Avaliação do desempenho de redes LLNs baseadas nas recomendações 6LoWPAN e RPL

Bruno Silvestre[1], **José Gonçalves Pereira Filho**[2], **Silvana Rossetto**[3], **Vinicius Barcellos**[2]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)

[2]Depto. de Informática – Universidade Federal do Espírito Santo (UFES)

[3]Depto. de Ciência da Computação – Universidade Federal do Rio de Janeiro (UFRJ)

Resumo. *Um trabalho importante por parte do IETF vem sendo realizado com o objetivo de introduzir o protocolo IPv6 nas LLNs (Low-power and Lossy Networks). Um dos objetivos desse esforço é facilitar o uso compartilhado, por diferentes aplicações, de uma variedade de dispositivos inteligentes de sensoreamento (como é o caso das aplicações voltadas para o contexto de "cidades inteligentes"). Duas das recomendações do IETF para essa finalidade se destacam: a camada de adaptação 6LoWPAN (RFC6282) e o protocolo de roteamento RPL (RFC6550). Neste artigo, avaliamos o desempenho geral de uma LLN — em termos de eficiência, consumo de memória e tolerância a falhas — usando como referência duas implementações open source das recomendações IETF no contexto de redes de sensores sem fio.*

Abstract. *Considerable work has been done from IETF with aim to introduce IPv6 stack implementation in LLNs (Low-power and Lossy Networks). One of the goals of this effort is to facilitate the shared use of a variety of smart sensors by different applications, as is the case of smart city applications. Two of the recommendations of the IETF for this purpose are: the 6LoWPAN adaptation layer (RFC6282) and the RPL routing protocol (RFC6550). In this article, we evaluate the overall performance of a LLN — in terms of efficiency, memory consumption and fault tolerance — using as reference two open source implementations of IETF recommendations in the context of wireless sensor networks.*

1. Introdução

O impulso dado nos últimos anos para o desenvolvimento e implantação de sistemas inteligentes de monitoramento e atuação remota, incluindo por exemplo, aplicações *smart grid* e *smart building*, criou demandas urgentes para a implantação de redes IP formadas por centenas ou milhares de dispositivos de sensoreamento de baixo custo e baixo consumo. O passo seguinte (e já emergente) é a integração e ampliação dessas redes para possibilitar o desenvolvimento de aplicações dentro do conceito de "cidades inteligentes" (*smart city*), criando novas possibilidades de serviços e facilitando a administração das grandes cidades.

Na última década, a pesquisa na área de Redes de Sensores sem Fio (RSSFs) tem contribuído significantemente para o desenvolvimento de aplicações de sensoreamento em larga escala, mas a ausência de uma infraestrutura de comunicação IP dificulta a interoperabilidade dessas redes com a Internet e, por consequência, limita a possibilidade

de integração entre diferentes aplicações. Em função das suas características e restrições particulares (em termos de capacidade de processamento, armazenamento e fonte de energia), diferentes protocolos de comunicação foram projetados para as RSSFs, normalmente acoplados diretamente ao código das aplicações. As primeiras experiências de implementação da pilha IP em dispositivos de baixo consumo, entretanto, demonstraram a viabilidade dessa iniciativa e mostraram que a implementação de uma arquitetura geral de rede para esses dispositivos tem custo similar às implementações de sistemas de comunicação que não aderem a nenhuma arquitetura ou padrão conhecido [Dunkels 2003, Hui and Culler 2008].

Para atender a essa necessidade de padronização, o IETF (*Internet Engineering Task Force*) especificou recentemente protocolos e camadas de adaptação que permitem executar o protocolo IPv6 sobre redes IEEE 802.15.4 (protocolo da camada de enlace). O grupo de trabalho 6LoWPAN (acrônimo para *IPv6 over Low power Wireless Personal Area Networks*) sugeriu a inclusão de uma camada de adaptação na pilha IP, entre a camada de rede e a camada de enlace, para permitir a fragmentação e desfragmentação de pacotes IPv6 em quadros IEEE 802.15.4, e efetuar a compressão do cabeçalho IPv6 [Kushalnagar et al. 2007, Hui and Thubert 2011]. Outro grupo de trabalho, denominado RoLL (acrônimo para *Routing Over Low-power and Lossy networks*), especificou o protocolo RPL (*IPv6 Routing Protocol for Low-power and lossy networks*) [Winter and et.al. 2012] como o protocolo de roteamento padrão para pacotes IPv6 sobre redes LLNs.

Os benefícios do esforço de implementação da pilha de protocolos IP nos dispositivos que formam as LLNs incluem: (i) facilidade de interoperabilidade e uso compartilhado da infraestrutura de comunicação entre aplicações em uma mesma LLN; e (ii) possibilidade de integração dessas aplicações com outras aplicações baseadas no protocolo IP, permitindo maior flexibilidade tanto na disseminação dos dados sensoreados quanto no monitoramento da operação da própria rede. Entretanto, de nada valem esses benefícios se o desempenho obtido ou o custo associado com a sua implementação for inadequado para as especificidades das aplicações que executam nessas redes.

