4
Implementação

Neste capítulo vamos apresentar nossa implementação para um sistema que suporta a biblioteca de funções parametrizáveis e a engrenagem da FSM definidas no capítulo 3.

Vamos apresentar na seção 4.1 uma visão geral para a arquitetura de execução e os protocolos básicos de comunicação. Em seguida, na seção 4.2, abordaremos a nossa implementação através de diferentes operações na rede de sensores sem fio. Na seção 4.3 vamos apresentar o padrão de implementação utilizado no nosso modelo de camadas. Para exemplificar o uso do modelo e a respectiva implementação, vamos descrever o fluxo de execução de uma operação simples de agregação. No final do capítulo, discutimos a nossa experiência no desenvolvimento do sistema utilizando o TinyOS/NesC e abordarmos o procedimento para adicionar novos eventos e ações.

4.1
Visão operacional geral

De forma geral, a operação das aplicações em rede de sensores sem fio assume a existência de uma estação servidora para recebimento de informações coletadas. A nossa arquitetura considera que esta estação servidora também será usada para gerenciar as operações de reconfiguração da aplicação.

Executamos na estação servidora um processo JAVA para controle das operações e troca de mensagens com a rede de sensores sem fio. Essa troca de mensagens utiliza um mote especial conectado com a estação servidora via interface serial (USB) e que acessa a rede de sensores via rádio. Esse mote é chamado de estação base (*Base Station*).

A nossa implementação considera que o mesmo código executável será executado em todos os motes da rede. A diferença de comportamento é definido com a carga dos parâmetros de configuração e da tabela de transições da FSM. A informação a ser carregada no mote é armazenada na estação servidora no formato de arquivo XML.

Na Figura 4.1, temos o diagrama com os módulos utilizados no servidor.

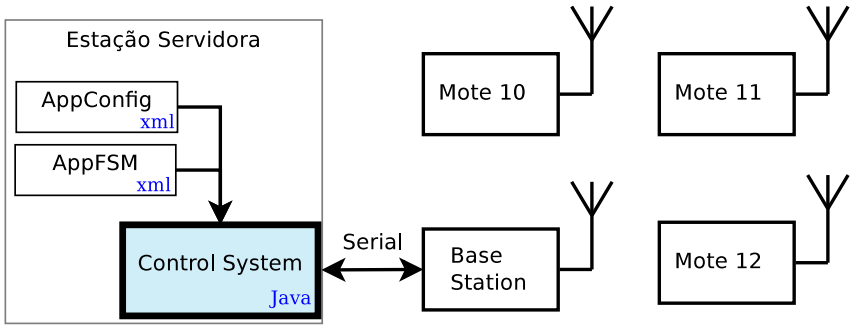

Figura 4.1: Módulos utilizados na estação servidora

Os testes mais exaustivos e monitorados foram executados num ambiente simulado baseado na ferramenta TOSSIM[14]. Para realiza-los foi necessário construir, além do nosso sistema e do módulo de controle em JAVA, alguns módulos auxiliares. Esses módulos estão descritos no apêndice C.

4.1.1 Protocolos de comunicação

A comunicação entre os motes é a base para as principais operações nas redes de sensores sem fio. Normalmente as aplicações necessitam de alguma interação entre motes vizinhos e na maioria das vezes é necessário encaminhar alguma informação para a estação servidora. A implementação dos protocolos de comunicação é parte fundamental do nosso modelo de programação, pois é onde implementamos as funções básicas para a maior parte das simplificações oferecidas.

A partir da análise do capítulo 3, implementamos protocolos para roteamento de mensagens, disseminação de dados e formação de grupos de nós. O foco principal no nosso trabalho não é a implementação da melhor solução de comunicação. Então optamos por simplificar ao máximo o trabalho de implementação e procuramos utilizar recursos já implementados no TinyOS. Para isso utilizamos os protocolos de disseminação de dados DIP [20] e o de coleta de dados CTP [21].

Troca de mensagens entre a rede e a estação servidora

O roteamento de/para a estação base foi implementado utilizando uma adaptação do protocolo CTP-Collection Tree Protocol [21]. O CTP disponibiliza uma interface para envio de mensagens de qualquer nó da rede para um nó definido como nó raiz. Esse processo de roteamento é transparente ao usuário do protocolo. Na inicialização do CTP são trocadas mensagens específicas entre os motes para definir, com base na intensidade do sinal de rádio, quais

motes serão os nós roteadores. No nosso caso o mote estação base funciona como nó raiz da hierarquia para o protocolo de coleta.

Como o CTP só resolve o roteamento no sentido da rede para a estação base, implementamos uma extensão que permite o roteamento no sentido inverso. Com o intuito de reduzir a utilização de memória RAM do mote, a nossa extensão assume que o processo de roteamento sempre é disparado de um nó da rede para a estação base, quando então, uma resposta, poderá ser roteada no sentido inverso. Isso permite trabalharmos com um tabela mínima de roteamento, onde só são armazenadas as informações das últimas solicitações.

