TinySDN: Enabling Multiple Controllers for Software-Defined Wireless Software Networks

B. T. de Oliveira, L. B. Gabriel and C. B. Margi, *Senior Member*, *IEEE*

***Abstract*— Software-Defined Networking (SDN) has been envisioned as a way to reduce the complexity of network configuration and management, enabling innovation in production networks. While SDN started focusing on wired networks, there were proposals specifically for Wireless Sensor Networks (WSN), but all of them required a single controller to be coupled to the sink. This paper presents TinySDN, a TinyOS-based SDN framework that enables multiple controllers within the WSN. It comprises two main components: the SDN-enabled sensor node, which has an SDN switch and an SDN end device, and the SDN controller node, where the control plane is programmed. TinySDN was designed and implemented to be hardware independent. Experiments were conducted on COOJA simulator, and results concerning delay and memory footprint are presented.**

***Keywords*— Wireless Sensor Networks, Software-Defined Networking, Software-Defined Wireless Sensor Networks.**

I. INTRODUÇÃO

REDES definidas por software (ou SDN, do inglês *software-defined networking*) é um paradigma emergente que utiliza um software logicamente centralizado para controlar o comportamento de uma rede. Foi concebido com o objetivo de reduzir a complexidade de configuração e gerenciamento de redes, permitindo pesquisa e inovação em redes de produção. As primeiras iniciativas de implementação de SDN foram direcionadas às redes cabeadas, entretanto este paradigma despertou o interesse da comunidade de redes sem fio e, então, diversas empresas que atuam na área de comunicação sem fio e redes móveis juntaram-se a iniciativas relacionadas [1]. Desde então, algumas propostas de adoção de SDN no contexto de redes sem fio foram apresentadas, como SDN em redes de substituição [2] e em redes de compartilhamento de banda larga [3].

Considerando as redes de sensores sem fio (RSSF), um recurso interessante que pode ser implementado através de dispositivos com suporte a SDN é o gerenciamento de nós e de seus recursos. Por exemplo, quando o controlador determina a rota a ser utilizada, ela pode considerar a energia disponível em dado nó (ou conjunto de nós), poupando os nós com pouca energia para execução de tarefas indispensáveis e proporcionando um maior tempo de vida à rede. Além disso, o controlador pode detectar quando um dado nó sensor falhar, pois deixará de receber as mensagens esperadas deste. Ainda, os nós de RSSF são frequentemente considerados dispositivos descartáveis e baratos que podem ser implantados para uma tarefa específica, mas em *iCities* ou cidades inteligentes, os nós sensores devem coletar, processar e transmitir diferentes tipos de dados para diferentes aplicações [4]. Se esses nós sensores e outros dispositivos de coleta de dados forem gerenciados pelo paradigma SDN, pode-se obter uma melhor utilização da infraestrutura subjacente através de roteamento dinâmico e reconfiguração de tarefas dos dispositivos.

As propostas anteriores relacionadas a RSSF definidas por software incluem o Flow-Sensor [5], o Sensor OpenFlow [6], o SDWN [1] e o trabalho de Gante *et al.* [7]. No entanto, esses trabalhos não abordam características comuns de RSSF, como a possível interrupção e atraso na comunicação, a limitação de energia e o comprimento dos quadros de enlace reduzido. Além disso, os dispositivos típicos em redes sem fio de múltiplos saltos e RSSF têm apenas um módulo rádio para transmitir e receber em uma determinada frequência em um dado momento e, portanto, os pacotes de controle são transmitidos pelas mesmas conexões que os pacotes de dados (característica conhecida como *in-band control*).

Neste trabalho apresenta-se o TinySDN, uma arquitetura que possibilita a utilização de múltiplos controladores em redes de sensores sem fio definidas por software. Sua implementação e parte de seu projeto são baseados no sistema operacional TinyOS. Os principais desafios abordados são: (i) *in-band control*, diferente das redes definidas por software cabeadas, que podem lançar mão de enlaces dedicados ao tráfego de pacotes de controle; (ii) maior latência na comunicação; (iii) quadros da camada de enlace muito reduzidos; e (iv) suprimento de energia limitado.

O paradigma SDN considera a separação dos planos de dados e de controle. Contudo, apesar dos recentes esforços [8], dispositivos típicos de RSSF têm apenas um rádio para transmitir ou receber em uma determinada frequência e em um determinado momento. Assim, em redes sem fio, planos de dados e de controle devem compartilhar o mesmo link de comunicação e largura de banda disponível. Esta característica limita a quantidade de dados que pode ser transmitida através de uma determinada conexão e potencialmente aumenta o atraso. Portanto, este trabalho é focado na redução do tráfego de controle sobre a RSSF. Além disso, por causa do compartilhamento da banda entre tráfego de dados e de controle é importante diferenciar fluxos de controle e fluxos de dados.

Outra questão que deve ser considerada é a largura de banda limitada fornecida pelos dispositivos compatíveis com o padrão IEEE 802.15.4. Comparada a controladores em redes cabeadas, a latência de comunicação em redes sem fio é acrescida pela média de apenas 250 Kbps e os múltiplos saltos a serem percorridos até alcançar o nó *gateway* e, em seguida,

B. T. de Oliveira, Escola Politécnica da Universidade de São Paulo (EPUSP), São Paulo, São Paulo, Brasil, btrevizan@larc.usp.br.