Neste artigo avaliamos o desempenho geral de uma LLN — em termos de eficiência, consumo de memória e tolerância a falhas — usando como referência duas implementações *open source* das recomendações IETF (6LoWPAN e RPL) no contexto de RSSFs (TinyOS e Contiki).

O restante deste texto está organizado da seguinte forma. Na próxima Seção apresentamos as principais recomendações dos grupos de trabalho 6LoWPAN e RoLL para compressão de cabeçalho IPv6 e roteamento em LLNs. Na Seção 3, avaliamos o desempenho e custo computacional dessas recomendações usando como referência as implementações disponibilizadas pelo TinyOS e Contiki. Na Seção 4 destacamos outros trabalhos de avaliação das recomendações 6LoWPAN e RPL. Por fim, na Seção 5, apresentamos as conclusões e comentários finais sobre o trabalho realizado.

2. 6LoWPAN e RPL

Para integrar as LLNs com a pilha de protocolos IP é necessário reduzir a complexidade do cabeçalho IP (considerando as limitações de memória, capacidade de processamento, e fonte de energia das LLNs), e lidar apropriadamente com as características particulares das interconexões de comunicação dessas redes. Nesta seção, apresentamos as princi-

pais recomendações dos grupos de trabalho 6LoWPAN e RoLL da IETF com relação à compressão de cabeçalho IPv6 e ao algoritmo de roteamento proposto para LLNs.

2.1. Compressão de cabeçalho 6LoWPAN

O protocolo IPv6 define um MTU (*Minimum Transmission Unit*) de 1280 bytes. Dessa forma, toda a rede IPv6 deve ser capaz de transferir pacotes de 1280 bytes (cabeçalho e dados) sem a necessidade de fragmentação. A primeira proposta de compressão do cabeçalho IPv6 foi descrita na RFC4944 [Montenegro et al. 2007] em 2007. Essa proposta inicial foi melhorada pela RFC6282 [Hui and Thubert 2011] em 2011. Algumas implementações atuais, incluindo a implementação `blip/TinyOS` que avaliaremos nas seções seguintes, segue o mecanismo de compressão descrito em uma versão intermediária do padrão, descrita no documento "draft-ietf-6lowpan-hc-06" [Hui and Thubert 2009] de 2009, o qual já incluiu melhorias ao método de compressão original.

O cabeçalho IPv6 comprimido é chamado LOWPAN_IPHC e assume que os seguintes campos do cabeçalho IPv6 serão de uso comum nas comunicações em redes 6LoWPAN: *Version* é sempre 6; *Traffic Class* e *Flow Label* são ambos iguais a 0; *Payload Length* pode ser inferido das camadas inferiores (cabeçalho de fragmentação 6LoWPAN ou cabeçalho IEEE 802.15.4); *Hop Limit* pode receber um valor padrão; e os endereços designados para interfaces 6LoWPAN podem ser formados usando o prefixo de enlace local ou um pequeno conjunto de prefixos de roteamento designados para 6LoWPAN.

2.2. Algoritmo de roteamento RPL

As LLNs são definidas como redes nas quais os roteadores tipicamente operam com severas restrições em termos de capacidade de processamento, armazenamento e fonte de energia. As interconexões de comunicação, por sua vez, são caracterizadas por elevadas taxas de perdas de pacotes, baixas taxas de transmissão e alta instabilidade. Para lidar com esses desafios particulares e minimizar o *overhead* de controle, a IETF propôs um protocolo de roteamento IPv6 específico, chamado RPL [Winter and et.al. 2012].

O RPL é um protocolo do tipo *distance vector* que suporta três categorias de padrões de tráfego. Na primeira, *multipoint-to-point*, os nós periodicamente enviam mensagens para um ponto de coleta específico, por exemplo um nó *sink*. Na segunda, *point-to-multipoint*, o tráfego originado em um nó *sink* tem como destino dispositivos específicos dentro da LLN. Por último, a comunicação *point-to-point* também é suportada.

O RPL introduz uma série de conceitos que o torna bastante flexível, embora relativamente complexo. A formação da topologia da rede é baseada no conceito de Grafos Acíclicos Dirigidos (*Directed Acyclic Graph – DAG*) com uma estrutura em forma de árvore que define rotas *default* entre nós da LLN. Para aumentar a confiabilidade em ambientes LLN, o RPL adota o conceito de "diversidade espacial". Assim, diferentemente de uma árvore tradicional, onde um nó está associado a um único *nó pai*, em um grafo RPL um nó pode estar associado a vários *pais*, criando caminhos alternativos em direção a um destino.

O protocolo organiza os nós como um conjunto de *Destination Oriented DAGs (DODAGs)*. Nos DODAGs, os nós mais populares, como o nó *sink* e os nós *gateways*, são os nós raízes dos DAGs. Uma instância do protocolo RPL (*RPL Instance*), identificada univocamente por um *RPLInstanceID*, pode conter múltiplos DODAGs, cada um deles

identificado por um único DODAGID. Cabe observar que uma rede LLN pode ter várias instâncias RPL executando concorrentemente.