Utilizamos a implementação do CTP para rotear a mensagem de solicitação até o nó raiz e implementamos uma interface para o envio de mensagens no sentido inverso. A nossa extensão intercepta o CTP e registra em cada nó roteador as informações necessárias para efetuar o roteamento de retorno.

Disseminação da tabela FSM

A disseminação da FSM pela rede foi implementada utilizando o protocolo DIP[20]. O DIP dissemina automaticamente um pequena quantidade de dados pela rede de sensores sem fio, garantindo a entrega da informação. Como existe um limite na quantidade de dados em uma mensagem via rádio, tivemos que executar essa distribuição em várias partes. O sistema atual está preparado para uma tabela FSM com no máximo 40 transições. Essa tabela é sempre difundida com 5 blocos/mensagens, sendo que não é garantida a sequência da entrega. O processo de disseminação mantém um controle de versão para a tabela FSM. Esse controle de versão permite identificar o início de uma nova disseminação com a chegada do primeiro bloco do DIP. O final da disseminação é identificado quando todos blocos estiverem com a nova versão.

Formação de grupos de nós (NHops)

O protocolo de formação de grupo de motes (NHops) foi implementado utilizando a função básica de comunicação do TinyOS *Active Message*(AM). Essa função permite o envio de uma mensagem para um nó específico ao alcance do rádio (1-hop) ou o envio de uma mensagem broadcast para todos os nós vizinhos ao alcance do rádio. A formação do grupo é feita por meio da difusão de uma mensagem broadcast que se propaga por uma quantidade predefinida de saltos (hops). Quando receber uma mensagem desse tipo, o mote verifica o contador de saltos e, se necessário, reenvia a mensagem incrementando esse contador. Essa mensagem tem um identificador único composto por um número sequencial e o ID do mote origem. Também carrega alguns parâmetros, como

o ID do mote que repassou a mensagem, a quantidade máxima de saltos, a quantidade de saltos atual, a informação requisitada e o identificador do grupo.

Cada mote só considera válida a primeira mensagem recebida para cada identificação única (Sequencial + IdOrigem), e ao recebê-la registra esses valores em uma lista. Isso permite a criação da rota de retorno para o mote que iniciou o processo de difusão. Para cada pedido recebido, caso o mote faça parte do mesmo grupo do nó originador, o mote retorna o valor solicitado utilizando a informação de roteamento. Um nó intermediário, mesmo que não faça parte do grupo identificado na mensagem, deve rotear a mensagem de retorno.

A seguir, apresentamos uma visão resumida do pseudo-código do nosso algoritmo para formação do grupo. Todos os nós executam o mesmo código. O processo é iniciado com a execução do procedimento `RequestValues()` em um dos nós do grupo.

```
RequestValues(ReqNum):
    Enviar mensagem broadcast NHops.

Recebimento da msg NHops:
    Se sequencial já recebido ou a msg é sua,
      então descartar msg e retornar.
    Fazer nó pai igual ao nó origem.
    Se não alcançou limite de saltos,
      então enviar a mensagem broadcast NHops.
    Se não for do mesmo grupo,
      então descartar msg e retornar.
    Executar a leitura do sensor identificado pela mensagem.
    Enviar mensagem NHopsReturn para o nó pai.

Recebimento da msg NHopsReturn:
    Se a msg for para mim,
        então encaminhar para a aplicação o valor recebido;
        senão encaminhar a msg para o nó pai.
```

A implementação atual utiliza duas filas de eventos, uma de saída e outra de entrada. Essas filas permitem a concorrência entre diferentes operações e ao mesmo tempo compatibilizam o código com o modelo split-phase de nesC/TinyOS.

4.2 Etapas de operação

Para a implementação do sistema atual, identificamos um conjunto de operações necessárias para o funcionamento completo de uma aplicação na rede de sensores sem fio. Descrevemos a implantação de uma aplicação considerando que ela ocorre em três etapas: Inicialização, Configuração e Execução da Aplicação. Opcionalmente temos ainda uma etapa de Eleição. A seguir descrevemos essas etapas.

4.2.1 Etapa: Inicialização

Essa etapa é responsável pela inicialização da rede de sensores sem fio. Tem como principais funções a inicialização interna (local) do mote e a formação da topologia hierárquica para o protocolo de coleta CTP.

A inicialização do mote é equivalente a um procedimento de "boot", onde o TinyOS e os componentes do sistema são inicializados. O código executável do nosso sistema é o mesmo em todos os motes, então é necessário inicializar o mote conforme os parâmetros de configuração e a tabelas FSM previamente carregados.

Para formar a topologia hierárquica do CTP, os nós trocam mensagens entre si, identificando as rotas mais curtas para alcançar o nó raiz, isto é, identificando em cada mote quem é o mote imediato na hierarquia (mote pai).