L. B. Gabriel, Escola Politécnica da Universidade de São Paulo (EPUSP), São Paulo, São Paulo, Brasil, lucasbg@usp.br.

C. B. Margi, Escola Politécnica da Universidade de São Paulo (EPUSP), São Paulo, São Paulo, Brasil, cintia@usp.br.

o controlador. Ao instalar o controlador no nó sorvedouro, como feito em [5], [6] e [1], a latência diminui em comparação com o cenário anterior, mas ainda produz alta latência de comunicação. A fim de diminuir ainda mais a latência total, exploramos a localidade, como sugerido por Schmid e Suomela [9], usando múltiplos controladores na RSSF e os aproximando dos nós sensores controlados.

Ademais, as informações de controle da SDN que precisam ser trocadas devem ser acomodadas dentro de um quadro de camada de enlace, e o padrão IEEE 802.15.4 determina que o quadro é de, no máximo, 127 bytes. Por fim, o aumento na transmissão de dados de controle pode levar a um acréscimo no consumo de energia, o que é uma preocupação dado o suprimento de energia limitado nas RSSF. O TinySDN evita o crescimento do tráfego de controle dispensando o uso de mecanismos de garantia de entrega fim-a-fim e lidando com as possíveis perdas de pacotes.

As principais contribuições deste trabalho incluem a concepção e implementação do TinySDN, uma arquitetura que permite o uso de múltiplos controladores em redes de sensores definidas por software, junto a sua implementação baseada no TinyOS e independente de plataforma de hardware. A arquitetura é composta por dois componentes principais: o **nó sensor SDN**, que faz o papel de um *switch* SDN e um dispositivo sensor final; e o **nó controlador SDN**, no qual o plano de controle é programado. Os experimentos foram realizados no simulador COOJA e os resultados apresentados, que incluem métricas como tempo de inicialização da rede, latência e consumo de memória, atestam a viabilidade da arquitetura.

II. TRABALHOS RELACIONADOS

A primeira iniciativa para aplicação do paradigma de redes definidas por software para redes de sensores sem fio foi o Flow-Sensor [5], que propõe nós sensores com os principais recursos do OpenFlow [10]. O objetivo é obter nós sensores mais confiáveis em comparação com nós sensores típicos, dado que os dispositivos da rede podem ser facilmente monitorados, controlados e reconfigurados, permitindo uma gestão dinâmica do ponto de vista de custo, eficiência energética e desempenho da rede.

As propostas seguintes, Sensor OpenFlow [6] e SDWN [1], propõem a separação clara entre o plano de dados e o plano de controle, e um componente central que padroniza a comunicação entre estes dois planos; permitindo reconfiguração dinâmica das tarefas dos sensores e promovendo o compartilhamento de recursos de hardware e funcionalidades implementadas entre diferentes aplicações da rede. Além disso, SDWN inclui outros recursos mais avançados como agregação de dados em rede, configuração de ciclos de trabalho e flexibilidade para definir regras e ações com o objetivo de permitir otimizações *cross-layer*.

O trabalho de Gante *et al.* [7] apresenta uma proposta de arcabouço para aplicação do paradigma SDN para a gestão de RSSF. Além dos benefícios destacados pelos trabalhos anteriores, os autores incluem localização precisa e descoberta de topologia como vantagens do uso SDN em RSSF. Em sua proposta, o controlador SDN deve ser implementado como componente do nó sorvedouro da rede. Duas camadas principais são consideradas neste arcabouço: (i) *middleware*, que trata das funções de rede (como mapeamento, tabelas de fluxos e aplicações do controlador), e (ii) aplicação, onde localização e algoritmos de rastreamento são executados. Detalhes e desafios de implementação não são discutidos, e a distribuição de controladores é deixada como uma questão em aberto.

III. A ARQUITETURA DO TINYSDN

TinySDN é uma arquitetura que, junto a sua implementação, permite múltiplos controladores em redes de sensores sem fio definidas por software, bem como o emprego do paradigma SDN em plataformas de RSSF compatíveis com o sistema operacional TinyOS [11]. O TinySDN converte o nó sensor sem fio em um nó com suporte ao paradigma SDN, chamado de **nó sensor SDN**, sendo este um elemento constituído por um *switch* SDN e um dispositivo final (ou *end device*) onde são programadas as tarefas das aplicações da RSSF. Assim, o componente *switch* SDN é programado através de um controlador SDN participante da mesma rede, o qual é denominado **nó controlador SDN**.

A Fig. 1 apresenta os dois tipos de nós especificados na arquitetura TinySDN. Cada **nó sensor SDN**, onde é executado o plano de dados, conecta-se através de uma comunicação sem fio de múltiplos saltos a um **nó controlador SDN**, onde a lógica do plano de controle é executada, permitindo assim a interação entre os dois planos.

