Redes de sensores sem fio: visão geral e um exemplo de implementação

Silvana Rossetto, Flávia C. A. da Costa [1]
Vagner Sacramento, Leandro Alexandre, Rodrigo Neves, Salomão Pinheiro [2]

[1]Universidade Federal Fluminense (UFF), Pólo Universitário de Rio das Ostras (PURO)

[2]Universidade Federal de Goiás (UFG)

Resumo. *Redes de Sensores Sem Fio (RSSFs) são formadas por dispositivos de tamanho e custo reduzidos, com capacidade de processamento e sensoriamento, conectados através de um meio sem fio. As RSSFs apresentam características que as distinguem das redes tradicionais, entre elas, limitação de recursos computacionais, execução de aplicações específicas, requerimento de vida longa, presença de sensores que requerem acoplamento com o mundo físico e condições ambientais dinâmicas. Neste trabalho, apresentamos uma visão geral sobre as RSSFs, incluindo exemplos de áreas de aplicação, problemas e características específicas dessas redes, plataformas de hardware e soluções de software disponíveis para uso, e um exemplo prático de implementação de uma aplicação.*

1. Introdução

Redes de Sensores Sem Fio (RSSFs) são formadas como uma coleção de sensores e/ou atuadores e nós computacionais conectados via uma rede para coletar, processar e disseminar informações sobre tópicos de interesse em um ambiente físico [Akyildiz et al. 2002, Chong and Kumar 2003, Culler et al. 2004]. A noção de RSSFs está associada à definição de entidades que ao invés de operarem isoladamente, colaboram para alcançar um propósito definido de monitoramento de uma área ou de supervisão de um processo particular.

Tipicamente, os dispositivos coletam, processam e transmitem a informação capturada (e.g., temperatura do ambiente, velocidade de deslocamento de um objeto, etc.) para uma estação base, via rede sem fio. A capacidade de processamento real nas redes de sensores é obtida somente com a agregação dos vários dispositivos que individualmente capturam a informação de interesse, e, em conjunto, transformam informações do mundo físico em dados que podem ser avaliados e usados para finalidades diversas.

As RSSFs resultam da convergência de três tecnologias: micro-processadores, comunicação sem fio e micro-sistemas eletro-mecânicos [Hill et al. 2004]. Essas redes apresentam características que as distinguem das redes tradicionais, entre elas, limitação de recursos computacionais, execução de aplicações específicas, requerimento de vida longa, presença de sensores que requerem acoplamento com o mundo físico e condições ambientais dinâmicas (em ambientes abertos, por exemplo, os sensores são expostos às interpéries da natureza). RSSFs promovem oportunidades de pesquisa em diferentes áreas da engenharia de computação, entre elas, a computação ubíqua [Weiser 1993, Hightower and Borriello 2001], e requerem soluções específicas para a construção de compiladores, linguagens de programação, sistemas operacionais, protocolos de comunicação e modelos de programação [Estrin et al. 2002].

Uma rede de sensores pode prover acesso à informação em qualquer tempo e em qualquer lugar, coletando, processando, analisando e disseminando dados. Essas redes permitem criar ambientes inteligentes, com sensores capazes de coletar dados com maior acurácia. Combinando o denso sensoreamento próximo aos fenômenos físicos, i.e., fatos, aspectos ou ocorrências passíveis de observação, com as tecnologias de software que formam a Internet, é possível instrumentar o mundo com maior fidelidade [Culler et al. 2004].

O desenvolvimento de aplicações que exploram as possibilidades oferecidas pelas redes de sensores requer o estudo e compreensão dos limites físicos dessas redes (e.g., bateria, capacidade de processamento, armazenamento e comunicação) e a experimentação de protocolos, algoritmos e modelos de programação que se adequem às suas características e limitações particulares [Chong and Kumar 2003].

O objetivo deste documento é oferecer uma visão geral sobre o funcionamento, as oportunidades e as limitações principais das RSSFs. Organizamos o documento da seguinte forma: na Seção 2, destacamos alguns exemplos de aplicações que usam RSSFs; na Seção 3, caracterizamos as RSSFs destacando as diferenças entre essas redes e as redes convencionais; na Seção 4, apresentamos uma visão geral da pilha de protocolos nas RSSFs (camada física, de enlace e de rede); na Seção 5, apresentamos os principais padrões definidos para as RSSFs; na Seção 6, mostramos alguns sistemas operacionais especialmente desenvolvidos para RSSFs; na Seção 7, discutimos aspectos particulares para tratar as questões de localização e sincronização em RSSFs; na Seção 8, apresentamos modelos de programação propostos para RSSFs, destacando o modelo de base de dados; na Seção 9, discutimos os requisitos especiais e as soluções propostas para a questão de segurança nas RSSFs; por fim, na Seção 10, discutimos um exemplo de implementação e na Seção 11 apresentamos as considerações finais deste trabalho.

2. Exemplos de aplicações usando RSSFs

As RSSFs podem ser aplicadas em diferentes cenários, entre eles: monitoramento ambiental; controle de processos industriais; medicina; agricultura de precisão; provisão de segurança em ambientes públicos; monitoramento do tráfego de veículos; entre outros. RSSFs são ditas de *aplicação-específica*, uma vez que são desenvolvidas para um domínio específico.

Monitoramento ambiental A possibilidade de monitorar características como temperatura, luminosidade, correntes de ar e poluição atmosférica podem ser usadas para otimizar o controle de ambientes fechados (por exemplo, controle automático do aquecimento e resfriamento de salas em um edifício), ou permitir o monitoramento de ambientes externos. RSSFs podem ser usadas para a detecção de incêndios ou de abalos sismícos, e atuarem orientando as pessoas para seguirem rotas de fugas mais adequadas dentro de um edifício.

Em ambientes abertos, as RSSFs podem ser usadas, por exemplo, para monitorar o ambiente natural de animais silvestres e o comportamento desses animais em função das mudanças climáticas [Arampatzis et al. 2005].

Monitoramento e controle industrial Na indústria, a utilização de sensores está voltada para a redução de custos da produção e o incremento do desempenho e da manutenabilidade do maquinário usado no processo produtivo. Monitorar a "saúde" de uma máquina por meio da determinação da vibração ou do nível de lubrificação e a inserção de sensores em locais inacessíveis pelos seres humanos (por exemplo, dentro de uma caldeira) são exemplos das possibilidades de utilização das RSSFs no ambiente industrial. Outra classe de aplicação é a utilização de sensores sem fio para instrumentar todo o processo produtivo de uma fábrica, garantindo o atendimento de exigências sanitárias (e/ou outros requisitos de operação) com a possibilidade de redução de custos [Chong and Kumar 2003].

Medicina Na área de saúde, existe uma expectativa de crescimento do uso das RSSFs em função do desenvolvimento de sensores biológicos, compatíveis com a tecnologia de circuitos integrados. Esses sensores são capazes de detectar materiais biológicos como enzimas e ácidos, podendo trazer grandes benefícios para a aplicações farmacêuticas.

Uma classe particular de aplicação das RSSFs na Medicina é o monitoramento do desempenho de atletas, por exemplo, fazendo leituras da taxa de pulsação e de respiração via sensores sem fio e enviando esses valores para um computador base capaz de analisá-los. Outra classe de aplicação é o acompanhamento médico domiciliar, por exemplo, sensores sem fio podem ser usados para monitorar a taxa de glicose e a pressão arterial de um paciente e enviar esses valores para o PDA de um agente de saúde. Outros exemplos de possibilidades de uso das RSSFs na Medicina incluem: a construção de retinas artificiais, o monitoramento de órgãos transplantados e o prognóstico de câncer [Callaway 2003].

Agropecuária Na agricultura, uma aplicação de RSSFs é no controle automático da irrigação. Sensores sem fio podem ser posicionados próximos às raízes das plantas para monitorar a taxa de umidade do solo, e acionar os dispositivos de irrigação quando necessário.

Outra classe de aplicação das RSSFs é no controle e manejo de animais. Sensores colocados em cada animal determinam a necessidade de tratamento contra parasitas e outras infecções. Na pecuária extensiva, os sensores podem ser usados para o rastreamento dos animais. De forma similar, as RSSFs podem ser usadas para monitorar animais silvestres, por exemplo, obtendo informações sobre as rotas de deslocamento desses animais ou sobre os limites geográficos necessários para a sua sobrevivência em grupo [Callaway 2003].

Controle de tráfego O monitoramento e controle do tráfefo de automóveis tem sido implementado, em especial nas grandes cidades, usando câmeras que coletam imagens e enviam para uma central de operação. Entretando, a utilização desses sensores (câmeras) e da rede de comunicação necessária para conectá-los exige, normalmente, um custo financeiro elevado, limitando o seu uso. A utilização de pequenos dispositivos, com capacidade de processamento e comunicação embutida, podem ser dispostos em todas as vias e suas interceções para contar o número de veículos e estimar a velocidade deles. A interação entre grupos de sensores, permite obter uma visão geral do tráfego de veículos e prever

possíveis pontos de congestionamento. Outra possibilidade é a fixação de sensores nos automóveis permitindo que os mesmos troquem informações sobre congestionamentos e planejem rotas alternativas em função dessas informações [Chong and Kumar 2003].

3. Funcionamento, caracterização, topologias e desafios

Nesta seção, discutimos os princípios de funcionamento e as características principais das RSSFs; as topologias mais comuns e os desafios para a construção e operacão dessas redes.

3.1. Conceitos e princípios de funcionamento

Dois conceitos importantes relacionados às RSSFs são os conceitos de *sensor* e de *atuador*. Um *sensor* é um dispositivo que converte energia física, biológica ou química em um sinal elétrico. Um *atuador* é um dispositivo que recebe um sinal elétrico e o converte em um ação física, biológica ou química. Como um exemplo, considere um ambiente industrial com dispositivos que coletam gradientes de temperatura de um líquido inflamável. Quando o gradiente atinge um certo limiar, válvulas de escape precisam ser acionadas para minimizar os riscos de explosão. Nesse cenário, temos *sensores* que continuadamente medem o valor da temperatura, e *atuadores* que interferem no ambiente manipulando as válvulas [Marrón and Minder 2006].

Outros termos usados são: *transducer*, *sensores inteligentes* e *sensores inteligentes enredados*. O termo *transducer* é normalmente usado para referenciar tanto um sensor quanto um atuador. *Sensores inteligentes* são sensores acoplados a micro-processadores, os quais são capazes de prover informações sobre parâmetros de observação ou identificar estados de controle (e.g., em movimento ou em epouso), em outras palavras, sensores permitem aumentar a "inteligência" do sistema. *Sensores inteligentes enredados*, ou RSSFs, têm como objetivo monitorar e/ou controlar um ambiente sem ou com pouca intervenção humana. O suporte para comunicação pode ser oferecido por um tipo especial de MANET (*Mobile Ad Hoc Network*) [Tanenbaum 2003, Callaway 2003], ou por um tipo de WPAN (*Wireless Personal Area Network*) [Tanenbaum 2003, Callaway 2003]. A infra-estrutura e os mecanismos de operação são normalmente dependentes do domínio da aplicação.

Os elementos básicos que compõem os nós que formam as RSSFs são destacados na Figura 1. São eles: processador, memória, fonte de energia, transceptor (envio e recepção de dados) e sensor [Culler et al. 2004].

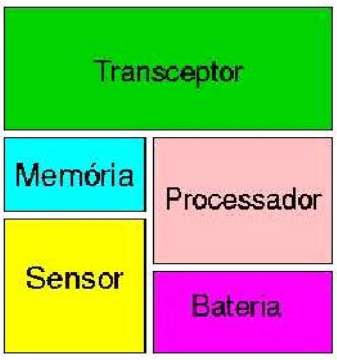

Figura 1. Estrutura básica dos dispositivos que formam as RSSFs.

3.2. Formação das redes

Em boa parte das aplicações com potencial para explorar a tecnologia das RSSFs, um grande número de *nós sensores* são espalhados em uma área definida, e um pequeno número de nós com maior capacidade computacional, denominados *estações base* (e.g., PCs conectados à Internet ou centrais de controle), são dispostos para oferecerem interfaces para os usuários ou observadores da rede. Nessa arquitetura de sistema, um usuário pode obter informações sobre o ambiente físico disparando consultas à rede por meio das estações base. Outra forma de interação entre usuários e a rede é o registro de interesse por determinados eventos. Por exemplo, um usuário pode registrar interesse pelo evento de deslocamento de um objeto em uma determinada área; quando o evento ocorre uma consulta é automaticamente disparada para informar a direção ou a velocidade de deslocamento do objeto.

Antes de enviar informações para as estações base, os sensores dentro de uma determinada área interagem entre si para agregar ou fundir os dados obtidos e gerar a informação esperada. Por exemplo, sensores que detectam a presença de um objeto realizam uma operação de agregaçao de dados para compor características que permitam identificar o objeto. A Figura 2 ilustra a arquitetura de uma rede usada para a monitoração de espécieis.

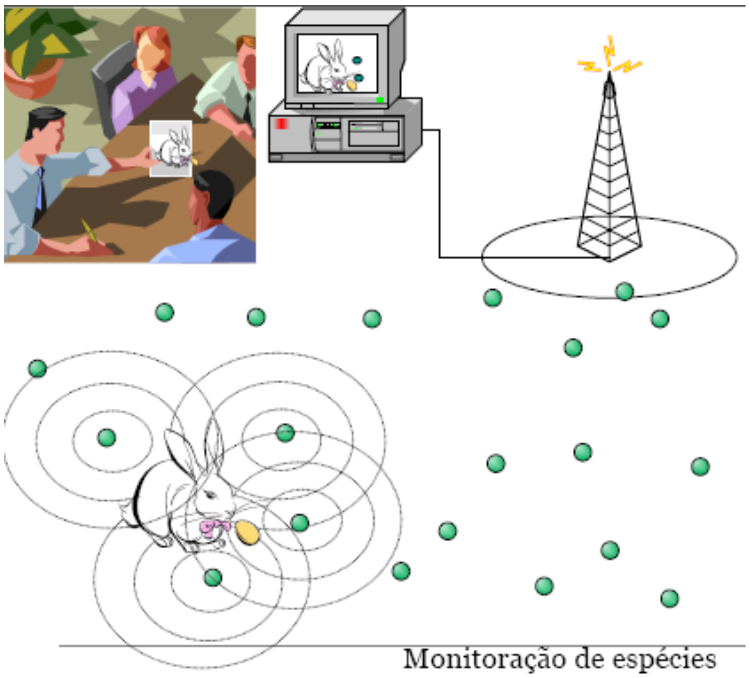

Figura 2. Exemplo de uma rede de sensores para rastreamento de animais.

