5
Avaliação

A nossa avaliação foi dividida em duas partes. Na primeira exercitamos nosso modelo de programação com a implementação das três aplicações de referência definidas na subseção 3.1.1. Na segunda parte, fazemos uma avaliação preliminar da facilidade de programação utilizando o modelo de programação proposto e o impacto no tempo de vida útil da rede com a utilização da configuração remota.

5.1
Implementação das aplicações de referência

Nessa seção aplicamos o modelo de programação proposto na construção das três aplicações de referência descritas na seção 3.1.1. Usamos a primeira aplicação de referência para explicar em mais detalhes a utilização do modelo de programação. No apêndice B.2 apresentamos os diagramas e as tabelas de configuração para as três aplicações.

Conforme a necessidade da aplicação, pode-se definir uma ou mais máquinas de estados para cada operação de coleta.

A separação entre a configuração dos componentes e a configuração da FSM permitiu um grande reaproveitamento das definições das FSMs. Com isso, temos a mesmas FSMs de inicialização e alarme para as aplicações 1 e 2, e a mesma FSM de monitoração para as aplicações 2 e 3.

5.1.1
Aplicação 1 - Alarme de incêndio florestal

Essa aplicação foi definida na subseção 3.1.1 e tem como objetivo detectar rapidamente focos de incêndio em florestas. A sua implementação utilizou duas máquinas de estados. A primeira é a extensão da máquina de estados da inicialização para controle da suspensão do alarme. A segunda máquina executa as operações de monitoração e alarme utilizando os parâmetros de configuração da Coleta 1.

Para comandar a suspensão do alarme utilizamos a ação *sendCustom01* que envia uma mensagem genérica para os motes do mesmo grupo. O retorno

para a situação normal é disparado por um temporizador também genérico *CustomTimer01*.

Para a operação efetiva do alarme, configuramos o temporizador da Coleta 1 com o período de leitura do sensor local de temperatura. O resultado da leitura de temperatura é comparado com o respectivo parâmetro de comparação. Quando o resultado dessa comparação for verdadeiro, é disparada uma operação de agregação para contabilizar os valores de temperatura dos motes do mesmo grupo. Essa contabilização num primeiro estágio sumariza a quantidade de valores que atendam ou não uma determinada condição de comparação. Em um segundo estágio compara o resultado dessa sumarização com outro valor parametrizado. Em seguida, se o resultado da agregação for verdadeiro, são enviados o alarme para a estação base e a mensagem genérica para os motes do mesmo grupo.

Primeira máquina de estados

Na Figura 5.1, apresentamos o diagrama com as transições da primeira máquina de estado.

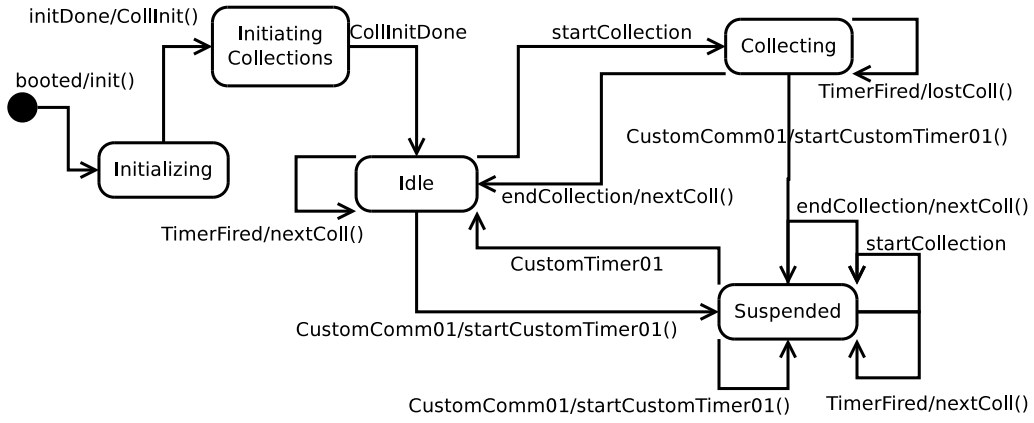

Figura 5.1: Aplicação 1 - FSM para inicialização do mote e controle geral da aplicação

Podemos dividir essa máquina nas etapas Inicialização, Ocioso, Coletando e Suspenso. Na Inicialização é executado um procedimento interno padrão para inicialização sistêmica (estados Initializing e Initiating Collections). No estado Ocioso (Idle), o mote estará pronto para executar uma nova coleta. O estado Coletando (Collecting) indica que o mote está executando uma operação de coleta. O estado Suspenso (Suspended) indica que o mote recebeu um comando para inibir a execução das próximas coletas.