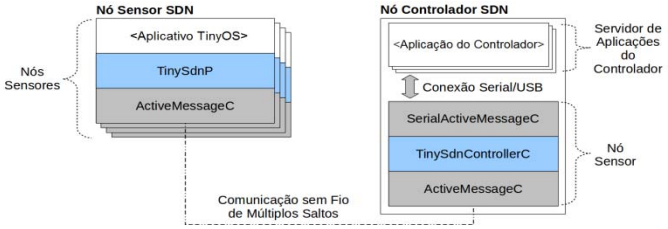


Figura 1. Visão geral da arquitetura TinySDN e seus componentes.

Em relação a garantia de entrega fim-a-fim para a comunicação sem fio de múltiplos saltos, é considerado que o mecanismo de garantia de entrega salto-a-salto do padrão IEEE 802.15.4 [12] é suficiente e perdas de pacote podem ocorrer. Serviços de segurança fim-a-fim podem ser obtidos a partir do uso se soluções com o WSN-ETESec [13].

A seguir são descritos os dois tipos de nós, bem como a estratégia empregada pelo **nó sensor SDN** para encontrar o **nó controlador SDN** e estabelecer uma conexão entre eles, o processo de especificação de fluxos e ações relacionadas, e o processo de coleta de informações de topologia da rede.

A. Nó Sensor SDN

Este é o componente que é instanciado nos nós sensores. Como discutido em [6], dispositivos finais comumente são considerados periféricos às redes definidas por software e consequentemente estão fora do escopo do OpenFlow [10], principal projeto baseado no paradigma SDN atualmente.

Todavia, nós sensores se comportam como dispositivos finais por gerarem pacotes para a transmissão de dados coletados através de seus componentes de sensoriamento, além de também encaminharem dados, como os *switches* SDN. Assim, o nó sensor com recursos SDN é um tipo de nó que desempenha os dois papéis: *switch* SDN e dispositivo final SDN.

Cada **nó sensor SDN** deve se associar a um **nó controlador SDN** para receber especificações de fluxos e requisitar os mesmos quando necessário. Como ilustrado na Fig. 1, o **nó sensor SDN** é dividido em três partes: **aplicativo TinyOS**, **TinySdnP** e **ActiveMessageC**.

O componente chamado de **aplicativo TinyOS** é equivalente ao dispositivo final e é escrito pelo programador (ou usuário) da rede, conforme a aplicação RSSF. Este componente gera pacotes de dados e então os insere na rede utilizando uma interface de programação provida pelo componente **TinySdnP**.

O **TinySdnP** é o componente principal do TinySDN, responsável por verificar se um pacote recebido corresponde a alguma entrada especificada nas tabelas de fluxos e assim realizar a ação relacionada, ou então enviar uma requisição de especificação de fluxo para o **nó controlador SDN**. Consequentemente, também é responsável por realizar a atualização das tabelas de fluxos quando recebe uma especificação de fluxo vinda do **nó controlador SDN**.

O **ActiveMessageC** é o componente do TinyOS que gerencia e provê a interface de programação que permite a interação com o módulo de rádio do nó sensor. É usado pelo TinySDN para realizar todas as tarefas relacionadas à comunicação sem fio, tais como encaminhamentos de pacotes de dado e controle e coleta de informações de topologia.

B. Nó Controlador SDN

Nó controlador SDN é o nó que realiza tarefas referentes ao controlador SDN, ou seja, aplica definições de aplicações SDN, criando e gerenciando fluxos na rede. O TinySDN permite o uso de múltiplos nós, então é possível adicionar mais de uma instância deste elemento na rede.

Como pode ser observado na Fig. 1, ele é composto por dois módulos de hardware diferentes: o **módulo nó sensor** e o módulo **servidor de aplicações do controlador**.

Módulo nó sensor - É executado em um nó sensor, sendo este responsável pela comunicação com cada **nó sensor SDN** usando a interface **ActiveMessageC**. Ele utiliza a interface **SerialActiveMessageC** (componente de comunicação serial do TinyOS) para encaminhar as mensagens recebidas da rede para o módulo **servidor de aplicações do controlador** e receber mensagens, vindas do módulo servidor, a serem enviadas para a rede. O componente chamado **TinySdnControllerC** tem como função adaptar mensagens e gerencia esta comunicação.

Módulo servidor de aplicações do controlador - Contém a lógica do plano de controle, ou seja, armazena aplicações do controlador e gerencia fluxos, bem como as informações de topologia da rede.

C. Especificação de Fluxos e Ações Relacionadas

O **nó sensor SDN** é responsável por classificar pacotes dentre diferentes fluxos e executar a ação relacionada. Duas ações são especificadas atualmente, as ações "*forward*" e "*drop*", semelhante ao OpenFlow [10]. A ação "*forward*" consiste em realizar encaminhamentos de pacote para um próximo nó especificado. A ação "*drop*" indica que o pacote deve ser descartado, bloqueando ou desativando um fluxo.

Existem dois tipos de fluxo: fluxos de controle, utilizados para manter o tráfego de pacotes de controle entre **nós sensores SDN** e **nós controladores SDN**; e fluxos de dados, utilizados pela aplicação para tráfego de dados. Fluxos são especificados como entradas nas tabelas de fluxos, sendo que cada entrada é composta por quatro campos: campo de identificação específica; campo de ação relacionada ao fluxo, que especifica a ação a ser realizada; o valor, campo que atribui um valor específico relacionado à ação (para a ação "*forward*", por exemplo, este campo especifica qual é o nó destino para o próximo salto do pacote); e contador, campo incrementado cada vez que um pacote correspondente a um dado fluxo é recebido pelo **nó sensor SDN**, provendo estatísticas sobre este fluxo.