Em geral, um requisito base para as RSSFs é a necessidade de completar a computação e disseminação dos dados capturados dentro de um intervalo de tempo definido, permitindo que o usuário da rede (um ser humano ou um dispositivo atuador) tome as ações necessárias no tempo devido. Por exemplo, um objeto que está sendo monitorado

pode ter já saído da área de cobertura da rede quando a informação de deslocamento foi recebida, caso o requisito de tempo real não tenha sido atendido. O requisito de tempo real gera o desafio técnico de como coordenar e controlar eficientemente a operação dos nós sensores (captura dos dados, interação com os nós vizinhos, processamento dos dados e disseminação da informação) atendendo a requisitos de tempo real em uma rede sem garantia total de conectividade [Stankovic et al. 2003].

Mecanismos para *auto-configuração* e *auto-adaptação* ganham ênfase no cenário das RSSFs [Prehofer and Bettstetter 2005]: em certos domínios de aplicação (e.g., monitoramento ambiental), a rede precisa se formar sem intervenção humana; e a adição e remoção de nós não deveria exigir a reinicialização da rede [Collier and Taylor 2004]. Outro requisito particular é a adaptação das rotas usadas para a comunicação entre os nós, tomando como base as condições dinâmicas da rede. A Figura 3 [Loureiro et al. 2003] ilustra as etapas para o estabelecimento de uma RSSF especialmente projetada para o monitoramento de um espaço geográfico. A primeira etapa é a definição da região de interesse para o monitoramento; em seguida os sensores são "lançados" nessa área (trata-se aqui de um tipo particular de aplicação onde é inviável o posicionamento manual de cada nó da rede); o próximo passo é a ativação dos nós que começam a executar o código da aplicação; a etapa seguinte destaca a organização dos nós para executar a aplicação definida; e, por fim, a última etapa ilustra a troca de dados entre eles.

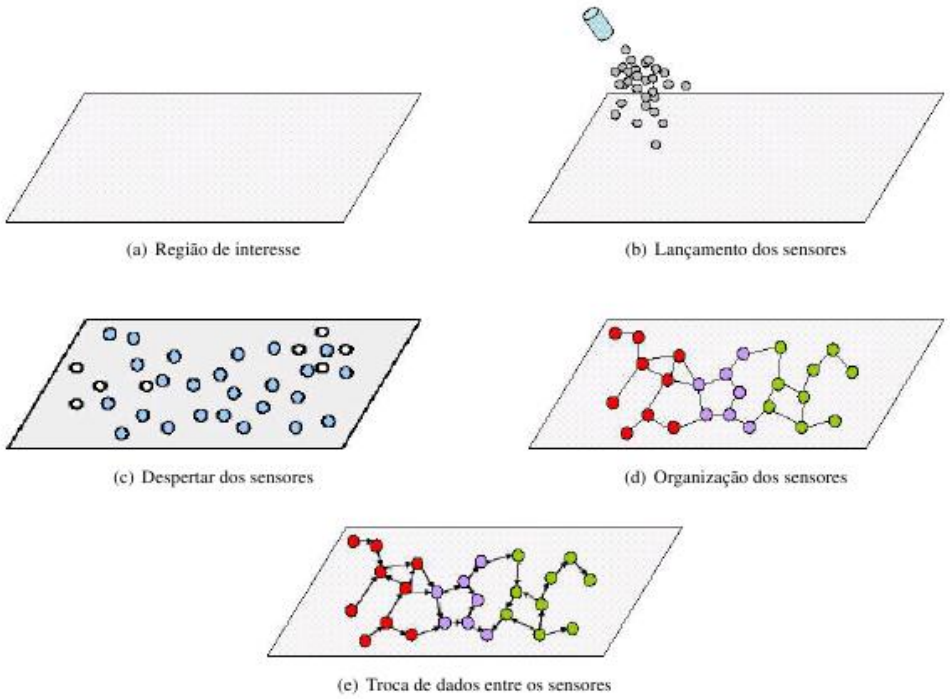

Figura 3. Etapas para o estabelecimento de uma RSSF.

Uma vez estabelecida, a RSSF pode estabeler ligações com redes cabeadas (e com maior capacidade de processamento) e transmitir os dados coletados e pré-processados pelos sensores até o usuário final através dessas redes. A Figura 4 [Akyildiz et al. 2002] ilustra essa idéia. Nesse caso, os nós da rede (*sensor nodes*) coletam informações dentro de uma região específica (*sensor field*). As informações coletadas são enviadas para o

sorvedouro (*sink*) que as encaminha para o usuário final (*user*) por meio de satélites e da Internet (*Internet and satellite*).

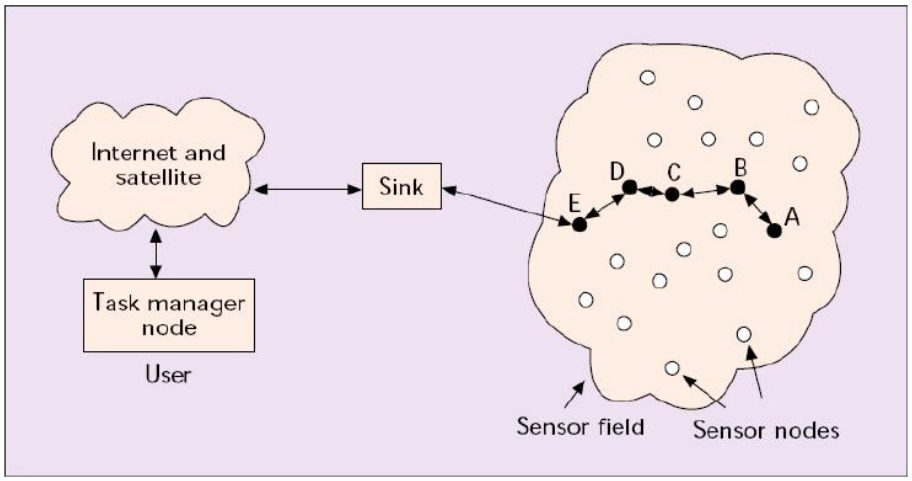

Figura 4. Arquitetura de operação de uma RSSFs.

3.3. Características, topologias e desafios principais

Entre as características particulares das RSSFs, destacam-se:

- Severas restrições de hardware e software;
- Integração com o mundo físico e eventos de tempo real;
- Atividade intermitente dos nós (mesmo sem a mobilidade dos nós, a topologia é dinâmica);
- Grande número de elementos distribuídos;
- Comunicação multi-saltos (a confiabilidade da rede não de expressa na capacidade de garantir a comunicação fim-a-fim entre fonte e destino, mas em mecanismos que garantam tolerância a desconexões em transferências de dados ponto-a-ponto);
- Necessidade de cooperação entre os nós para coletar e processar dados;
- Adaptação às condições de operação (e.g., adaptação das rotas com base em condições dinâmicas da rede, ou adaptação da operação da rede para refletir os interesses do observador).

Como a fonte de energia dos nós da rede é limitada, a *gerência de energia* é uma questão crucial nas RSSFs e requer o desenvolvimento de soluções de software que permitam aos nós se adequarem às condições dinâmicas da rede. As possibilidades de adaptação incluem, por exemplo, a redução do número de nós ativos quando a frequência de ocorrência dos eventos observados dimimui, ou a redução dos intervalos de leitura dos sensores quando a energia residual nos mesmos atingiu um certo limiar. No nível de desenvolvimento de soluções de software, as alternativas para reduzir o consumo de energia envolvem simplificações das soluções e algoritmos em função das aplicações (ou dos domínios de aplicação), além da cooperação entre os nós de forma a aumentar a vida útil da rede (e.g., nós vizinhos alternam períodos de ativação e desativação para recebimento e reenvio de mensagens). A forte limitação do consumo de energia faz com que soluções usadas em outras redes sem fio (e.g., `Bluetooth`) [Tanenbaum 2003, Callaway 2003] não sejam diretamente aplicáveis nas RSSFs.

Os nós que formam as RSSFs são tipicamente envolvidos em diferentes tipos de processamento: (a) processamento relacionado à infra-estrutura da rede, e.g., acesso ao meio e roteamento de mensagens; (b) processamento local dos dados coletados, e.g., tradução e calibração; e (c) processamento de correlação de dados, e.g., fusão, contagem e agregação. Uma característica que distingue as RSSFs das demais redes é que normalmente o interesse não está nos dados coletados por um nó específico, mas sim nos dados coletados e processados por um grupo de nós.

As RSSFs podem ser classificada de acordo com os seguintes aspectos:

- Composição
  - *homogênea*: nós com a mesma capacidade de hardware;
  - *heterogênea*: nós com diferentes capacidades de hardware;
- Organização
  - *hierárquica*: organização dos nós em grupos;
  - *plana*: ausência de organização em grupos;
- Mobilidade
  - *estacionária*: sem mobilidade dos nós, do observador ou do fenômeno monitorado (monitoramento de edifícios, pontes, salas, etc.);
  - *móvel*: com mobilidade dos nós, do observador ou do fenômeno monitorado;
- Densidade
  - *balanceada*: distribuição e concentração dos nós ideal para o objetivo da rede;
  - *densa*: alta concentração de nós por unidade de área (permite redundância, elimina ruídos, estende a vida útil da rede);
  - *esparsa*: baixa concentração de nós por unidade de área;
- Distribuição
  - *irregular*: distibuição não-uniforme dos nós;
  - *regular*: distribuição uniforme dos nós na área monitorada;
- Tarefas executadas
  - *sem variação*: a tarefa executada por cada nó não muda;
  - *com variação*: mudanças das tarefas com o tempo (tarefas esporádicas);
- Intervalo de amostragem
  - *periódica*: intervalos regulares de coleta;
  - *contínua*: coleta continuada;
  - *reativa*: resposta a eventos de interesse ou a requisições de um observador;
  - *tempo real*: maior quantidade de dados possível no menor intervalo de tempo.

Os desafios associados às RSSFs podem ser classificados em três categorias básicas:

1. *Computação*: alocar hardware limitado para atividades concorrentes (leitura dos sensores, processamento e envio de dados), tratar a visualização e o armazenamento de dados, viabilizar a colaboração entre os nós, lidar com ambientes dinâmicos, dispor de mecanismos para descoberta de nomes e serviços, etc.;
2. *Comunicação*: descobrir nós conectados e o melhor caminho entre fonte e destino, tratar o empacotamento dos dados e questões de segurança, lidar com conexões não-confiáveis e com largura de banda limitada, etc.;
3. *Medição*: gerenciar a operação dos sensores, tratar a representação e a aquisição das medidas, lidar com diferentes tipos de dispositivos com calibrações distintas, considerar representações imprecisas do mundo real e incertezas nas medidas, garantir a sincronização/coordenação das ações dos nós, etc..

4. Pilha de protocolos

Assim como nas redes convencionais, as RSSFs também requerem a composição de uma pilha de protocolos para permitir a comunicação entre os nós da rede [Tanenbaum 2003]. Boa parte dos desafios na área de RSSFs está nas camadas de comunicação — as soluções desenvolvidas devem tratar quetões relacionadas ao consumo de energia e ao controle da topologia da rede; e atender requisitos de tempo real.

As características particulares das RSSFs, em particular seu modelo centrado em dados (o foco principal da formação da rede é a coleta de informações pelos nós), faz com que as soluções e protocolos projetados para as redes convencionais precisem ser adaptados ou completamente redesenhados para as RSSFs [Stankovic et al. 2003]. Nesta seção, discutimos as principais características das camadas física, de enlace e de rede nas RSSFs.

4.1. Camada física

A *camada física* em uma pilha de protocolos tem como objetivo transmitir bits através de um canal de comunicação. As soluções desenvolvidas para essa camada dependem do meio físico usado para a comunicação e envolvem interfaces mecânicas, elétricas, e de medidas de tempo [Tanenbaum 2003, Callaway 2003].

Entre os meios de comunicação possíveis de serem usados nas RSSFs, destacam-se [Callaway 2003, Loureiro et al. 2003]:

1. *Óptico*: não requer espaço para uma antena de transmissão, mas precisa de um linha de sinal (alinhamento) entre transmissores e receptores, a comunicação pode ser *passiva* ou *ativa*;
2. *Infra-vermelho*: a comunicação é direcional e tem alcance de $1m$, assim como no meio óptico, não requer espaço para a antena;
3. *Rádio frequência (RF)*: a comunicação é baseado em ondas eletromagnéticas e o maior desafio é o espaço necessário para a antena, contudo é mais fácil de usar, quando comparada aos outros meios de comunicação sem fio, e tem maior aceitação comercial.

Nas RSSFs utiliza-se tipicamente a banda de frequência ISM — banda "não-licenciada" — com os seguintes valores:

- 433MHz e 868MHz na Europa
- 902-928MHz nos EUA
- 2,40-2,48GHz em todo o mundo

As propriedades dos sinais RF dependem da frequência do sinal. Sinais de *baixa frequência* podem atravessar obstáculos, e a potência cai com a distância. Sinais de *alta frequência* geram ondas menos "onduladas", o que dificulta a penetração de obstáculos (fenômeno dos "quiques"), além de serem mais susceptíveis a interferências de equipamentos elétricos.

O rádio pode ser configurado para modos de operação distintos, entre eles: *transmitindo*, *recebendo*, *em espera* (ativado, mas sem transmitir ou receber), *dormindo* (desativado). Os dois últimos modos vizam minimizar o consumo de energia. Entre os modelos comerciais de rádio para os nós sensores, temos o modelo CC2420 (banda ISM de 2,4GHz); e o modelo CC1000 (bandas ISM de 315MHz, 433MHz, 868MHz e 915MHz).