Os principais eventos considerados nessa máquina são *TimerFired* que indica que um temporizador periódico de uma coleta foi disparado; *startCol-*

lection que indica que uma nova coleta foi disparada; *endCollection* que indica que uma coleta foi finalizada; *CustomComm01* que indica que a mensagem genérica 01 foi recebida pelo mote. Essa mensagem genérica é usada para colocar o mote no estado Suspenso; *CustomTimer01* que indica que o temporizador genérico 01 foi disparado. Esse temporizador genérico é usado para retirar o mote do estado Suspenso.

As ações utilizadas são *startCustomTimer01* para iniciar a contagem de tempo para o temporizador genérico 01; *nextColl* para disparar o próximo evento *startCollection*; *lostColl* para cancelar o próximo evento *startCollection*, é utilizado quando o temporizador dispara antes da finalização de uma coleta.

No apêndice B.2, a Tabela B.4 apresenta todos os parâmetros de configuração e a Tabela B.7 apresenta a configuração para a tabela de transições da FSM.

Segunda máquina de estados

Na Figura 5.2, apresentamos o diagrama com as transições da segunda máquina de estado.

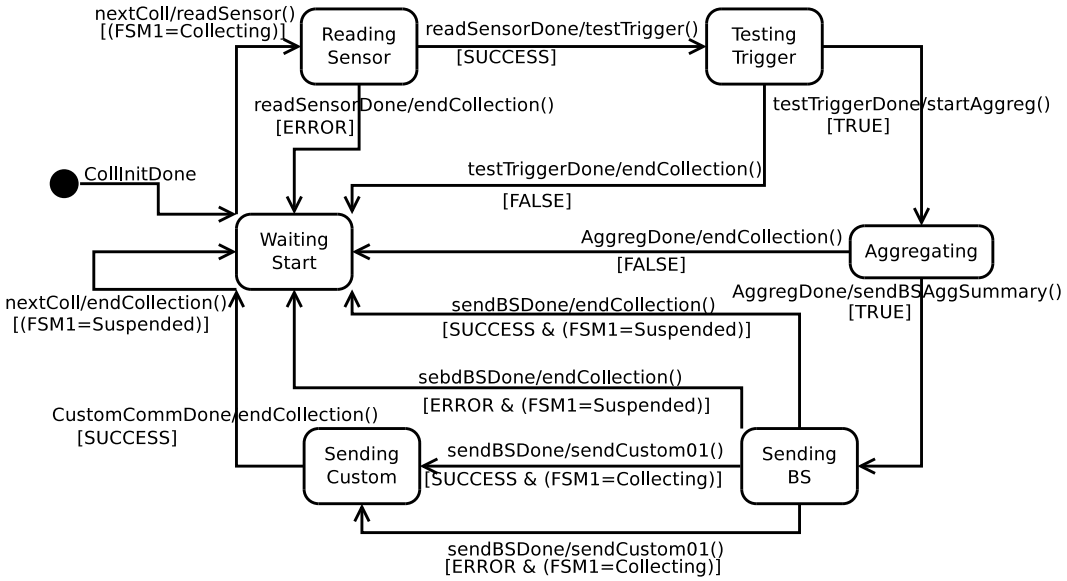

Figura 5.2: Aplicação 1 - FSM para geração do alarme

O diagrama pode ser dividido nas etapas de Alarme Local, Cálculo do Sumário, Envio de Dados e Suspensão dos Vizinhos. Na etapa de Alarme Local, temos os estados *Waiting Start* que fica aguardando um evento para início de uma coleta, *Reading Sensor* que fica aguardando a resposta do sensor de temperatura e *Testing Trigger* que aguarda a resposta do teste de valor do alarme. Na etapa para Cálculo do Sumário, o estado *Aggregating* indica que

está sendo executada uma operação de agregação para calcular o sumário dos valores de temperatura do grupo de nós. Na etapa Envio de Dados, o estado *Sending BS* indica que o mote está aguardando a finalização do comando para envio de dados para a estação base. Na etapa Suspensão dos vizinhos, o estado *Sending Custom* indica que o mote está aguardando a finalização do comando enviado para os motes vizinhos.