O RPL suporta aplicações com diferentes requisitos por meio da definição de Funções Objetivo (*Objective Functions - OFs*). Basicamente, uma *OF* especifica como o RPL deve selecionar seus pais e possíveis sucessores, ou seja, como selecionar caminhos no DODAG. As *OFs* definem de que maneira métricas de roteamento e funções relacionadas são empregadas pelos nós para computarem o seu *Rank* dentro de um DODAG. Em última instância, as OFs restringem ou otimizam as rotas selecionadas. Para garantir interoperabilidade de comunicação entre diferentes aplicações, o RPL provê uma função objetivo básica denominada **OF0** [Thubert 2011], a qual seleciona rotas com base no número de saltos até o nó raiz do DODAG. Observa-se ainda que é possível definir *OFs* por aplicação (ex: prover rotas sensíveis à latência para aplicações de tempo real).

Para construir e manter o DODAG, o protocolo RPL define mensagens ICMPv6 denominadas *DODAG Information Object - DIO* para a descoberta de vizinhos e o estabelecimento de rotas. Na formação da topologia, cada nó raiz constrói um pacote *DIO* e o envia para todos os filhos. Qualquer filho que decide se juntar ao DAG repassa o *DIO* adiante, para os seus próprios filhos. O *DIO* contém um valor do *Rank* do nó, que é incrementado quando o filho se junta ao DAG. Além disso, os nós podem armazenar um conjunto de pais e irmãos candidatos, que podem ser usados se o pai preferencial está incapacitado de rotear tráfego. O *DIO* também é usado para indicar a *OF*. Por fim, a propagação de rotas é implementada com o algoritmo Trickle [Levis et al. 2011] (escalona o envio de mensagens *DIO* visando sempre minimizar a quantidade de DIOs transmitidos e garantir um tempo de convergência baixo para a rede).

Como redes LLNs são bastante dinâmicas, o RPL também fornece facilidades para incorporar métricas de natureza dinâmica, tais como *ETX - Estimated number of Transmissions* [Couto et al. 2005] (facilita a descoberta de caminhos de maior vazão e minimiza o número total de transmissões de um pacote até o seu destino). Além disso, se um nó detecta a inexistência de uma rota em direção à raiz, um mecanismo de "reparo local" é acionado para encontrar uma rota alternativa. Quando uma sequência de reparos locais levarem a uma topologia da árvore ineficiente, um mecanismo de "reparo global" pode ser usado [Korte et al. 2012].

3. Avaliação do 6LoWPAN e RPL usando as implementações do TinyOS e Contiki

Diferentemente das redes sem fio tradicionais, os dispositivos que formam as redes LLN requerem sistemas operacionais *lightweight*, otimizados para uso em ambientes com severas restrições de processamento, armazenamento e energia. Os sistemas operacionais TinyOS [Hill et al. 2000] e Contiki [Dunkels et al. 2004] ocupam posições de destaque nesse cenário. Ambos são adotados no desenvolvimento de aplicações para RSSFs e são sistemas operacionais de código aberto, que podem ser embarcados em diversos tipos de dispositivos computacionais e objetos inteligentes.

Nesta seção, usamos as implementações mais recentes das técnicas de compressão 6LoWPAN e do protocolo de roteamento RPL disponibilizadas por esses sistemas operacionais (`blip2/TinyRPL` e `uIPv6/ContikiRPL`) para avaliar o custo computacional e desempenho alcançado em uma LLN.

3.1. Principais aspectos da implementação 6LoWPAN/RPL no TinyOS

A implementação 6LoWPAN disponibilizada atualmente com o TinyOS é chamada `blip` (versão 2.0) [TinyOS-Blip 2012]. Ela implementa compressão do cabeçalho (seguindo as definições apresentadas em [Hui and Thubert 2009]), e os protocolos DHCPv6 para atribuição de endereços e PPP para comunicação com redes externas. TinyRPL é a implementação TinyOS do protocolo de roteamento RPL. A implementação corrente do TinyRPL é baseada na revisão 17 do draft RPL [Winter and et.al. 2010] e acompanha a implementação `blip2.0` do TinyOS 2.1.2. O TinyRPL adota os seguintes valores de configuração: (i) para o tamanho máximo das filas nos roteadores: 3 pacotes; (ii) para o intervalo de retransmissão de pacotes na camada de enlace: 103 ms;

3.2. Principais aspectos da implementação 6LoWPAN/RPL no Contiki

O Contiki disponibiliza uma implementação da pilha 6LoWPAN chamada `SICSLowPan` que segue as especificações RFC4944 [Montenegro et al. 2007], *draft-hui-6lowpan-interop-00* [Hui et al. 2007] e *draft-hui-6lowpan-hc-01* [Hui 2008]. `ContikiRPL` é a implementação do RPL disponível no Contiki e é baseada na RFC6550 [Winter and et.al. 2012]. A implementação atual adota os seguintes valores como padrões: (i) para o tamanho máximo das filas nos roteadores: 4 pacotes; (ii) para o intervalo de retransmissão de pacotes na camada de enlace: 128 ms;

3.3. Ambiente de experimentação

Para avaliar as implementações TinyOS e Contiki, usamos o ambiente de simulação do Contiki em todos os experimentos. O ambiente é definido em duas partes: (i) simulador de rede COOJA [Contiki-COOJA 2012]; e (ii) emulador de motes (nós da rede) MSPSim, o qual combina a emulação dos ciclos de execução de motes baseados no microcontrolador MSP430 (ex., TmoteSky e TelosB) com a emulação do transceiver de rádio CC2420.

