na Seção 4.2) da chegada dos tópicos de interesse assinados pela aplicação do usuário. No serviço de agregação, os dados são combinados de acordo com a função e as políticas estabelecidas pela aplicação do usuário na fase de assinatura. O Mires é responsável por transmitir os dados processados pelo serviço de agregação aos demais nós.

Em resumo, a aplicação do usuário necessita informar o Mires somente que tópicos (temperatura, luminosidade ou umidade) são de interesse. O *middleware* recebe esta mensagem de configuração (tópicos, função e políticas de agregação) e envia os dados pedidos em direção à aplicação do usuário. Desta maneira, o Mires faz o controle e a comunicação das leituras dos sensores, reduzindo assim o *"gap"* semântico do desenvolvimento da aplicação.

5.3 Consumo de Energia

Para avaliar o desempenho do *middleware* Mires foram realizadas diversas simulações em redes de sensores compostas por 20, 40 e 60 nós sensores estacionários e homogêneos, distribuídos randomicamente, em uma área de 100 x 100 m^2. O tempo de simulação foi de 1000 segundos.

Os experimentos foram realizados no simulador TOSSIM [26], utilizando o módulo de cálculo de consumo de energia denominado PowerTossim [27]. A execução deste módulo possibilita a geração de um arquivo contendo os registros do consumo de energia de cada nó na rede. Os parâmetros de simulação foram ajustados para corresponder a uma rede real baseada no nó sensor Mica2 [28]. A quantidade de energia inicial de cada nó sensor foi definida como 100 Joules. A freqüência de amostragem e a geração de mensagens também foram configuradas para serem a mesma em ambas aplicações. Seguindo esta metodologia foram realizadas medições referentes ao consumo de energia (rádio, CPU e sensor). Os resultados apresentados a seguir representam uma média de 30 replicações para um intervalo de confiança de 95%.

As medições de consumo de energia e ocupação de memória do Mires foram comparadas com a aplicação *Surge*, que faz parte do pacote de instalação do TinyOS 1.1.6. A principal motivação para esta comparação é fato que tanto a aplicação Surge como a aplicação de monitoramento implementada neste trabalho (ver Seção 5.2) foram programadas para realizarem o monitoramento do ambiente e o envio dos dados para a estação base. A aplicação Surge é um simples programa de leitura e transmissão de dados. Nesta aplicação, os nós sensores periodicamente coletam as leituras dos sensores e as roteiam para a estação base, que é responsável por receber os dados e graficamente exibi-los para o usuário final. Para tornar a comparação mais fidedigna, o Mires utilizou o mesmo algoritmo de roteamento adotado pela aplicação Surge. Este algoritmo forma uma *spannig tree* onde cada nó mantém o endereço e a profundidade do seu pai na árvore. Na transmissão de dados, o nó tenta selecionar o pai com menor profundidade e melhor qualidade estimada do link.

Como o objetivo é avaliar o impacto do consumo de energia introduzido pela camada de *middleware*, as medições realizadas avaliaram o consumo de energia do Mires com e sem o serviço de agregação. A idéia em avaliar o Mires sem nenhum serviço adicional (o que inclui o serviço de agregação implementado) é apresentar o consumo de energia necessário para fazer uso dos serviços básicos da camada de *middleware.*

A Figura 9 representa o consumo médio de energia da rede para a aplicação Surge e para o Mires com e sem o serviço de agregação. O serviço de agregação foi configurado para sumarizar os dados recebidos a cada intervalo de 10 segundos através da média aritmética dos valores. Como pode ser observado na Figura 9(a), referente a uma rede com 20 nós sensores, a aplicação surge apresentou menor consumo de energia (≅ 18 Joules) se comparado com o consumo do Mires (≅ 20 Joules) para o mesmo período de simulação (1000 segundos). A diferença entre os consumos foi de apenas 2 Joules, o que representa uma diminuição no tempo de vida da rede usando o *middleware* Mires. Este consumo adicional é introduzido principalmente pelo serviço *publish/subscribe,* que é responsável por toda interação entre os componentes da rede.

