An Extended Software-Defined Approach for Reprogramming Low-end IoT Devices

Luis Eduardo Lima and Valério Rosset

***Abstract*—The usage of the Software-Defined Sensor Networks (SD-WSNs) as support for applications of the Internet of Things (IoT) has been the focus of recent research to reduce the complexity of configuration and management. This simplification allows greater flexibility for the configuration of these networks in operating time. However, although more flexible than traditional approaches, the existing SD-WSN architectures found in the literature have limitations mainly with regard to the reprogramming of devices in operating time. In this article, we propose a novel approach called Extended Software-Defined Wireless Sensor Network (ESD-WSN), in which the concept of the SDNs is extended to the application layer allowing the reprogramming of low-end devices in operation time without the need for firmware update. The proposed architecture was validated by simulation and an experimental study using very constrained low-end devices. The simulation/experimental results showed the effectiveness, flexibility, and efficiency of the proposal.**

***Index Terms*—Software-Defined Networks, Network Architecture, Internet of Things.**

I. Introdução

O paradigma de Redes Definidas por Software (SDNs, *Software-Defined Networks*) possibilita a redução da complexidade das tarefas de reconfiguração e gerenciamento em redes de computadores. Nesse contexto, o uso de SDNs permite mudanças de configuração ou introdução de novas funcionalidades em dispositivos de rede *on the Fly* (i.e. em tempo de operação da rede) sem necessidade de interrupção dos serviços da rede [1]. Essa simplificação se deve à separação das funções relacionadas ao controle lógico da rede e ao encaminhamento de pacotes realizados neste contexto, respectivamente, pelos controladores e *switches* de rede [2].

O interesse em aplicar o paradigma de SDNs ao contexto das Redes de Sensores Sem fio (WSN, *Wireless Sensor Networks*) levou a uma nova classificação de soluções de redes, chamada de Redes de Sensores Definidas por Software (SD-WSN, *Software-Defined Wireless Sensor Network*) [3], [4]. Neste contexto, destacam-se as arquiteturas propostas em [5], [6], [7] e [1]. Nesses estudos o protocolo *OpenFlow* [8] é estendido para permitir a disseminação e instalação de regras de fluxo de dados, que são criadas baseadas nas informações coletadas pelos nós sensores.

Desse modo é possível priorizar fluxos importantes ou eliminar dados irrelevantes para o propósito da aplicação. Este recurso é interessante porque permite a reconfiguração da rede para atender os requisitos das aplicações *on the fly* sem necessidade de intervenção humana ou atualização de *firmware* dos nós sensores.

No entanto, uma vez que as abordagens presentes na literatura permitem apenas a configuração de regras de fluxo de pacotes, elas ficam restritas a um conjunto de aplicações homogêneos, ou seja, aquelas que compartilham os mesmos requisitos de coleta de dados na camada de aplicação. Em contrapartida, por razões de otimização de custos em ambientes urbanos, uma SD-WSN pode servir a uma variedade de aplicações heterogêneas de Internet das Coisas (IoT, *Internet of Things*) com diferentes requisitos que podem ser habilitados ou instalados *on the fly* [9]. Assim, permitir a reconfiguração da camada de aplicação dos nós sensores para atender aos requisitos de aplicações heterogêneas em tempo de execução e sem a necessidade de atualização do *firmware* desses dispositivos é um desafio quando se considera sua implementação em dispositivos de baixo custo (*low-end devices*). Embora a existência de abordagens de reprogramação baseada em *scripts* para dispositivos IoT de baixo custo (e.g., [10]), sua utilização não é possível em dispositivos com capacidade de memória de acesso aleatório (RAM) muito restrita (ou seja, com menos de 10 kB de RAM) [11].

Diante disso, neste artigo apresentamos uma solução baseada na extensão de SDNs para a camada de aplicação de dispositivos IoT de baixo custo muito restritos para que possam ser reconfigurados por controladores SDN, chamada de *Extended Software-Defined Wireless Sensor Network (ESD-WSN)*. Desta forma, propomos ir além do controle tradicional dos fluxos de pacotes, utilizando regras de controle para configurar outros parâmetros que permitam redefinir as operações da camada de aplicação dos nós sensores. Além da tabela de fluxo, apresentamos uma tabela para armazenar o agendamento de atividades e outra para armazenar as ações dos nós sensores e atuadores.

