Um método para otimização de aplicações de Redes de Sensores através de análises do código executável

Jônatas C. Oliveira [1], Carlos M. S. Figueiredo[2], Eduardo F. Nakamura[2]

[1]Departamento de Ciência da Computação - Universidade Federal do Amazonas (UFAM)
Av. Gen. Rodrigo Octávio Jordão Ramos, 3000, Coroado I, 69077-000 Manaus, AM.

[2]Fundação Centro de Análise, Pesquisa e Inovação Tecnológica - FUCAPI
Av. Danilo de Matos Areosa, 381, Distrito Industrial - 69075-351 Manaus, AM.

jco@dcc.ufam.edu.br,{eduardo.nakamura,mauricio.figueiredo}@fucapi.br

Abstract. *In Wireless Sensor Networks, the concern of developing efficient applications is a constant need due to its limited resources. Especially, frameworks of development, operating systems and compilers are concern in generating efficients executables codes. However, optimization opportunities are still possible to find through analysis of the application executable code. We present a method that, through the analysis and simulation of the executable code of the sensor node, we can find optimization opportunities, and new considerations of application efficiency are observed. We utilized the method on the TinyOS's Antithief Application, reducing 6,2% the microprocessor time in active state.*

Resumo. *Em Redes de Sensores, a preocupação com o desenvolvimento de aplicações mais eficientes é uma necessidade constante devido a limitação de recursos. Particularmente, ferramentas de desenvolvimento, SOs e compiladores se preocupam em gerar executáveis eficientes. No entanto, oportunidades de melhoria na aplicação ainda podem ser encontradas através de análises do seu código executável. Apresentamos um método que possibilita encontrar oportunidades de otimizações, além de que novas considerações da eficiência da aplicação sejam observadas. Aplicamos o método na aplicação Antithief do TinyOS, reduzindo em 6,2% o tempo ativo do processador.*

1. Introdução

Rede de Sensores Sem Fio (RSSF)[Akyildiz et al. 2002] tem obtido atenção dos pesquisadores devido ao seu potencial de ser empregado como infraestrutura para computação ubíqua. Devido a limitação de recursos do nó sensor, como energia, processamento e armazenamento, todo o projeto de uma RSSF visa usufruir o máximo da eficiência desses recursos. No entanto, as otimizações dos compiladores para códigos de RSSFs, assim como nos compiladores de uso comum, têm como principais parâmetros de otimização o processamento e o tamanho de código, enquanto a energia consumida pelo código não é levada em consideração, que, em RSSF, é o recurso mais escasso. Poucos trabalhos têm sido propostos na literatura que abordam esse problema nos compiladores de RSSF [Lane and Campbell 2006, Zhang et al. 2009].

Outro problema a ser destacado é que o Sistema Operacional (SO) em RSSFs mais utilizado, TinyOS [Levis et al. 2004], possui um conjunto de funções que foram escritas para uso geral, ou seja, para qualquer que seja o contexto da RSSF, não permitindo assim, a otimização dessas funções de acordo com a aplicação. Dessa forma, o trabalho visa demonstrar que códigos gerados para RSSFs muitas vezes ainda podem ser otimizados, através da utilização de análises do código executável. Ao final, pretende-se apresentar um método capaz de oferecer possíveis otimizações a partir dessas análises. Tais análises de código são: análise das interrupções do código, análise do consumo de energia, e o Grafo de Fluxo de Controle (GFC) do código. Elas são análises implementadas, em parte, pela ferramenta Avrora [Titzer et al. 2005], e posteriormente estendidas por nós, de acordo com o que era necessário de se obtê-las.

O artigo está organizado da seguinte forma: na seção 2 são descritos os trabalhos relacionados. Na seção 3, é apresentado o método proposto e a ferramenta utilizada pelo método. Na seção 4, aplica-se o método nas aplicações Blink e Antithief como estudos de caso, e são discutidos os resultados. E, por fim, na seção 5, conclui-se e cita-se os trabalhos futuros.