Na tabela de fluxo de dados, o campo de identificação é a identificação do fluxo, que é usado para identificar os fluxos e correspondência (ou *matching*) dos pacotes de dados. A Tabela I contém um exemplo de tabela de fluxo de dados e algumas entradas.

TABELA I. EXEMPLO DA TABELA DE FLUXO DE DADOS.

IDENTIFICAÇÃO DO FLUXO	AÇÃO	VALOR	CONTADOR
1	*DROP*	-	100
2	*FORWARD*	5	10
101	*FORWARD*	10	27

No caso da tabela de fluxo de controle, o campo de identificação é o identificador do nó destinatário do fluxo de controle. A tabela II contém um exemplo de tabela de fluxo de controle.

TABELA II. EXEMPLO DA TABELA DE FLUXO DE CONTROLE.

IDENTIFICAÇÃO DO NÓ DESTINO	AÇÃO	VALOR	CONTADOR
0	*FORWARD*	4	5
1	*FORWARD*	4	2
7	*FORWARD*	6	2

D. Processo de Busca pelo Nó Controlador SDN e Comunicação com o Mesmo

A primeira tarefa do **nó sensor SDN**, a partir da inicialização da rede, é encontrar um **nó controlador SDN** e associar-se ao ele. Para isto, o TinySDN utiliza o protocolo CTP (*collection tree protocol*) [14] em segundo plano. O CTP é um protocolo que constrói uma árvore de roteamento, escoando todos os dados gerados na rede de sensores para um nó raiz, ou sorvedouro. Este é um protocolo amplamente adotado nas aplicações TinyOS de múltiplos saltos.

Selecionou-se o CTP considerando duas principais características desejáveis:

- **Independência de hardware -** o CTP usa valores de ETX (ou número de transmissões esperadas) como métrica. Este valor é derivado do componente do TinyOS *Four-Bit Link Estimator* [15], um componente que estima qual é a qualidade da conexão entre nós vizinhos, considerando o total de mensagens entregues e o total de mensagens perdidas, ao invés de utilizar o RSSI (do inglês *received signal strength indication*), que é um recurso específico de alguns módulos de rádio.
- **Múltiplas raízes -** O CTP possibilita o uso de múltiplas raízes, ou múltiplos nós sorvedouros e, consequentemente, a criação de múltiplas árvores de roteamento em uma rede. Quando múltiplas raízes estão presentes na mesma rede, cada nó adota a raiz mais conveniente para ele, tornando-se um elemento de sua árvore. A raiz mais conveniente é que apresenta menor custo para manter a comunicação a partir do nó sensor, considerando a métrica ETX. Os **nós controladores SDN** do TinySDN são configurados como raízes do CTP, permitindo que os **nós sensores SDN** os encontrem, se associem a eles e mantenham a comunicação.

No CTP, assim que a rede é inicializada, as raízes têm ETX=0 e os demais nós participantes têm ETX=∞. Cada nó descobre a existência de seus nós vizinhos através de respostas obtidas a partir do envio de pacotes de *beacon* em *broadcast*, então o vizinho com menor ETX será considerado o melhor vizinho. Assim, cada nó atualiza o valor do ETX seguindo a equação abaixo, onde "*le*(*id*)" é a função que prove a estimativa da qualidade do link para determinado nó vizinho.

$$\text{ETX} = [\text{ETX do melhor vizinho}] + \text{le}(\text{ID do melhor vizinho})$$

Uma vez que o CTP está estabilizado, o **nó sensor SDN** já é capaz de enviar a requisição de associação para o **nó controlador SDN** (configurado como raiz) através da rota do CTP, encaminhando um pacote contendo esta requisição ao seu melhor vizinho. Imediatamente após receber a requisição, o controlador encaminha ao **nó sensor SDN** requerente uma especificação de fluxo de controle.

E. Coleta de Informação de Topologia

O processo de coleta de informações de topologia do TinySDN é dividido em 2 etapas: (i) cada **nó sensor SDN** reconhece seus vizinhos e mede a qualidade de conexão com os mesmos, e (ii) envia as informações coletadas para o **nó controlador SDN** ao qual ele se associou, através de um pacote de CTP ou fluxos de controle. Estas etapas são executadas periodicamente de acordo com a configuração de tempo feita pelo projetista da rede.

Para reconhecer os vizinhos, o **nó sensor SDN** envia pacotes em *broadcast* e aguarda a resposta de seus vizinhos. Assim que recebe a resposta, cada nó que respondeu ao *broadcast* é adicionado à tabela de vizinhos conjuntamente com a qualidade de conexão entre eles.

A Tabela III contém um exemplo de tabela de vizinhos. A coluna ID do vizinho contém o ID dos nós vizinhos e a coluna Qualidade de conexão contém o valor estimado da qualidade de conexão medida entre eles. O menor valor de qualidade de conexão medido entre o nó vizinho e o sensor representa o nó com melhor qualidade. Decidiu-se adotar esta representação para tornar a representação da conexão mais fácil no peso dado aos grafos da estrutura de dados.