O nosso sistema requer, independentemente do tipo de aplicação, uma máquina de estados mínima para controle da inicialização do mote. Essa máquina pode ter seu uso estendido para a aplicação do usuário.

4.2.2 Etapa: Configuração

Essa etapa implementa o controle da reconfiguração remota em duas fases. Primeiro uma única tabela FSM é disseminada e depois são enviados os parâmetros individuais de cada mote.

O processo de configuração inicia a partir da estação servidora, que carrega os dados da tabela FSM na estação base. Esses dados são distribuídos para todos os motes da rede pelo protocolo de disseminação DIP. Quando um mote recebe a primeira notificação da disseminação, imediatamente suspende a execução da aplicação do usuário.

Cada mote, ao completar o recebimento da tabela, solicita ao servidor os seus parâmetros individuais de configuração através de um protocolo de comunicação para carga de parâmetros. A estrutura de dados dos parâmetros de

configuração está dividida em quatro mensagens. Nesse protocolo o mote envia uma mensagem solicitando um bloco de parâmetros e aguarda a estação servidora enviar os dados. Ao receber uma mensagem com parte dos parâmetros, o mote solicita o próximo bloco. Se após um tempo o mote não receber os dados, ele reenvia a mesma solicitação. Ao final da carga dos parâmetros, o mote libera a execução da aplicação do usuário, agora com a nova configuração.

4.2.3 Etapa: Execução da Aplicação

Uma execução sempre deve ser iniciada por um evento. Normalmente esse evento é o disparo de um temporizador periódico. A partir desse evento, pode-se disparar ações que retornam outros eventos. A sequência entre eventos e ações são definidas pela tabela de transições da FSM da aplicação. O comportamento das ações é definido pelos parâmetros. As principais ações e eventos disponíveis na nossa biblioteca estão definidos na Tabela 4.1.

Tabela 4.1: Principais ações e eventos disponíveis para a aplicação

Componente	**Ação**	**Evento**
Timer Periódico	–	Disparo para uma nova Coleta (TimerFired)
Sensor Local	Iniciar leitura (readSensor())	Resultado da leitura (sensorDone)
	Comparar resultado com parâmetro (testValue())	Resultado da comparação (testValueDone)
Mote Infos	Verificar se é coordenador (testCoord())	Resultado da verificação (testCoordDone)
Agregação	Disparar a agregação indicada nos parâmetros (startAggreg())	Resultado da agregação (AggregDone)
Timer customizado	Iniciar contagem (startTimerX())	Timer finalizado (timerXFired)
Comunicação	Enviar comando para os motes do mesmo grupo (sendComm())	Recebimento de comando de outro mote (recComm)
	Enviar dados para o servidor (sendBS())	Confirmação do envio (sendDone)

A implementação atual permite a configuração de até três operações de coleta simultâneas, que denominamos de Coleta 1, Coleta 2 e Coleta 3. Para isso disponibiliza três conjuntos de parâmetros que tornam possível configurar de forma independentes essas operações, incluindo o temporizador periódico. Com isso é possível definir diferentes máquinas de estados para cada operação de coleta com diferentes períodos. Por exemplo, a Coleta 1 pode enviar a cada

30 minutos o valor médio da temperatura de um grupo de nós, enquanto que a coleta 2 pode monitorar a cada 5 segundos a temperatura para gerar um alarme de incêndio.

Apesar de permitirmos o disparo de até três temporizadores, o sistema só executa uma operação de coleta de cada vez. Para não perdermos possíveis disparos concorrentes, implementamos uma fila de tarefas e transferimos para a FSM da aplicação a responsabilidade de buscar a próxima tarefa ao final da execução da tarefa corrente. Basta a FSM executar a ação "endCollection()" ao final de uma operação de coleta e esperar o evento "nextColl" para iniciar uma nova coleta.

A execução da aplicação pode utilizar dois tipos de protocolos de comunicação. O principal é o protocolo NHops, que é utilizado em todas operações que requerem comunicação entre os motes do mesmo grupo. O outro protocolo utilizado é o de envio de dados para a estação servidora.

4.2.4 Etapa opcional: Eleição

O processo de eleição de coordenador de grupo é opcional e deve ser configurado explicitamente. Quando configurado, é disparado automaticamente após a etapa de inicialização do mote. Caso ainda não exista um mote eleito para o grupo, é executada uma eleição completa. Caso exista um coordenador, esse será informado ao mote recém inicializado. A implementação atual elege o mote com mais energia (voltagem da bateria). Pode-se definir o período em que o mote coordenador deve solicitar uma nova eleição, dessa forma permitindo o revezamento dos motes coordenadores. A reeleição do mote com mais energia permite uma melhor distribuição do uso de energia na rede de sensores sem fio.