4.2. Camada de enlace

A camada de enlace é responsável pela delimitação de quadros, e pelo controle de erros, de fluxo, e de acesso ao meio de transmissão [Tanenbaum 2003].

As redes sem fio normalmente usam um único canal de rádio com comunicação *half-duplex* (comunicação bidirecional e não-simultânea). O rádio com uma única frequência pode apenas transmitir ou receber dados a cada instante, por isso o protocololo CSMA/CD, usado nas redes cabeadas [Tanenbaum 2003], não se aplica bem a essas redes. Outra consequência do uso de uma única frequência é que as colisões são detectadas apenas nos receptores, já que o transmissor não consegue "escutar" o canal. Além disso, a ausência de um tempo global — característica das redes *ad hoc* — exige que os nós escutem o canal por mais tempo para detectar se o mesmo está ocupado ou não.

Dois problemas particulares que aparecem nas redes sem fio são o *problema do nó escondido* e o *problema do nó exposto*. O *problema do nó escondido* ocorre quando um nó não é capaz de detectar um nó competindo pelo acesso ao meio porque ele está distante (fora do seu raio de cobertura). O *problema do nó exposto* aparece quando um nó detecta uma ocupação falsa do meio, i.e., ele deixa de transmitir porque outro nó no seu raio de cobertura está transmitindo, entretanto o nó destino da sua mensagem não está na linha de alcance do nó que está transmitindo (em sistemas de pequeno alcance, várias transmissões podem ocorrer simultaneamente se elas têm destinos diferentes e esses destinos estão fora de alcance uns dos ountros).

Uma solução adotada para as redes sem fio é adicionar um diálogo *request-to-send-clear-to-send* (RTS-CTS-DATA) ao protocolo CSMA [Tanenbaum 2003]. Essa alternativa é nomeada CSMA/CA (*Collision Avoidance*). Os nós que desejam transmitir um quadro enviam um RTS para o destinatário e esperam um CTS para iniciar a transmissão, ou seja, eles anunciam aos vizinhos quais são os nós "candidatos" a nó transmissor e receptor. Considere como exemplo dois nós A e B. Se **A** envia um RTS para **B**:

- os nós que escutam o RTS estão próximas de A e devem permanecer em silêncio até a recepção do CTS;
- os nós que escutam o CTS estão próximas de B e devem permanecer em silêncio durante a transmissão de dados.

Essa solução trata os problemas dos nó exposto e do nó escondido, mas não elimina completamente a possibilidade de colisões (colisões podem ocorrer com os pacotes RTS).

Uma extensão ao CSMA/CA é o protocolo MACA (*Multiple Access Collision Avoidance*) [Ye and Heidemann 2004]. Esse protocolo estende o CSMA/CA adicionando um campo de duração nos pacotes RTS e CTS, indicando a quantidade de dados para a transmissão. Dessa forma, os outros nós sabem o tempo necessário de "*back-off*" e podem desligar os seus rádios (vão para o modo de operação *dormindo*) durante esse tempo (*escuta de canal virtual*). O protocolo MACAW estende o MACA incluindo uma mensagem `ACK` após cada pacote de dados (RTS-CTS-DATA-ACK) [Bharghavan et al. 1994].

Desafios da camada de enlace nas RSSFs As RSSFs provêem uma infra-esturura de computação e comunicação diferente das demais redes sem fio. Essas diferenças se devem não somente às características físicas dos dispositivos que formas as RSSFs (dispositivos

de tamanho reduzido e severas restrições de consumo de energia), mas também pelas características das aplicações visionadas para essas redes. Por exemplo, aplicações que envolvem o rastreamento de objetos ou a detecção de eventos de interesse (parâmetros que alcançam determinado limiar). Como consequência, os requisitos para a camada de enlace nas RSSFs são diferentes daqueles das redes tradicionais. Destacamos aqui alguns desses requisitos [Stankovic et al. 2003]:

- *Requisito de tempo real*: como base para a pilha de comunicação, a camada de enlace deve garantir que o tempo entre a detecção de um evento, o processamento e a disseminação da informação atendam aos requisitos de tempo real da aplicação;
- *Decentralização*: como as RSSFs podem envolver um grande número de nós sem a garantia de que cada um deles opere de forma continuada (sem desligamento da rede), os algoritmos da camada de enlace devem ser distribuídos;
- *Gerência de energia*: a limitação do consumo de energia tem duas implicações para a camada de enlace — a primeira é que as soluções desenvolvidas para essa camada precisam economizar energia, e.g., evitando a necessidade de escuta contínua do meio; a segunda implicação é que é preciso levar em conta que os nós nem sempre estarão ativos, eles podem estar temporariamente desligados (para economizar energia) ou suas fontes de energia já chegaram ao fim;
- *Flexibilidade*: embora existam aplicações típicas para as RSSFs, certas especificidades podem aparecer na forma como as aplicações exploram a rede, os protocolos de acesso ao meio precisam ser flexíveis para acomodar, por exemplo, padrões distintos de tráfego na rede (tráfigo continuado, tráfego em rajada, etc.).

Exemplos de protocolos para a camada de enlace nas RSSFs incluem o protocolo S-MAC [Ye and Heidemann 2004] e o protocolo B-MAC [Polastre et al. 2004].

4.3. Camada de rede

A camada de rede é responsável por garantir a entrega de pacotes da fonte ao destino, usando ou não roteadores intermediários. Para cumprir essa tarefa, é preciso conhecer a *topologia de comunicação* e escolher os caminhos mais adequados até o destino. Um cuidado importante é não sobrecarregar alguns caminhos, deixando outros com pouco uso [Tanenbaum 2003].

Desafios da camada de rede nas RSSFs Enquanto muitas das idéias aplicadas nos algoritmos de roteamento utilizados nas redes convencionais podem ser adaptadas para as RSSFs, existem diferenças significativas entre essas redes que impossibilitam seu uso direto [Stankovic et al. 2003, Callaway 2003]. As RSSFs são dinâmicas — a possibilidade de nós entrarem e deixarem a rede durante a sua vida útil é constante (mesmo considerando nós estacionários, essa dinamicidade pode ocorrer quando a estratégia de desligar regularmente alguns nós para economizar energia é adotada). Como o meio de comunicação é sem fio, a taxa de mensagens perdidas pode ser alta, além disso, a camada de rede também deve se preocupar com os requisitos de tempo real das aplicações. Em geral, a identificação específica de um nó (identificador ou IP do nó) é menos importante do que a localização geográfica do nó. Por exemplo, no rastreamento de um objeto, a aplicação quer saber apenas *onde* o objeto está, e não *quem* está enviando a informação.

Esses fatores fazem com que as soluções baseadas em *tabelas de rotas*, com manutenção do estado global da rede [Tanenbaum 2003], sejam muito custosas para as

RSSFs, além de poderem não funcionar bem. Em [Costa et al. 2005], discutimos essas questões e sugerimos uma abordagem para a disseminação de informações sem a manutenção de tabelas de rotas. Prover garantias de tempo real é um desafio particular para as RSSFs. Esquemas para reservas de canais em geral não escalam bem devido à necessidade de manter informações sobre o estado das ligações e sinalizar sobrecargas nos canais [Stankovic et al. 2003].

As RSSFs são semelhantes às redes MANETs (*Mobile Ad Hoc Networks*) em relacão à comunicação multi-saltos, mas apresentam características particulares:

- a forma de comunicação típica das RSSFs é *unidirecional*, no sentido dos nós fontes para a estação base (*multicast* invertido);
- a probabilidade de dados redundantes é maior;
- os modelos de coleta de dados dependem do tipo de aplicação (e.g. coleta contínua, periódica ou esporádica);
- existe uma limitação maior do consumo de energia.

Endereçamento e descoberta de nós nas RSSFs O enderaçamento nas RSSFs pode ser de dois tipos principais [Loureiro et al. 2003]: (i) *endereçamento espacial*: baseado nas coordenadas geográficas dos nós (e.g. $47^o39'N$, $122^o18'W$); (ii) *endereçamento com atributos*: utiliza atributos externos à topologia e relevantes para a aplicação (e.g. *próximo ao edifício Y*, *dentro da sala X*)

O roteamento pode ser classificado como: (i) *plano*, a atividade de roteamento é tratada de forma idêntica por todos os nós da rede; ou *hieráquico*, estabelece duas classes de nós (nós fonte e líderes de grupo).

As tarefas de *descoberta* e *localização* de nós pode ser feita das seguintes formas:

- Usar a informação de *nós âncoras* cuja posição (coordenada geográfica) foi configurada na implantação da rede, por exemplo, os nós inferem suas localizações a partir da intensidade do sinal medido (usa-se um modelo de rádio que define a degradação do sinal RF com a distância), ou do número de saltos entre os nós (quando a distância entre os nós é conhecida);
- Usar um protocolo que define endereços locais únicos, i.e., nós vizinhos devem ter endereços distintos (a gerência de endereços locais é mais fácil que a gerência de endereços globais).

Levando em conta as características das RSSFs e as especificidades das suas aplicações, o paradigma de *agregação e/ou fusão* de dados dentro da própria rede destaca-se como uma solução adequada para essas redes. O foco tradicional de protocolos centrados em endereço muda para uma abordagem *centrada em dados*, a qual permite a consolidação de dados redundantes. O paradigma pode ser visto como um conjunto de métodos para combinar dados provenientes de diferentes nós em uma única informação. Um cuidado importante deve ser tomado com relação a dados específicos (e.g. localização dos nós fonte) que podem ser necessários em determinadas aplicações.

4.3.1. Protocolos para a camada de rede das RSSFs

Destacamos nesta seção alguns protocolos de rede que podem ser usados nas RSSFs, suas vantagens e desvantagens.

Protocolo básico de *inundação* Cada nó retransmite todos os pacotes recebidos, a menos que o número máximo de saltos tenha sido atingido, ou o nó destino é o próprio nó [Tanenbaum 2003]. Sua principal vantagem é a de não requerer manutenção da topologia e algoritmos para a descoberta de rotas. As principais desvantagens são:

- *Implosão*: situação onde mensagens duplicadas são enviadas para o mesmo nó;
- *Sobreposição de dados*: se dois nós compartilham a mesma região de observação eles coletam dados similares, como resultado os nós vizinhos recebem mensagens duplicadas;
- *Desperdício de recursos*: não leva em conta a quantidade de energia disponível no nó.

O protocolo de inundação é normalmente usado como referência para a comparação de outras soluções (o que é feito, por exemplo, em [Costa et al. 2005]).

Protocolos com negociação A idéia básica é que os nós enviam dados que descrevem os dados lidos (e.g., imagens), antes de enviar a informação completa. Um exemplo desse tipo de protocolo é a família de protocolos SPIN [Heinzelman et al. 1999]. Três tipos de mensagens são usadas (ADV-REQ-DATA):

- o nó envia um descrição dos dados em uma mensagem ADV;
- o vizinho interessado responde com uma mensagem REQ, e então o dado completo é enviado;
- o vizinho repete o processo para os seus nós vizinhos, disseminando a informação para todos os nós interessados.

Protocolos baseado em *clusters* A rede é organizada como um conjunto de grupos (*clusters*), onde cada nó pertence ao menos a um grupo. Todo grupo tem um líder que exerce controle local sobre o grupo. O líder do grupo é responsável pela agregação dos dados coletados pelos nós do grupo e pela comunicação com os outros grupos. Um exemplo de protocolo de roteamento baseado em *cluster* é apresentado em [Younis et al. 2002].

Protocolo de difusão direcionada Neste protocolo [Intanagonwiwat et al. 2000], os dados gerados pelos sensores são nomeados por pares *atributo-valor* e as requisições são disseminadas na forma de *interesses* por uma determinada informação. O objetivo é economizar energia dos nós, mantendo canais de comunicação eficientes entre os sensores e a estação base. Os interesses são recebidos e armazenados pelos nós; quando um nó possui um dado de interesse de outro nó, ele repassa esse dado para o nó que enviou a requisição. A agregação de dados é favorecida, permitindo aos nós intermediários agregarem seus dados em um único pacote. Essa abordagem evita a redundância de transmissões e o reduz o número de mensagens na rede.

Comparando o paradigma da difusão direcionada com protocolos de roteamento tradicionais, a difusão é centrada em dados e a comunicação é vizinho-a-vizinho e não fim-a-fim como nas redes tradicionais. Não há roteadores, e cada nó é capaz de interpretar os dados e mensagens de interesse. Essa opção justifica-se pelo fato das RSSFs serem específicas para uma tarefa. Os nós não precisam ter uma identificação global única, entretanto precisam distinguir seus vizinhos. A comunicação local implica que a definição

do caminho não usa métricas sobre a topologia global. Isso permite reduzir o custo de manter a informação sobre a topologia da rede, em consequência, os caminhos podem ser sub-ótimos, mas as técnicas de agregação (se bem definidas) podem reduzir essa perda.

5. Padrões para RSSFs

Um esforço tem sido feito para padronizar as diferentes camadas de protocolos de comunicação das RSSFs. O sucesso dessas redes, como tecnologia para o desenvolvimento de aplicações, depende do sucesso desse esforço de padronização, o qual visa basicamente permitir a interoperabilidade entre os dispositivos que formam as RSSFs. Dois desses esforços de padronização são o padrão WPAN IEEE 802.15.4 (*Low Rate Wireless Personal Area Network*) e a família de padrões IEEE 1451 [Callaway 2003].

IEEE 802.15.4 [Callaway 2003] O escopo desse padrão é definir a especificação das camadas física e de enlace para dispositivos com complexidade reduzida, baixo custo, consumo de energia limitado e conectividade via rede sem fio. O padrão prevê uma variedade de aplicações, com requisitos distintos. Por exemplo, aplicações que requerem uma vazão alta de dados com baixo retardo na transmissão, como é o caso dos teclados sem fio; ou aplicações que permitem baixa vazão de dados e toleram atrasos nas transmissões, como é o caso de aplicações para agricultura de precisão e controle ambiental.

