1
Introdução

As Redes de Sensores Sem Fio (RSSF) são constituídas por pequenos dispositivos computacionais (chamados *motes*) com capacidade de sensoreamento físico e severas restrições em termos de capacidade de processamento, armazenamento e consumo de energia. Essas redes são usadas em aplicações de monitoramento e requerem normalmente um grande número de nós implantados diretamente no meio físico. Algumas aplicações necessitam que os nós sejam distribuídos em grandes espaços geográficos ou em regiões de difícil acesso.

O trabalho de desenvolvimento de aplicações para RSSF é impactado pelos mesmos desafios encontrados no desenvolvimento de aplicações para sistemas distribuídos que utilizam plataformas tradicionais. Porém, alguns desses desafios são aumentados pela escassez de recursos computacionais dos motes, pelo modelo de programação tipicamente adotado que é orientado a eventos e pela tecnologia de comunicação sem fio com características de redes ad-hoc. Por outro lado, em boa parte dos casos, as aplicações que executam nas RSSF seguem uma estrutura comum de execução, permitindo identificar funcionalidades que se repetem em diferentes cenários. Por exemplo, funcionalidades locais, como a leitura periódica de sensores e o envio dos dados coletados para uma estação base; ou funcionalidades distribuídas, como a fusão ou agregação das medidas coletadas por nós vizinhos.

Um dos desafios típicos da computação distribuída que tem a sua complexidade aumentada no cenário de RSSF é a necessidade de realizar ajustes na aplicação após a sua implantação. Esses ajustes podem ser uma simples alteração no período de monitoração, uma correção no fluxo de processamento ou até a ativação ou desativação de pequenas funcionalidades. A plataforma de software de uso mais comum para o desenvolvimento de aplicações em RSSF não disponibiliza, de forma nativa, opções para reconfiguração remota após a implantação da aplicação e, na maioria dos casos, é inviável recuperar todos os motes para aplicar novas alterações. As alternativas mais comuns para reprogramação dos nós sensores envolvem a carga remota e substituição de componentes inteiros, dessa forma, indo muito além de uma simples reconfi-

guração e podendo consumir mais recursos da rede do que o necessário.

Uma solução para esses desafios é disponibilizar para o desenvolvedor um conjunto de funcionalidades básicas que possam ter a sua execução definida por meio de alguns parâmetros, e de um controle de fluxo simplificado. Esse conjunto de funcionalidades representaria alguns padrões de interação típicos de RSSF e pode ser visto como uma biblioteca de funções pronta para uso, atendendo a diversos tipos de aplicações. Essa biblioteca atenderia basicamente as aplicações de monitoração e alarme, incluindo o controle de roteamento de mensagens pela rede. Alguns componentes mais específicos poderiam ser adicionados em tempo de projeto. Essa solução permite a simplificação do desenvolvimento de novas aplicações, minimizando as dificuldades da programação distribuída e evitando a necessidade de escrever códigos na linguagem nativa da plataforma. Também possibilita a implementação de um processo de reconfiguração remota em ponto pequeno, por meio do envio de parâmetros de configuração dos componentes da aplicação, ao invés da abordagem de substituição de componentes inteiros. Tudo isso permite uma mudança no perfil do desenvolvedor. O novo desenvolvedor não precisa conhecer os detalhes de programação no nível mais baixo e poderá direcionar os esforços para o entendimento e configuração da aplicação. O foco principal do nosso trabalho é baseado nessa ideia, e visa minimizar as dificuldades de desenvolvimento e reconfiguração de aplicações para RSSF incorporando no próprio modelo de programação a capacidade de reconfiguração das aplicações em tempo de execução e a possibilidade de reutilizar padrões de interação comuns entre os nós da rede.

1.1 Abordagem

Para a identificação do conjunto básico de funcionalidades que irão compor a nossa biblioteca de funções, tomamos como referência padrões de interação típicos das aplicações para RSSF. Esses padrões são identificáveis, pois, de forma geral, as aplicações em RSSF executam operações semelhantes, como por exemplo a leitura periódica de sensores, a fusão de dados envolvendo grupos de nós vizinhos e o roteamento de mensagens até a estação base. Para obtermos uma gama razoável de padrões, procuramos diferentes fontes, incluindo exemplos de aplicações para RSSF e propostas de modelos para macroprogramação em RSSF. A ideia de macroprogramação em RSSF toma como base que toda aplicação requer naturalmente a interação entre os nós individuais, então, ao invés de projetar as aplicações a partir do código de cada nó, propõe-se o inverso, que a aplicação seja projetada considerando a

rede como um todo e que em uma etapa seguinte o código necessário seja instanciado em cada nó individual. Por isso os modelos de macroprogramação, normalmente, incluem alguns padrões de interação típicos.