Podemos dividir o protocolo de eleição em fases. Na primeira fase um mote sem coordenador solicita aos motes do mesmo grupo quem é o coordenador atual. Se o mote receber uma mensagem de retorno indicando o coordenador, o processo é interrompido e o mote armazena o ID do coordenador. Se o mote não receber um retorno, então dispara a segunda fase.

Na segunda fase, um mote sem coordenador envia, para os motes do mesmo grupo, uma mensagem solicitando o início de uma eleição. Cada mote deve responder ao mote solicitante o seu voto com o respectivo valor de voltagem da bateria. O mote solicitante identifica qual é o mote do grupo com maior voltagem e, em seguida, informa a este mote que ele deverá ser o novo coordenador. O novo coordenador deve informar aos demais elementos do grupo que ele será o novo coordenador. Para o caso em que mais de um

mote inicializa simultaneamente a segunda fase, prevalece como solicitante o mote com o maior ID.

Todas as mensagens da etapa de eleição foram implementadas utilizando o protocolo de formação de grupos NHops.

4.3 Padrão de implementação para as camadas funcionais

Mantivemos o modelo de camadas da arquitetura funcional apresentado na seção 3.2 e definimos um padrão básico que orienta a implementação de cada camada. A nossa implementação segue a arquitetura resumida na figura 3.2 e inclui as funções definidas na tabela 4.2.

Optamos por utilizar o conceito de FSM também como forma de definir parte do fluxo de controle de cada camada. A engrenagem da FSM interage com o restante da camada por meio de eventos e ações. O corpo principal da camada deve implementar as funções para atender as ações e eventos da FSM. A camada também é responsável pela implementação das funções para as interfaces externas.

Na Figura 4.2 apresentamos um modelo de duas camadas utilizando o padrão de implementação proposto. A padrão é composto pela engrenagem da FSM, a definição da FSM, o conjunto de ações disponíveis e por uma ou mais interfaces externas. Tanto as interfaces externas como a ações internas podem gerar eventos para a FSM.

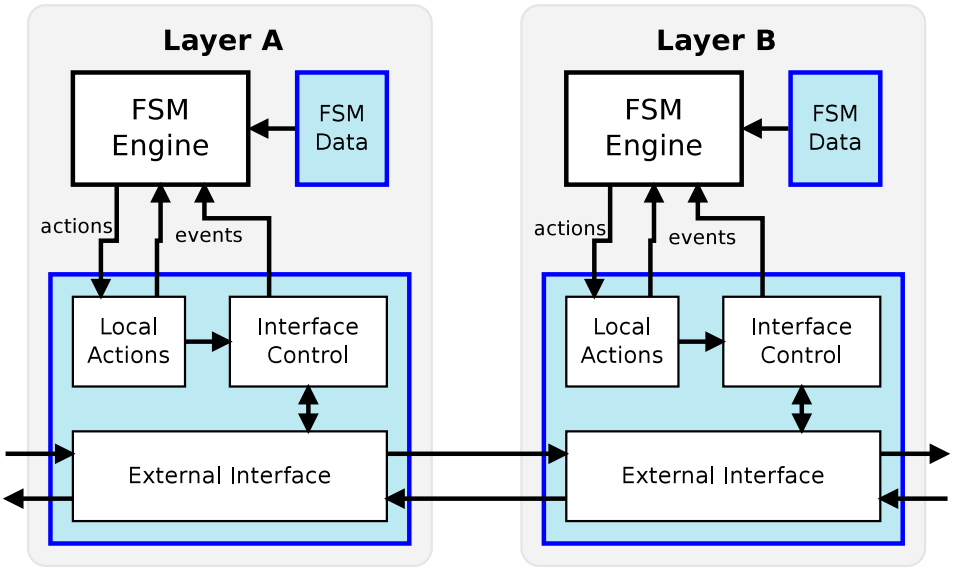

Figura 4.2: Arquitetura básica das camadas funcionais

No apêndice D temos mais detalhes sobre as interfaces das camadas e a lista completa dos componentes NesC implementados. A versão atual não