2. Trabalhos Relacionados

Os trabalhos relacionados se limitaram aos simuladores de RSSF que oferecem funcionalidades de análises de código.Simulação em RSSFs tem sido muito estudada nesta década, já que o funcionamento incorreto da rede ou de estimações são irreversíveis. A maioria dos simuladores em RSSF [Sundresh et al. 2004, S. Park and Srivastava 2000, Levis et al. 2003] foram destinados mais a analisar a rede em si, abstraindo o *hardware* e o código executável do nó sensor, inviabilizando o uso deles em nossa análise.

Os simuladores Avrora [Titzer et al. 2005] e Ipen [Joe et al. 2008], contudo, simulam em nível de instrução. O Ipen foi desenvolvido com o único objetivo de analisar o consumo de energia, enquanto o Avrora é uma ferramenta que fornece simulação e um conjunto de análises para códigos escritos para nós sensores Mica2 e MicaZ. Ele é flexível, oferecendo uma infraestrutura que facilita o acréscimo de novas funcionalidades. A finalidade de tais análises é facilitar o entendimento do projetista quanto a execução de sua aplicação. Porém, entendemos que esse conjunto de análises do Avrora somadas a outras análises por nós implementadas podem ser mais úteis que uma simples observação, permitindo localizar otimizações na aplicação. Por isso, escolhemos o Avrora, estendendo em cima dele nossas análises, para a aplicação do método proposto.

3. Método Proposto e Ferramenta Estendida

É apresentado um método que, a partir de análises do código executável do nó sensor, oferecidos por uma ferramenta estendida do Avrora, possibilita que se encontre otimizações na aplicação. O método é composto por uma análise crítica e um guia de uso. A primeira é a descrição de que maneira cada análise oferecida pela ferramenta poderia gerar alguma otimização na aplicação, enquanto a segunda é útil para que o usuário final da ferramenta possa aplicar o método através de uma sequência de

passos na tentativa de se obter alguma otimização da aplicação. A efetuação da otimização detectada é feita manualmente pelo usuário. É importante ressaltar que as análises são dependentes do cenário da simulação, cabendo ao usuário especificar o cenário de acordo com a rede real, para que as otimizações sejam efetivas. A apresentação é, então, dividida para cada análise de código da ferramenta. As análises de código da ferramenta são: grafo de fluxo de controle, análise do consumo de energia e análise das interrupções. Para uma melhor compreensão das análises, pode-se observar as figuras e tabelas de cada uma, nos estudos de caso.

3.1. Grafo de Fluxo de Controle

Um GFC é uma representação do fluxo de execução do código, que permite uma observação detalhada da execução do código do nó sensor através dos dados inseridos nele. O GFC é uma funcionalidade do Avrora, que foi expandida.

Os dados inseridos nos nós do GFC são: nome do bloco (endereço no código), quantidade de ciclos do processador, alteração do estado de algum componente do nó sensor e quantidade de vezes que o caminho foi percorrido. Os nós se classificam em: entrada de alguma interrupção, entrada de procedimento, nó simples que termina com retorno simples ou retorno de uma interrupção, e nó simples.

Uma funcionalidade estendida no GFC do Avrora é a marcação dos trechos de código mais executados durante o tempo de execução do código. Essa funcionalidade consiste na marcação, que se faz pela colorização, dos nós que são mais executados a fim de destacá-los visualmente. A percentagem dos nós mais executados que se queira marcar é passado como parâmetro no uso dessa funcionalidade da ferramenta.

Análise Crítica do GFC

O GFC do Avrora permite observar como o programa é estruturado e o mapeamento do código em alto nível para o código *assembly* realizado pelo compilador. Com a extensão, pode-se observar os comportamentos dos componentes do nó sensor, e da execução do código ao longo do tempo, pode-se realizar cálculos do consumo de processamento entre os caminhos de execução do código. Além disso, através da classificação dos nós do grafo, observa-se o que ocorre no tratamento de cada interrupção da aplicação e das chamadas de funções. A seguir, apresenta-se detalhadamente cada análise crítica que se pode fazer do GFC, e seus respectivos guias de uso.