Os experimentos realizados para redes compostas por 40 e 60 nós sensores, Figuras 9(b) e 9(c) respectivamente, mostram comportamentos semelhantes aos apresentados na Figura 9(a). As diferenças de consumo são aproximadamente de 2 Joules para a rede com 40 nós e de 3 Joules para a rede com 60 nós.

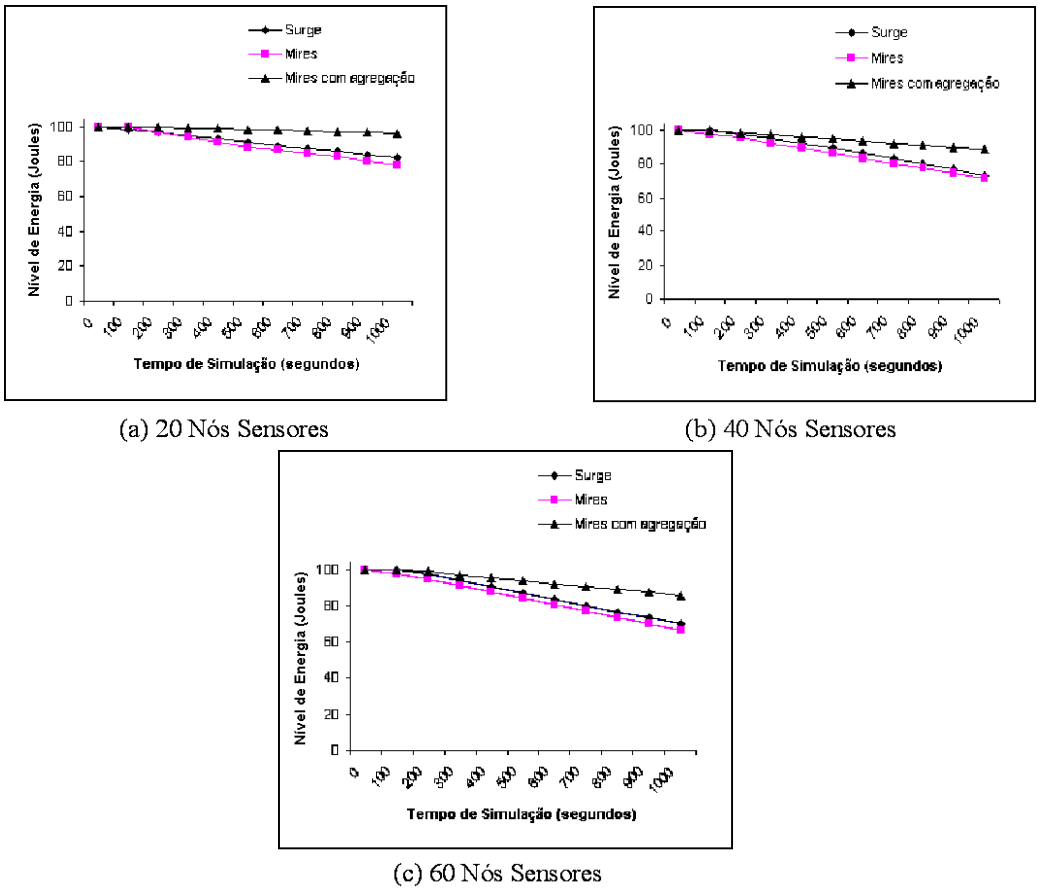

Figura 9. Consumo de energia do Mires sem e com o serviço de agregação para uma rede composta de 20, 40 e 60 nós sensores.