O padrão IEEE 802.15.4 permite conexões *estrela* ou *ponto-a-ponto*. Como consequência, permite uma variedade de topologias de rede e algoritmos de roteamento. O acesso aos canais é baseado em contenção, via protocolo CSMA/CA [Tanenbaum 2003]. Para atender às aplicações que requerem retardo reduzido na transmissão de mensagens, o padrão IEEE 802.15.4 permite a reserva de fatias de tempo do canal de comunicação para dispositivos individuais (nesse caso, o protocolo CSMA/CA não precisa ser seguido).

O IEEE 802.15.4 incorpora características para minimizar o consumo de energia. Entre elas está a definição de um byte, anexado a cada quadro recebido pela camada física antes de ser enviado para a camada de enlace, com a indicação da qualidade da ligação. O nó que recebe essa informação pode usá-la para:

- indicar a qualidade do canal de comunicação;
- controlar a energia do nó transmissor;
- auxiliar algoritmos de localização (calcular a distância entre os nós vizinhos em funçao da força do sinal recebido) e roteamento (estabelecer rotas de acordo com a qualidade dos canais).

O padrão IEEE 802.15.4 incorpora duas camadas físicas: (i) *banda baixa*, 868.0-868.6MHz (Europa) e 902-928MHz (América); (ii) *banda alta*, 2.4GHz (todo o mundo).

ZigBee Alliance [Callaway 2003] O padrão IEEE 802.15.4 não padroniza as camadas de comunicação mais altas (rede e transporte). Para garantir a interoperabilidade entre dispositivos que operam no padrão IEEE 802.15.4, o comportamento dessas camadas também precisa ser especificado. A criação dessa especificação tem sido objeto de estudo de um consórcio, denominado *ZigBee Alliance*, formado por fabricantes de dispositivos, provedores de serviços, usuários de aplicações, etc. (muitos deles envolvidos também na definição do padrão IEEE 802.15.4).

Família IEEE 1451 [Callaway 2003] O desenvolvimento de padrões para as RSSFs não é conduzido apenas para a interface de comunicação. A família de padrões IEEE 1451 tem sido desenvolvida para tratar as questões de compatibilidade entre sensores e atuadores. Os fabricantes encontravam dificuldades para desenvolver dispositivos compatíveis com os diferentes protocolos de comunicação em rede. A solução foi o desenvolvimento de uma interface que consiste de um protocolo simples de comunicação usado por todos os sensores.

Entre os benefícios dessa padronização, está a criação do TEDS (*Transducer Electronic Data Sheet*). O TEDS provê um caminho através do qual sensores e atuadores podem descrever eles mesmos para os sistemas de medição, de controle e, de forma geral, para qualquer dispositivo na rede de interconexão. Exemplos de parâmetros de descrição de um dispositivos incluem: fabricante, calibração, parâmetros de desempenho, entre outros. Com esse tipo de padronização, a comunicação na rede fica independente do tipo do dispositivo (e.g., termômetro, barômetro, controlador de robôs, etc.).

6. Sistemas operacionais

Os Sistemas Operacionais (SOs) facilitam o desenvolvimento de aplicações provendo uma abstração conveniente e segura dos recursos de hardware [Silberschatz et al. 2004]. SOs para PCs e servidores alocam *threads* de execução para os processos, mapeiam endereços virtuais para localizações na memória, manipulam o acesso aos discos, à rede e a outros periféricos. Essa separação entre SO e aplicações (essencial na computação convencional) é menos comum nos sistemas embutidos, onde as aplicações estão mais diretamente ligadas a um hardware particular. Isso se deve, em parte, às limitações dos recursos de hardware e ao fato das aplicações serem altamente especializadas.

RSSFs são sistemas embutidos, porém de propósito geral, permitindo uma variedade de aplicações e componentes heterogêneos. Um nó da rede normalmente precisa gerenciar vários dispositivos, incluindo sensores, rádio, memória e processador. Quando uma operação é iniciada em um dispositivo, esse dispositivo passa a executar concorrentemente com o processador principal até terminar. Essas características conduzem à necessidade de desenvolver sistemas operacionais específicos para redes de sensores.

Alguns exemplos de SOs desenvolvidos para RSSFs são: TinyOS [Hill et al. 2000], Mantis [Bhatti et al. 2005], e SOS [Han et al. 2005]. Nesta seção, discutimos as diferentes estratégias adotadas por cada um deles para lidar com os principais requisitos dos dispositivos que formam as RSSFs.

6.1. TinyOS

O TinyOS [Hill et al. 2000] implementa um modelo baseado em componentes, com interfaces de duas fases, comunicação baseada em eventos e um mecanismo para postergar a execução de chamadas de procedimentos. O TinyOS pode ser definido como um *framework* de programação para redes de sensores, com um conjunto de componentes que permite construir um sistema operacional específico para cada aplicação, i.e., apenas os componentes do sistema operacional necessários para uma aplicação específica, são compilados junto com a aplicação.

Um programa TinyOS consiste de um grafo de componentes de software que usam três abstrações de programação: *comandos*, *eventos* e *tarefas*. *Comandos* são usados

para requisitar serviços, por exemplo, o envio de uma mensagem, e terminam imediatamente. Tipicamente, o código associado a um comando armazena os parâmetros de uma requisição e condicionalmente escalona uma tarefa para ser executada posteriormente. Um *evento* sinaliza o término de um serviço, como por exemplo, o envio de uma mensagem, ou a ocorrência de um evento de hardware. Um tratador de evento pode depositar informações no ambiente do componente, escalonar tarefas, sinalizar outros eventos ou chamar comandos. *Tarefas* são as unidades básicas de execução (procedimentos) que permitem postergar a execução das chamadas de procedimentos. Quando uma tarefa é chamada, ela não é executada imediatamente, mas sim escalonada para execução futura. Essa abordagem permite concluir a rotina de tratamento de um evento (e.g., o recebimento de uma mensagem), e posteriormente manipular o conteúdo da mensagem (tarefa mais longa). A Figura 5 ilustra a relação entre essas abstrações de programação.

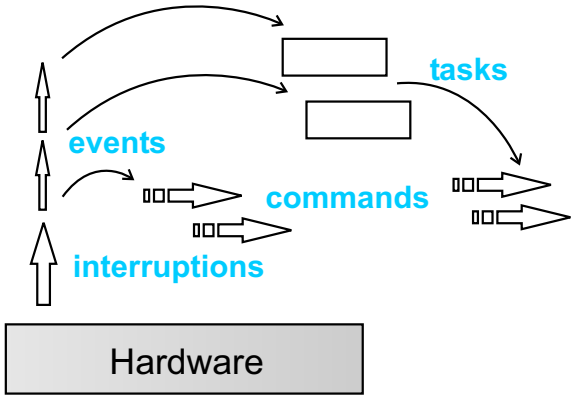

Figura 5. Abstrações de programação do TinyOS.

O TinyOS executa uma única aplicação com duas linhas de execução: as *tarefas* e os tratadores de *eventos de interrupção*. Uma vez escalonada, uma tarefa executa até terminar, ou seja, não existe preempção entre as tarefas. Os tratadores de eventos de hardware são executados quando uma interrupção de hardware ocorre e também executam até terminar, mas podem interromper as tarefas e outros tratadores de interrupção.

Para garantir que a execução de uma tarefa não postergue indefinidamente a execução de outras tarefas, o código executado por elas deve ser curto, i.e., operações longas devem ser particionadas em várias tarefas. Além disso, como uma tarefa precisa terminar para que a próxima seja tratada, o código de uma tarefa não pode bloquear ou ficar em espera ocupada. O laço principal de uma aplicação TinyOS implementa um *escalonador de tarefas* responsável por escalonar as tarefas para execução sempre que o processador torna-se disponível. Na Seção 10, usamos o TinyOS para implementar uma aplicação para RSSFs e aproveitamos para aprofundar um pouco mais a discussão sobre as suas características.

6.2. Mantis

O objetivo principal de projeto do Mantis [Bhatti et al. 2005] é reduzir a curva de aprendizado necessária para o desenvolvimento de protótipos e de aplicações básicas para redes de sensores. Para isso, o sistema operacional Mantis adere ao modelo clássico de sistemas

operacionais. Em particular, adota-se a programação *multithreading*, com filas de prioridade, e o mecanismo de semáforos para a sincronização entre os processos. A Figura 6 mostra a arquitetura básica do Mantis.

A gerência das *threads* é sustentada por um sub-conjunto da API de *threads* do POSIX. O desafio principal é como adequar esse modelo de programação com as restrições severas de recursos computacionais dos sensores, e integrar o modelo de *multithreading* com a idéia de *threads* que "dormem" enquanto nenhum trabalho é solicitado. O kernel mantém uma tabela de *threads*, alocada estaticamente, com uma entrada para cada *thread*. Assim, o número máximo de *threads* no sistema é pre-determinado (o número padrão é 12).

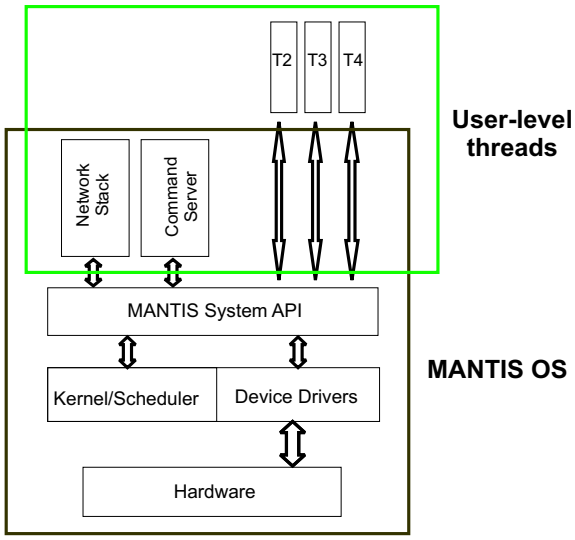

Figura 6. Arquitetura básica do sistema operacional MantisOS.

6.3. SOS

O SOS [Han et al. 2005] prioriza a adaptação dinâmica das aplicações. O projeto do SOS foi inspirado na arquitetura do TinyOS — ele define um modelo de componentes e é um sistema operacional dirigido a eventos. A principal diferença é que o SOS permite instalar e modificar os componentes de uma aplicação depois de implantada. Para favorecer a dinamicidade do sistema, o SOS mantém uma estrutura modular e um mecanismo de escalonamento de mensagens com prioridades. A Figura 7 mostra a arquitetura básica do SOS.

O sistema consiste de módulos que podem ser carregados em tempo de execução e de um kernel comum que implementa troca de mensagens, memória dinâmica, carga de módulos, entre outros serviços. Um módulo implementa uma tarefa ou função específica, e pode ser comparado em funcionalidade com um componente TinyOS. A interação entre os módulos pode se dar de duas formas: através de chamadas diretas a funções dentro dos módulos ou através de troca de mensagens. Um esquema de registro de funções é implementado pelo kernel através do qual os módulos registram suas funções públicas ou requisitam a referência para funções oferecidas por outros módulos.

Quando uma função é invocada, o controle é transferido para o módulo que implementa a função, caracterizando um tipo de interação síncrona entre os módulos. As

mensagens, por outro lado, são assíncronas e reproduzem o comportamento das tarefas no TinyOS. O SOS implementa um tipo de escalonamento cooperativo depositando as mensagens em filas de prioridade para serem executadas posteriormente. O escalonador principal do SOS seleciona uma mensagem de uma fila de prioridade e a remete para o tratador de mensagens do módulo destinatário.

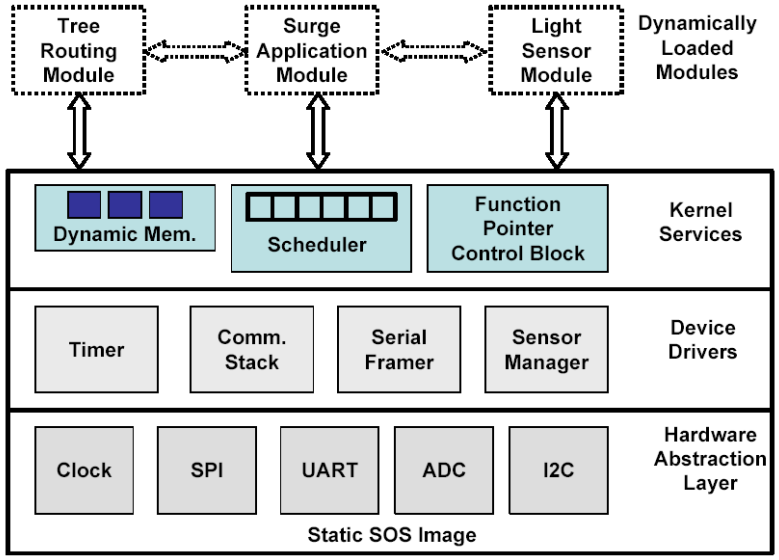

Figura 7. Arquitetura básica do sistema operacional SOS.

7. Algoritmos de localização e sincronização

Nesta seção destacamos aspectos particulares para o tratar as questões de localização e sincronização nas RSSFs.

7.1. Algoritmos de localização

Certas aplicações precisam saber "onde" um evento foi detectado (um foco de incêncio, por exemplo), ou classificar/reconhecer objetos localizados para tomar uma ação particular, ou ainda requisitar dados de uma área geográfica específica. Para isso é preciso conhecer a *localização física* no espaço [Hightower and Borriello 2001]. A informação de localização permite ainda estudar as propriedades de cobertura da rede e podem auxiliar os algoritmos de roteamento.

Um sistema de localização pode prover dois tipos de informação:

- *física* (coordenadas geográficas, e.g. $47^o39'17''N$, $122^o18'23''W$);
- *simbólica* (idéias abstratas, e.g. próximo ao edifício Y, dentro da sala X).

As aplicações podem gerar/inferir informações simbólicas a partir da localização física, por exemplo, qual é a impressora mais próxima do usuário, ou que atividade uma pessoa deve estar executando.

Um sistema de localização pode ser classificado como *absoluto* ou *relativo*. Em um *sistema de localização absoluta*, usa-se a mesma referência para todos os objetos. Em um *sistema de localização relativa*, cada objeto tem sua própria base de referência (e.g.

na busca por vítimas em um desastre, o operador pode obter a posição das vítimas em relação a ele).