Para definir o controle de fluxo usamos a ideia de máquinas de estados finitos (*Finite State Machine*) (FSM). No modelo de FSM temos um motor de execução que, de acordo com o estado corrente, dispara ações de saída a partir de eventos de entrada. Com isso o desenvolvedor pode indicar quais funcionalidades irá utilizar e qual será a sequência de operação das mesmas. O controle de fluxo proposto permite a criação de várias máquinas de estado dentro de uma mesma aplicação e ainda possibilita a dependência entre estados de diferentes máquinas, facilitando a construção de aplicações mais complexas, ou a visão de uma mesma RSSF usada para diferentes finalidades.

Tanto o fluxo da FSM quanto o comportamento funcional de cada componente da aplicação são definidos por parâmetros. A possibilidade de configuração dinâmica é alcançada simplesmente permitindo que esses parâmetros sejam alterados remotamente. Como exemplo, em uma aplicação de coleta periódica de temperatura, pode-se alterar o período de coleta ou alterar o fluxo operacional para calcular a média da temperatura dos nós do mesmo grupo. Também é possível adicionar, em tempo de execução, um novo fluxo para essa aplicação, como um alarme de temperatura alta, mantendo o fluxo anterior inalterado.

1.2 Objetivos e contribuições

Nesta seção apresentamos os objetivos do nosso trabalho e detalhamos as principais etapas para alcançar esses objetivos. Também identificamos as principais contribuições geradas ao término do trabalho.

1.2.1 Objetivo Geral

O nosso objetivo principal é estudar um modelo de programação que facilite a construção de novas aplicações e que permita, de forma simples, a reconfiguração remota dessas aplicações.

1.2.2 Objetivo Detalhado

Para alcançarmos o objetivo proposto, dividimos o trabalho em etapas. A seguir, identificamos essas etapas e descrevemos o objetivo de cada uma.

Identificação do conjunto de funcionalidades básicas - O objetivo dessa etapa é identificar e definir um conjunto de funcionalidades básicas que possam suportar a construção de diversos tipos de aplicações para RSSF. Complementando o conjunto de funcionalidades, incluímos a definição do conjunto de parâmetros que permitirão ajustes no comportamento dessas funcionalidades.

Definição do controle de fluxo - Nessa etapa definimos um mecanismo de controle de fluxo das aplicações baseado em máquinas de estados finitos, assim como o conjunto de eventos e ações básicas que serão usados pelas aplicações.

Implementação - Nessa etapa implementamos a nossa biblioteca de funções parametrizáveis, que representam o conjunto de funcionalidades básicas definidas. Também implementamos a infraestrutura para o controle de fluxo e os módulos de suporte para reconfiguração remota.

Avaliação - Essa etapa teve o objetivo de avaliar o quanto é simples a programação de aplicações para RSSF usando o modelo proposto, e qual é o impacto da reconfiguração remota na vida útil da rede.

1.3 Trabalhos relacionados

Um dos primeiros trabalhos a propor simplificação para programação de plataformas de sensores no estilo motes foi o TinyDB[1]. O TinyDB disponibiliza um conjunto de funções para agregações simples e permite reconfigurar a coleta de dados da rede por meio de comandos remotos baseados numa linguagem simplificada do estilo SQL. O TinyDB limita-se às aplicações com características de coleta de dados, com possibilidade de agregações em uma topologia hierárquica de roteamento e sem interação local entre os nós da rede. A nossa proposta disponibiliza um conjunto maior de funções para agregações com a possibilidade de reconfiguração, mas, diferentemente do TinyDB, a agregação é feita a partir de grupos locais de nós e enviada para o nó raiz. O nosso modelo de agregação não fica restrito à topologia da árvore de roteamento.

Um conjunto de trabalhos mais recentes, com foco em modelos de programação mais adequados para RSSF, está ligado ao conceito de *macroprogramação*. Dentre os representantes dessa linha, destacamos Regiment[2] e Cosmos[3]. A macroprogramação em RSSF visa simplificar o processo de construção de novas aplicações, disponibilizando ao desenvolvedor abstrações

simplificadas para operações mais complexas, principalmente operações distribuídas entre grupos de nós. No conceito de macroprogramação, o programador é habilitado a trabalhar com uma abstração em que a rede é um sistema composto por grupos de sensores. Essa abstração permite a geração de programas de uma forma mais simples. Uma dificuldade encontrada nessas soluções é que, embora a visão de programação da rede como um todo seja mais adequada, em geral os componentes básicos que devem ser executados nos nós individuais ainda precisam ser implementados diretamente pelo desenvolvedor, desviando a atenção para os nós individuais em uma etapa seguinte ao projeto da aplicação. O foco do nosso trabalho é justamente o desenvolvimento desses componentes básicos, incluindo algumas soluções que facilitam o processamento distribuído, como agregações de valores e roteamento de mensagens.