TABELA III. EXEMPLO DA TABELA DE VIZINHOS.

IDENTIFICAÇÃO DO NÓ VIZINHO	QUALIDADE DA CONEXÃO
5	10
8	50
9	12

O TinySDN implementa dois métodos que podem se complementar para medir a qualidade de conexão entre os nós: o *Link Estimator* [15], utilizando o componente do TinyOS; e o já citado RSSI, para plataformas que possuem o recurso. Para as plataformas específicas que são providas de RSSI, a medida da qualidade da conexão é mais precisa; por outro lado, a utilização do *Link Estimator* não depende da plataforma de hardware. Assim, o RSSI pode ser enviado junto ao valor do *Link Estimator* para aumentar a acurácia da visão da rede nos nós controladores SDN.

É interessante notar que o RSSI é uma medida de intensidade de sinal recebida pelo rádio, ou seja, é uma medida de qualidade de conexão do ponto de vista do receptor e não do emissor, ao contrário do *Link Estimator*, que fornece a qualidade de conexão do ponto de vista do emissor. Estas informações devem ser consideradas pelo controlador na hora de coletar as informações de topologia da rede e na representação nas estruturas de dados como, por exemplo, grafos direcionados.

IV. IMPLEMENTAÇÃO E EXPERIMENTOS

Uma implementação completa do TinySDN foi produzida utilizando a linguagem de programação nesC [16], linguagem adotada no arcabouço de programação do TinyOS. Os testes de validação da implementação foram executados na plataforma de RSSF TelosB [17]. O nó controlador SDN foi implementado como software independente, também implementado em nesC, especificamente para fornecer uma prova de conceito e experimentação.

Os experimentos para obtenção de métricas de desempenho foram executados no simulador COOJA [18], um simulador de redes de sensores sem fio que adota uma abordagem híbrida, capaz de simular o nível de rede e o nível de sistema operacional com base no programa nativo do nó sensor, integrando o comportamento do nó com o ambiente de simulação de rede utilizando a *Java Native Interface*. Desta forma, o simulador permite o uso do código compilado para a plataforma (ou seja, exatamente o mesmo código binário a ser

carregado no dispositivo físico) emulando a execução de um sensor TelosB real no emulador TI MSP430.

Dado que o COOJA executa programas compilados para a plataforma real, é possível emular tanto programas feitos para Contiki OS [19] quanto aplicações TinyOS. Assim, foi adotado este ambiente por conta da facilidade de montagem de topologias e isolamento de interferências, e pelo uso de emulação, que permite o teste das funções do sistema em um recurso similar a um nó sensor real.

O principal objetivo é medir e comparar o tempo que os nós geradores de pacotes levam para estabelecer os fluxos de dados com o nó sorvedouro e entregar seu primeiro pacote utilizando a arquitetura TinySDN. Isso, tanto para uma rede de sensores sem fio definida por software com um único nó controlador SDN quanto para dois controladores, aferindo assim a eficiência da utilização de múltiplos controladores. Como as topologias testadas são pequenas, estas não requerem alto poder de processamento ou alta capacidade de memória para serem processadas, então em nossos cenários de testes os nós controladores SDN também foram instalados em nós sensores TelosB (emulados).

Devido ao fato de que os nós sensores levaram 4,166 segundos para inicializarem a partir do início das simulações, subtraímos este valor de todas as medidas de tempo (com exceção da latência para entrega de pacotes através de fluxos já estabelecidos ou via CTP).

A. Descrição dos Cenários de Testes

Nos experimentos foram considerados três cenários diferentes com sete sensores com o TinySDN instalado (ou seja, sete nós sensores SDN), sendo que dois deles são nós geradores de pacotes, quatro deles apenas executam tarefas de encaminhamento e um nó funciona como nó sorvedouro. O número de nós controladores SDN é variável de acordo com o cenário proposto. Os cenários *a* e *b* representam uma rede de sensores sem fio definida por software com um único controlador e o cenário *c* representa o caso de múltiplos controladores. Estes cenários de testes estão representados na Fig. 2, em que o nó de identificação 4 é o sorvedouro, os nós de identificações 1 e 7 são os geradores de pacotes e os demais nós sensores SDN apenas executam encaminhamento.

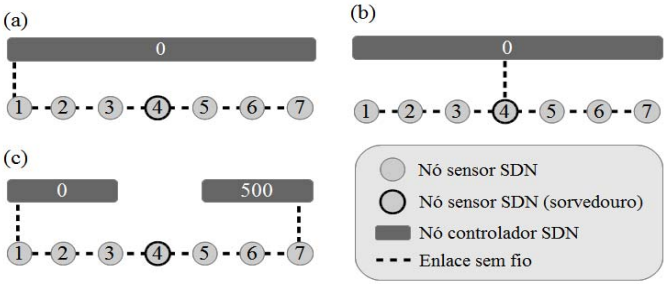


Figura 2. Cenários de testes.

