Interligando RSSFs com a Internet Utilizando IPv6

Matheus Nascimento, Bruno Silvestre, Lucas Zenha[1]

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

{matheusnacimento,brunoos,lucaszenha}@inf.ufg.br

Abstract. *Internet of Things is a concept for interconnection and interaction of everyday objects with Internet. WSN plays an important role in this interconnection, but there are challenges to be overcome due to the nature of WSN. This work presents studies related to interconnection of sensor networks via IPv6.*

Resumo. *Internet das Coisas é um conceito de interligação e interação dos objetos do cotidiano com a infraestrutura de Internet existente. RSSF cumpre um papel importante nessa interligação, mas há desafios a serem vencidos devido a natureza do funcionamento das RSSF. Neste trabalho apresentamos estudos relacionados à interligação de RSSF via IPv6.*

1. Introdução

O avanço na tecnologia de equipamentos embarcados tem levado à sua disseminação, ocasionando cada vez mais a instrumentação dos ambientes. Isso permite a coleta de informações, monitoramento e até interação com o ambiente por meio do computador. A interligação de redes de sensores sem fio (RSSF), objetos do cotidiano e a Internet cumpre um importante papel nessa tendência de uma infraestrutura global e integrada, formando o conceito de Internet das Coisas.

Uma RSSF é normalmente composta por um grande conjunto de nós, muitos deles dotados de sensores, outros apenas de apoio no funcionamento da própria rede [Sohraby et al. 2007]. Os nós são pequenos equipamentos que contêm basicamente bateria, memória, processador, sensores e rádio. Eles possuem restrições de capacidade de processamento, armazenamento, e consumo de energia. A comunicação é via rádio e pode seguir padrões como Bluetooth ou IEEE 802.15.4, por exemplo. O objetivo é ter equipamentos de baixo custo e que podem ser usados em grande volume.

Podemos citar como exemplo o uso mais clássico de rede de sensores no monitoramento de mecanismos em ambientes industriais. Mais recentemente, temos o conceito de *Smart Grid*, controlando a distribuição de energia elétrica. Essas redes também estão presentes na área da medicina, possibilitando o monitoramento dos sinais vitais e controle de medicamentos. Nas cidades, temos sensoriamento de volume de chuvas, umidade relativa do ar, nível de poluição, tráfego de veículos, etc.

Os desafios agora se concentram em como realizar a interconexão das RSSF com a tecnologia de comunicação que já temos hoje em nossas máquinas (*desktops*, *tablets*, celulares, etc.). Diferentemente de outras redes, as RSSFs vêm sendo tratadas de forma muito específica para cada uma das aplicações que as utilizam devido às características desse tipo de rede.

Neste trabalho nós apresentamos um levantamento do uso de IPv6 em RSSF que permite a comunicação direta de aplicações com os nós sensores, utilizando a infraestrutura de que dispomos hoje. Como parte dos estudos, analisamos a implementação *blip* de IPv6 para RSSF [WEBS 2011] e, para exemplificar a interoperabilidade com os sensores, desenvolvemos uma pequena aplicação que é embarcada nos nós sensores, e que permite recolher dados e interagir com o equipamento por meio de requisições HTTP.

Na seção 2 apresentamos o levantamento de paradigmas de desenvolvimento para RSSF (não-IP) comumente utilizado para RSSF. Na seção 3 apresentamos as características, limitações e adaptações no protocolo IPv6 para se adequar à RSSF. Na seção 4 apresentamos a nossa aplicação exemplo de interligação de RSSF sobre IPv6. Finalmente, na seção 5, fazemos as considerações finais sobre o trabalho.

2. Modelos Tradicionais de Programação para RSSF

Inicialmente, a comunidade científica pensou que, dada a diferença entre os requisitos das RSSFs e os da Internet, toda a estrutura de serviços deveria ser reconsiderada [Estrin et al. 1999]. Foi assumido que a arquitetura em camadas não poderia ser utilizada, uma vez que as RSSFs têm diversas restrições de recursos. Imaginaram que a única maneira de conseguir a robustez e escalabilidade necessária para uma RSSF seria usando algoritmos e protocolos específicos.