Alguns sistemas estabelecem que cada objeto deve inferir sua própria localização. Essa abordagem garante privacidade, uma vez que nenhuma entidade pode conhecer a localização do objeto, a menos que o próprio objeto decida publicar essa informação. Outros sistemas requerem que os objetos periodicamente emitam informações para que a infra-estrutura externa os localizem (e.g. sistemas de localização pessoal com *badges*). Nessa abordagem, a carga computacional nos objetos é reduzida.

As abordagens para descoberta de localização consitem de duas fases: (i) estimativa da distância; e (ii) combinação das distâncias. Nas RSSFs, os métodos mais comuns para estimar a distância são:

- *Indicador de intensidade do sinal* (RSSI);
- *Tempo de propagação*;
- *Ângulo de recepção do sinal*.

Para a fase de combinação, as alternativas mais comuns incluem:

- *Triangulação*: interceção entre círculos ou relações trigonométricas;
- *Análise de cena*: requer o conhecimento de características do ambiente para comparar as cenas (uma cena pode corresponder a fenômenos físicos, por exemplo, a intensidade do sinal);
- *Proximidade*: um dispositivo próximo a um ponto de referência.

O GPS (*Global Positioning System*) é um sistema de navegação por satélite que auxilia na descoberta de localização. Ele é constituído de uma rede de 24 satélites, colocados em órbita pelo Departamento de Defesa Americano. Os satélites circulam em uma órbita precisa transmitindo sinais para a Terra (funciona em qualquer parte do mundo, 24h por dia). Os receptores GPS comparam o tempo que o sinal foi transmitido com o tempo de recepção e então calculam a distância do satélite. Comparando a distância de outros satélites, o receptor determina a posição do usuário. As principais dificuldades com relação ao uso do GPS nas RSSFs são: os sinais emitidos pelos satélites não alcançam ambientes fechados (ou com vegetação densa), o uso de GPS requer consumo alto de energia e antenas com tamanho capaz de captar os sinais dos satélites.

Algumas questões que precisam ser observadas nos sistemas de localização para RSSFs envolvem o fato dos dispositivos serem de tamanho reduzido (baixa potência), com transmissor/receptor RF (alcance radial). Para uma solução ser escalável, a descoberta da localização deve ficar a cargo dos receptores e não dos pontos de referência. A preocupação em encontrar soluções com baixo consumo de energia deve ser levada em conta, e, por fim, a granularidade da localização vai depender do número de pontos de referência disponíveis.

Rádio idealizado e localização baseada em conectividade Um modelo de rádio idealizado é definido por Buluso et.al. [Bulusu et al. 2000]. Os autores propõem um método de localização baseado em conectividade. A idéia é explorar a comunicação RF entre os dispositivos. Um número fixo de nós na rede, com regiões de cobertura que se sobrepõem, são usados como *pontos de referência*. Esses nós transmitem sinais (*beacons*)

periódicos e usam uma métrica de conectividade simples para inferir a proximidade a um subconjunto desses pontos de referência e então se localizarem.

O modelo de rádio idealizado tem as seguintes características:

- rádio esférico de propagação perfeito;
- raio de transmissão idêntico para todos os rádios.

Esse modelo idealizado não é apropriado para ambientes fechados onde pode ocorrer reflexão dos sinais de rádio. Para ambientes externos, o modelo requer ausência de obstruções.

O conjunto de *pontos de referência* são situados em posições conhecidas (x_1, y_1) a (x_n, y_n) e transmitem *beacons* periódicos com suas posições. Assume-se que os pontos de referência vizinhos são sincronizados para evitar que os sinais de *beacons* se sobreponham. Dentro de um intervalo T, todos os pontos de referência transmitem um sinal de *beacon*. Cada nó móvel escuta o meio durante um intervalo de tempo fixo t e coleta todos os *beacons* que ele recebe dos pontos de referência. O nó receptor infere a *proximidade* a um conjunto de pontos de referência para os quais a conectividae excede um limiar (e.g. 90%). A localização do nó é definida, então, como a região de interceção das regiões de conectividae dos pontos de referência.

A localização dos pontos de referência pode ser pré-determinada ou definida via GPS. A distribuição não-uniforme dos nós requer maior densidade da rede. A solução apresentada requer a sincronização do parâmetro T e da taxa de amostragem dos *beacons*.

7.2. Algoritmos de sincronização

A *sincronização do tempo* visa prover uma escala de tempo comum para os nós de uma rede. Conhecer o momento em que um evento ocorreu permite:

- Co-relacionar informações sobre a atividade dos nós;
- Ter uma visão global do sistema observado;
- Determinar causalidade;
- Reconstruir a sequência exata dos eventos.

Nas RSSFs, uma das possibilidades trazidas com a sincronização é a integração dos dados de vários sensores. Essa integração permite: (a) previnir notificações redudantes; (b) combinar valores para oferecer uma visão mais precisa dos fenômenos monitorados; e (c) saber se os sensores estão tratando o mesmo evento [Sivrikaya and Yener 2004, Elson and Romer 2003, Elson and Estrin 2001].

Os dispositivos computacionais são normalmente equipados com um oscilador que implementa uma aproximação $C(t)$ do tempo real t. A frequência angular do oscilador determina a taxa na qual o relógio executa. Todos os relógios são sujeitos a um "desvio" (*clock drift* ou *clock skew*): frequência de oscilação que varia com as condições físicas. Assim, o relógio local pode ser aproximado como: $C_i(t) = a_i t + b_i$, onde a_i é a frequência do relógio e b_i é a diferença em relação ao tempo real.

Os relógios locais de dois nós da rede podem ser comparados através da equação: $C_1(t) = a_{12}C_2(t) + b_{12}$, onde a_{12} é o desvio relativo e b_{12} o *offset* relativo entre os relógios dos nós 1 e 2. Se dois relógios estão perfeitamente sincronizados, então: (i) os relógios têm a mesma frequência de oscilação; (ii) o *offset* relativo é 0, i.e., os relógios têm o mesmo valor a cada instante.

O problema da sincronização em uma rede com n dispositivos corresponde ao problema de equalizar os relógios dos computadores dos diferentes dispositivos. Equalizar os valores instantâneos (corrigindo o *offset*) não é suficiente para a sincronização. Um esquema de sincronização deve: (i) equalizar as *frequências dos relógios* e os *offsets*; ou (ii) repetidamente corrigir os *offsets* para manter os relógios sincronizados todo o tempo. A sincronização pode ser (i) *global*: tenta equalizar $C_i(t)$ para todos $i = 1..n$; ou (ii) *local*: tenta equalizar $C_i(t)$ para um conjunto de nós.

As soluções de sincronização podem ser classificadas em três tipos básicos:

1. *Ordenar eventos ou mensagens*: compara os relógios apenas para ordenar os eventos $E_1, E_2..E_n$ e não para sincronizá-los;
2. *Manter relógios relativos*: cada dispositivo usa seu próprio relógio mas mantém a informação sobre o desvio e *offset* relativo em relação aos demais nós;
3. *Manter relógios sincronizados*: a meta é preservar uma escala de tempo global na rede.

O não-determinismo da rede faz a sincronização uma tarefa difícil. O envio de um pacote com *timestamp* para sincronização pode sofrer atrasos variáveis até chegar ao receptor. As principais fontes de erro nos métodos de sincronização:

- *Tempo de envio*: construção da mensagem e acesso ao rádio;
- *Tempo de acesso*: espera pelo meio;
- *Tempo de propagação*: depende da velocidade de transmissão;
- *Tempo de recepção*: tratamento da mensagem.

Um dos trabalhos pioneiros na construção de uma solução de sincronização para as RSSFs propõe que os nós sejam sincronizados apenas quando necessário (sincronização *post-facto*) [Elson and Estrin 2001]. Após a ocorrência de um evento, estima-se o *offset* dos nós no momento do evento. Considera-se que os nós estão normalmente fora de sincronismo. Quando um estímulo é recebido, cada nó armazena seu *timestamp* e um nó age como um *beacon*: difunde um pulso de sincronização, os nós que recebem esse pulso sincronizam momentaneamente seus relógios. No restante dessa seção, destacamos outros dois protocolos de sincronização para RSSFs: o RBS [Elson et al. 2002] e o TPSN [Ganeriwal et al. 2003].

Protocolo de sincronização RBS A idéia central do protocolo RBS(*Reference-Broadcast Synchronization*) [Elson et al. 2002] é usar um "terceiro elemento" para a sincroniação: ao invés de sincronizar emissor/receptor, sincroniza um conjunto de receptores. Os nós periodicamente difundem *beacons* para os seus vizinhos. Os nós *receptores* usam o tempo de chegada como referência, assim elimina-se o não-determinismo do lado do emissor. Os nós vizinhos trocam entre si seus tempos de recepção e tentam estimar os *offsets* relativos. A vantagem dessa abordagem é a de remover as fontes não determinísticas do problema. A desvantagem é a sobrecarga da troca de mensagens.

Protocolo de sincronização TPSN O protocolo TPSN (*Timing-Sync Protocol for Sensor Networks*) [Ganeriwal et al. 2003] funciona em duas fases:

1. *Descoberta de nível*: cria uma topologia hierárquica da rede e associa cada nó a um nível;

2. *Sincronização*: o nó no nível i se sincroniza com o nó no nível $i-1$.

Na descoberta de nível, um nó é escolhido como raiz da árvore (nível 0). O nó raiz inicia o algoritmo enviando um pacote *level_discovery* que contém a identidade e nível do emissor. Os vizinhos do nó raiz se atribuem o nível 1 e enviam também um pacote *level_discovery*. Quando um nó define seu nível, ele descarta novas mensagens de descoberta.

Na fase de sincronização, pares de nós trocam duas mensagens entre si (assume-se que o desvio do relógio e o atraso de propagação é constante durante o intervalo da troca de mensagens). Um dos nós inicia o algoritmo enviando um pacote *time_sync* no tempo local T_1 (o pacote inclui o nível e o valor de T_1). O outro nó recebe esse pacote no instante $T_2 = T_1 + \Delta + d$, onde Δ é o desvio relativo dos relógios e d é o atraso de propagação. Uma resposta é enviada no tempo T_3, incluindo os valores T_1, T_2 e T_3.

8. Modelos de programação

Diferentes modelos de programação têm sido estudados e propostos para as redes de sensores [Kasten and Romer 2005, Welsh and Mainland 2004] [Hadim and Mohamed 2006, Heinzelman et al. 2004] [Liu and Martonosi 2003]. Nesta seção, destacamos dois desses modelos: o modelo de base de dados, implementado pelo TinyDB [Madden et al. 2005]; e o modelo de máquina virtual, implementado pelo Matè [Levis and Culler 2002].

TinyDB O TinyDB [Madden et al. 2005] é um exemplo de modelo de programação para RSSFs que trata a rede como uma base de dados virtual. A idéia central dessa abordagem é usar técnicas tradicionais de processamento de dados, com foco em baixo consumo de energia. Dado um conjunto de dados, aplica-se técnicas de agregação e filtragem de dados antes disseminá-lo, visando minimizar a comunicação na rede.

A proposta do TinyDB consiste em explorar características especiais das redes de sensores (além das técnicas tradicionais), por exemplo, o fato dos sensores terem controle sobre quando, onde e como os dados são coletados e enviados para processamento; ao invés de assumir a existência dos dados a priori.

Algumas questões que precisam ser tratadas nesse modelo são:

1. Quando as amostras para um requisição particular deveriam ser feitas?
2. Quais nós possuem dados significantes para uma requisição particular?
3. Em qual ordem as amostras deveriam ser feitas e como a amostragem deveria ser intercalada com outras operações?
4. Vale a pena o custo computacional e de comunicação para processar uma amostra particular?

No TinyDB, as tuplas com dados dos sensores são armazenadas em uma tabela chamada *sensors* que implementa uma linha por nó (dispositivo) e uma coluna para cada atributo (ex., luminosidade, temperatura, etc.). As entradas nessa tabela são adquiridas apenas quando necessárias para processar uma requisição, e são normalmente armazenadas por um curto intervalo de tempo ou disseminadas diretamente na rede. Fisicamente, a tabela *sensors* é particionada entre todos os dispositivos da rede, i.e., cada dispositivo produz e armazena suas próprias leituras. Assim, para comparar leituras de diferentes sensores é preciso coletá-las em algum nó comum.

As requisições no TinyDB são similares a requisições SQL, e consistem de cláusulas SELECT-FROM-WHERE-GROUPBY. As tuplas são produzidas em intervalos de amostragens bem definidos, os quais são passados como parâmetro para as requisições. Um exemplo de requisição é mostrado a seguir:

```
SELECT nodeid, light, temp
FROM sensors
SAMPLE PERIOD 1s FOR 10s
```

O TinyDB explora a possibilidade de *agregagação de dados*. A medida que os dados percorrem a árvore de roteamento (construção lógica definida para as consultas), eles são agregados de acordo com a função de agregação e particionamento especificado na requisição. Uma diferença semântica entre as requisições SQL e TinyDB é que a saída é uma sequência de valores, ao invés de um único valor agregado. O código abaixo mostra um exemplo de agregação de dados:

```
SELECT AVG(temp), room
FROM sensors
WHERE floor = 6
GROUP BY room
HAVING AVG(temp) > threshold
SAMPLE PERIOD 30s
```

Nesse exemplo, os sensores no sexto andar são particionados de acordo com a sala onde eles estão localizados. O resultado da requisição reporta todas as salas onde a média da temperatura é maior que o limiar dado (*threshold*). As atualizações são enviadas a cada 30s.

O TinyDB permite a definição de *eventos* como mecanismo para iniciar uma coleta de dados. Os eventos podem ser gerados explicitamente, como resposta de outra requisição ou pelo sistema operacional. No código abaixo a sinalização de um evento é usada para iniciar uma requisição:

```
ON EVENT bird-detect(loc):
SELECT AVG(light), AVG(temp), event.loc
FROM sensors AS s
WHERE dist(s.loc, event.loc) < 10m
SAMPLE PERIOD 2s FOR 30s
```

Sempre que o evento *bird-detect* ocorre, a requisição é emitida a partir do nó que detectou o evento. Os valores médios de luminosidade e temperatura são coletados de nós a uma distância máxima de 10m, a cada 2s, durante 30s. O uso de eventos permite que o sistema permaneça em modo *sleep*, ao invés de ficar testando uma condição continuadamente.