A mesma topologia foi empregada em todas as simulações (Figura 1). Foram utilizados um nó *sink* e 40 nós que assumem papéis de roteador e/ou emissor de informação (de acordo com os cenários de avaliação descritos abaixo). A área simulada é de 240x240 metros, com o alcance de rádio para comunicação de 50m, e mais 50m de interferência (gerando as duas áreas em volta do nó i mostradas na Figura 1). Todas as mensagens trocadas entre os nós são implementadas na camada de aplicação e usam o protocolo de transporte UDP. Além disso, o simulador foi configurado com 0% de taxa de perda de pacotes por ruído do ambiente. Dessa forma, podemos assumir que qualquer perda de pacote acontece pelo fato do nó não estar pronto para o recebimento ou estouro de buffer (questões que estão diretamente relacionadas ao comportamento dos nós e aos protocolos das camadas de enlace e rede que queremos avaliar). É importante destacar que os parâmetros de configuração (como *timeout* de retransmissão, tamanho de fila na interface de rede do nó, etc.) de cada sistema operacional foram mantidos com seus valores *default*.

3.4. Perda de pacotes

O objetivo deste experimento é verificar a taxa de perda de pacotes com a variação do número de mensagens trafegando na rede. Para isso, variamos o número de nós emissores e o intervalo de tempo entre a transmissão das mensagens em cada nó. Todas as mensagens são destinadas ao nó *sink* que apenas recebe e contabiliza as mensagens. Neste experimento, todas as mensagens possuem um *payload* de 12 bytes.

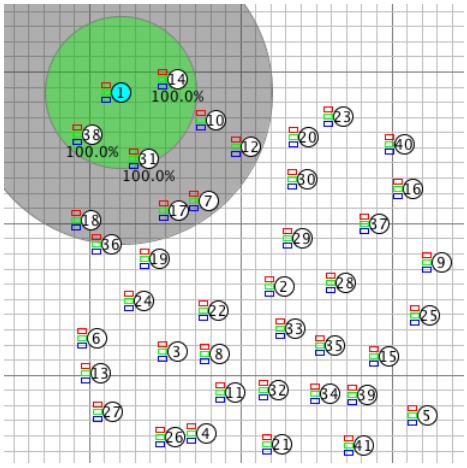

Figura 1. Topologia da rede usada nos experimentos.

Quatro cenários foram experimentados com relação ao percentual de nós da rede que enviam pacotes: **25%** dos nós são emissores/roteadores, 75% são apenas roteadores; **50%** dos nós são emissores/roteadores, 50% são apenas roteadores; **75%** dos nós são emissores/roteadores; 25% são apenas roteadores; e **100%** dos nós são emissores/roteadores. Além disso, variamos o intervalo com que os emissores enviam as mensagens em 2, 4 e 8 segundos de forma a emular as características de diferentes aplicações. Não há, entretanto, combinação de intervalos de tempo distintos dentro de um mesmo experimento.

A escolha dos nós emissores e roteadores para cada um dos 12 casos de teste (4 cenários com 3 frequências de emissão) foi feita de forma aleatória. Realizamos 5 simulações de 1 hora para cada caso. As porcentagens de perda de pacote, com o respectivo intervalos de confiança, são mostrados na Figura 2.

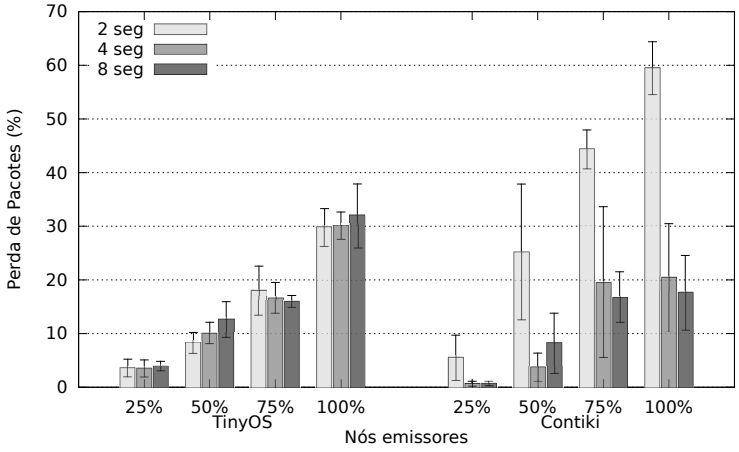

Figura 2. Perda de pacotes no TinyOS e Contiki.