Os principais eventos utilizados são *nextColl* que indica que existe um evento de coleta disponível na fila; *readSensorDone* que indica que a leitura do sensor foi finalizada; *testTriggerDone* que retorna verdadeiro ou falso para o teste de comparação do valor de alarme; *AggregDone* que retorna o resultado da operação de agregação; *sendBSDone* que indica a finalização da ação de envio de dados para a estação base; *CustomCommDone* que indica a finalização da ação de envio do comando genérico.

As ações utilizadas são *readSensor* para disparar uma leitura do sensor de temperatura; *testTrigger* para disparar a comparação do resultado do sensor com o parâmetro de alarme; *startAggreg* para iniciar a operação de agregação; *sendBSAggSummary* para enviar os dados do cálculo do sumário para a estação base. *sendCustom01* para enviar o comando genérico 01 para os motes do mesmo grupo. *endColection* retira o evento de coleta da fila para poder processar o próximo evento.

Nessa aplicação utilizamos o recurso de dependência entre máquinas para inibir uma coleta quando o mote estiver no estado Suspenso. Note no diagrama da Figura 5.2 que algumas condições de transições foram definidas por `[(FSM1=Collecting)]` ou por `[(FSM1=Suspended)]`. Isso indica que essas transições só ocorrem quando o estado da máquina 1 (FSM1) for o estado indicado na condição.

No apêndice B.2, a Tabela B.4, apresenta todos os parâmetros de configuração e, as Tabelas B.7 e B.8, apresentam as configurações para as tabelas de transições das FSMs.

5.1.2 Aplicação 2 - Monitor de temperatura e Alarme de incêndio predial

Essa aplicação foi definida na subseção 3.1.1 e tem como objetivos monitorar a temperatura média dos ambientes do prédio e gerar alarmes em caso de incêndio. A sua implementação utilizou três máquinas de estados. Na primeira máquina estendemos, da mesma forma que na aplicação anterior, a máquina de estados da inicialização para controle da suspensão do alarme. Para a operação efetiva do alarme, definimos uma segunda máquina de estados que utiliza os parâmetros da Coleta 1. Para a operação de monitoração periódica,

definimos uma terceira máquina de estados que utiliza os parâmetros da Coleta 2.

As duas primeiras máquinas operam da mesma forma que a aplicação 1, diferindo apenas quanto aos parâmetros de configuração. Na terceira máquina, o temporizador periódico da Coleta 2 dispara a ação para verificar se o mote é um mote coordenador. Caso o resultado da comparação seja verdadeiro, é disparada uma operação de agregação para calcular a temperatura média dos motes do mesmo grupo. Em seguida o resultado da agregação é enviado para a estação base.

No apêndice B.2, as figuras B.3, B.4 e B.5, apresentam os diagramas das máquinas de estados dessa aplicação. Na Tabela B.5, temos todos os parâmetros de configuração e, nas Tabelas B.9,B.10 e B.11, temos a configuração para as tabelas de transições.

5.1.3 Aplicação 3 - Estacionamento urbano e Monitoração de vagas

Essa aplicação foi definida na subseção 3.1.1 e tem como objetivos auxiliar motoristas na reserva de uma vaga de estacionamento e informar para uma central de controle a situação das vagas em cada região. A sua implementação também utilizou três máquinas de estados. Na primeira máquina estendemos a máquina de estados da inicialização para simplesmente controlar a ação de reserva. Esse controle é disparado manualmente a partir do acionamento de uma chave local (solicitação de vaga), onde é gerado um evento de início de coleta.

Para a operação de monitoração periódica, vamos definir uma segunda máquina de estados que utiliza os parâmetros da Coleta 1. O funcionamento é similar ao monitoramento da Aplicação 2.

Para a operação de reserva de vaga, vamos definir uma terceira máquina de estados que utiliza os parâmetros da Coleta 2. Essa coleta é disparada por um evento gerado pela chave na primeira máquina de estado, que dispara uma operação de agregação. Essa operação de agregação executa todos os passos necessários para a reserva e confirmação de um recurso num grupo de motes. Após uma reserva, um novo evento de disparo da coleta vai gerar uma operação de finalização da reserva que vai providenciar o cancelamento da reserva atual.

No apêndice B.2, as figuras B.6, B.7 e B.8, apresentam os diagramas das máquinas de estados dessa aplicação. Na Tabela B.6, temos todos os parâmetros de configuração utilizados e, nas Tabelas B.12 e B.13, temos as configurações para as tabelas de transições.