Tabela 4.2: Biblioteca de funções parametrizáveis

Funcionalidade	**Descrição**
Camada Controle da Aplicação	
Controle FSM	Interpreta a FSM, reage aos eventos e dispara ações
Temporizador Periódico	Controla o disparo periódico dos eventos de coleta
Temporizador Customizável	Controla o disparo eventual de um evento sob demanda da aplicação
Configuração	
Dados FSM	Armazena a configuração atual da FSM
Parâmetros	Armazena a configuração atual para controle das funções da biblioteca
Funções Locais	
Leitura Sensor	Lê o sensor indicado
Camada Agregação	
Controle Agregação	Controla a operação de agregação com os outros motes
Temporizador Agregação	Indica a finalização de uma agregação
Funções Agregação – Parcial	Executa o cálculo parcial da agregação durante o recebimento dos valores
Funções Agregação – Final	Executa o cálculo final da agregação
Camada Agrupamento	
Validação Grupo	Verifica se o grupo da mensagem é o mesmo grupo do mote
Controle Mensagens	Controla o recebimento e envio das mensagens de agregação e eleição
Controle Eleição	Controla a eleição do mote coordenador de grupo
Camada Comunicação	
Reconfigura FSM	Controla o recebimento dos dados de uma nova tabela FSM
Reconfigura Parâmetros	Controla o recebimento dos dados de configuração de controle das funções da biblioteca
Recebe Mensagens	Executa o recebimento, seleção e encaminhamento das mensagens
Envia Mensagens	Controla o envio ou reenvio de mensagens

implementa integralmente todas as funcionalidades identificadas no capítulo 3. A nossa implementação foi realizada usando a linguagem NesC[12] e o sistema operacional TinyOS[11]. Para os testes reais usamos a plataforma de hardware MICAz[22]. Os testes mais exaustivos e monitorados foram executados no simulador TOSSIM[14], que acompanha o TinyOS.

4.4 Exemplo de um fluxo de execução

Escolhemos como exemplo a operação de agregação porque esse tipo de operação utiliza todas as camadas da nossa implementação. Em especial, mostra o comportamento da formação de grupo independente do funcionamento da FSM da aplicação. Para esse exemplo vamos considerar uma aplicação que a cada 30 minutos calcula a temperatura média de cada galpão de uma industria.

Vamos explicar o fluxo seguindo a mesma visão de camadas da arquitetura funcional. Começaremos da visão do usuário do sistema, na camada de aplicação, partindo da definição da FSM e dos parâmetros de configuração. Em seguida descreveremos o comportamento de cada camada para atender a operação exemplo. Detalharemos um pouco mais a camada de comunicação para mostrar o comportamento do protocolo NHops na formação de grupos.

Visão do usuário

A Figura 4.3 ilustra a configuração dos sensores para o cenário exemplo.

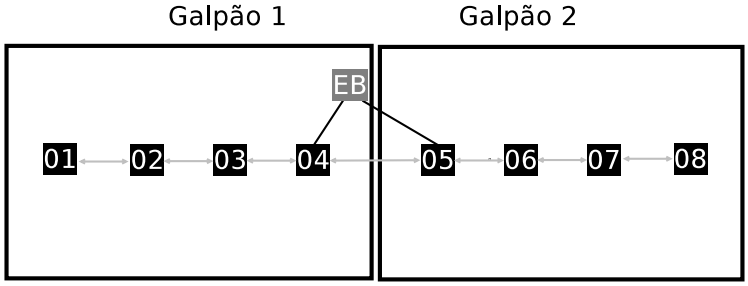

Figura 4.3: Configuração para uma aplicação de monitoração de temperatura

Para configurar a aplicação vamos definir o temporizador da Coleta 1 com o valor de 30 minutos, o cálculo de agregação para "MÉDIA" e o sensor utilizado será o de temperatura. Também vamos definir o parâmetro identificador de grupo dos motes do primeiro galpão com o valor "1" e dos motes do segundo galpão com o valor "2". Vamos utilizar o recurso de eleição de líder para definir um mote coordenador para cada grupo. Esse mote será responsável por disparar a agregação.

A máquina de estados para essa aplicação encontra-se na figura 4.4. Essa máquina controla três ações: (1)Verifica se é um mote coordenador de grupo; (2) Dispara a operação de agregação; (3) Envia dados para a estação base.

Para a nossa explicação vamos detalhar somente duas transições. Assumindo que logo após o evento do temporizador, será chamada a ação que verifica se o mote é o mote coordenador. Na sequência precisamos de uma

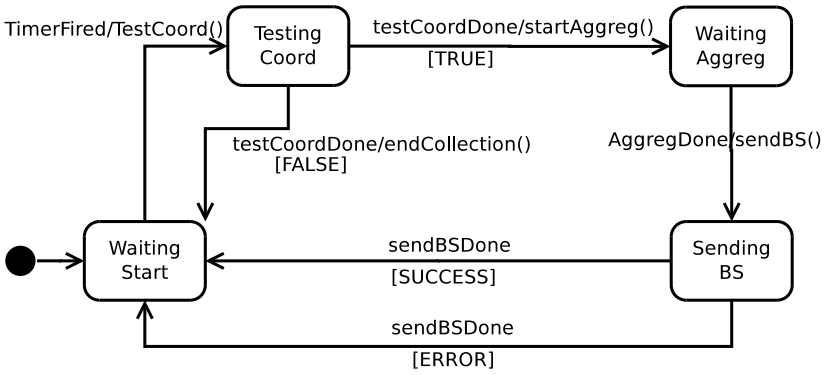

Figura 4.4: FSM para monitoração periódica da temperatura média

transição que reaja ao evento de resposta dessa verificação - testCoordDone - com valor verdadeiro (TRUE) e que dispare uma ação de agregação *startAggerg*. A segunda transição deve reagir ao evento de finalização da agregação - AggregDone - e disparar a ação de envio do resultado para a estação servidora *sendBS*. Na Tabela 4.3, temos um exemplo para essas duas transições.