Os eventos também podem servir como condição de parada, acrescentando uma cláusula do tipo:

```
STOP ON EVENT(param) WHERE cond(param)
```

Finalmente, uma requisição pode sinalizar um evento, como mostrado a seguir:

```
SELECT nodeID, temp
WHERE temp > threshold
OUTPUT ACTION SIGNAL hot(nodID, temp)
SAMPLE PERIOD 10s
```

Nesse caso, os eventos são sinalizados localmente (para o próprio nó).

As requisições no TinyDB podem ser classificadas como:

- *Monitoramento*: requisitam um valor continuadamente sobre as condições da rede, por exemplo, bateria dos nós, nós vizinhos em uma determinada topologia;
- *Exploratória*: examinam o estado de um nó em um dado instante;
- *Aninhada*: são possíveis via pontos de materialização e eventos;
- *Atuadora*: os usuários podem determinar uma ação física, por exemplo, ligar ou desligar um sensor;
- *Entrega postergada*: quando a frequência de um fenômeno é maior que a capacidade de transmissão do rádio.

Quando um nó recebe uma requisição, ele deve decidir se essa requisição se aplica localmente e/ou deve ser retransmitida para os nós filhos na árvore de roteamento. Uma requisição se aplica a um nó se existe a probabilidade do nó produzir os resultados para a requisição (em alguns casos pode não ser uma decisão simples). Se a requisição não se aplica, e o nó não possui filhos para os quais a requisição se aplica, então a sub-árvore pode ser "podada" minimizando os custos. Para manter informações sobre os valores dos atributos dos nós filhos, o TinyDB usa uma estrutura de dados chamada árvore de roteamento semântica *semantic routing tree* (SRT).

Usando a árvore de roteamento semântica, a escolha do nó pai leva em conta não apenas a qualidade das ligações entre os nós, mas também propriedades semânticas. Cada nó armazena um intervalo unidimensional representando os valores que um atributo A pode assumir em cada um dos nós filhos. Quando uma requisição q com predicado definido sobre A chega no nó n, esse nó verifica se os possíveis valores de A nos seus filhos podem satisfazer à requisição: (i) se sim, o nó se prepara para receber os resultados e repassa a requisição; (ii) se não, a requisição não é repassada.

O processamento das requisições consiste de uma sequência de operações em cada nó durante um intervalo de tempo. Quando os nós "acordam", eles lêem os sensores e aplicam os operadores sobre os dados gerados localmente e recebidos de seus vizinhos, e então enviam os resultados para seu nó pai. A cláusula ON EVENT deve ser tratada especialmente: quando um evento ocorre em um nó, esse nó dissemina a requisição, especificando ele mesmo como a raiz; o nó coleta os resultados e os envia para a estação base ou para um ponto de materialização.

Algumas questões que precisam ser observados referem-se ao custo para manter a árvore (entrada e saída de filhos) e o número de mensagens adicionais para avisar o pai que ele tem um filho (dependência da taxa de troca de pai). O uso da árvore semântica de decisão é uma solucao boa quando, por exemplo, coordenadas geográficas fixas estão envolvidas.

Matè Outro modelo de programação para RSSFs é o *modelo de máquina virtual*. O foco principal nesse caso é a necessidade de reprogramar a rede de sensores, ajustando seus parâmetros até a recarga completa da aplicação. Como o custo de comunicação nas redes de sensores é alto, o uso de máquinas virtuais é um caminho para representar de forma concisa os programas.

O interpretador de *bytecode* Maté [Levis and Culler 2002], projetado como um componente do sistema operacional TinyOS, implementa uma máquina virtual que oferece um conjunto reduzido de primitivas de alto nível. O objetivo principal é reduzir o tamanho das aplicações desenvolvidas para as redes de sensores, e então minimizar o custo de energia para transmití-las pela rede. Usando uma representação mais concisa e de alto nível, os autores argumentam que além de facilitar a reprogramação dos nós da rede, é possível simplificar a interface de programação do TinyOS.

Os programas são particionados em cápsulas com 24 instruções (cada uma com 1 byte de tamanho). Esse limite permite que uma cápsula seja empacotada dentro de uma mensagem TinyOS. O modelo de comunicação usa a abstração de mensagens ativas, permitindo que o programa envie uma mensagem que é automaticamente roteada até o seu destino. Do outro lado, quando um pacote é recebido, ele é automaticamente enfileirado como uma tarefa a ser processada.

O Maté define três contextos distintos de execução que correspondem a três eventos: temporização, recepção de mensagem ou requisição de envio de mensagem. Cada contexto possui duas pilhas: uma de operandos, usada pelas instruções para manipular dados; e outra de endereços de retorno, usada para chamar subrotinas. A pilha de eventos de temporização persiste entre várias execuções, de forma que os valores deixados na pilha podem ser usados em diferentes invocações. Os contextos usados para receber e enviar mensagens estabelecem que os dados que devem ser manipulados estarão sempre no topo da pilha e por isso essas pilhas não são persistentes.

As instruções definidas pelo Maté escondem o assincronismo da programação com o TinyOS. Por exemplo, quando o comando para enviar uma mensagem é chamado, o contexto de execução é suspenso até que o evento que sinaliza que a mensagem foi enviada seja recebido. O mesmo tratamento é dado para a requisição da leitura de um valor pelos sensores: a requisição é feita ao componente apropriado e o contexto é suspenso até que o valor esteja disponível.

O exemplo abaixo ilustra o uso do Matè:

```
gets    # obtém um contador
pushm   # obtém um buffer
clear   # esvazia o buffer
add     # incrementa o contador
send    # envia um pacote com o valor do contador
```

9. Segurança

Os requisitos básicos de segurança em uma rede incluem:

- *Confidencialidade*: não permitir que os dados sejam lidos por terceiros;
- *Autenticação*: garantia da origem da mensagem, i.e., partes não Autorizadas não devem participar da rede (a necessidade de autenticidade pode ser maior que a de confidencialidade, por exemplo, em um alarme de incêndio);
- *Integridade*: as mensagens não podem ser alteradas;
- *Garantia de dados atuais*: não permitir a repetição de mensagens antigas.

Nas RSSFs, o processamento "in-network" para agregação/fusão de dados difere-se do padrão de comunicação fim-a-fim das redes convencionais que adotam mecanismos

de segurança como SSH, SSL/TLS, IPSec, etc. [Tanenbaum 2003]. A natureza do meio de transmissão sem fio, usado nas RSSFs, permite que uma transmissão seja "escutada" por todos os receptores dentro do raio de alcance do sinal. Essa possibilidade implica em maior vulnerabilidade a ataques de segurança. Além disso, diferente das redes convencionais, as RSSFs podem ser implantadas em áreas acessíveis, implicando em maior risco de ataques físicos (remoção dos nós da rede). Com a limitação de recursos dos nós, as RSSFs tornam-se vulneráveis também a ataques de consumo de recursos (e.g., demanda por processamento até o esgotamento da bateria do dispositivo). Basicamente, as RSSFs requerem proteção contra escuta não-permitida, injeção e modificação de pacotes não-autorizada [Perrig et al. 2004].

Além da demanda por soluções particulares para segurança da rede, a tecnologia baseada no uso de sensores traz consigo a necessidade de discussão sobre requisitos básicos de privacidade:

- Empregadores "espionando" empregados;
- Comerciantes "espionando" consumidores;
- Vizinhos "espionando" outros vizinhos;
- Agências de segurança "espionando" locais públicos, etc..

A tecnologia por si só não é capaz de lidar com os problemas de privacidade. São necessárias leis governamentais e outras respostas tecnológicas para lidar com essas questões. Informar as pessoas sobre a presença de sensores e aquisição de dados é uma medida particularmente importante.

Chaves criptográficas O projeto de uma rede segura requer o uso de chaves criptográficas. Entre os desafios para as RSSFs está o fato do padrão de comunicação nessa redes exigir o estabelecimento de chaves com os nós vizinhos e com o nós que agregam dados. Outro desafio é que a criptografia fim-a-fim requer o estabelecimento de chaves entre todos os pontos, e é incompatível com a participação passiva dos nós (nós no raio de alcance de um transmissor tornam-se potenciais receptores desse nó) [Perrig et al. 2004].

Entre as possíveis soluções para o uso de chaves criptográficas, destacam-se:

- *Toda a rede compartilha a mesma chave*: o problema é que o comprometimento de um nó pode revelar a chave secreta de toda a rede;
- *Chave simétrica única entre cada par de nós*: o problema nesse caso é que a solução não é escalável;
- *Criptografia de chave pública*: nessa abordagem, um nó pode estabelecer uma chave segura com qualquer outro nó da rede, o problema é que para as RSSFs as primitivas de criptografia de chave pública são muito custosas;
- *Estação base certificadora*: nesse caso cada nó compartilha apenas uma chave com a estação base e estabelece chaves com outros nós via estação base, o problema é que a estação base centraliza um ponto de falha.

Captura de nós Um problema de segurança particular das RSSFs é a *captura de nós*, uma vez que, em boa parte das aplicações, os nós sensores são posicionados em locais não protegidos. Nesses cenários, um nó pode ser fisicamente removido, os segredos criptográficos podem ser capturados e o nó reprogramado e reintroduzido na rede. O desafio

torna-se, então, garantir que a rede continue operando corretamente, mesmo quando alguns nós são comprometidos. Para isso, uma alternativa é replicar o estado da rede e usar consenso ou outras técnicas para detectar inconsistências.

Um exemplo de uso dessa estratégia é a adoção de protocolos de rotemanento que enviam a mesma mensagem por diferentes caminhos, e verificam a consistência do pacote no nó destino. Outro exemplo são aplicações que requisitam mais de um relato sobre o mesmo evento. O mecanismo de defesa baseada em redundância é particularmente adequada para RSSFs, considerando que um algomerado de pequenos dispositivos pode ser mais capaz de prover uma operação segura da rede do que poucos dispositivos mais sofisticados [Perrig et al. 2004].

Negação de serviço Um ataque de negação de serviço é qualquer evento que prejudica ou elimina a capacidade da rede de executar a sua funcionalidade, por exemplo, falhas de hardware, erros de software, exaustão dos recursos, condições ambientais, etc.. Determinar se uma falha é resultado de um ataque intencional de negação de serviço é um desafio por si só. Nas RSSFs, ataques de negação de serviço podem ocorrer nas camadas física, de enlace e de rede [Wood and Stankovic 2002].

Na camada física, um ataque de negação de serviço pode se dar de duas formas principais:

- *Obstrução*: sinais de alta intensidade na mesma frequência usada pelos nós da rede, nesse caso uma forma de defesa é tentar isolar a região atacada;
- *Remoção*: a remoção ou destruição de um nó pode não ser distinguível de um comportamento de falha, i.e., um ataque desse tipo pode não ser detectado, por isso, quando possível, os nós deveriam apagar seu conteúdo quando removidos da rede.

Na camada de enlace, as possíveis formas de negação de serviço incluem:

- *Colisão*: violação do controle de acesso ao meio, e.g., transmitindo em instantes não permitidos;
- *Exaustão*: solicitação continuada de acesso ao meio, e.g., um nó enviando pacotes RTS forçando o nó destino a enviar pacotes CTS (uma alternativa é ignorar requisições excessivas);
- *Injustiça*: abuso de um esquema de prioridade para degradar o serviço do canal.

Em relação à camada de rede, as RSSFs são mais vulneráveis, uma vez que cada nó é um roteador em potencial. Os ataques de negação de serviço podem ser dar das seguintes formas:

- *Negligência*: o nó não repassa as mensagens recebidas (usando uma referência randômica ou arbitrária) ou prioriza apenas as suas mensagens (uma alternativa é usar várias rotas para enviar a mesma mensagem);
- *Direção invertida*: o nó repassa mensagens por caminhos incorretos (e.g., injeção de informações de roteamente maliciosas na rede).

Como as RSSFs normalmente já devem ser projetadas para continuar funcionando na ocorrência de falhas, essa robustez contra problemas físicos pode previnir algumas classes de ataques.

Requisitos de segurança para RSSFs Os requisitos de segurança para as RSSFs podem ser sumarizados da seguinte forma:

- *Gerência de segurança em grupo*: são necessários protocolos de gerência de grupos para garantir a entrada segura de novos membros e permitir a comunicação segura entre eles (os dados enviados pelo grupo para a estação base devem ser autenticados para garantir a procedência);
- *Agregação segura dos dados*: todos os locais de agregação devem ser seguros, uma alternativa é fazer amostragens aleatórias de pequenas frações dos nós e verificar se o comportamento está correto;
- *Detecção de intrusos*: requer uma solução distribuída e com baixo custo de comunicação, energia e uso da memória; particularmente importante é conhecer bem as possíveis formas de ataque (o uso de grupos seguros pode ser uma alternativa promissora para a detecção descentralizada);
- *Definição de níveis de segurança*: uma estratégia para minimizar os custos com segurança é considerar mensagens com maior ou menor exigência de segurança.

9.1. Soluções de segurança para RSSFs

Destacamos duas soluções desenvolvidas para lidar com os requisitos de segurança nas RSSFs: SPINS [Perrig et al. 2002] e TinySec Karlof/etal/04.

SPINS SPINS *Security Protocols for Sensor Networks* [Perrig et al. 2002] consiste de um conjunto de protocolos otimizados para RSSFs: (1) o protocolo $SNEP$, que trata das questões de confidencialidade, autenticação par-a-par, e garantia de dados atuais; e (2) o protocolo $\mu TESLA$, que permite disseminação (*broadcast*) autenticada. Os padrões de comunicação considerados são:

- estação base $>$ todos os nós;
- estação base $>$ nó individual;
- nó individual $>$ estação base.

SPINS usa apenas primitivas de criptografia simétrica e seu funionamento requer a sincronização dos nós. Cada nó recebe uma chave secreta mestre, compartilhada com a estação base, e todas as demais chaves são derivadas da chave mestre. As partes que se comunicam mantêm um estado comum que inclui um contador de mensagens.