5.2 Avaliação da facilidade de programação e do impacto da configuração remota

Para a segunda parte dos testes, avaliamos o nosso sistema sob dois aspectos: (i) facilidades de configuração e reconfiguração de aplicações; (ii) impacto adicional em termos de comunicação para a reconfiguração de uma aplicação em execução.

No primeiro aspecto, as métricas utilizadas foram a quantidade de linhas na tabela de transições para a FSM e o número de parâmetros necessários para a implementação da aplicação. Também fizemos uma implementação alternativa diretamente em NesC para uma comparação relativa. Nesse caso utilizamos a quantidade de linhas dos arquivos fontes e a quantidade de operações representadas pelo caractere ";". Apesar de não serem métricas semelhantes, a interpretação desses valores apoia a nossa comparação. Adicionalmente apresentamos o espaço utilizado de memória RAM e ROM para cada implementação.

No segundo aspecto, a métrica utilizada foi a quantidade de mensagens enviadas na rede. Conforme [23] podemos considerar o envio de uma mensagem como a operação de maior custo em relação ao consumo de bateria e, consequentemente, a operação que mais afeta o tempo de vida útil de um nó. Para contabilizar o número de mensagens enviadas, executamos a aplicação no simulador TOSSIM habilitando os logs das funções de comunicação.

No próximo item apresentamos a aplicação utilizada como referência para nossos testes. Em seguida, abordamos cada teste e apresentamos os resultados específicos.

5.2.1 Aplicação de referência

Em nossos testes, vamos considerar uma aplicação de monitoração predial, em que o motes estão distribuídos de forma equidistante em três ambientes. Em cada ambiente há um grupo de nove motes. Cada mote só consegue se comunicar com os vizinhos no raio de alcance do rádio, independentemente de estarem ou não no mesmo grupo. Para conseguir formar os grupos configuramos o parâmetro de salto máximo (n_hops_max) igual a dois. A estação servidora (fonte de reconfiguração da aplicação) está conectada ao nó central (25) através da estação base (EB). Essa configuração requer, propositalmente, que os protocolos utilizados precisem rotear a maioria das mensagens.

A Figura 5.3 apresenta a nossa rede de referência. As setas indicam o alcance da comunicação entre os motes.

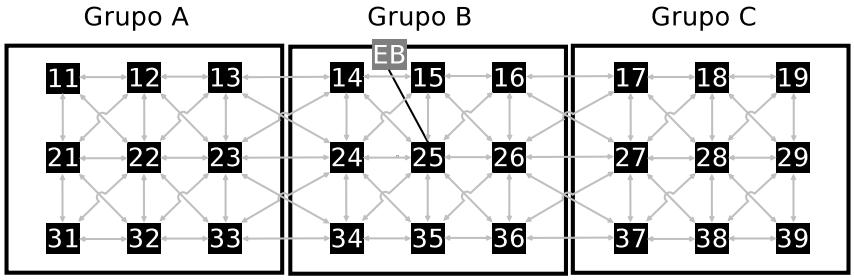

Figura 5.3: Rede de referência

5.2.2 Primeiro Teste: Desenvolvendo uma nova aplicação

O primeiro teste teve o objetivo de exercitar a criação de uma nova aplicação no nosso sistema. Para isso, definimos uma aplicação de coleta periódica de dados. A nossa aplicação de teste calcula, periodicamente, a temperatura média de cada grupo de nós da rede de referência e, em seguida, envia esse valor para a estação servidora. O nó centralizador de cada grupo é definido num processo de eleição de líder em que o nó com a maior disponibilidade de bateria é eleito. Com o objetivo de distribuir o consumo, periodicamente é executado um processo para reeleição do líder.

Como citado no capítulo 3, a configuração da aplicação utiliza um conjunto de parâmetros e uma tabela de transições como definição do fluxo de funcionamento da aplicação. Para essa aplicação foi preciso definir apenas seis parâmetros, que incluem o período da coleta, o tipo de operação de agregação e a definição do grupo. Já em relação à FSM foi necessário definir 15 transições. Sendo oito para a primeira máquina de estados que inclui a inicialização obrigatória do mote e sete para o controle da aplicação de coleta. Na Tabela 5.1 apresentamos um resumo da quantidade utilizada de parâmetros, de transições e de espaço de memória. Essas quantidades indicam o esforço da atividade de programação usando o sistema proposto. Essa atividade é equivalente à programação direta em NesC, na qual o esforço pode ser medido pela quantidade de linhas de código ou operações.