Tabela 4.3: Transições utilizadas para o exemplo de fluxo

CurrState	**Event**	**Action**	**NewState**
TestingCoord	testCoordDone,TRUE	startAggreg()	WaitingAggreg
WaitingAggreg	AggregDone	sendBS()	Sending

O próximo passo é gerar os arquivos XML com a configuração dos parâmetros e da tabela de transições. Recomendamos a utilização de alguma ferramenta para geração dos arquivos XML. Atualmente utilizamos arquivos de planilhas que geram automaticamente o conteúdo XML no formato requerido pelo programa de carga, bastando que o usuário preencha as tabelas de dados. Essas planilhas são pré-configuradas com as ações e eventos disponibilizados pelo versão atual do sistema. No apêndice E apresentamos os formatos XML para os arquivos de parâmetros e para a tabela FSM.

A carga dos dados de configuração pode ocorrer de duas formas. Na primeira forma, o desenvolvedor pode inserir no código fonte do sistema os valores para o funcionamento da aplicação. Essa configuração será utilizada imediatamente ao iniciarmos um mote. Na segunda forma, o usuário da aplicação pode gerar novos arquivos de configuração e comandar uma configuração remota após a inicialização dos motes da rede. Na inicialização, um mote vai ser sincronizado com a configuração mais recente do restante da rede.

Fluxo de execução

Na Figura 4.5 apresentamos um diagrama com o fluxo de execução para o nosso exemplo. Representamos o mote coordenador (Request Mote) e um mote vizinho (Neighbors Mote) para mostrar a interação entre diferentes motes. As setas em azul (linha contínua) indicam o fluxo para formação do grupo e as setas em verde (pontilhado) indicam o fluxo de retorno dos valores para o cálculo da média.

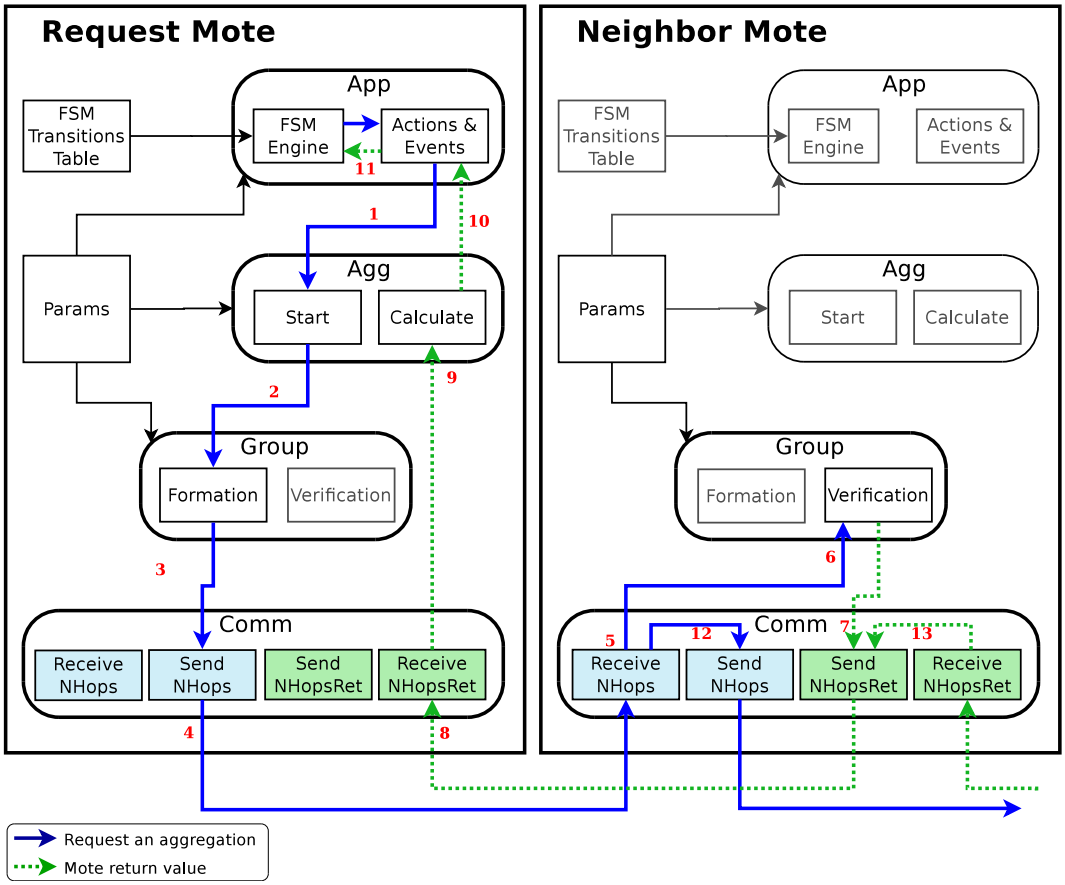

Figura 4.5: Exemplo de fluxo de execução para uma aplicação de agregação

Os números associados às setas na figura 4.5 representam as etapas do fluxo que são descritas a seguir:

1 Dispara Agregação - O controle da FSM, baseado na transição selecionada, dispara a ação startAggreg(). Essa ação executa a função equivalente na interface da camada de agregação.

2 Consulta os parâmetros do cálculo de agregação, lê o sensor local e solicita os valores de temperatura do restante do grupo.

3 Consulta os parâmetros para formação do grupo e monta parte da estrutura de dados da mensagem.

4 Completa o restante da estrutura de dados e envia, via rádio, uma mensagem do tipo NHops.

5/12 [5]O mote vizinho recebe uma mensagem NHops e encaminha para a verificação de grupo. / [12]Se necessário, encaminha a mensagem para a próxima camada de vizinhos.

6 Verifica se o mote pertence ao mesmo grupo do mote solicitante. Se sim, monta a estrutura de dados com o valor do sensor local e solicita o retorno do valor para o mote solicitante.

7 Completa o restante da estrutura de dados e envia uma mensagem do tipo NHopsReturn.

8/13 [8]Recebe as mensagens NHopsReturn com os valores dos motes do mesmo grupo. / [13]Se necessário, encaminha a mensagem para o mote solicitante.

9 Calcula a média parcial para cada valor recebido.

10 Após o timeout da agregação, calcula o valor da média final e envia para a camada de aplicação.

11 Recebe o retorno da agregação e gera um evento para a FSM - AggregDone.

4.5 Experiências no desenvolvimento e novas funcionalidades

Nesta seção, levantamos alguns pontos de nossa experiência com o desenvolvimento do sistema na plataforma TinyOS/NesC, principalmente ajustes necessários para atender à restrição de memória RAM. Em seguida, vamos abordar o procedimento para inclusão de novas funcionalidades no sistema.

4.5.1 Experiências no desenvolvimento

O processo inicial de desenvolvimento foi feito utilizando somente o ambiente simulado do TOSSIM. Esse ambiente não limita a memória utilizada pela aplicação. Inicialmente, assumimos uma linha de implementação que permitia grande flexibilidade na operação da aplicação. Como exemplo de flexibilidade temos a execução de mais de uma operação de agregação disparada simultaneamente no mesmo mote. Para isso a nossa implementação utilizou várias filas de dados como forma de isolar as camadas funcionais e permitir o processamento concorrente entre diferentes operações de coleta. A utilização dessas filas também permitiu que uma ação fosse dividida em várias partes. Possibilitando dessa forma atender um requisito do modelo de execução do TinyOS que recomenda a quebra do código em pequenas atividades.

Com a linha adotada no início da implementação somada à característica do TinyOS de alocação de memória de forma estática, o sistema acabou atingindo cinco vezes o limite de memória RAM disponível em um mote do tipo MICAz. Os maiores ofensores foram a multiplicação das filas para diferentes protocolos de comunicação e diferentes operações. Por exemplo, quatro filas com uma estrutura de dados com 5 itens de 20 bytes vai ocupar um pouco mais que 400 bytes, o que representa 10% dos 4kB disponíveis no MICAz.

Outro ponto investigado foi a forma que o compilador trata as constantes do programa. No caso específico do compilador para processadores AVR (utilizado no MICAz) essas constantes são mantidas em memória RAM, pois a memória de programa (ROM) tem modo de acesso diferenciado da RAM. Foi necessário utilizar funções específicas desses processadores para alocar algumas tabelas de constantes em memória ROM e liberar mais espaço na memória RAM.

Após uma minuciosa análise do uso de memória, tivemos que rever a estratégia de isolamento entre camadas por meio de filas, restringir a concorrência de algumas operações e rever o tamanho e uso de alguns vetores utilizados no programa. Então reescrevemos parte do código utilizando uma fila na camada de aplicação e duas filas centrais (recebimento e envio) para os protocolos de comunicação. Na nova versão, se ocorrer o disparo concorrente entre duas agregações, a segunda operação só é processada após o termino da primeira.

Para teste da versão atual do sistema, implantamos uma aplicação exemplo em sensores MICAz e executamos alguns testes simples em uma rede com três motes desse tipo. O mote MICAz[22] da Crossbow Technology usa um microcontrolador Atmel ATmega128L 8-bit, com 4kB de RAM, 128kB para

memória de programa, utiliza o chip de rádio CC2420 e é alimentado por um par de baterias do tipo AA. O rádio opera na frequência de 2.4GHz no padrão IEEE 802.15.4/ZigBee com taxa de transmissão de dados de 250kbps.