TinySec Outra solução de segurança para as RSSFs é o TinySec [Karlof et al. 2004]. Trata-se de um pacote genérico de segurança cuja idéia é permitir que os desenvolvedores possam integrá-lo nas suas aplicações. O TinySec pode ser visto como uma plataforma extensível para ser incorporada em protocolos de nível mais alto. O protocolo visa aproveitar características das redes de sensores (como largura de banda limitada) para minimizar o custo das soluções de segurança. O TinySec é implementado para o sistema operacional TinyOS e seu foco é autenticidade, integridade e confidencialidade das mensagens. O protocolo não trata ataques de consumo de recursos, captura ou remoção de nós.

O TinySec permite duas opções de segurança:

1. *Criptografia e autenticação* (*TinySec-AE*): criptografa os dados e autentica o pacote;

2. *Autenticação* (*TinySec-Auth*): apenas autentica o pacote;

O TinySec requer a definição de mecanismos para distribuição e compartilhamento das chaves criptográficas através da rede. O TinySec é independente do mecanismo adotado.

10. Exemplo de implementação

Nesta seção, ilustramos o uso de RSSFs por meio de um exemplo de aplicação. Adotamos o sistema operacional TinyOS [Hill et al. 2000] e plataformas de nós sensores da família Mica Motes [1]. A aplicação é implementada usando a linguagem *nesC* — uma extensão da linguagem C desenvolvida especialmente para incorporar os conceitos e o modelo de execução do TinyOS. Apresentamos inicialmente a aplicação acompanhada de uma visão geral da linguagem nesC (baseada no manual de programação do TinyOS escrito por Philip Levis [Levis 2006]). Em seguida, mostramos exemplos de plataformas para execução de aplicações TinyOS e um roteiro básico para instalação da aplicação exemplo na plataforma Mica2 [2].

10.1. A linguagem nesC

A linguagem *nesC* [Gay et al. 2003] — uma extensão da linguagem C — foi projetada para incorporar os conceitos e o modelo de execução do TinyOS. Os programas nesC são construídos a partir de um conjunto de *componentes* com *interfaces bi-direcionais* bem definidas. O comportamento de um componente é especificado em termos das interfaces que ele oferece ou usa. As interfaces oferecidas representam a funcionalidade provida pelo componente aos seus usuários, enquanto as interfaces usadas representam as funcionalidades de que o componente precisa para executar o seu trabalho.

As interfaces definem um canal de interação multi-funcional entre dois componentes: o *provedor* e o *usuário*. O provedor de uma interface deve implementar o conjunto de funções denominadas *comandos*; e o usuário de uma interface deve implementar o conjunto de funções denominadas *eventos*. Como exemplo, a interface `AMSend` define o comando `send` para enviar uma mensagem e o evento `sendDone` para sinalizar que a mensagem foi enviada:

```
interface AMSend {
 command error_t send(am_addr_t addr,message_t* msg,uint8_t len);
 event void sendDone(message_t* msg, error_t error);
}
```

Essa estrutura permite representar interações complexas entre os componentes, por exemplo, o registro de interesse em algum evento e a informação da função de retorno que deverá ser executada quando o evento acontecer. Todos os comandos TinyOS que codificam computações demoradas, como o envio de mensagens, devem ser não-bloqueantes: o término da execução deve ser sinalizado separadamente por meio de um evento. Os componentes que usam esses comandos devem definir tratadores para os eventos que poderão ser sinalizados em resposta aos comandos chamados. No caso da interface `AMSend`, um componente não pode chamar o comando `send` se ele não implementar o evento `sendDone`.

[1] http://www.xbow.com
[2] http://www.xbow.com

Operações que envolvem o hardware são normalmente realizadas em duas fases, ao invés de serem bloqueantes, i.e., a conclusão de uma operação é feita através de uma função de retorno. A abordagem adotada pelo TinyOS é diferente da abordagem adotada pelos sistemas operacionais convencionais: ao invés de construir uma visão síncrona, através de linhas de execução distintas, as operações de duas fases no nível de hardware são também operações de duas fases no nível de software.

A linguagem nesC define dois tipos fundamentais de componentes: `módulo` (*module*) e `configuração` (*configuration*). Os módulos provêem o código das aplicação implementando as interfaces, enquanto as configurações são usadas para conectar outros componentes através das interfaces oferecidas e usadas por eles. Toda aplicação nesC é descrita através de um componente de configuração principal que conecta os demais componentes.

10.2. Desenvolvendo aplicações para o TinyOS

Uma aplicação básica de sensoriamento consiste em capturar periodicamente os valores físicos monitorados pelos sensores (por exemplo, luminosidade ou temperatura) e enviar esses valores para uma estação base (normalmente um nó conectado com a rede cabeada). Tomaremos a aplicação `RadioSenseToLeds` como referência. Trata-se de uma aplicação nesC que solicita periodicamente a leitura dos sensores e usa a rede sem fio para transmitir os valores medidos para uma estação base. `RadioSenseToLeds` é intencionalmente simples (não executa operações avançadas como a agregação de dados ou técnicas sofisticadas de roteamento de mensagens). O nosso objetivo é apenas experimentar a programação dos sensores.

Os dois componentes principais da aplicação `RadioSenseToLeds` são o módulo que implementa a aplicação — `RadioSenseToLedsC.nc` — e o componente de configuração responsável por conectar os demais componentes — `RadioSenseToLedsAppC.nc`. Toda aplicação TinyOS requer um arquivo de configuração, o qual é usado como arquivo fonte pelo compilador nesC para gerar o código executável. Apresentamos uma versão simplificada dos componentes que implementam a aplicação `RadioSenseToLeds` para manter o foco nos conceitos mais importantes para a nossa discussão.

A Figura 8 mostra o código principal da aplicação. O componente principal da aplicação, chamado `RadioSenseToLedsC`, declara as interfaces usadas por ele: `Leds`, `Boot`, `Receive`, `AMSend`, `Packet`, `Read<uint16_t>`, `SplitControl` (renomeada para RadioControl) e `Timer<TMilli>` (renomeada para MilliTimer). Essas declarações significam que o componente `RadioSenseToLedsC` pode chamar qualquer um dos comandos oferecidos por essas interfaces e deve implementar os eventos sinalizados por elas.

A interface `SplitControl` define os comandos para iniciar e finalizar um componente TinyOS. Todos os componentes que requerem inicialização ou podem ser desligados e religados durante a execução da aplicação devem prover essa interface. A especificação da interface `SplitControl` é mostrada a seguir:

```
interface SplitControl {
  command error_t start();
  event void startDone(error_t error);
  command error_t stop();
  event void stopDone(error_t error);
}
```

```
module RadioSenseToLedsC {
  uses { interface Leds;  interface Boot;
    interface Receive;  interface AMSend;
    interface Timer<TMilli> as MilliTimer;
    interface Packet;  interface Read<uint16_t>;
    interface SplitControl as RadioControl;
  }
}
implementation {
  message_t packet;
  bool locked = FALSE;

  event void Boot.booted() {
    call RadioControl.start();
  }

  event void RadioControl.startDone(error_t err) {
    if (err == SUCCESS) {
      call MilliTimer.startPeriodic(1000);
    }
  }

  event void RadioControl.stopDone(error_t err) {}

  event void MilliTimer.fired() {
    call Read.read();
  }

  event void Read.readDone(error_t result, uint16_t data) {
    if (locked) {return;}
    else {
      radio_sense_msg_t* rsm;

      rsm = (radio_sense_msg_t*)call Packet.getPayload(&packet, NULL);
      if (call Packet.maxPayloadLength() < sizeof(radio_sense_msg_t))
            return;
      rsm->error = result;
      rsm->data = data;
      if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
                  sizeof(radio_sense_msg_t)) == SUCCESS) {
        locked = TRUE;
      }
    }
  }

  event message_t* Receive.receive(message_t* bufPtr,
                                   void* payload, uint8_t len) {
    //...
    return bufPtr;
  }

  event void AMSend.sendDone(message_t* bufPtr, error_t error) {
    if (&packet == bufPtr) locked = FALSE;
  }
}
```

Figura 8. Código principal da aplicação *RadioSenseToLeds*.

```
configuration RadioSenseToLedsAppC {}
implementation {
  components MainC, RadioSenseToLedsC as App, new SensorMts300C();
  components ActiveMessageC, LedsC;
  components new AMSenderC(AM_RADIO_SENSE_MSG);
  components new AMReceiverC(AM_RADIO_SENSE_MSG);
  components new TimerMilliC();

  App.Boot -> MainC.Boot;

  App.Receive -> AMReceiverC;
  App.AMSend -> AMSenderC;
  App.RadioControl -> ActiveMessageC;
  App.Leds -> LedsC;
  App.MilliTimer -> TimerMilliC;
  App.Packet -> AMSenderC;
  App.Read -> SensorMts300C.Light;
}
```

Figura 9. Componente de configuração da aplicação *RadioSenseToLeds*.

A interface `Leds` é usada para controlar os *leds* dos dispositivos. As interfaces `AMSend` e `Receive` são responsáveis pelo envio e recepção de mensagens, respectivamente. A interface `Packet` é usada para manipular a estrutura de dados usada para o envio de mensagens na rede.

A interface `Read` é usada para interagir com os sensores requisitando a leitura de valores físicos como, temperatura, luminosidade, ruído, etc.. Ela define um comando para requisitar a leitura e um evento para sinalizar que a leitura foi concluída. A descrição da interface é mostrada abaixo:

```
interface Read<val_t> {
  command error_t read();
  event void readDone(error_t result, val_t val);
}
```

O argumento passado para a interface (`val_t`) define o tipo do valor que será lido pelo sensor. Na aplicação exemplo, o tipo definido é `<uint16_t>`.

O módulo `RadioSenseToLedsC` usa uma instância da interface `Timer<TMilli>` renomeandoa-a para `MilliTimer`. A construção `Timer<TMilli>` apenas indica uma interface genérica de temporizador, com a precisão requerida (nesse caso, mili-segundos). A interface `Timer` define o comando `startPeriodic`, o qual recebe como parâmetro o intervalo de sinalização do temporizador; o comando `stop` para parar o temporizador; e o evento `fired` que é sinalizado a cada intervalo de tempo.

A Figura 9 mostra o componente de configuração da aplicação. Ele inclui a especificação dos componentes usados: `MainC`, `RadioSenseToLedsC` (renomeado para `App`), `LedsC`, `SensorMts300C`, `TimerMilliC<TMilli>`, `AMSenderC`, `AMReceiverC`, `ActiveMessageC`; e a forma como as interfaces desses componentes são conectadas.

O componente `MainC` é o componente executado no início de uma aplicação TinyOS, por isso toda aplicação deve incluí-lo. O componente `SensorMts300C` im-

plementa as interfaces da plataforma de sensores `MTS300` [3].

A linguagem nesC usa o operando $->$ para associações básicas entre componentes e interfaces: a seta aponta do usuário para o provedor de uma interface.

A sentença `App.Read -> SensorMts300C.Light` estabelece que a interface `Read`, usada pelo componente `RadioSenseToLedsC`, é conectada com a interface `Light`, provida pelo componente `SensorMts300C`. A interface `Light` é uma interface `Read` implementada para leitura de sensores de luminosidade. Assim, quando o comando `Read.read` for chamado na aplicação, a leitura que será recebida corresponderá a uma medida de luminosidade.

A sentença `App.Boot -> MainC.Boot` estabelece que a interface `Boot`, usada pelo componente `RadioSenseToLedsC`, é conectada com a interface `Boot` provida pelo componente `MainC`. Assim, quando o evento `Boot.booted` for sinalizado (indicando que a aplicação terminou de ser carregada), o tratador desse evento no código da aplicação `RadioSenseToLedsC` será executado.

A sentença `App.AMSend -> AMSenderC` estabelece que a interface `AMSend`, usada pelo componente principal da aplicação, é conectada com a interface `AMSend`, provida pelo componente `AMSenderC`. Assim, quando o comando `AMSend.send` for chamado na aplicação, a implementação oferecida pelo componente `AMSenderC` é que será executada.

Os componentes `TimerMilliC` e `ActiveMessageC` são usados para implementar as interfaces `App.MilliTimer<TMilli>` e `App.RadioControl`, respectivamente. Os componentes `AMReceiverC` e `AMSenderC` implementam as interfaces para recepção e envio de mensagens, respectivamente. Esses componentes recebem como parâmetro o identificador das mensagens que serão trocadas (`AM_RADIO_SENSE_MSG`). Esse valor é definido no arquivo de cabeçalho `RadioSenseToLedsC.h`. O componente `AMSenderC` também é usado para implementar a interface `Packet`. Por fim, o componente `LedsC` é usado para implementar a interface `Leds`.

A Figura 10 destaca os principais componentes da aplicação `RadioSenseToLeds`.

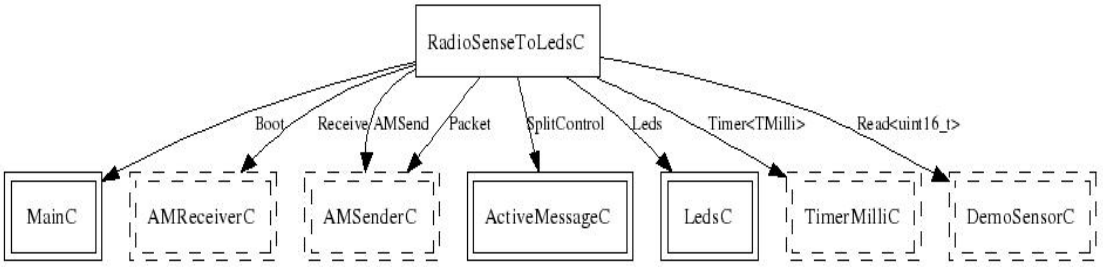

Figura 10. Componentes usados na aplicação *RadioSenseToLeds*.