A Figura 5.4.a apresenta a FSM para controle da coleta e a Figura 5.4.b mostra os parâmetros usados.

Executamos essa aplicação no simulador do TinyOS para contabilizar a quantidade de envios em cada etapa de funcionamento da aplicação. Na Tabela 5.2, apresentamos o resultado da sumarização entre cada etapa e os respectivos contadores. Essa tabela também incluí os contadores do processo de reconfiguração do segundo teste, explicado no próximo item. Na Tabela 5.3, extraímos da tabela anterior as quantidades de mensagens enviadas para as operações CTP Inic (*Inicialização CTP*), Eleição, Reeleição (*Reeleição* -

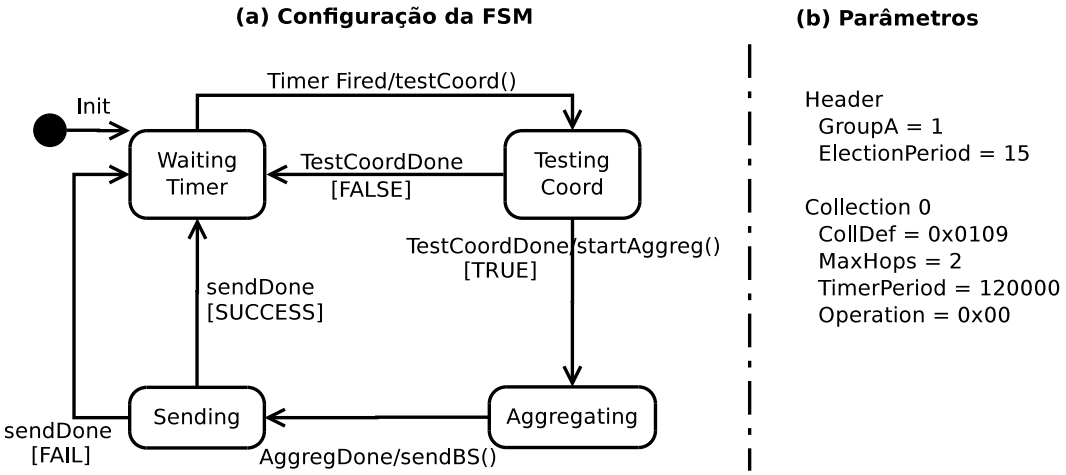

Figura 5.4: (a) FSM para controle de uma aplicação de coleta e (b) Definição dos respectivos parâmetros.

Tabela 5.1: Primeiro Teste: Resultado para as métricas da implementação no modelo proposto

	Quantidade
Parâmetros	6 valores
Transições de FSM	15 registros
Memória RAM	3.666 bytes
Memória ROM	50.436 bytes

Coll06) e Coleta (*valor médio das coletas*). Na Figura 5.5, temos uma visão da Tabela 5.2 com a evolução dos contadores durante a execução do teste. Nessa figura não identificamos a Reeleição, pois essa se mistura com a coleta 6.

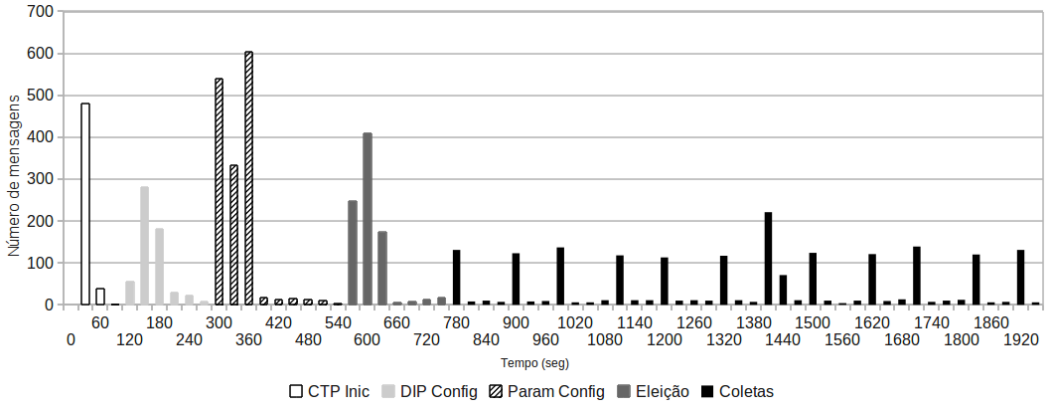

Figura 5.5: Distribuição no tempo de cada etapa de operação

O protocolo CTP troca mensagens periodicamente para a manutenção

Tabela 5.2: Sumarização da quantidade de mensagens enviadas para cada etapa e os respectivos contadores