O restante deste artigo está organizado conforme segue. A Seção II apresenta a revisão da literatura. Na Seção III descrevemos a arquitetura proposta. A validação da arquitetura ESD-WSN proposta, descrevendo os cenários simulados, modelagem, métricas e os resultados obtidos é apresentada na Seção IV. Detalhes sobre o protótipo desenvolvido e a avaliação experimental da arquitetura ESD-WSN são apresentados na Seção V. Finalmente, na Seção VI apresentamos as conclusões e direções futuras.

Luis Eduardo Lima is with Science and Technology Department, Federal University of São Paulo, Brazil e-mail:llima@unifesp.br.

Valério Rosset is with Science and Technology Department, Federal University of São Paulo, Brazil e-mail:vrosset@unifesp.br.

Manuscript received date; revised date

TABELA I
ANÁLISE COMPARATIVA DA LITERATURA RELACIONADA A SD-WSNS

Arquitetura	Regras de Corresp.	Operadores	Ciclo de Trabalho	Implementação	Particularidades
SDWN [1]	3	= e $\neq$	sim	não	Suporte a agregação
SD-WSN [7]	2	=, $\neq$, $>$ e $<$	não	não	Compatível com OpenFlow
SDN-WISE [6], [12]	3	=, $\neq$, $>$, $>=$, $<$ e $<=$	sim	*testbed* simulação	Abordagem stateful
TinySDN [5]	nd	nd	não	*testbed* simulação	Múltiplos controladores
IT-SDN [13]	nd	nd	não	*testbed* simulação	OpenSource ContikiOS
ESD-WSN	2	=, $\neq$, $>$, $>=$, $<$ e $>=$	sim	*testbed* simulação	OpenSource Arduino

*nd utilizado para *não definido*;

II. TRABALHOS RELACIONADOS

Dentre a literatura relacionada, consideramos as principais abordagens nas quais os conceitos fundamentais de SDNs são aplicados a WSNs, nomeadamente, [1], [7], [6], [12], [5] e [13]. Nesta seção descrevemos brevemente cada uma dessas abordagens.

Em [1] os autores propõem um modelo conceitual para a aplicação de SDNs em WSNs. Na abordagem proposta uma extensão foi introduzida pela adição da possibilidade de definição de regras de fluxo baseadas em dados de carga útil além das informações presentes no cabeçalho dos pacotes. Com esta abordagem foi possível controlar o fluxo de acordo com os dados coletados pela aplicação dos nós sensores, permitindo descartar pacotes duplicados, agregar dados e aumentar a prioridade para dados mais importantes.

O estudo apresentado em [7], por sua vez, propõe uma extensão para o protocolo *OpenFlow* [8]. Os autores propõem uma nova arquitetura definindo um protocolo padrão de comunicação entre os planos de dados e controle chamado *Sensor OpenFlow* (SOF). A ideia é fazer o plano de dados programável pela manipulação das tabelas de fluxo em cada sensor utilizando o protocolo SOF. Similar a [1], os autores propuseram manter a compatibilidade com o protocolo *OpenFlow* por meio da funcionalidade *OpenFLow eXtensible Match* (OXM).

A arquitetura proposta em [6], chamada de *Software Defined Networking for WIreless Networks* (SDN-WISE), implementa o conceito de [1] e [7], e apresenta resultados experimentais. Os autores propõem uma abstração de memória de estados (*stateful memory*), baseado nos conceitos definidos em [14]. Isso habilita os nós serem programados como uma máquina de estado finito permitindo a execução de tarefas baseadas em estados e excluindo a necessidade de comunicação com o controlador da rede. A arquitetura SDN-WISE foi validada no ambiente de simulação OMNeT++ [15] e em nós reais. Nas simulações os autores avaliaram essencialmente dois cenários: *i)* a eficiência do roteamento considerando o intervalo de tempo de envio dos pacotes de descoberta e *ii)* a eficiência do roteamento considerando o tempo de vida das entradas nas tabelas de fluxo. Os resultados indicaram que os intervalos mais longos entre envios de pacotes de descoberta e o tempo de vida mais longo das entradas das tabelas de fluxo implicam em maior eficiência de roteamento. Uma versão estendida do SDN-WISE com suporte para virtualização de funções de rede (NFV, *Network Function Virtualization*) foi apresentada em [12].