Dentre os trabalhos que abordam a reprogramação remota, temos os que utilizam um modelo de máquina virtual simplificado, como Maté [4], ou os trabalhos baseados no DELUGE [5] que carregam o programa original na linguagem nativa. A linguagem utilizada por Maté é bem simples e permite acesso apenas aos recursos básicos do mote, não disponibilizando uma biblioteca de componentes de alto nível que facilite a construção de novas aplicações. Os trabalhos que utilizam o DELUGE precisam fazer a carga completa da aplicação em conjunto com o sistema operacional. Abordagens mais recentes possibilitam a recarga em separado de componentes originais da aplicação [6], permitindo reduzir o custo da reprogramação remota. O nosso modelo permite a reconfiguração do controle de fluxo e dos parâmetros de funcionamento dos componentes. Dessa forma, o desenvolvedor pode reconfigurar as operações mais complexas, não necessitando recarregar o código executável da aplicação e do sistema operacional.

O trabalho de Kasten e Romer propõe um modelo de programação baseado em máquinas de estados para RSSF, chamado OSM [7], cujo objetivo é simplificar a programação dos motes. OSM utiliza um processo de compilação para gerar código na linguagem nativa dos nós sensores e não aplica o conceito de reprogramação. Como outros trabalhos, OSM também considera que o desenvolvedor deve construir os componentes mais básicos na linguagem de programação dos nós sensores. Como OSM, o nosso modelo utiliza o conceito de FSM para o controle de fluxo. No nosso caso, a configuração da FSM é interpretada, enquanto que em OSM, é compilada para a plataforma de execução.

Inspirado em sistemas multiagentes e inteligência artificial distribuída, temos RuleCaster [8]. RuleCaster usa a linguagem RCAL, um compilador e

um ambiente de execução. Com RCAL o programador especifica as regras das transições entre estados e as possíveis ações. O compilador converte o programa RCAL em atividades que são distribuídas aos nós. O sistema de execução interpreta essas atividades, reagindo aos eventos e atuando com ações. Essas atividades geradas pelo compilador funcionam como uma linguagem intermediária e, associadas ao interpretador do sistema de execução, permitem a reconfiguração remota. O conceito de FSM utilizado no nosso modelo é equivalente ao conceito de linguagem intermediária de RCAL. A diferença é que disponibilizamos previamente um conjunto de funcionalidades para serem utilizadas pela nossa linguagem intermediária.

Com foco específico num modelo para Finite State Machine (FSM) temos Statecharts[9] e Statemate[10]. O modelo de Statecharts propõe uma notação gráfica para configuração de máquinas de estados, permitindo combinações complexas como paralelismo e hierarquias. Statemate define uma semântica para o modelo de Statecharts. O nosso modelo de programação foi inspirado no mesmo modelo de Statecharts. No nosso caso, as atividades são definidas por uma tabela de transição de estados que representa uma FSM.

O nosso principal diferencial em relação aos trabalhos citados acima é a combinação de um mecanismo para configuração do fluxo de processamento da aplicação com uma biblioteca de funções parametrizáveis, a partir da qual, em boa parte dos casos, o desenvolvedor não precisará construir nenhum componente adicional. Essa abordagem permitiu combinar no nosso modelo os principais pontos atendidos de forma isolada em outros trabalhos.

1.4 Organização do documento

No capítulo 2, apresentamos, para contextualizar a leitura dos capítulos seguintes, o ambiente de desenvolvimento do TinyOS/NesC .

Os três capítulos seguintes apresentam a execução do trabalho. O capítulo 3 apresenta o modelo de programação proposto. O capítulo 4 apresenta a implementação do sistema utilizando os recursos do TinyOS. No capítulo 5 mostramos o processo e a execução dos testes de avaliação do nosso trabalho. Para finalizar, o capítulo 6 apresenta as conclusões do trabalho em função dos objetivos iniciais. Nos apêndices é possível encontrar algumas informações complementares citadas durante o texto principal.