A. Eliminação de Código

O nesC automaticamente detecta os nós do grafo que não são alcançáveis e os retira do código compilado, eliminando assim *código morto*. Porém, outra forma de eliminação de código é a detecção de componentes que foram adicionados pelo TinyOS e não serão úteis na aplicação. Essa detecção ocorre através da observação dos nomes dos nós de entrada de funções e saltos, pois a entrada da execução de algum componente do TinyOS sempre é um nó de entrada de uma função. Sendo assim, verifica-se os componentes inclusos pelo TinyOS e retiram-se aqueles desnecessários pela aplicação. Logo, necessita-se de um conhecimento dos componentes do TinyOS, assim como do funcionamento da aplicação.

Guia de Uso: (i) detectar funções no GFC que são inicializações de algum

componente; (ii) observar a aplicação em alto nível e verificar se a mesma necessita desses componentes adicionados pelo TinyOS; (iii) caso algum componente não seja necessário, descartar a inclusão do mesmo na aplicação, alterando as bibliotecas do SO.

B. Otimização do Consumo de Processamento

O último ponto destacado na eliminação de código consequentemente reduzirá o consumo de processamento da aplicação, mesmo que esta redução seja mínima. No entanto, o ponto de maior destaque na redução nos ciclos de processador está na funcionalidade de destaque dos nós do grafo mais percorridos, pois, possibilita que otimizações no código sejam feitas apenas para os locais do código mais percorridos.

Guia de Uso: (i) gerar GFCs para diferentes percentagens dos trechos de código mais executados, com o objetivo de definir qual a melhor percentagem de código a ser otimizada, de acordo com alguma métrica estabelecida pelo usuário; (ii) selecionar um conjunto de técnicas de otimização de código, cabendo ao usuário definir quais as mais apropriadas para a aplicação em desenvolvimento; (iii) efetuar as técnicas nas funções que possuem trechos de código mais executados.

C. Gerenciamento dos Componentes do Nó Sensor

O GFC gerado permite observar o comportamento dos componentes ao longo da execução do código. Aprimora-se esse estudo com a informação do custo de ciclos de processamento em cada bloco, pois, permite o cálculo do limite inferior e superior de tempo que um componente será ativado ou desativado. Assim, espera-se que o estudo do comportamento dos componentes contribua para o desenvolvimento de gerenciadores de componentes mais eficientes [Paleologo et al. 1998, Klues et al. 2007].

Guia de Uso: (i) encontrar no GFC os nós que contém alterações dos estados dos componentes; (ii) para cada nó do GFC que altera o estado de um determinado componente, fazer cálculos para se encontrar a distância mínima e máxima em ciclos de processamento até outro nó do GFC que poderia alterar o estado do mesmo componente; (iii) contar quantas vezes cada estado do componente é alterado ao longo da execução; (iv) utilizar tais dados estatísticos como auxílio na construção de gerenciadores de componentes do nó sensor.

3.2. Análise do Consumo de Energia

A análise do consumo de energia observa o custo de energia de cada componente do nó sensor durante o tempo de execução da aplicação. E mais especificamente, para cada componente, ela fornece a quantidade de tempo gasto e a energia consumidade de cada estado que foi transitado durante a simulação. A funcionalidade foi implementada pelo próprio Avrora.

Análise Crítica do Consumo de Energia

A vantagem é prever o consumo de energia de cada componente do nó sensor e do tempo de permanência em cada um de seus estados, assim como o tempo de vida da rede também. O principal objetivo dessa análise é servir como uma métrica de qualidade do gerenciamento de energia da aplicação, e também para usá-la para

comparações em casos de otimização.

Guia de Uso: (i) verificar quais os estados de ativação alcançados para cada componente durante a simulação e o tempo em cada estado, oferecendo um auxílio ao *profiling* da aplicação. O *profiling* é uma modelagem do comportamento da aplicação através de uma análisa dinâmica da mesma, útil para que otimizações sejam aplicadas de acordo com esse comportamento.