No cenário *a* há um único nó controlador SDN, diretamente ligado ao nó 1. Assim, embora ambos os nós de remetentes sejam equidistantes ao nó sorvedouro, espera-se que o nó 1 entregue sua primeira mensagem antes nó 7, uma vez que está mais próximo do controlador e, portanto, tende a receber especificações de fluxo antes.

O cenário *b* é muito semelhante ao cenário *a*, sendo a única diferença os nós geradores de pacotes serem equidistantes ao nó controlador SDN, que está conectado ao nó 4 (o nó sorvedouro). Desta forma, espera-se que esta alteração equilibre o tempo de estabelecimento dos fluxos para os nós geradores de pacotes e, consequentemente, o tempo para a entrega das primeiras mensagens.

No cenário *c* há dois controladores de SDN (nó 0 e nó 500) que, juntos, cobrem todos os sete nós sensores SDN, dividindo a rede em dois subdomínios. Durante o processo de busca por controlador, os nós 1, 2 e 3 foram atribuídos ao nó controlador SDN de identificação 0, enquanto os nós 4, 5, 6 e 7 foram atribuídos ao nó controlador SDN 500. Vale a pena notar que o nó 4 poderia tanto associar-se ao nó controlador SDN 0 quanto ao nó 500, mas o resultado do CTP indicou que era menos custoso comunicar-se com o nó raiz 500. Este cenário representa um possível cenário com múltiplos controladores.

Em relação ao consumo de memória, foram utilizadas as ferramentas *MSP430-size* e *MSP430-ram-usage* [20] para estimar o uso de memória flash programável (ROM) e RAM, respectivamente. Assim, foi possível comparar o uso de memória para a mesma aplicação utilizando o TinySDN e um protocolo tradicional, no caso o CTP.

B. Resultados

A Fig. 3 contém um gráfico que mostra as diferenças de tempo para a associação dos nós geradores de pacotes a um nó controlador SDN, incluindo o tempo da especificação reativa de fluxo de controle para o controlador nos nós sensores SDN.

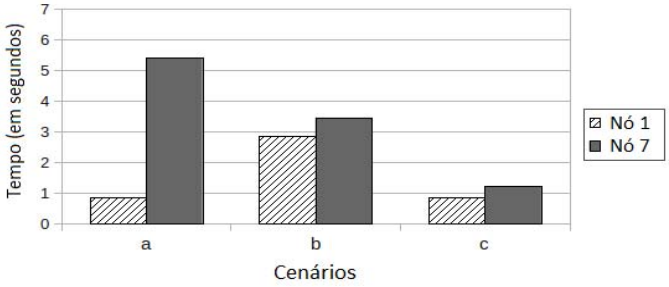


Figura 3. Tempo para associação ao nó controlador SDN.

No cenário *a*, o nó 1 levou menos de 1 segundo para associar-se ao controlador, enquanto o nó 7 levou mais de 5 segundos. Como esperado, o cenário *b* apresentou um resultado mais equilibrado, o nó 1 nó levou pouco menos de 3 segundos e o nó 4 cerca de 3,5 segundos. No entanto, como pode ser observado na Tabela IV, ambos os cenários de único controlador SDN apresentam aproximadamente a mesma média para a associação ao controlador, um pouco menos de 3,3 segundos.

No cenário *b*, teoricamente os nós 1 e 7 deveriam completar a associação ao controlador simultaneamente. No entanto, na simulação no COOJA observou-se que o nó 1 foi inicializado antes que o nó 7. Assim, o nó 1 iniciou o envio mensagens antes, obtendo vantagem no tempo. Isso também

ocorreu nos resultados para o cenário *c*.

TABELA IV. TEMPO MÉDIO PARA ASSOCIAÇÃO A UM NÓ CONTROLADOR SDN E PARA ENTREGA DO PRIMEIRO PACOTE DE DADOS (EM SEGUNDOS).

Evento	Cenário		
	A	B	C
Associação	3,26	3,25	1,07
Primeiro pacote	4,17	3,5	2,36

De acordo com os resultados apresentados na Fig. 3, o tempo para associação ao controlador no cenário de múltiplos controladores levou cerca de 1 segundo para ocorrer, ou seja, menos de metade do tempo se comparado aos cenários com um único controlador SDN, o que é devido à distribuição dos controladores.

A Fig. 4 apresenta tempos para a entrega dos primeiros pacotes entregues a partir de ambos os nós do remetente em todos os cenários. Os resultados são semelhantes para o tempo para associação ao controlador. Para o cenário *a*, o nó 1 é quase três vezes mais rápido do que nó 7, como esperado, uma vez que realizou a associação ao controlador antes e está mais próximo do nó controlador SDN. Esta tendência também ocorre nos cenários *b* e *c*.

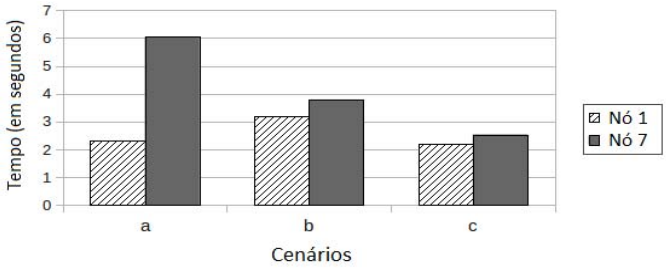


Figura 4. Tempo para entrega do primeiro pacote de dados.