Geralmente são seguidos três modelos de desenvolvimento de aplicações: Centrado em Posição, Dados ou Nós [Niculescu 2005]. Eles caracterizam como é a visão da rede e como os nós agem quanto à disseminação de informações, dando mais ênfase para três partes principais: descoberta, consultas e roteamento, respectivamente.

No modelo Centrado em Posição os nós utilizam técnicas e/ou equipamentos, como por exemplo GPS, para descobrir sua posição. Nesse modelo temos que não há tabelas de roteamento, a mensagem é roteada para o nó cuja posição é mais próxima do nó destino, ou seja, todo o funcionamento da rede se baseia nas posições dos nós.

No modelo Centrado em Dados a RSSF pode ser vista como um banco de dados que armazena os dados sensoriados e responde a consultas de alto nível. O aspecto central é que apenas os dados sensoriados importam e não quais são os nós que os capturaram. Esse modelo escala bem, já que as tabelas de roteamento crescem pelos tipos de dados consultados e não pela quantidade de nós. Porém, os protocolos utilizados na disseminação não são gerais e dependem do tipo da informação.

Já o modelo Centrado em Nós representa a maneira tradicional de vermos as rede. Cada nó possui um identificador, o qual é a base do roteamento. As soluções são muito bem estudadas, pois é o modelo usado na maioria das redes de computadores. Devido à divisão em camadas, apresenta uma grande flexibilidade na modificação de protocolos e implementações de novas soluções.

Seguindo esses paradigmas, o desenvolvimento de aplicações para RSSF levou a uma arquitetura de interligação com a Internet muito usada atualmente, que pode ser vista na Figura 1. A RSSF é baseada em um protocolo interno próprio, e isso gera a necessidade de se utilizar um roteador, na camada de aplicação, responsável por efetuar a tradução de requisições IP para o protocolo interno da RSSF.

Essa abordagem permite a interligação de RSSF já operantes, no entanto, apre-

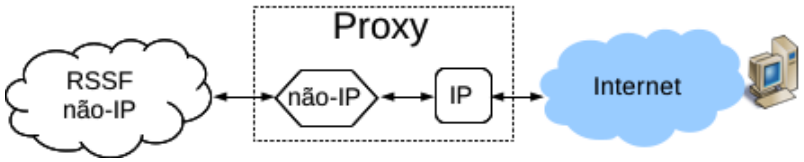

Figura 1. Interligação de RSSF que não utiliza IP com a Internet.

senta diversos problemas, como por exemplo, esconde o acesso direto aos nós para as aplicações baseadas em IP. Dunkels *et al.* [Dunkels and Vasseur 2008] dizem que o roteador de protocolo têm inerentemente uma modelagem complexa e são difíceis de gerenciar e construir. E a cada nova funcionalidade adicionada a um determinado nó sensor, o roteador também deve ser atualizado.

3. Uso de IP em RSSF

A primeira implementação de IP para nós sensores foi programada para o Contiki [Dunkels 2003]. Isso gerou uma pequena revolução na comunidade científica, e a viabilidade do protocolo IP para RSSF foi repensada [Neves and Rodrigues 2010]. Nessa arquitetura baseada em IP, os nós sensores implementam diretamente o protocolo. Um ou mais roteadores de borda são empregados para fazer a conversão de camadas de mais baixo nível, transformando requisições vindas de redes cabeadas ou WiFi, por exemplo, para o padrão de rádio da RSSF, como pode ser visto na Figura 2.

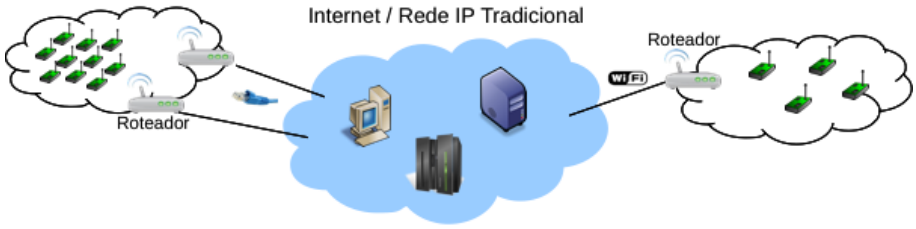

Figura 2. Arquitetura de RSSF baseada em IP.