A linguagem nesC define a construção `call <command>` para chamar um comando; a construção `post <task>` para escalonar a execução de uma tarefa; e a construção `signal <event>` para sinalizar a ocorrência de um evento. As funções que imple-

[3] http://www.xbow.com

mentam os comandos são marcadas com o identificador `command`, enquanto as funções que implementam os tratadores de eventos são marcadas com o identificador `event`. As funções que implementam tarefas são marcadas com o identificador `task`. Essas funções não recebem parâmetros de entrada e não devolvem valores.

Em linhas gerais, o funcionamento da aplicação `RadioSenseToLeds` é o seguinte: um temporizador é definido para sinalizar em intervalos de 1 seg (`MilliTimer.startPeriodic(1000)`). A cada intervalo de sinalização, uma leitura do sensor é disparada (`call Read.read()`). Quando a leitura é finalizada, o valor lido é empacotado e transmitido pela rede com o comando `AMSend.send()`. Quando uma mensagem é recebida pela rede, o estado do *led* verde (led1) é alternado. A variável `locked` é usada para impedir que uma nova leitura do sensor seja empacotada antes do valor lido anteriormente ter sido de fato transmitido. O evento `AMSend.sendDone()` é usado para reiniciar o valor dessa variável.

Os campos que compõem a informação principal das mensagens trocadas pela aplicação são definidos em uma estrutura de dados no arquivo `RadioSenseToLeds.h` Essa estrutura de dados é mostrada a seguir:

```
typedef nx_struct radio_sense_msg {
  nx_uint16_t error;
  nx_uint16_t data;
} radio_sense_msg_t;
```

10.3. Instalando uma aplicação nos nós da rede

Diferentes plataformas de hardware (e.g., mica2, micaZ, telos, mica2dot) podem ser usadas com o TinyOS. Usaremos a plataforma Mica2, da família Mica Motes, para a execução do exemplo discutido. A família Mica Motes é construída a partir do microcontrolador ATmega128L [4]. O ATmega128L possui 128KB de memória de código (memória *flash*) e 4KB de memória de dados (SRAM). Alguns exemplos de plataformas da família Mica Motes são: Mica2, Micaz e Mica2Dot [5]. A Figura 11 mostra o mote Mica2 e uma placa (*board*) usada para comunicação serial do mote com o PC.

As aplicações TinyOS vêm com um arquivo `Makefile` que permite selecionar a plataforma de execução e chamar o compilador nesC (`ncc`) com as opções apropriadas. Para instalar a aplicação no mote Mica2, o seguinte comando deve ser executado:

```
make <platform> (re)install,<n> <programmer>,<port>
```

platform refere-se ao tipo de Mote, *n* define um identificador para o Mote (pode ser omitido), *programmer* refere-se ao MIB (*Mote Interface Programming Board*) e *port* especifica a porta serial na qual o MIB está conectado.

O exemplo a seguir ilustra o uso da plataforma Mica2 com a MIB510 [6] conectado à porta USB:

```
make mica2 (re)install mib510,/dev/ttyUSB0
```

Usando Mica2 é necessário selecionar a frequência de rádio compatível com o Mote. A faixa de frequência permitida varia de acordo com o tipo de Mote. O Mote

[4]http://www.atmel.com/products/AVR
[5]http://www.xbow.com
[6]http://www.xbow.com

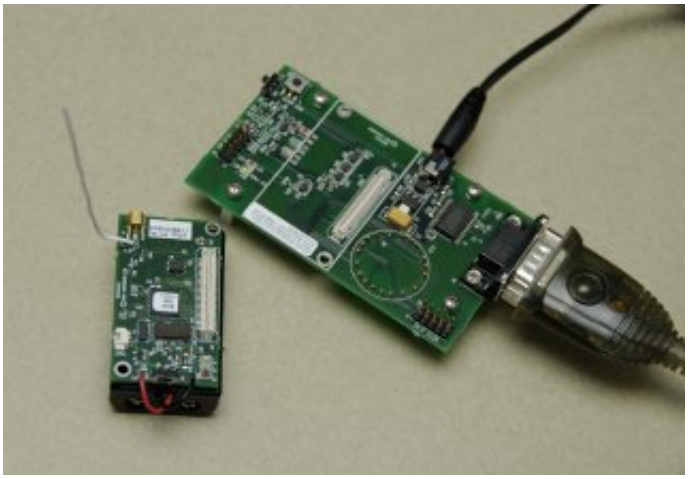

Figura 11. Mote Mica2 e a placa (*board*) para comunicação serial com o PC.

Mica2, modelo MPR400, usa a frequência 915MHz. Nesse caso o comando de instalação é:

```
PFLAGS=-DCC1K_DEF_FREQ=916400000
make mica2 (re)install mib510,/dev/ttyUSB0
```

Quando a instalação conclui com sucesso, o comando retorna uma mensagem do tipo:

```
writing TOS image
cp build/mica2/main.srec build/mica2/main.srec.out
installing mica2 binary using mib510
uisp -dprog=mib510 -dserial=/dev/ttyUSB0 --wr_fuse_h=0xd9
  -dpart=ATmega128 --wr_ fuse_e=ff  --erase
  --upload if=build/mica2/main.srec.out
Firmware Version: 2.1
Atmel AVR ATmega128 is found.
Uploading: flash

Fuse High Byte set to 0xd9

Fuse Extended Byte set to 0xff
rm -f build/mica2/main.exe.out build/mica2/main.srec.out
```

A MIB510 possui um processador *on-board in-system* (ISP) chamado Atmega16L que é usado para programar os Motes. O código é carregado no ISP, através da porta serial RS-232, e depois para o Mote. Ambos, o ISP e o Mote, compartilham a mesma porta serial. O ISP monitora os pacotes recebidos pela porta serial para detectar um padrão especial. Quando esse padrão é detectado a comunicação serial do Mote é desativada e o ISP assume o controle da porta serial. Devido ao compartilhamento da porta serial, o comando de instalação pode retornar com erro (o ISP não consegue detectar o padrão). Nesse caso a operação deve ser repetida até que possa ser completada com sucesso.

Os Motes Mica2 não fazem o chaveamento automático entre as fontes de energia externa e da bateria. Assim, é recomendável manter o Mote "desligado" (*in off*) sempre que ele estiver acoplado ao MIB. Se o MIB estiver chaveado em "on" (ver label SW2), a

estação base não poderá receber dados, apenas transmití-los.

Leitura dos sensores Para os sensores, usaremos dispositivos MTS300 (medem temperatura e luminosidade) [7]. Para permitir a leitura de valores dos sensores MTS300 é necessário informar a localização do diretório com a implementação do *sensor board*. Para isso, o comando de instalação deve ser modificado para:

```
SENSORBOARD=mts300
PFLAGS=-DCC1K_DEF_FREQ=916400000
make mica2 (re)install mib510,/dev/ttyUSB0
```

11. Considerações finais

Neste minicurso, apresentamos uma visão geral sobre as RSSFs, incluindo exemplos de áreas de aplicação, problemas e características específicas dessas redes, plataformas de hardware e soluções de software disponíveis para uso. Discutimos os principais desafios para a implantação dessas redes, e mostramos um exemplo prático de implementação de uma aplicação.

Conhecer os cenários reais de funcionamento das redes de sensores é uma meta especialmente importante para desenvolver soluções de software viáveis envolvendo essas redes. O erro randômico natural que envolve a comunicação nas RSSFs, por exemplo, torna os problemas clássicos de comunicação sem fio ainda mais desafiadores. O caminho para o desenvolvimento de soluções para as RSSFs envolve conhecer as características e os limites dessas redes e suas áreas particulares de aplicação. Com base nesse conhecimento é que devem ser projetados os algoritmos, técnicas, protocolos e arquiteturas de desenvolvimento que levem em conta esses limites e permitam implementar casos reais de uso.

Nosso objetivo é que o embasamento teórico, dado através da visão geral sobre as RSSFs, e o exercício prático de construção de uma aplicação real sirvam de motivação para os alunos aprofundarem o estudo sobre as diferentes possibilidades de aplicação das RSSFs e sobre os desafios que ainda precisam ser tratados para que essas redes sejam de fato viabilizadas.

Referências

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

Arampatzis, T., Lygeros, J., and Manesis, S. (2005). A survey of applications of wireless sensor and wireless sensor networks. In *13th Mediterranean Conference on Control and Automation*, Limassol, Cyprus. "IEEE".

Bharghavan, V., Demers, A., Shenker, S., and Zhang, L. (1994). Macaw: a media access protocol for wireless lan's. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 212–225, New York, NY, USA. ACM Press.

[7] http://www.xbow.com

Bhatti, S., Carlson, J., Dai, H., Deng, J., Rose, J., Sheth, A., Shucker, B., Gruenwald, C., Torgerson, A., and Han, R. (2005). MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.*, 10(4):563–579.

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

Callaway, E. H. (2003). *Wireless Sensor Networks*. CRC Press.

Chong, C. and Kumar, S. (2003). Sensor networks: Evolution, opportunities, and challenges. *Proceedings of IEEE*, 91(8).

Collier, T. C. and Taylor, C. (2004). Self-organization in sensor networks. *Journal of Parallel and Distributed Computing*, 64:866–873.

Costa, P., Picco, G. P., and Rossetto, S. (2005). Publish-Subscribe on sensor networks: a semi-probabilistic approach. In *Proceedings of the 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, Washington DC, USA.

Culler, D., Estrin, D., and Srivastava, M. (2004). Overview of sensor networks. *IEEE Computer*, 37(8):41–49.

Elson, J. and Estrin, D. (2001). Time synchronization for wireless sensor networks. In *15th International Parallel and Distributed Processing Symposium*, pages 1965–1970. ieeexplore.ieee.org.

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

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

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

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

Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., and Culler, D. (2003). The nesC language: A holistic approach to networked embedded systems. *Proceedings of Programming Language Design and Implementation (PLDI)*.

Hadim, S. and Mohamed, N. (2006). Middleware: Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online*, 7(3):1.

Han, C., Kumar, R., Shea, R., Kohler, E., and Srivastava, M. (2005). A dynamic operating system for sensor nodes. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications and Services*, pages 163–176, New York, NY, USA. ACM Press.

Heinzelman, W., Murphy, A., Carvalho, H., and Perillo, M. (2004). Middleware to support sensor network applications. *IEEE Network*, 18:6–14.

Heinzelman, W. R., Kulik, J., and Balakrishnan, H. (1999). Adaptive protocols for information dissemination in wireless sensor networks. In *5th annual ACM/IEEE inter-*

national conference on Mobile computing and networking, pages 174–185, New York, NY, USA. ACM Press.

Hightower, J. and Borriello, G. (2001). Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66.

Hill, J., Horton, M., Kling, R., and Krishnamurthy, L. (2004). The platforms enabling wireless sensor networks. *Commun. ACM*, 47(6):41–46.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K. (2000). System architecture directions for networked sensors. In *9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, New York, NY, USA. ACM Press.

Intanagonwiwat, C., Govindan, R., and Estrin, D. (2000). Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *6th Annual International Conference on Mobile Computing and Networking (MobiCom'00)*, pages 56–67, New York, NY, USA. ACM Press.

Karlof, C., Sastry, N., and Wagner, D. (2004). TinySec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 162–175, New York, NY, USA. ACM Press.

Kasten, O. and Romer, K. (2005). Beyond event handlers: programming wireless sensors with attributed state machines. In *International Conference on Information Processing in Sensor Networks*.

Levis, P. (2006). Tinyos programming manual. http://www.tinyos.net.

Levis, P. and Culler, D. (2002). Maté: a tiny virtual machine for sensor networks. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 85–95, New York, NY, USA. ACM Press.

Liu, T. and Martonosi, M. (2003). Impala: a middleware system for managing autonomic, parallel sensor systems. In *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 107–118, New York, NY, USA. ACM Press.

Loureiro, A. A. F., Nogueira, J. M. S., Ruiz, L. B., Mini, R. A., Nakamura, E. F., and Figueiredo, C. M. S. (2003). Wireless sensors networks. In *21st Brazilian Symposium on Computer Networks*, pages 179–226, Natal, RN, Brazil. Tutorial.

Madden, S. R., Franklin, M. J., Hellerstein, J. M., and Hong, W. (2005). Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173.

Marrón, P. J. and Minder, D. (2006). Embedded WiSeNts Research Roadmap. Technical report, Embedded WiSeNts Consortium. http://www.embedded-wisents.org/.

Perrig, A., Stankovic, J., and Wagner, D. (2004). Security in wireless sensor networks. *Commun. ACM*, 47(6):53–57.

Perrig, A., Szewczyk, R., Tygar, J. D., Wen, V., and Culler, D. E. (2002). SPINS: security protocols for sensor networks. *Wireless Networks*, 8(5):521–534.

Polastre, J., Hill, J., and Culler, D. (2004). Versatile low power media access for wireless sensor networks. In *2nd International Conference on Embedded Networked Sensor Systems*, pages 95–107, New York, NY, USA. ACM Press.

Prehofer, C. and Bettstetter, C. (2005). Self-organization in communication networks: principles and design paradigms. *IEEE Communications Magazine*.

Silberschatz, A., Gagne, G., and Galvim, P. B. (2004). *Fundamentos de Sistemas Operacionais*. LTC.

Sivrikaya, F. and Yener, B. (2004). Time synchronization in sensor networks: a survey. *IEEE Network*, 18(5):45–50.

Stankovic, J., Abdelzaher, T., Lu, C., Sha, L., and Hou, J. (2003). Real-time communication and coordination in embedded sensor networks. *Proceedings of the IEEE*, 91(7):Invited paper.

Tanenbaum, A. S. (2003). *Computer Networks*. PH PTR, 4 edition.

Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84.

Welsh, M. and Mainland, G. (2004). Programming sensor networks with abstract regions. In *USENIX/ACM Symposium on Network Systems Design and Implementation*.

Wood, A. D. and Stankovic, J. A. (2002). Denial of service in sensor networks. *Computing*, 35(10):54–62.

Ye, W. and Heidemann, J. (2004). Medium access control in wireless sensor networks. pages 73–91.

Younis, M., Youssef, M., and Arisha, K. (2002). 10th ieee international symposium on modeling, analysis and simulation of computer and telecommunications systems. In *IEEE Integrated Management of Power Aware Communications*, 129-136. IEEE Computer Society.