A utilização do serviço de agregação no Mires possibilitou a obtenção de níveis de consumo inferiores ao do Surge, conforme pode ser observado nos gráficos da Figura 9. Com o serviço de agregação configurado para sumarizar os dados recebidos a cada intervalo de 10 segundos, os consumos de energia caíram para 4, 12 e 15 Joules para as redes com 20, 40 e 60 nós respectivamente. Isto ocorreu porque o serviço de agregação foi implementado em cada nó sensor, permitindo que o nó conduza a redução de dados na rede, o que possibilitou uma redução no número de transmissões de mensagens e, conseqüentemente, do consumo de energia. Outros experimentos foram realizados variando o tempo de agregação e os resultados obtidos apresentaram economia proporcional à redução do envio e recebimento de mensagens pelo rádio.

5.4 Ocupação de Memória

O processo de compilação do código fonte nesC para o Mica2 gera um relatório informando em bytes a ocupação da memória ROM e RAM. No Mica2, a ROM é uma memória flash de 128KB (interna ao ATmega128L) utilizada para armazenar o programa (segmentos TEXT e DATA). Já a RAM do ATmega128L é uma SRAM de 4KB, também interna, utilizada para armazenar os dados da aplicação (segmento BSS).

A Tabela 1 mostra os valores de ocupação de memória da aplicação Surge e do Mires com e sem o serviço de agregação. Como pode ser observado, a aplicação Surge ocupa 12,9 % do espaço de memória ROM e 47% da RAM do Mica2. Já os valores

obtidos através do processo de compilação do *middleware* Mires mostram que a ocupação é de apenas 12,8% da ROM e 48,9% da RAM. Essa diferença mínima entre os valores obtidos é devido às características de implementação das camadas de aplicação do Surge e do Mires.

Tabela 1. Ocupação de memória em bytes do Mires e Surge.

Modo Utilizado	ROM (bytes)	% ROM	RAM (bytes)	% RAM
Surge	16902	12,9	1926	47,0
Mires	16824	12,8	2001	48,9
Mires com agregação	20666	15,8	2472	60,3

Com a adição do serviço de agregação, os valores de ocupação do Mires passaram a ser 15,8% da ROM e 60,3% da RAM, restando 84,2% e 39,7% de espaço na ROM e RAM respectivamente. Este espaço livre pode ser utilizado para a implementação de novos serviços. Portanto, os resultados demonstram que é viável, sob ponto de vista de ocupação de memória, a utilização do Mires na plataforma Mica2.

6 Conclusão e Trabalhos Futuros

Este artigo apresentou os passos do projeto, implementação e avaliação do consumo de energia de *middleware* para RSSF. O *middleware* proposto fornece dois serviços básicos, comunicação e agregação, que foram implementados na linguagem nesC[20] para o sistema operacional baseado em eventos TinyOS[13]. O principal componente do Mires é o serviço de comunicação (*publish/subscribe*) que possibilita a comunicação assíncrona entre as aplicações de monitoramento. Além disto, o *middleware* desenvolvido também fornece a noção de tópicos para enriquecer o serviço de comunicação e o de agregação que interage com o algoritmo de roteamento da camada de rede.

A relevância deste trabalho vem de três pontos principais. Primeiro, os serviços fornecidos pelo *middleware*, definidos na fase de projeto, foram especialmente projetados levando-se em consideração as questões de consumo de energia nos nós sensores, as limitações do sistema operacional para redes de sensores e a natureza das RSSF. Segundo, a implementação apresentada (baseada em componentes) permite que serviços sejam reusáveis e que a integração de serviços adicionais seja feita com facilidade. Por fim, os passos da validação do *middleware* incluíram uma análise de dois dos pontos principais da avaliação de um *middleware* para RSSF: o consumo de energia e a ocupação de memória. Desta forma, foi possível definir mais claramente os benefícios (diminuição do número de transmissões em função do serviço de agregação) e as limitações (aumento do uso da memória) de um *middleware* para RSSF.