3.3. Análise das Interrupções

Análise das interrupções observa as interrupções de *hardware*. Ela é bastante útil para relatar o comportamento da aplicação, principalmente em Sistemas Operacionais orientados a eventos como o TinyOS. A funcionalidade é disponibilizada pelo Avrora, sem a necessidade da extensão da mesma. Ela é uma análise dinâmica da aplicação, onde se obtém uma tabela ao final da simulação com todas as interrupções que ocorreram, o intervalo de tempo entre uma mesma interrupção e o período de latência de cada. A tabela 1 é um exemplo de algumas das interrupções que ocorreram na aplicação Antithief.

Tabela 1. Exemplo de Tabela de Interrupção

Num	Nome	Chamadas	Separação	Latência	Wakeup
10	TIMER2 COMP	0	-	-	-
11	TIMER2 OVF	0	-	-	-
12	TIMER1 CAPT	8178	14583	161	4
13	TIMER1 COMPA	4104	29223	12.4	4
14	TIMER1 COMPB	0	-	-	-
15	TIMER1 OVF	263	389074	4.456	4

Análise Crítica das Interrupções

A tabela de interrupções permite observar quais interrupções não ocorreram durante a simulação. Dessa forma, mesmo que a tabela de interrupções não seja essencial para saber quais interrupções não possuem tratamento, ela facilita a observação em comparação ao GFC, pois, este último pode ser muito grande. A verificação de quais tratamentos de interrupções podem ser ativados durante a execução do código é necessário, pois, alguns componentes são ativados apenas nesses tratamento de interrupções.

A informação entre os intervalos das interrupções são úteis apenas para observar se alguma interrupção ocorre muito frequentemente, podendo levar a degradação do sistema. O período de latência é importante para ser acrescentado nos cálculos entre tempo máximo e mínimo de intervalo entre a alteração de estado de algum componente, pois a latência soma-se com os valores dos ciclos do processamento no nó de cada entrada no tratamento de interrupção.

Guia de Uso: (i) verificar quais interrupções ocorreram durante a simulação e, através disso, ter indícios da execução dos componentes do nó sensor. Um exemplo seria verificar quantas interrupções do rádio ocorreram durante a execução; (ii) utilizar os intervalos das interrupções para auxiliar a construção do *profiling* da

aplicação, sabendo quais componentes serão muito acionados; (iii) verificar se os intervalos das interrupções são o mínimo suficiente para não degradar o sistema; (iv) utilizar os períodos de latência como auxílio em um cálculo mais preciso dos ciclos de processamento máximo e mínimo para alteração do estado de um componente do nó sensor.

4. Estudos de Caso

4.1. Estudos de Caso - Blink

Aplicamos o método proposto na aplicação Blink, do TinyOS 2.1, usada pra testar os temporizadores e os *Leds* do nó sensor. Por sua simplicidade facilitar a observação do método, optou-se por apresentar primeiramente este estudo de caso. Simulamos quatro horas de sua execução, pois, como a aplicação só usa os temporizadores, executando sempre as mesmas operações em um ciclo curto de tempo, quatro horas é suficiente para que as execuções de inicialização não exerçam influência no *profiling* da aplicação.

As otimizações aplicadas foram duas. A primeira foi a detecção do componente CC1000, que é o rádio, que não seria útil nessa aplicação, por isso o removemos manualmente no TinyOS, e simulamos novamente. A segunda otimização foi do processador. Verificou-se pela análise de energia que somente dois estados eram transitados no processador, e que no GFC, eles tinham a mesma quantidade de vezes transitadas. Dessa forma, verificamos que sempre que a função MCU-Sleep do TinyOS executava, ela transitava para o mesmo estado. Então, a otimização consistiu em retirar as condições do código para o cálculo de qual seria o estado a ser transitado.

Tabela 2. Aplicação do Blink antes e depois do método ser aplicado.

-	Energia - MCU	Redução	Tamanho do Código	Redução
Antes do Método	0,3583 Joules	-	2198 Bytes	-
Ponto 1 - CC1000	0,3581 Joules	0,005%	2052 Bytes	6,6%
Ponto 2 - McuSleep	0,3374 Joules	5,8%	2070 Bytes	5,8%
Aplicação de ambos	0,3370 Joules	5,8%	1924 Bytes	12,4%

Ao aplicá-las, obtivemos otimizações de consumo de energia e tamanho do código que são expressas na tabela 2. A segunda coluna mostra a quantidade de energia consumida em cada etapa do método, a terceira apresenta a percentagem reduzida em comparação a aplicação antes de qualquer otimização ser efetuada, a quarta coluna é o tamanho de código em cada etapa, enquanto a última coluna é a percentagem de redução do tamanho do código em comparação a aplicação antes das otimizações.