	Contador					
Etapa	**Ack**	**Eleição**	**CTP**	**DIP**	**Config**	**Coleta**
Inicialização (CTP)	0	0	511	0	0	0
Configuração (DIP)	0	0	53	499	0	0
Eleição	75	735	14	45	0	0
Configuração (Param)	606	0	404	49	257	0
Coll01	30	0	18	24	0	79
Coll02	58	0	26	25	0	160
Coll03	30	0	10	25	0	71
Coll04	30	0	22	28	0	60
Coll05	30	0	20	26	0	67
Coll06/Reeleição	69	116	16	26	0	78
Coll07	29	0	9	26	0	79
Coll08	58	0	37	23	0	155
Coll09	30	0	17	24	0	72
Coll10	30	0	9	26	0	80

Tabela 5.3: Quantidade de mensagens enviadas para cada tipo de operação

Etapa	**CTP Inic**	**Eleição**	**Reeleição**	**Coleta**
Aplicação	–	810	150	127
CTP	511	14	8	18

da topologia de roteamento, deixando um resíduo de mensagens durante a execução da aplicação. Isso pode ser observado na Tabela 5.3 para as operações que não utilizam o CTP como o processo de eleição.

Implementação alternativa em NesC

Com o objetivo de fornecer uma base de comparação, construímos uma versão alternativa da aplicação de teste usando somente o NesC/TinyOS. Assumindo que esse tipo de comparação sempre será relativa, adotamos várias simplificações que facilitaram a codificação da mesma. As principais simplificações foram:

- Reorganizamos a sequência dos motes e formamos os grupos pelas dezenas dos identificadores, i.e., grupo A é formado pelos motes de 11 até 19.
- Fixamos o mote central do grupo como o mote coordenador. Com isso a comunicação do grupo foi implementada utilizando somente 1-hop de alcance e também não implementamos o processo para eleição e reeleição.
- Implementamos a comunicação no sentido da rede para a estação base utilizando somente o protocolo CTP.

Utilizamos como principais elementos um temporizador periódico para a coleta, um protocolo simplificado para agregação entre vizinhos 1-hop, o sensor de temperatura (demoSensor), um controle para acumulo e cálculo da média e o protocolo CTP para envio de informação para estação base.

Mantivemos variáveis de estado da execução para poder controlar a execução, impedir o sobreposição entre coletas e identificar o destino da leitura do sensor. Não implementamos nenhum mecanismo para reconfiguração remota da aplicação. Para alterar alguma coisa é necessário carregar localmente todo o código do mote.

Na Tabela 5.4 apresentamos o resultado para as métricas da implementação alternativa da aplicação do primeiro teste.

Tabela 5.4: Primeiro Teste: Resultado para as métricas da implementação alternativa em NesC

	Quantidade
Linhas	366 linhas
Operações	188 caracteres ";"
Memória RAM	1.678 bytes
Memória ROM	25.206 bytes

5.2.3 Segundo Teste: Modificando uma aplicação

Para essa parte do teste, vamos supor que, depois de instalada a aplicação, precisemos incluir uma monitoração do tipo alarme que envie para a estação servidora uma mensagem quando a temperatura ultrapassar determinado valor de referência. Com isso podemos aumentar o período de coleta original, diminuindo o consumo de energia.

Essa modificação pode ser vista como a inclusão de mais uma aplicação na rede, ativando uma segunda operação de coleta e adicionando uma nova máquina de estado. Em nossa implementação, a aplicação necessitou definir cinco novos parâmetros, dentre os quais estão o período de leitura do sensor local, o valor de referência e a operação de comparação para gerar o alarme. Também modificamos o período de coleta da configuração anterior. A nova máquina de estados requer sete transições, alterando de 15 para 22 o número total de transições de estados. Na Tabela 5.5 apresentamos um resumo da quantidade utilizada de parâmetros, de transições e de espaço de memória. Note que a quantidade de memória não é alterada, pois a nossa implementação já reserva o espaço total permitido para os parâmetros e transições. A Figura 5.6.a apresenta a segunda FSM para controle do alarme e a Figura 5.6.b mostra a estrutura de dados com a lista de parâmetros atualizada.