Em relação ao Mires, alguns aspectos podem ser observados. Primeiramente, o *middleware* demonstra que o modelo de comunicação *publish/subscribe* (comunicação assíncrona) é mais adequado para aplicações de monitoramento contínuo de dados em redes de sensores sem fio do que o tradicional modelo de comunicação *request/response* (comunicação síncrona). Em particular, como as mensagens são armazenadas em uma fila, se o *link* de comunicação não estiver disponível por alguma razão, as mensagens não serão perdidas e podem ser encaminhadas para o destinatário em um momento

apropriado. Um segundo aspecto é que o *middleware* foi projetado para facilitar o desenvolvimento de aplicações de monitoramento de ambientes. Os desenvolvedores podem usar uma simples API que os possibilita construir aplicações abstraindo de detalhes dos mecanismos de comunicação da rede. Finalmente, novos serviços de *middleware* são também facilmente adicionados ao Mires. O desenvolvedor precisa somente ter conhecimento de quatro operações básicas para integrar um novo serviço.

Os próximos passos do desenvolvimento do Mires é torná-lo mais robusto às mudanças de topologia e a falhas de nós. Devido à limitação de recursos das RSSF e o custo de adaptação ser alto mesmo para *middleware* tradicionais, explorar as interações entre as camadas da pilha de protocolo e as camadas do *middleware* (abordagem *cross-layer*) deve ser uma alternativa para reduzir o custo da comunicação e melhorar o desempenho do sistema. Além disso, testes usando sensores reais também estão sendo planejados. Finalmente, a adição de novos serviços como *trading,* gerenciamento de recursos, reconfiguração, rastreamento de objetos e gerenciamento de grupos [6] estão sendo projetados para o Mires.

Referências

[1] I.F.Akyildiz, W. Su, Y. Sankarasubramaniam e E. Cayirci, “A Survey on Sensor Networks”, IEEE Communications Magazine, pp. 102-114, Aug. 2002.

[2] A. Cerpa et al., “Habitat Monitoring: Application Driver for Wireless Communications Technology”, ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, Costa Rica, Apr. 2001.

[3] G.J. Pottie e W.J. Kaiser, “Wireless Integrated Network Sensors”, Communications of the ACM, Vol. 43, no. 5, pp. 51-58, Mai. 2000.

[4] E. Souto, G. Guimarães, et al., “Mires: A Publish/Subscribe Middleware for Sensor Networks”, Springer's Personal and Ubiquitous Computing Journal, 2005.

[5] E. Souto, G. Guimarães, et al., “A Message-Oriented Middleware for Sensor Networks”, in Middleware 2004, International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC), Toronto, Ontario, Canada, 2004.

[6] Mardoqueu Vieira and Nelson Rosa, "A reconfigurable group management middleware service for wireless sensor networks," MPAC '05: 3rd international workshop on Middleware for pervasive and ad-hoc computing. Grenoble, France: ACM Press, 2005.

[7] W. Heinzelman, A. Chandrakasan e H. Balakrishnan, “Energy-Efficient Communication Protocol for Wireless Microsensor Networks”, IEEE Hawaii International Conference on System Sciences, Hawaii, EUA, Janeiro 2000.

[8] S. Lindsey and C. S. Raghavendra, "PEGASIS: Power Efficient GAthering in Sensor Information Systems”, IEEE Aerospace Conference, Montana, EUA, Mar. 2002.

[9] W. Heinzelman, A. Murphy, H. Carvalho e M. Perillo, “Middleware to Support Sensor Network Applications”, IEEE Network Magazine Special Issue, pp. 6-14, Jan. 2004.

[10] Y. Yu, B. Krishnamachari e V.K. Prasanna, “Issues in Designing Middleware for Wireless Sensor Networks”, IEEE Network Magazine, Vol. 18, Issue 1, pp. 15-21, Jan. 2004.

[11] K. Römer, O. Kasten, e F. Mattern, “Middleware Challenges for Wireless Sensor Networks”, ACM SIGMOBILE Mobile Communication and Communications Review, vol. 6, No. 2, Oct. 2002.