O TinyOS apresentou pouca variação na perda de pacotes em relação ao intervalo de envio das mensagens. Por outro lado, o Contiki apresentou perdas muito maiores quando o intervalo entre as mensagens foi de 2 segundos. Isso mostra que o Contiki foi muito mais sensível a um número maior de pacotes trafegando pela rede. Nos demais

intervalos, o Contiki se mostrou, em média, melhor (25%, 50% e 100%) ou comparável (75%) com o TinyOS, mas o intervalo de confiança mostra uma instabilidade maior do Contiki.

3.5. *Round-Trip Time* (RTT)

Neste segundo experimento, medimos o tempo de comunicação entre pares de nós da rede (considerando que o segundo nó é sempre o nó *sink*). Escolhemos 3 nós emissores de acordo com as suas distâncias, em saltos, até o nó *sink*. Nosso objetivo nesse caso foi analisar o impacto mínimo do roteamento no tempo de comunicação de acordo com o número de saltos.

Os nós escolhidos para o experimento foram o 19, 11 e 5 (vide Figura 1). O primeiro se encontra a 3 saltos do *sink*, o segundo a 6 saltos e o último a 11 saltos. Realizamos 5 simulações para cada nó (1 hora para cada simulação). O nó envia um pacote de 1 byte para o *sink*, que responde com o mesmo pacote. Após receber a resposta, o nó repete o processo. Isso se dá durante toda a simulação. Medimos então o tempo gasto desde o envio até o recebimento de cada mensagem.

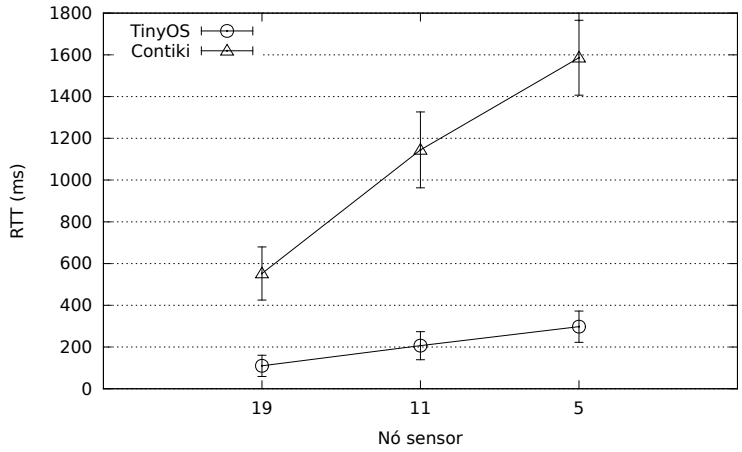

Figura 3. RTT no TinyOS e Contiki.

A Figura 3 mostra os valores obtidos nas 5 simulações, para o TinyOS e o Contiki. Observamos que a medida que o nó emissor se afasta do nó receptor, o tempo de ida e volta do pacote (RTT) aumenta pois somam-se os atrasos de enfileiramento e reenvio de pacotes em cada nó ao longo do caminho. Esse incremento do RTT também mostrou uma tendência linear com o aumento de saltos. Além disso, nos nossos testes, o Contiki se mostrou em média 5 vezes mais lento que o TinyOS.

3.6. Entrega de pacotes

Neste experimento avaliamos a capacidade de entrega de pacotes considerando como referência aplicações baseadas em alarme, quando um determinado evento ocorre e um nó deve manter o *sink* atualizado o mais rápido possível.

Realizamos uma alteração no código fonte do TinyOS para a realização deste cenário. Por padrão, não temos um retorno das camadas internas do `blip2.0` sobre o sucesso do envio de uma mensagem. A alteração foi necessária para incluir um evento

sendDone de pacotes UDP para a aplicação, que é disparado pelas camadas inferiores da pilha `blip2.0`[1]. Usamos esse evento para disparar o envio da próxima mensagem, criando um fluxo contínuo de envios para o *sink*. O Contiki também não possui nenhuma forma de informar à aplicação sobre o resultado do envio. Entretanto, o código do Contiki é mais complexo e as modificações necessárias pareciam ser mais intrusivas do que as realizadas no TinyOS. Por isso, esse experimento foi realizado apenas no TinyOS.

Da mesma forma que o experimento anterior, utilizamos os nós 19, 11 e 5 para verificar o comportamento do roteamento de acordo com o tamanho da rota. Além disso, variamos o tamanho dos pacotes em 1, 40 e 80 bytes. Como o tamanho útil de um pacote UDP em redes 6LoWPAN é cerca de 80 bytes, os valores escolhidos representam um pacote de tamanho mínimo, mediano e máximo sem a ocorrência de fragmentação.

Para a realização do experimento envolvendo o nó 5, tivemos que aumentar o tamanho da tabela de rotas dos nós de 20 para 40, pois o nó 1 recebia o pacote mas ocorria um erro no reenvio porque os nós não possuiam um caminho de volta para o nó 5. Executamos 5 simulações para cada um dos 9 cenários (3 nós com 3 tamanhos de pacote), sendo cada simulação de 1 hora. A Figura 4 apresenta os dados coletados.