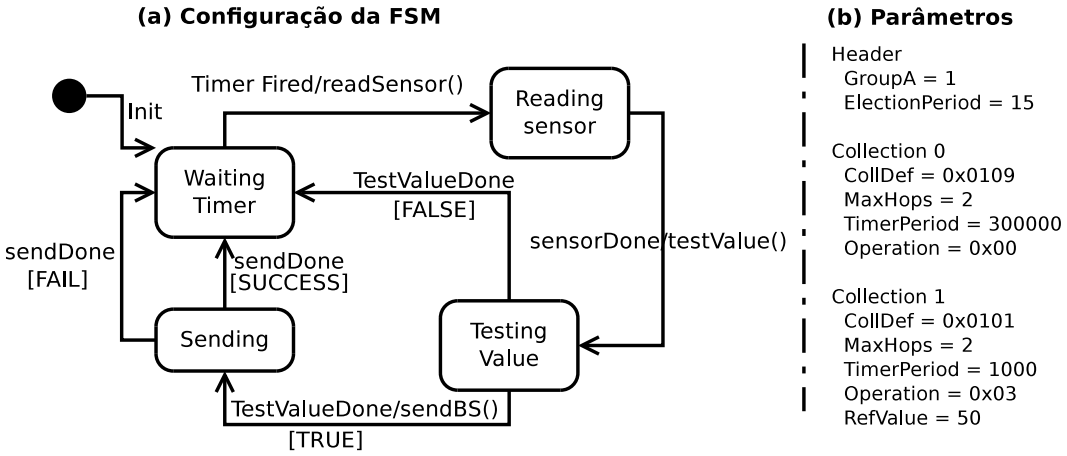

Figura 5.6: (a) FSM para controle de uma aplicação de alarme e (b) Definição dos respectivos parâmetros

Tabela 5.5: Segundo Teste: Resultado para as métricas da implementação no modelo proposto

	Quantidade
Parâmetros	11 valores
Transições de FSM	22 registros
Memória RAM	3.666 bytes
Memória ROM	50.436 bytes

Executamos essa aplicação no simulador e contabilizamos a quantidade de mensagens adicionais por conta da reconfiguração remota. Apresentamos esses valores na Tabela 5.6.

Tabela 5.6: Quantidade de envios para a operação de reconfiguração remota

Etapa	**DIP Inic**	**Param Inic**
Aplicação	–	863
CTP	–	404
DIP	499	49

Como o CTP, o protocolo do DIP também troca mensagens periódicas para manutenção do sincronismo dos dados, deixando um resíduo de mensagens durante a execução da aplicação. Isso pode ser observado durante operações que não utilizam o DIP como o processo de inicialização dos parâmetros.

Implementação alternativa em NesC

Para a execução do segundo teste na versão alternativa, alteramos os arquivos fontes utilizados no primeiro teste para incluir a nova funcionalidade. Tivemos que utilizar mais um temporizador periódico para controle do alarme,

adaptamos a mensagem de envio de dados para a estação base para acomodar o envio da média e do alarme e implementamos o teste de alarme. O controle da execução teve um aumento de complexidade, pois agora existe a concorrência entre a operação de agregação e a operação de alarme. Para isso estendemos as variáveis de estado e condicionamos várias operações a essas variáveis. Também foi necessário implementar um mecanismo simplificado para postergar a execução de uma operação disparada durante a execução de outra operação.

Na Tabela 5.7 apresentamos o resultado para as métricas da implementação alternativa da aplicação do segundo teste. Esse valores se referem a toda aplicação, que inclui as operações de agregação e alarme.

Tabela 5.7: Segundo Teste: Resultado para as métricas da implementação alternativa em NesC

	Quantidade
Linhas	446 linhas
Operações	231 caracteres ";"
Memória RAM	1.688 bytes
Memória ROM	25.438 bytes

5.2.4 Análise do impacto da reconfiguração

A partir dos valores experimentais podemos projetar uma operação de longo prazo e calcular o custo de reconfiguração em função da quantidade de coletas. Vamos considerar nossa implementação para a aplicação de coleta periódica executando na configuração de rede de referência com os três grupos de coleta. A partir dos valores experimentais, podemos projetar uma operação de longo prazo e calcular o custo de reconfiguração em função da quantidade de coletas.