No entanto, essa primeira implementação seguiu a versão 4 do IP (IPv4). Dado que se espera que uma RSSF seja composta por uma grande quantidade de nós, ficou claro que a quantidade de endereços fornecidos pelo IPv4 era inadequado para uso com RSSF (várias RSSFs com vários nós sensores) — de fato, a quantidade já era apontada como insuficiente para o número de computadores conectados à Internet. Outras características de RSSF, como a dinamicidade da topologia e a dificuldade de manutenção dos nós devido ao tamanho da rede e a localização desses nós, contribuíam para que o IPv4 não fosse um protocolo muito adequado para esse tipo de rede.

Por outro lado, as melhorias propostas no IP versão 6 (IPv6), sucessor do IPv4, continham mecanismos que eram interessantes para as RSSFs. A grande quantidade de endereços foi o primeiro passo. Além disso, na definição do IPv6 estão descritos mecanismos de distribuição de endereços, sendo um deles conhecido como *stateless*. Não há um serviço central de distribuição de endereço, os nós geram os endereços baseados em prefixos e identificadores da camada de enlace. O mecanismo de descoberta de vizinhos do IPv6 também permite que os próprios nós se auto-organizem para o roteamento de mensagens [Narte et al. 2007].

Apesar desses fatores incentivadores do uso de IPv6, o protocolo não foi proposto levando em considerações as limitações dos nós sensores. O principal fato contra a adoção do protocolo foi o tamanho do MTU, que deve ser no mínimo de 1280 bytes. A título de comparação, o padrão IEEE 802.15.4, usado nas RSSFs, permite um *payload* de 127 bytes, então, teríamos um *overhead* em torno de 15% e 30% somente para transportar os cabeçalhos do IPv4 e IPv6, respectivamente [Hui and Culler 2008].

Foi iniciado então estudos para adaptar o IPv6 às RSSFs, que resultou no padrão 6LoWPAN [Kushalnagar et al. 2007, Montenegro et al. 2007, Hui and Culler 2008], o qual define técnicas para transmitir pacotes IPv6 sobre redes IEEE 802.15.4, por exemplo, compressão dos cabeçalhos da pilha (incluindo ICMP, UDP e TCP) para poupar espaço, e esquemas de fragmentação. Basicamente, foi inserida uma camada de adaptação entre as camadas de rede e enlace.

4. Estudo de Caso

Este estudo de caso visa apresentar uma aplicação desenvolvida para RSSF que permite a leitura direta dos sensores, seguindo a arquitetura vista na Figura 2. Desenvolvemos um pequeno servidor HTTP que é executada em cada nó sensor, assim, podemos utilizar navegadores, ferramentas (wget, curl) ou bibliotecas HTTP para recuperar as informações diretamente de um dos nós sensores da rede.

O sistema foi desenvolvido em *TinyOS* 2.1.1 [Levis et al. 2004, TinyOS 2011], que é um dos sistemas operacionais mais usado na área de pesquisa de RSSF. O TinyOS conta com uma implementação de 6LoWPAN que está sendo desenvolvida na universidade de Berkeley, na Califórnia, e se chama *blip* (*Berkeley IP implementation for low-power networks*) [WEBS 2011]. As funcionalidades mais importantes que o blip provê atualmente são a descoberta de vizinhos, seleção de rota padrão, e roteamento ponto-a-ponto. O blip também oferece os protocolos ICMP e UDP, e há uma pilha TCP ainda experimental, mas suficiente para nossa aplicação.