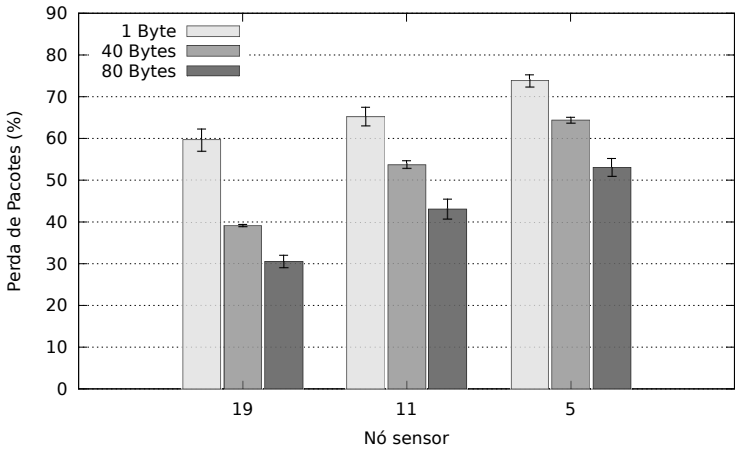

Figura 4. Entrega de pacotes em aplicação alarme no TinyOS.

Podemos observar que os pacotes menores possuem maior perda. Isso sugere que é mais vantajoso acumular uma certa quantidade de dados e enviar um pacote maior para rede. No entanto, vale lembrar que nosso cenário de experimentação não leva em conta interferências (por ruídos) no ambiente e que pacotes maiores têm maior probabilidade de sofrer tal interferência.

3.7. Recomposição de rotas

Como último experimento, testamos a capacidade do RPL de recompor rotas após a falha ou desativação temporária de nós. Na topologia que adotamos, o nós tendem a selecionar as mesmas rotas na maioria das execuções de simulação, mesmo quando alguns deles são desativados no andamento da simulação. Consideramos então 5 simulações que utilizaram as mesmas rotas. Também aumentamos o TTL das mensagens de 16 (padrão) para 64, devido ao tamanho das rotas.

[1] Alteração disponível em http://www.inf.ufg.br/~brunoos/tinyos/UDPSendDone.patch

Para este teste nós utilizamos apenas o nó 5 como emissor de mensagens (de 1 byte) a cada 5 segundos, e os demais 39 nós como roteadores. Cada vez que o nó *sink* recebe uma mensagem, desligamos o rádio de um dos nós da rota, forçando o algoritmo de roteamento a recompor a rota. Em alguns casos, desligamos um vizinho do nó da rota como forma de evitar a mudança de rota simplesmente para esse vizinho, obrigando o RPL a fazer uma reconfiguração maior. Medimos então o intervalo de tempo entre duas recepções de mensagens do *sink*. Esses intervalos indicam por quanto tempo o nó 5 ficou sem conexão com o *sink*.

	Rota	Nós Desativados	Tempo	σ
1	5-39-35-28-29-30-12-10-14-1	39	9,791	0,053
2	5-15-35-28-29-30-12-10-14-1	15	42,986	5,666
3	5-41-34-33- 2-22-24-36-17-31-1	33 e 35	11,683	4,979
4	5-41-34-21-11- 3-24-36-17-31-1	3 e 8	48,904	26,380
5	5-41-34-21-11-26-27- 6-24-36-17-31	36	19,589	24,478
6	5-41-34-21-11-26-27- 6-24-19-17-31-1	31	437,500	439,503
7	5-41-34-32-11-26-27- 6-24-19-17- 7-12-10-14-1	7 e 17	753,953	214,040
8	5-41-34-32-11-26-27- 6-24-22- 2-29-30-12-10-14-1	29 e 30	?	?

Tabela 1. Tempo para o TinyOS recompor rotas.

Na Tabela 1 temos as rotas percorridas pelo pacote do emissor até o *sink*, os nós que foram desativados, o tempo médio e o desvio padrão (em segundos) gasto pelo TinyOS para reconstruir a rota. Nos casos das rotas 1 e 5 temos reconfigurações bem pontuais. Na primeira o nó 39 foi substituído pelo vizinho 15 e na segunda o nó 19 substituiu o nó 36. Mas, comparando as duas situações tivemos uma variação do dobro de tempo, e se considerarmos a média somada ao desvio padrão, podemos notar uma variação de até 4 vezes. Ao desligar os nós 3 e 8 (rota 4) e o nó 31 (rota 6), o reestabelecimento de rota foi efetuado contornando esses nós desligados, i.e., aumentando o caminho com novos nós. Podemos notar que a diferença de tempo entre os dois casos chega a ser de 15 vezes.

No caso da rota 8, não houve reconstrução durante o tempo de simulação (lembrando que a simulação foi configurada para 1 hora), entretando foi possível verificar que havia ainda uma rota: 2-28-37-16-40-23-20-12-10-14-1. Se somarmos a coluna *Tempo* e ainda considerarmos os 12 segundos que foi dado para a estabilização da rede antes que o nó 5 enviasse a primeira mensagem, os nós 29 e 30 foram desligados por volta de 30 minutos de simulação, em média. Isso significa que mesmo depois de 30 minutos o TinyOS não conseguiu refazer a rota para o nó 5.