4.2. Estudos de Caso - AntiThief

O estudo de caso Blink foi útil como demonstração de uso da ferramenta. Este outro foi escolhido para que o método fosse realizado numa aplicação mais realística e típica de RSSF do TinyOS, denominada AntiThief, para a plataforma MicaZ. A

aplicação consiste de leituras periódicas do sensor de luz por cada sensor, e este mesmo verifica se a quantidade de luz alcançou o limite adotado e, caso não, alerta a existência de um ladrão ligando o *Led* Vermelho conjuntamente com um aviso de alerta ao nó *sorvedouro* através de uma comunicação com múltiplos saltos.

Simulou-se a aplicação na ferramenta Avrora estendida em três diferentes tempos: dois minutos, seis minutos e dez minutos para a plataforma MicaZ. A rede foi composta por três nós, sendo o nó 0, o nó sorvedouro. Os nós foram distribuídos em uma linha, sendo que cada nó só possui alcance aos nós mais próximo à esquerda e à direita. Escolheu-se o nó 1 para ser analisado pela ferramenta, pois, como ele é um nó intermediário, terá que receber e transmitir pacotes, executando obrigatoriamente os trechos de código que incluem tais funções. Os eventos, detecção de ladrão por parte de um nó sensor, foram programados para ocorrer sempre uma vez por minuto no nó 1, e uma vez a cada dois minutos no nó 2, em todos os três tempos de simulação. Os tempos escolhidos de simulação foram curtos, pois, uma simulação longa geraria muitas modificações dos estados dos componentes no GFC, dificultando a análise, visto que ela depende de figuras e é feita manualmente, incluindo os cálculos de ganhos em processamento e energia. Dessa forma, pode-se concluir que o cenário aqui apresentado limita a garantia dos resultados das otimizações somente para esse cenário, porém, não se anula que a abordagem proposta aqui pôde obter otimizações.

A. Grafo de Fluxo de Controle

Para cada um dos três tempos de simulação, três diferentes colorações foram aplicadas. Os percentuais de cinco, dez e vinte dos blocos mais executados do código foram coloridos de azul. Então, tem-se nove distintas simulações e GFCs, sendo que a diferença entre os que possuem a mesma quantidade de tempo de simulação é somente a coloração.

A tabela 3 mostra para cada tempo de simulação e para cada percentagem dos blocos mais executados (cinco, dez e vinte porcento), passada por parâmetro na funcionalidade do GFC, a percentagem do tempo de execução nesse percentual dos blocos mais executados sobre o tempo de execução total do código.

Tabela 3. Relação entre os blocos mais executados e tempo total de execução para o Antithief.

Tempo de Simulação(minutos)	5% dos Blocos	10% dos Blocos	20% dos Blocos
2 min	79,1%	89,5%	97,6%
6 min	79,2%	89,6%	97,6%
10 min	79,3%	89,6%	97,7%

A figura 1 mostra para o mesmo trecho do GFC, para duas colorações distintas, para o tempo de simulação de dez minutos, comprovando que há mais nós azuis quando é elevada a percentagem de coloração dos blocos mais executados. As três informações nos nós do grafo são: nome do bloco (endereço), número de ciclos de processamento do bloco e a quantidade de vezes que este foi executado, respectivamente.