A fim de comparar os cenários propostos de redes de sensores sem fio definidas por software com uma RSSF tradicional, foram medidos os tempos de entrega dos primeiros pacotes de solicitação de associação para nós controladores SDN (ou seja, o primeiro pacote entregue via CTP). Esses tempos são apresentados na Tabela V.

TABELA V. TEMPO PARA A RECEPÇÃO DO PRIMEIRO PACOTE CTP POR CADA NÓ CONTROLADOR SDN (EM SEGUNDOS).

Identificação do nó controlador	Cenário		
	A	B	C
0	0,84	0,89	0,84
500	-	-	1,21

Considerando a média destes valores (0,97 segundos), o cenário *b*, cenário com um único nó controlador de SDN que apresentou o melhor desempenho, leva em média cerca de 3,5 vezes o tempo do CTP para entregar o primeiro pacote. O cenário *c*, que contém múltiplos controladores, apresenta desempenho melhor que o cenário *b*, leva cerca de 2,4 vezes do tempo do CTP para a mesma tarefa. Esta desvantagem em relação ao CTP é esperada, uma vez que o TinySDN utiliza o próprio CTP para a busca de controladores.

Também foi medida a latência para entrega de pacotes para o sorvedouro através de fluxos TinySDN já estabelecidos e via CTP. Tanto o CTP quanto o TinySDN para os três cenários testados apresentaram latência média de cerca de 33 milissegundos, o que indica que o processo do TinySDN para classificação dos pacotes não introduz um atraso considerável quando comparada com um protocolo tradicional, uma vez que o maior atraso está concentrado na inicialização de rede.

Em relação à ocupação de memória pelo TinySDN, a Tabela VI contém os resultados estimados. Observou-se que a aplicação de teste utilizando o TinySDN requer 52,89% mais RAM e 10,53% mais ROM em comparação à aplicação de teste utilizando apenas o CTP. O aplicativo de teste utilizando o TinySDN requereu (no total, incluindo componentes do TinyOS) 31,68% de RAM e 51,96% de ROM disponível no dispositivo utilizado para os testes, que possui 48 KBytes de ROM e 10 KBytes de RAM. Desta forma, resta memória considerável para comportar outras possíveis aplicações de RSSF em execução nos nós sensores. Portanto, a alocação de memória introduzida pelo TinySDN não degrada ou impede o uso dos recursos dos dispositivos pelas aplicações de RSSF, provendo simultaneamente a flexibilidade na comunicação inerente ao paradigma SDN. Ainda assim é possível reduzir o consumo de memória do CTP instanciado no TinySDN, otimizando a integração entre suas implementações.

TABELA VI. OCUPAÇÃO DE MEMÓRIA DE UM PROTOCOLO TRADIOCIONAL DE RSSF E DO TINYSDN (EM BYTES).

Protocolo	RAM	ROM
CTP	2072	22562
TinySDN	3168	24940

V. CONSIDERAÇÕES FINAIS

Neste artigo foi apresentado o TinySDN, uma arquitetura de múltiplos controladores para redes de sensores definidas por software baseada no sistema operacional TinyOS, junto com sua implementação independente de plataforma de hardware. O TinySDN possibilita a obtenção de flexibilidade na comunicação fornecida pelo paradigma SDN, enquanto que a alocação de memória introduzida não compromete a utilização de outros recursos dos dispositivos relacionados a aplicações de RSSF.

Os resultados dos experimentos indicam que, comparado ao protocolo CTP, o TinySDN não introduz atraso considerável na tarefa de encaminhamento após estabelecimento dos fluxos de dados. Os atrasos observados nos cenários experimentais afetam apenas os primeiros pacotes gerados, que precisam aguardar na fila de saída do nó sensor até que os fluxos de dados sejam estabelecidos reativamente, o que pode ser melhorado com o estabelecimento proativo dos fluxos de dados.

Considerando a importância do consumo de energia, pretendemos medir e modelar o consumo de energia do

TinySDN em trabalhos futuros, bem como estender os cenários experimentais para incluir mobilidade e integração com diferentes tipos de redes. Além disso, planejamos adicionar mais tipos de ações relacionadas aos fluxos de dados no TinySDN, como as ações de plano de dados propostas no artigo do SDWN [1].

Em relação às aplicações SDN, estamos trabalhando atualmente em uma biblioteca para facilitar o desenvolvimento de novos aplicativos de plano de controle para os nós controladores do TinySDN.

AGRADECIMENTOS

Os autores gostariam de agradecer ao Laboratório de Arquitetura e Redes de Computadores da Universidade de São Paulo (LARC), por prover toda a infraestrutura necessária para o desenvolvimento deste trabalho. Este trabalho é financiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processos FAPESP números 2013/15417-4 e 2014/06479-9.