3.8. Consumo de memória

Para estimar o consumo de memória da pilha IPv6, medimos o consumo total de memória ROM e RAM das aplicações de teste. Como as aplicações implementadas apenas enviavam/recebiam mensagens via UDP, consideramos que o consumo obtido é o mínimo necessário para implementar qualquer aplicação usando a pilha de protocolos IPv6. O consumo de memória médio no TinyOS foi: (i) ROM: 30 kB; (ii) RAM: de 7 a 8 kB. Esses valores foram obtidos durante a compilação do código para a plataforma `TelosB`. No Contiki usamos a ferramenta `msp430-size` para obter o consumo de memória das aplicações compiladas para a plataforma `Sky`. Os resultados obtidos foram: (i) ROM: de 40 a 42 kB; (ii) RAM: de 8 a 9.2 kB.

Podemos observar que o TinyOS oferece atualmente uma implementação da pilha IPv6 com menor requisito de memória RAM e ROM, o que é bastante relevante em se tratando de plataformas de nós sensores. Nos dois casos é possível executar aplicações básicas sobre a pilha IPv6 em plataformas como `TelosB` e `Sky` (ambas baseadas no microcontrolador MSP430, com 10kB de RAM e 48kB de memória de programa). Entretanto, para outras plataformas de uso comum na pesquisa com RSSF, como é o caso da plataforma `MicaZ` (microcontrolador Atmega128L, com 4kB de RAM e 128kB de memória de programa), as implementações atuais da pilha IPv6 não são executáveis.

4. Trabalhos relacionados

Outros trabalhos apresentam resultados de investigações com propósitos similares de avaliar a viabilidade e desempenho das implementações 6LoWPAN e RPL em RSSFs. Em [Cody-Kenny et al. 2009], os autores avaliaram o desempenho dos protocolos 6LoWPAN especificamente nas plataformas `MicaZ` e `TelosB`, considerando a comunicação ponto-a-ponto de um computador para um nó da RSSF. A implementação *b6loWPAN* do TinyOS foi usada e precisou ser adaptada para nós MicaZ em razão da limitação de memória RAM dessa plataforma, como discutido na Seção 3.8. Os autores destacam o impacto no desempenho da implementação em função da adaptação requerida. [Ko et al. 2011a] compararam o desempenho da implementação `blib/TinyRPL` com o protocolo de roteamento básico do TinyOS, chamado `CTP`. Os autores avaliaram especificamente o *overhead* de operação do protocolo RPL e seu desempenho em comparação com o CTP. Em [Ko et al. 2011c] e [Ko et al. 2011b] os autores avaliaram o desempenho do ContikiRPL e do TinyRPL com o objetivo de levantar questões relacionadas com a interoperabilidade entre as diferentes implementações do protocolo RPL. Os resultados demonstraram que as implementações são interoperáveis mas que certas escolhas de implementação e de componentes do sistema podem afetar o desempenho do roteamento de modo significativo.

As avaliações apresentadas neste artigo se diferenciam das anteriores e trazem novas contribuições de pesquisa pois levam em conta métricas adicionais, incluindo a avaliação da vazão do protocolo RPL e da sua capacidade de reconstruir rotas a partir da falha ou desligamento de parte dos nós da rede, requisitos fundamentais para aplicações em RSSFs. Além disso, avaliamos as versões mais atuais das duas implementações da forma como são distribuídas hoje pelo TinyOS e Contiki.

5. Comentários finais

A implementação da pilha de protocolos IP nas LANs possibilitará uma nova geração de aplicações onde a infraestrutura de comunicação poderá ser compartilhada de forma transparente. Além disso, qualquer dispositivo conectado à Internet poderá se comunicar diretamente com um nó (e.g., um sensor) na LAN remota. Embora essa comunicação direta não seja necessária em todos os casos, ela pode ser requerida para fins de manutenção ou adaptação de código.

Neste trabalho avaliamos duas das principais implementações da pilha de protocolos IPv6 para o contexto das RSSFs, disponibilizadas atualmente como parte dos sistemas operacionais TinyOS e Contiki. Os resultados obtidos em termos de eficiência (taxa de perda de pacotes e tempo de comunicação entre pares de nós) demonstram que as

duas implementações diferem significantemente e dependem diretamente do percentual de nós emissores versus roteadores na rede. Essas diferenças podem estar diretamente relacionadas com a Função Objetivo (configurada no algoritmo de roteamento para determinar como escolher o nó pai na árvore de roteamento e auxiliar nas decisões de redirecionamento), como já salientado por outros autores [Ko et al. 2011c]. Nesse sentido, uma direção de pesquisa é a investigação aprofundada sobre o impacto da escolha da função objetivo e suas configurações no desempenho geral da rede, incluindo as questões de interoperabilidade entre as diferentes implementações dos protocolos.