Análise Crítica do GFC

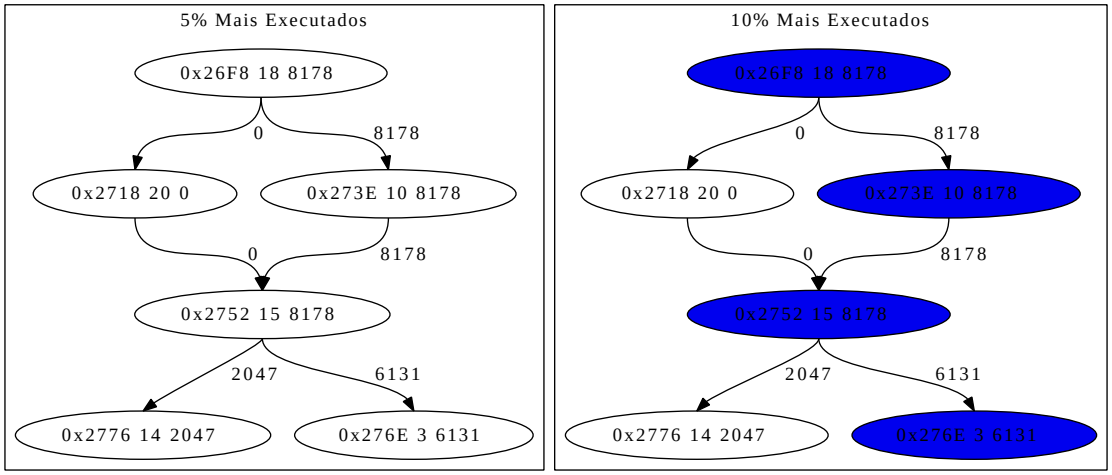

Figura 1. Grafos para um mesmo trecho de código da aplicação Antithief.

Assim como no caso do Blink, a etapa de análise crítica do GFC depende do conhecimento da estrutura do TinyOS, visto que as verificações de oportunidades de melhorias são aferidas de acordo com este conhecimento. Foi decidido aplicar a análise crítica somente em 10% dos nós mais executados do GFC, devido a análise crítica manual ser bastante trabalhosa, ao mesmo tempo em que o incremento da otimização para 20% difere em aproximadamente 8% a percentagem do tempo de execução da aplicação, segundo mostra a tabela 3. Também foi decidido somente o uso do tempo de execução de dez minutos, pois, apesar de os eventos ocorrerem sempre numa frequência constante nos nós 1 e 2, um maior tempo de simulação distribui de forma mais justa os tempos de execução pelos nós do GFC, visto os trechos do código de inicialização do nó sensor e da rede terão menor peso sobre o tempo total de execução.

Otimização do Consumo de Processamento

Nesse estudo, essa otimização teve por base as técnicas de otimização de desdobramento de *loops* e diretivas em linha. Tais técnicas diminuem o consumo de processamento, ao mesmo tempo em que incrementam o tamanho do código. Elas foram aplicadas nas funções que possuem blocos que estão inseridos nos 10% dos blocos de código mais executados durante a simulação. As otimizações foram realizadas em funções, porque não se poderia aplicar em apenas alguns blocos isolados das funções.

A tabela 4 apresenta algumas das funções do TinyOS no código executável que fazem parte dos blocos de código mais executados, e quais as otimizações aplicadas nelas (desdobramento de *loop*, diretivas em linha ou nenhuma). A terceira coluna apresenta o percentual da execução somente da função sobre a execução total de toda a aplicação, e a última coluna observa qual a redução de processamento na aplicação. A redução total foi de 6,28% do consumo de processamento efetuado pelas otimizações.

Tabela 4. Tabela das funções que contém blocos que estão entre os 10% dos blocos mais executados da aplicação Antithief.

Função	Otimizações	Fun./Apli.	Redução na Apli.
udivmodhi4	Loop/Em linha	20,60%	2,11%
udivmodsi4	Loop/Em linha	4,42%	0,77%
Atm128SpiP$SpiByte$write	Loop/Em linha	18,1%	2,28%
Atm128SpiP$sendNextPart	Em linha	20,6%	0,85%
Main	-	5,69%	-
SchedulerBasicP$TaskBasic$postTask	-	2,93%	-

B. Análise do Consumo de Energia

Através da quantidade total de ciclos reduzidos depois que todas as otimizações foram aplicadas no Antithief, conforme mostrado na tabela 5, calculou-se manualmente a energia consumida pelo processador após a otimização, retirando os ciclos do período ativo do processador e colocando-os no modo *idle*.