O nosso sistema ocupa atualmente 3.666 bytes de memória RAM e 50.436 bytes de memória ROM. A configuração da aplicação do usuário ocupa um espaço de memória já reservada no sistema, dessa forma não aumentando o uso de memória. O objetivo principal dessa avaliação foi confirmar que o modelo de programação proposto é adequado às limitações de memória dos dispositivos usados nas RSSF.

4.5.2 Inclusão de novas funcionalidades

Podem existir situações em que o conjunto de funcionalidades disponibilizadas não consigam atender alguma aplicação. Nessas situações, é necessário alterar o código existente e carregar todo os sistema no mote. Como essa carga não é uma ação de reconfiguração remota da aplicação, é preciso conectar cada mote, via cabo, ao sistema de desenvolvimento. A arquitetura em camadas funcionais e o modelo de configuração permite um grande reaproveitamento do código existente. Para cada nova funcionalidade, é necessário avaliar os pontos de impacto no sistema para garantir uma intervenção coordenada e que não impacte o funcionamento da parte existente.

Como identificado no item 4.5.1 sobre a nossa experiência no desenvolvimento, o desenvolvedor deve ter muita atenção na criação ou alteração das estruturas de dados. Facilmente uma lista de valores pode estourar a capacidade de memória do mote ou uma inclusão de novos parâmetros pode ultrapassar o limite de bytes de uma mensagem via rádio.

Como possíveis alterações podemos ter a inclusão de uma nova ação/evento, a inclusão de um novo cálculo de agregação e inclusão ou troca de algum protocolo de comunicação. A seguir, detalhamos essas alterações e explicitamos o respectivo impacto no sistema.

Inclusão de uma nova ação/evento

As ações e eventos disponíveis para utilização pela máquina de estado da aplicação do usuário são predefinidas no código executável carregado inicialmente no mote.

Para disponibilizar uma nova ação para a FSM da aplicação é necessário criar um novo identificador de ação e incluir a chamada da nova função na estrutura “switch/case” de controle das ações. O identificador da ação deve ser incluído no arquivo “AppMain.h” e o controle das ações é feito pela função

"FSM.execAction()" do arquivo "AppMainC.nc". A entrada da nova ação deve estar neste último arquivo. A implementação da nova ação pode estar distribuída pelas várias camadas do sistema ou simplesmente na camada de aplicação no arquivo "AppMainC.nc".

Para o caso de um novo evento, provavelmente gerado a partir de uma nova ação, deve-se também criar um novo identificador de evento no arquivo "AppMain.h" e executar uma chamada à função "FSM.enqueueEvt()" de dentro do arquivo "AppMainC.nc".

Deve-se também incluir os novos identificadores na ferramenta de configuração disponibilizada para o usuário, no nosso caso basta incluir a nova definição na planilha de configuração.

A nova funcionalidade pode reutilizar as funções das camadas inferiores ou aplicar pequenas modificações nas mesmas. Deve-se ter atenção especial para os casos que necessitem alterar a estrutura dos parâmetros de configuração. Se necessária a inclusão de um novo parâmetro, esse deve refletir na mensagem de configuração e consequentemente na ferramenta de configuração disponibilizada para o usuário. A estrutura de dados dos parâmetros fica no arquivo "WSNDyn.h" e as alterações devem ser refletidas no componente "MoteDataC.nc" que é responsável pelo acesso a esses dados.

Inclusão de um novo cálculo de agregação

A inclusão de um novo cálculo de agregação é uma operação bastante simples, basta criar um novo identificador para o cálculo e incluir a nova função na camada de agregação. O novo identificador deve ser criado no arquivo "WSNDyn.h", na seção "AO-Aggregation Operation". A nova função de agregação deve ser criada no arquivo "AggregMainP.nc". Pode-se utilizar as funções já existentes como template para a nova função.

Inclusão ou troca de algum protocolo de comunicação

As alterações nos protocolos devem ser muito bem avaliadas, pois impactam o funcionamento da maioria das operações do sistemas. Essas alterações podem ser bastante complexas sendo necessário que o desenvolvedor conheça a fundo o funcionamento do sistema. De forma geral, deve-se alterar as estruturas de dados das mensagens de comunicação e a lógica de controle do respectivo protocolo. A arquitetura em camadas facilita essas alterações, sendo possível fazer uma alteração na camada de comunicação e não impactar o funcionamento do restante do sistema. Aqui vale o comentário anterior sobre a estrutura dos parâmetros de configuração.

Os principais componentes envolvidos são “CommunicationP” e “Comm-MainP”. A estrutura de dados das mensagens está no arquivo “WSNDyn.h”.