A arquitetura utilizada no teste consiste de dois computadores, rodando Linux 2.6, em uma rede local cabeada IEEE 802.3 (Ethernet): um como cliente e outro como roteador de borda. Foram usados também três nós sensores TelosB, cada um dotado de três leds e sensores de luminosidade, temperatura e umidade, em uma rede IEEE 802.15.4. Dois dos nós sensores executam a aplicação desenvolvida e o terceiro é a *estação base*, a qual funciona como uma ponte para que o roteador de borda envie e receba pacotes para a RSSF. A Figura 3 mostra a arquitetura.

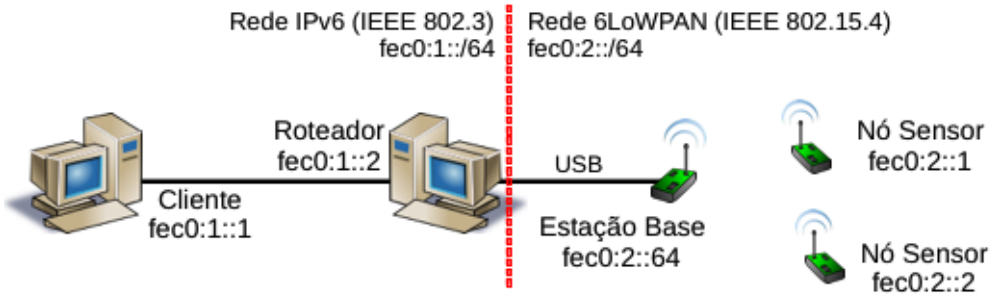

Figura 3. Arquitetura utilizada no estudo de caso.

Para a interligação dos dois computadores, utilizamos a pilha IPv6 nativa do Linux. Adicionamos os endereços IPv6 manualmente nas interfaces de rede, ambos utilizando a rede `fec0:1::/64`. Configuramos o cliente com o endereço `fec0:1::1` e a interface do computador roteador foi configurada com o endereço `fce0:1::2`. Mas, para que os pacotes endereçados à RSSF sejam encaminhados do cliente para o roteador, e finalmente para a RSSF, tivemos que configurar a tabela de rota do cliente.

Os nós sensores foram configurandos com endereços estáticos que são definidos no momento em que nossa aplicação é instalada no nó. Os dois nós sensores que farão a leitura do ambiente foram configurados para a rede `fec0:2::/64`, com os endereços `fec0:2::1` e `fec0:2::2`.

4.1. Ligação da Estação Base ao Computador Roteador

Para que os pacotes pudessem efetivamente ser entregues ao seu destino, uma das plataformas de sensores foi ligada diretamente ao roteador por meio de um cabo USB (estação base). Esse nó é carregado com uma aplicação que repassa todos os pacotes da RSSF para o roteador via USB, e vice-versa. Podemos dizer que o nó funciona como uma placa de rede para o roteador. O código do blip já inclui a aplicação que deve ser colocada na estação base para realizar a ponte descrita.

O blip também inclui um driver para Linux que faz o tunelamento de pacotes da rede cabeada para a RSSF, funciona em conjunto com o código da estação base. Esse driver cria uma interface de rede virtual no Linux (por exemplo, `tun0`) por onde os pacotes são enviados e lidos da RSSF. Configuramos a interface com endereço de rede `fec0:2::64`. Além do encaminhamento dos pacotes, o driver também fica responsável por fazer as alterações necessárias no cabeçalho dos pacotes IPv6 para se enquadrem no padrão 6LoWPAN, e poderem ser enviados para a RSSF.

Após os endereços serem configurados e o driver de tunelamento ser ativado, podemos utilizar aplicações usuais para redes IPv6, incluidas nas distribuições Linux, para testar a conexão com o sensor, por exemplo, ping6, nc6, tracert6, wget, e até o *sniffer* WireShark (usando a interface `tun0`) para verificar os pacotes que estão sendo trocados com a RSSF.

4.2. Aplicação HTTP para os Nós Sensores

A aplicação desenvolvida pode ser descrita em duas partes:

- Um temporizador que a cada segundo dispara a leitura dos sensores disponíveis na plataforma TelosB e armazena seus valores em variáveis internas;
- Um servidor HTTP simples que espera requisições e envia respostas contendo a última leitura armazenada dos sensores requisitados.