Tabela 5. Antithief - Reduções após efetuação das otimizações.

-	Redução após Otimizações
Estado Ativo da CPU	4102095 Ciclos (6,28%)
Energia CPU	0,00704949 Joules (1,14%)

4.3. Discussão dos Resultados

Embora a simulação do Antithief tenha ocorrido com apenas três nós integrando a rede, as funcionalidades de coleta e entrega, caracterizam a aplicação como típica de RSSF. Por ter um código bem maior que o Blink, o Antithief produziu um grafo muito mais extenso e complexo, de difícil detecção visual de otimizações, ao contrário do estudo de caso do Blink. Dessa forma, as otimizações se basearam nas técnicas de otimização de código utilizadas pelos compiladores, e foram aplicadas somente nos 10% dos blocos mais executados do grafo.

Observamos na tabela 5, que a redução do consumo de energia do processador foi somente de 1,14%, embora a redução do tempo de permanência no estado ativo ter sido de 6,28%. Isso ocorreu devido a aplicação ter sempre o rádio sempre em algum nível de atividade, impedindo assim, que o processador transite para um estado de menor ativação, pois, a possibilidade de uma interrupção do rádio necessita de um estado mínimo de ativação do processador. Em uma aplicação com coleta e entrega periódicas, onde o rádio é desligado regularmente, a energia do processador seria reduzida em maior escala.

5. Conclusão e Trabalhos Futuros

O desenvolvimento de *software* para RSSFs considera economia de recursos, seja pelas bibliotecas ou SO usado, ou seja, pelo compilador. Contudo, mostramos um método que após o código executável ser gerado, novas considerações da eficiência

da aplicação podem ser observadas e otimizações aplicadas através de análises do código, no TinyOS.

Aplicamos o método em dois estudos de casos, Blink e Antithief. Propõe-se como trabalhos futuros: a automatização do método, aplicar o método em um estudo de caso mais complexo, um estudo aprofudando das técnicas de otimização de código que diminuem o consumo de processamento enquanto aumentam o tamanho do código, e por fim, o uso do método para comparação dos SOs TinyOS e Contik.

Referências

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

Joe, H., Park, J., Lim, C., Woo, D., and Kim, H. (2008). Instruction-level power estimator for sensor networks. *ETRI Journal*, pages 47–58.

Klues, K., Handziski, V., Lu, C., Wolisz, A., Culler, D., Gay, D., and Levis, P. (2007). Integrating concurrency control and energy management in device drivers. In *SOSP '07: Proceedings of 21th ACM SIGOPS Symposium on Operating Systems Principles*, pages 251–264.

Lane, N. and Campbell, A. (2006). The influence of microprocessor instructions on the energy consumption of wireless sensor networks. In *EmNets 2006: The 3rd Workshop on Embedded Networked Sensors*.

Levis, P., Lee, N., Welsh, M., and Culler, D. (2003). Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 126–137.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., and Culler, D. (2004). Tinyos: An operating system for sensor networks. In *Ambient Intelligence 2005*.

Paleologo, G. A., Benini, L., Bogliolo, A., and De Micheli, G. (1998). Policy optimization for dynamic power management. In *DAC '98: Proceedings of the 35th annual Design Automation Conference*, pages 182–187.

S. Park, A. S. and Srivastava, M. B. (2000). Sensorsim: a simulation framework for sensor networks. In *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 104–111.

Sundresh, S., Kim, W., and Agha, G. (2004). Sens: A sensor, environment and network simulator. In *Proceedings of the 37th Annual Simulation Symposium*, pages 221–230.

Titzer, B., Lee, D. K., and Palsberg, J. (2005). Avrora: scalable sensor network simulation with precise timing. In *IPSN'05: The Fourth International Symposium on Information Processing in Sensor Networks*, pages 477–482.

Zhang, Z., Chan, W. K., Tse, T. H., Lu, H., and Mei, L. (2009). Resource prioritization of code optimization techniques for program synthesis of wireless sensor network applications. *Journal of Systems and Software*, 82(9):1376–1387.