REFERÊNCIAS

[1] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling sdns," in Proceedings of the 2012 European Workshop on Software Defined Networking, ser. EWSDN'12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/EWSDN.2012.12

[2] D. Venmani, Y. Gourhant, L. Reynaud, P. Chemouil, and D. Zeghlache, "Substitution networks based on software defined networking," in Ad Hoc Networks, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, J. Zheng, N. Mitton, J. Li, and P. Lorenz, Eds. Springer Berlin Heidelberg, 2013, vol. 111, pp. 242–259. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36958-2 17

[3] M. A. S. Santos, B. T. de Oliveira, C. B. Margi, B. Nunes, T. Turletti, and K. Obraczka, "Software-defined networking based capacity sharing in hybrid networks," in Proceedings of Capacity Sharing Workshop (CSWS'13), In conjunction with ICNP'13, May 2013.

[4] M. Naphade, G. Banavar, C. Harrison, J. Paraszczak, and R. Morris, "Smarter cities and their innovation challenges," Computer, vol. 44, no. 6, pp. 32–39, June 2011.

[5] A. Mahmud and R. Rahmani, "Exploitation of openflow in wireless sensor networks," in Computer Science and Network Technology (ICCSNT), 2011 International Conference on, vol. 1, 2011, pp. 594–600.

[6] T. Luo, H.-P. Tan, and T. Q. S. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks." IEEE Communications Letters, vol. 16, no. 11, pp. 1896–1899, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/icl/icl16.html#LuoTQ12

[7] A. D. Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in Communications (QBSC), 2014 27th Biennial Symposium on, June 2014, pp. 71–75.

[8] A. Sabharwal, P. Schniter, D. Guo, D. W. Bliss, S. Rangarajan, and R. Wichman, "In-band full-duplex wireless: Challenges and opportunities," IEEE JSAC Special Issue on Full-duplex Wireless Networks, accepted., 2014. [Online]. Available: http://arxiv.org/abs/1311.0456

[9] S. Schmid and J. Suomela, "Exploiting locality in distributed sdn control," in Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 121–126. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491198

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," SIGPLAN Notices, vol. 35, no. 11, pp. 93–104, 2000.

[12] IEEE Standard, "IEEE 802.15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (WPANs)," 2006.

[13] B. T. OLIVEIRA and C. B. MARGI, "Wsn-etesec: End-to-end security tool for wireless sensor networks," http://www.larc.usp.br/_cbmargi/wsnetesec, 2012.

[14] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, ser. SenSys '09. New York, NY, USA: ACM, 2009, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1644038.1644040

[15] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis, "Four bit wireless link estimation," in Proceedings of the Sixth Workshop on Hot Topics in Networks (HotNets VI), 2007.

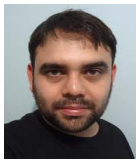
[16] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," in Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, ser. PLDI '03. New York, NY, USA: ACM, 2003, pp. 1–11. [Online]. Available: http://doi.acm.org/10.1145/781131.781133.

[17] MEMSIC, "Telosb datasheet," http://www.memsic.com/userfiles/files/DataSheets/WSN/telosb datasheet.pdf, 2011.

[18] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Crosslevel sensor network simulation with cooja," in Local Computer Networks, Proceedings 2006 31st IEEE Conference on, Nov 2006, pp. 641–648.

[19] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462.

[20] MSPGCC Development Team, "Gcc toolchain for msp430," http://sourceforge.net/projects/mspgcc/, 2013.



Bruno Trevizan de Oliveira é bacharel em sistemas de informação pela Escola de Artes, Ciências e Humanidades da Universidade de São Paulo (2009). É mestre em ciências pelo programa de pós-graduação em engenharia elétrica da Escola Politécnica da Universidade de São Paulo (2012) e atualmente é doutorando do mesmo programa de pós-graduação. Seus principais interesses de pesquisa incluem redes de sensores sem fio, Internet das coisas, redes definidas por software, criptografia aplicada e segurança de redes.

Lucas Batista Gabriel nasceu em Mogi das Cruzes-SP, Brasil (1992). Atualmente é aluno de graduação no curso de engenharia de computação na Universidade de São Paulo, Escola Politécnica (EPUSP), Brasil. Seus principais interesses de pesquisa incluem redes definidas por software, internet das coisas e sistemas embarcados.

Cíntia Borges Margi (membro do IEEE desde 2003 e membro *Senior* desde 2015) é professora associada do departamento de Engenharia de Computação e Sistemas Digitais (PCS) da Escola Politécnica da Universidade de São Paulo (EPUSP), onde atua desde junho/2010. É orientadora credenciada no programa de pós-graduação em Engenharia Elétrica da EPUSP na área de concentração Engenharia de Computação desde agosto/2010 para mestrado e dezembro/2012 para doutorado. Foi professora da Escola de Artes, Ciências e Humanidades da Universidade de São Paulo, curso de Sistemas de Informação, de fevereiro de 2007 a junho/2010. Possui doutorado em Engenharia de Computação pela University of California Santa Cruz (2006), mestrado em Engenharia Elétrica pela Universidade de São Paulo (2000) e graduação em Engenharia Elétrica Ênfase Computação e Sistemas Digitais pela Universidade de São Paulo (1997). Tem atuado na área de Arquitetura e Redes de Computadores, tendo como principal interesse redes de sensores sem fio (protocolos, sistemas, segurança, consumo e gerenciamento de energia, hardware) e redes definidas por software.