O TinyOS utiliza como linguagem de programação o *nesC*, um dialeto da linguagem C, mas com suporte a programação orientada a eventos. Esse paradigma de programação é apropriado para a programação dos sensores, dado que eles espelham bem o funcionamento do equipamento.

Apesar do paradigma de programação, o blip oferece uma API de programação para IPv6 semelhante a bem difundida API *socket* proposta pelo BSD e presente nos sistemas operacionais baseados em UNIX, bem como o Windows. A Listagem 1 mostra a assinatura de algumas funções do API TCP do blip.

```
interface Tcp {
  command error_t bind(uint16_t port);
  event bool accept(struct sockaddr_in6 *from,
                    void **tx_buf, int *tx_buf_len);
  command error_t connect(struct sockaddr_in6 *dest,
                          void *tx_buf, int tx_buf_len);
  command error_t send(void *payload, uint16_t len);
  event void recv(void *payload, uint16_t len);
  command error_t close();
  event void acked();
}
```

Listagem 1. API *socket* disponibilizada pelo blip

Os comandos (`command`) são funções que são chamadas pelo programa para realizar uma ação, já os eventos (`event`) são funções que o TinyOS chama para sinalizar a ocorrência de um evento. Por exemplo, para preparar o nó sensor de forma que ele receba conexão, o comando `bind` é chamado pela aplicação logo após o *boot* do nó. Quando uma nova conexão chega, o TinyOS invoca o evento `accept`. Note a ausência do `listen`, presente na API *socket* BSD: o `bind` faz o papel das duas funções, ou seja, houve uma simplificação em certos procedimentos.

Apesar de tentar replicar a API *socket*, há momentos em que o funcionamento do protocolo é exposto para a aplicação, por exemplo, o evento `accept` deve retornar (por meio dos parâmetros) um buffer que será usado pela camada TCP. Em sistemas operacionais como Linux ou Windows, esse buffer é controlado internamente pela camada. Outro detalhe é a exposição do evento de ACK. Isso é feito porque, devido à restrição de memória, o blip passa para a aplicação a responsabilidade de dimensionar o quanto ela necessita para o seu funcionamento. No caso do ACK, a aplicação não vai tentar enviar várias informações até ter certeza de que a anterior foi recebida, sem isso, poderia ocorrer um estouro de buffer interno de fragmentação, ocasionando perda de pacotes.

Em termos de interoperabilidade, a nossa aplicação seguiu o protocolo REST [Fielding and Taylor 2002] para exportar os recursos do nó sensor. Os recursos exportados pela aplicação são:

- **/**: informa todos os recursos disponíveis;
- **/sensor/temp**: retorna a temperatura;
- **/sensor/umi**: retorna a umidade;
- **/sensor/lum**: retorna a luminosidade;
- **/sensor/all**: retorna todos os valores acima;
- **/led/blue**: interage com o led azul;
- **/led/yellow**: interage com o led amarelo.

O conteúdo da reposta é compatível com a definição JSON [JSON.org 2006], o que facilita de certa forma a interação de aplicações com os nós sensores (dada a existência de bibliotecas para lidar com essa notação). Além disso, a notação JSON é mais compacta que XML, comumente usado na interconexão de aplicações. Por exemplo, ao requisitar via o navegador Firefox quais são os recursos disponíveis, usamos a URL `http://[fec0:2::1]/`. A Listagem 2 mostra a resposta dessa requisição.

```
{ "resources":[
    "/sensor/temp", "/sensor/hum", "/sensor/lum",
    "/sensor/all",  "/led/blue",   "/led/yellow"  ] }
```

Listagem 2. Resposta da requisição sobre recursos REST disponível nos nós sensores.

Como outro exemplo, o conteúdo referente à leitura da temperatura é `{"temp":1382}`. Nesse caso, o valor da temperatura é um inteiro de 16 bits retornado diretamente da leitura do sensor. O desenvolvedor deve recorrer ao manual do equipamento para saber como relacionar esse valor com uma escala de graus Celsius, por exemplo.