Os resultados obtidos sobre a capacidade de vazão do protocolo de roteamento também apontam para a necessidade de investigações mais detalhadas para determinar o quanto esses resultados podem estar associados ao protocolo de controle de acesso ao meio (CSMA-CA) da camada IEEE 802.15.4. Uma hipótese é que pacotes maiores sofrem menos perda porque favorem o algoritmo de deteção de ocupação do meio, reduzindo a ocorrência de colisões de pacotes.

Em termos de consumo de memória, foi possível constatar que as implementações atuais ainda não permitem o adoção da pilha IPv6 em plataformas de sensores com menos de 6KBytes de memória RAM (como é o caso da plataforma *micaZ*), mas já tornou viável o seu uso em outras plataformas com maior capacidade de memória, como é o caso das plataformas *telosB* e *sky*.

Em relação às novas especificações da IETF, de forma geral, as estratégias de compactação e roteamento demonstraram se adequar às especificidades das RSSFs. O RPL é capaz de lidar dinamicamente com a inatividade de certos nós na rede, mas ainda a um custo elevado em termos do tempo necessário para refazer as rotas estabelecidas.

Agradecimentos

Este trabalho foi parcialmente financiado pelo CTIC/RNP (através do projeto "CIA^2 – Construindo Cidades Inteligentes: da Instrumentação dos Ambientes ao desenvolvimento de Aplicações") e pela FAPERJ (processo n^o 112.552/2012).

Referências

Cody-Kenny, B., Guerin, D., Ennis, D., Simon Carbajo, R., Huggard, M., and Mc Goldrick, C. (2009). Performance evaluation of the 6LoWPAN protocol on MicaZ and TelosB motes. In *4th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, pages 25–30. ACM Press.

Contiki-COOJA (2012). www.contiki-os.org.

Couto, D., Aguayo, D., Bicket, J., and Morris, R. (2005). A high-throughput path metric for multi-hop wireless routing. *Wireless Networks*, 11(4):419–434.

Dunkels, A. (2003). Full TCP/IP for 8-bit architectures. In *Proceedings of the 1st International Conference on Mobile Systems Applications and Services (MobiSys)*, pages 85–98. ACM Press.

Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki — a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462. IEEE.

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

Hui, J. (2008). Compression format for IPv6 datagrams in 6LoWPAN networks. draft-ietf-6lowpan-hc-01.

Hui, J. and Culler, D. (2008). IP is dead, long live IP for wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys)*, pages 15–28. ACM Press.

Hui, J., Shelby, Z., and Culler, D. (2007). Interoperability test for 6LoWPAN. draft-ietf-6lowpan-interop-00.

Hui, J. and Thubert, P. (2009). Compression format for IPv6 datagrams in 6LoWPAN networks. draft-ietf-6lowpan-hc-06.

Hui, J. and Thubert, P. (2011). Compression format for IPv6 datagrams over IEEE 802.15.4-based networks. RFC6282.

Ko, J., Dawson-Haggerty, S., Gnawali, O., Culler, D., and Terzis, A. (2011a). Evaluating the performance of RPL and 6LoWPAN in TinyOS. In *Proceedings of the Workshop on Extending the Internet to Low power and Lossy Networks (IPSN/2011)*. ACM Press.

Ko, J., Eriksson, J., Tsiftes, N., Dawson-Haggerty, S., Terzis, A., Dunkels, A., and Culler, D. (2011b). ContikiRPL and TinyRPL: happy together. In *Proceedings of the Workshop on Extending the Internet to Low power and Lossy Networks (IPSN 2011)*.

Ko, J. G., Eriksson, J., Tsiftes, N., Dawson-Haggerty, S., Vasseur, J. P., Durvy, M., Terzis, A., Dunkels, A., and Culler, D. (2011c). Beyond interoperability–pushing the performance of sensor network IP stacks. In *ACM SenSys'11*.

Korte, K., Sehgal, A., and Schönwälder, J. (2012). A study of the RPL repair process using ContikiRPL. *Dependable Networks and Services*, pages 50–61.

Kushalnagar, N., Montenegro, G., and Schumacher, C. (2007). IPv6 over low-power wireless personal area networks (6LoWPANs): Overview, assumptions, problem statement, and goals. RFC4919.

Levis, P., Clausen, T., Hui, J., Gnawali, O., and Ko, J. (2011). The trickle algorithm. RFC6206.

Montenegro, G., Kushalnagar, N., Hui, J., and Culler, D. (2007). Transmission of IPv6 packets over IEEE 802.15.4 networks. RFC4944.

Thubert, P. (2011). RPL objective function zero. draft-ietf-roll-of0-20.

TinyOS-Blip (2012). docs.tinyos.net/tinywiki/index.php/blip_2.0.

Winter, T. and et.al. (2010). RPL: IPv6 routing protocol for low-power and lossy networks. draft-ietf-roll-rpl-17.

Winter, T. and et.al. (2012). Rpl: IPv6 routing protocol for low-power and lossy networks. RFC6550.