[12]P. Levis e D. Culler. “Maté: A Tiny Virtual Machine for Sensor Networks”, 10th ASPLOS, San Jose, CA, EUA, Oct. 2002.

[13]J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler e K. Pister, “System Architecture Directions for Networked Sensors”, ACM SIGOPS Operating Systems Review, Vol. 34, Issue 5, pp. 93-104, Dez. 2000.

[14]T. Liu e M. Martonosi, “Impala: a Middleware System for Managing Autonomic, Parallel Sensor Systems”, Ninth ACM SIGPLAN symposium on Principles and practice of parallel programming, San Diego, CA, EUA, Jun. 2003.

[15]P. Bonnet, J. E. Gehrke e P. Seshadri, “Querying the Physical World”, IEEE Personal Communications, Vol. 7, No. 5, pp. 10-15, Oct. 2000.

[16]B. Krishnamachari, D. Estrin e S. B. Wicker, “The Impact of Data Aggregation in Wireless Sensor Networks”, 22nd International Conference on Distributed Computing Systems, pp. 575-578, Viena, Áustria, Jul. 2002.

[17]E. Yoneki, "Mobile Applications with a Middleware System in Publish-Subscribe Paradigm", 3rd Workshop on Applications and Services in Wireless Networks, Suíça, Jul. 2003.

[18]G. Cugola, H e A. Jacobsen, “Using Publish/subscribe Middleware for Mobile Systems”, ACM SIGMOBILE Mobile Computing and Communications Review, Vol. 6(4), pp. 25-33. ACM Press, Nova Iorque, NY, EUA, Oct. 2002.

[19]M. Cilia, L. Fiege, C. Haul, A. Zeidler e A. P. Buchmann, “Looking into the Past: Enhancing Mobile Publish/Subscribe Middleware”, 2nd International Workshop on Distributed Event-Based Systems, San Diego, CA, EUA, Jun. 2003.

[20]D. Gay, P. Levis, D. Culler e E. Brewer, “nesC 1.1 Language Reference Manual”, documentação do TinyOS, disponível em www.tinyos.net, Mai. 2003.

[21]P. Buonadonna, J. Hill e D. Culler, “Active Message Communication for Tiny Networked Sensors”, 20th Annual Joint Conference of the IEEE Computer and Communications Societies, Anchorage, Alaska, EUA, Apr. 2001.

[22]B. Krishnamachari, D. Estrin e S. Wicker, "Modelling Data Centric Routing in Wireless Sensor Networks", 21th Annual Joint Conference of the IEEE Computer and Communications Societies, Nova Iorque, NY, EUA, Jun. 2002.

[23]S.R. Madden, M.J. Franklin, J.M. Hellerstein e W. Hong, “TAG: a Tiny Aggregation Service for Ad-hoc Sensor Networks”, Symposium on Operating Systems Design and Implementation, Boston, MA, EUA, Dez. 2002.

[24]Y. Yao e J. Gehrke, “The Cougar Approach to In-network Query Processing in Sensor Networks”, ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, EUA, Set. 2002.

[25]P. Bernstein, “Middleware: A Model for Distributed System Services”, Communications of the ACM, Vol. 39, No. 2, 1996.

[26]P. Levis, N. Lee, M. Welsh e D. Culler, “TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications”, 1st international conference on Embedded networked sensor systems, Los Angeles, Califórnia, EUA 2001.

[27]V. Shnayder, M. Hempstead, B. Chen, G. Allen e M. Welsh, “Simulating the power consumption of large-scale sensor network applications”, 2nd International Conference on Embedded Networked Sensor Systems, Baltimore, MD, EUA 2004.

[28]Crossbow Technology Inc., “Introducing Crossbow Technology”, Disponível em http://www.xbow.com/index.htm, Dez. 2005.