Na implementação atual, os sensores respondem ao GET do comando HTTP para retornar valores a aplicação. Caso seja necessário enviar algum comando de atuação sobre o nó sensor, utilizamos o comando PUT do HTTP (seguindo recomendação REST). Como exemplo, oferecemos a possibilidade da aplicação cliente ligar e desligar os dois leds do sensor (um azul e outro amarelo). Isso é feito com o cliente enviando o código "`PUT /led/blue HTTP/1.1`".

5. Considerações Finais

Neste trabalho, apresentamos um estudo da utilização de IPv6 em redes de sensores sem fio, como forma de interligação dessas redes com a infraestrutura da Internet (criando o conceito de Internet das Coisas). Esse estudo faz parte do projeto *Construindo Cidades Inteligentes: da Instrumetação dos Ambientes ao desenvolvimento de Aplicações (CIA*^{2}*)* que prevê a criação de um *testbed* para que pesquisadores possam testar aplicações em RSSFs. O projeto é financiado pela Rede Nacional de Ensino e Pesquisa (RNP) e conta ainda com a participação da UFRJ, UFES e PUC-Rio.

RSSFs sempre foram tratadas como um caso a parte devido suas limitações de recursos (memória, processamento e energia). No entanto, estudos de caso mostram que é possível utilizar a infraestrutura IPv6 nessas redes. Adaptações foram necessárias para que isso ocorresse, como mostra o padrão 6LoWPAN. Os estudos sobre o assunto ainda continuam, com a proposta de novos protocolos de roteamento, levando em consideração a natureza da interação dos sensores [Winter et al. 2011], bem como protocolos mais no nível de aplicação que sejam mais eficientes na troca de informações [Shelby and Frank 2010].

Apresentamos como exemplo da interação de aplicações existentes com RSSFs um pequeno servidor HTTP que é embutido nos nós sensores e permite, por exemplo, recuperar as informações por meio de navegadores web. Utilizamos a implementação IPv6 *blip*, disponível no TinyOS, para construir nossa aplicação. Podemos notar que, apesar do esforço de oferecer a mesma API de programação em *socket* dos sistemas operacionais de computadores, foi necessário adaptações para a API se acomodar ao ambiente com poucos recursos.

Referências

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. and Vasseur, J. (2008). IP for smart objects. Internet Protocol for Smart Objects (IPSO) Alliance. White Paper 1.

Estrin, D., Govindan, R., Heidemann, J., and Kumar, S. (1999). Next century challenges: Scalable coordination in sensor networks. In *5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 263–270.

Fielding, R. and Taylor, R. (2002). Principled design of the modern web architecture. *ACM Transaction Internet Technology*, 2:115–150.

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.

JSON.org (2006). RFC 4627 – the application/JSON media type for JavaScript Object Notation (JSON).

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

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 *in Ambient Intelligence*. Springer Verlag.

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

Narte, T., Nordmak, E., Simpson, W., and Soliman, H. (2007). RFC 4861 – neighbor discovery for IP version 6 (IPv6).

Neves, P. and Rodrigues, J. (2010). Internet protocol over wireless sensor networks, from myth to reality. *Journal of Communications*, 5(3):189–196.

Niculescu, D. (2005). Communication paradigms for sensor networks. *Communications Magazine*, 43(3):116–122.

Shelby, Z. and Frank, B. (2010). Constrained Application Protocol (CoAP) – Draft IETF.

Sohraby, K., Minoli, D., and Znati, T. (2007). *Wireless Sensor Networks: Technology, Protocols, and Applications*. John Wiley & Sons.

TinyOS (2011). TinyOS project. `http://www.tinyos.net/`.

WEBS, B. (2011). blip – Berkeley IP Information. `http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip`.

Winter, T., Thubert, P., Brandt, A., Clausen, T., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., and Vasseur, J. (2011). RPL: IPv6 Routing Protocol for Low power and Lossy Networks – Draft IETF.