Consideramos a quantidade total de mensagens utilizadas para reconfiguração como a soma do total de mensagens de um ciclo de disseminação da tabela FSM e do total de mensagens trocadas na rede para um ciclo de configuração dos parâmetros de cada mote. Para a coleta vamos considerar a quantidade de mensagens trocadas na rede para um ciclo de coleta. Basta dividir esses dois totais para calcular o impacto de uma reconfiguração em relação a uma coleta, com isso obteremos o custo de uma reconfiguração em quantidades de possíveis operações de coletas.

A partir dos valores das Tabelas 5.3 e 5.6, podemos dizer, conforme a equação 5-1, que uma reconfiguração equivale a aproximadamente 11 coletas.

$$Reconf = \frac{(DIP_{Inic} + Param_{Inic})}{Coleta} = \frac{552 + 1316}{170} = 11\ Coletas \qquad (5\text{-}1)$$

Supondo uma aplicação funcionando por três meses com coleta a cada 10 minutos, o custo de uma reconfiguração remota equivale 110 minutos de operação ou 0,083% do período de três meses. (3 meses = 131.760 minutos)

Adicionalmente aos nossos testes originais, vamos supor um novo protocolo de disseminação da FSM para reduzir ao máximo o valor de DIP Residual. Dessa forma, reduz-se a interferência do protocolo DIP, que é utilizado na reconfiguração, no cálculo do total de mensagens de coleta. A equação 5-2 mostra o resultado para esse novo cálculo:

$$Reconf = \frac{(DIP_{Inic} + Param_{Inic})}{Coleta} = \frac{552 + 1267}{145} = 12,5\ Coletas \qquad (5\text{-}2)$$

Para esse caso, o impacto será de 125 minutos com percentual de 0,095%.

É importante ressaltar que essa projeção considera o comportamento da rede como um todo. Quando olhamos a quantidade de envios para cada nó, identificamos valores maiores para nós roteadores e nós coordenadores. A nossa projeção não se altera muito, pois a diferença para os nós roteadores é equivalente tanto na reconfiguração remota como na operação normal. Já para os nós coordenadores, o processo de reeleição garante o revezamento em função da carga da bateria.

5.2.5 Discussão

Para subsidiar a nossa discussão, apresentamos na Tabela 5.8 um comparativo entre as implementações utilizando o modelo proposto e em NesC. Essa tabela mostra as diferenças entre a versão do primeiro teste (v1) e a versão incluindo a nova funcionalidade do segundo teste (v2). Também incluímos um valor aproximado do tempo gasto na construção e teste para cada implementação. A versão NesC foi simplificada e os números relativos ao tamanho de código seriam bem maiores caso implementássemos funcionalidades equivalentes à versão no modelo proposto. Esses valores refletem uma situação específica da nossa avaliação. Tivemos somente um usuário que já tinha experiência no modelo de programação proposto e no próprio NesC/TinyOS.

Comparando a nossa experiência com a implementação no modelo proposto e a utilizando NesC, podemos afirmar que uma solução parametrizável reduz o esforço necessário para reconfigurar uma aplicação. A quantidade de elementos alterados é pequena perto de uma intervenção em um programa NesC. O desenvolvimento em NesC tende a ser mais susceptível a erros de programação do que a configuração de módulos já prontos.

A facilidade introduzida pelo nosso modelo de programação acaba re-

Tabela 5.8: Resumo comparativo entre as duas implementações

item	**Tipo Implementação**	
	Modelo Proposto	**NesC/TinyOS**
Quant. v1	6 parâmetros e 15 transições.	366 linhas ou 188 operações
Quant. v2	11 parâmetros e 22 transições.	446 linhas ou 231 operações
Diferença	Parâmetros: altera 1 e inclui 5. Transições: inclui 7	Linhas: remove 14 e inclui 94
Esforço (v1+v2)	0,5 dia	2 dias
Memória utilizada	ROM: 50.436 bytes RAM: 3.666 bytes	ROM: 25.438 bytes RAM: 1.688 bytes

fletindo na quantidade de memória utilizada. No nosso sistema, a utilização de memória é definida pelos componentes do sistema previamente carregado e independe da configuração da aplicação. Já na implementação em NesC, o uso de memória tende a ser mais econômico, principalmente por carregar somente os componentes necessários para a versão específica da aplicação.