A TinySDN [5] é uma arquitetura SD-WSN desenvolvida para nós compatíveis com o sistema operacional TinyOS [16]. A TinySDN é baseada nos mesmos princípios introduzidos por [1] e [7] em que cada nó faz uso de duas tabelas de fluxo distintas, uma para o registro de regras para o tráfego de controle da rede e outra para o tráfego de dados da aplicação. A arquitetura proposta foi validada experimentalmente por meio de implementação em motes TelosB [17] e simulação em COOJA [18]. Os resultados mostraram que o TinySDN implementado precisa de uma alocação de memória um pouco maior do que as aplicações tradicionalmente implementadas com o mesmo propósito. A memória consumida pela implementação nos dispositivos TelosB foi respectivamente $31,68\%$ de RAM e $51,96\%$ de ROM, não caracterizando uma degradação dos recursos de memória disponíveis.

O IT-SDN, proposto em [13], é uma arquitetura inspirada no TinySDN incluindo melhorias de protocolo e implementação. O IT-SDN apresenta uma separação clara entre os três componentes principais, o Southbound Protocol (SB), o Neighbour Discovery Protocol (ND) e o Controller Discovery Protocol (CD). Isso permite que diferentes abordagens para esses protocolos sejam avaliadas. Para que cada protocolo seja substituído, as interfaces de protocolo devem ser bem definidas. Como exemplo, os autores citam que o protocolo ND deve conter duas funções, uma para iniciar o protocolo e outra para disparar ao receber um pacote.

Ao contrário do TinySDN, a arquitetura foi implementada no sistema operacional Contiki [19] que suporta vários modelos de dispositivos. A arquitetura foi testada nos dispositivos TelosB [17] e SensorTag [20] e modelada no simulador COOJA. Em [21], os autores avaliaram o desempenho do IT-SDN considerando a taxa de entrega de ponta a ponta, atraso na entrega e *overhead* de controle. Os cenários foram modelados no simulador COOJA considerando 16 a 81 nós transmitindo um pacote por minuto para o *Sink*. A taxa de entrega observada foi próxima a 100% para cenários entre 16 e 36 nós, caindo para cerca de 70% para 81 nós. O atraso na entrega dos pacotes apresentou um crescimento proporcional ao número de pacotes, chegando a pouco mais de 1s para o cenário com 81 nós. O atraso na entrega dos pacotes de controle, por sua vez, permaneceu estável e próximo a 1s em todos os cenários. A sobrecarga de controle, definida como o número de pacotes de controle enviados por nó por minuto, variou de $1,5$ a 2 pacotes por minuto.

A. Análise Comparativa

Para avaliar as arquiteturas encontradas na literatura, apresentamos nesta seção um comparativo de suas principais características. A Tabela I resume os resultados da análise comparativa no que diz respeito à estrutura da tabela de fluxo, como os limites de regras de correspondência, os operadores suportados, a operação de ciclo de trabalho, a implementação e as especificidades de cada abordagem.

Na tabela, a coluna *Regras de Correspondência* indica o número máximo de regras de correspondência simultâneas por entrada na tabela de fluxo. Dois são suficientes para definir faixas de valores, enquanto três podem garantir maior flexibilidade.

A coluna *Operadores* apresenta o conjunto de operadores lógicos correspondentes suportados por cada abordagem. A abordagem de [1] suporta apenas os operadores $=$ e $\neq$, enquanto [7] e [6] incluem os operadores $>$ e $<$. Além disso, [6] também suporta $>=$ e $<=$ adicionando mais flexibilidade ao intervalo de valores correspondentes.

A coluna *Ciclo de Trabalho* indica o suporte à configuração dessa característica por meio das ações disponíveis na tabela de fluxo, permitindo colocar nós em estado de consumo reduzido de energia. Apenas [1] e [6] apresentaram suporte para operação com *Ciclo de Trabalho*.

Pode-se notar que algumas arquiteturas não atingiram o estágio de *Implementação*. As abordagens [1] e [7] consistem nas primeiras proposições teóricas integrando o conceito SDN em WSN. Portanto, os trabalhos mais recentes têm tentado implementar seus conceitos em ambientes de simulação e *testbeds*. Dentre as *particularidades* de cada abordagem, destacamos as seguintes. O estudo do [1] apresenta suporte à agregação, porém, sem detalhes de implementação. O [7] fornece integração com Openflow através da funcionalidade *OpenFlow extensible match* (OXM). Em [6] a abordagem *stateful* traz um alto nível de otimização e autonomia para a operação da rede. O [5] adiciona suporte para interação de múltiplos controladores. Por fim, a proposta de [13] apresenta código-fonte aberto disponível na Internet, muito importante para reprodutibilidade.

Nesse artigo apresentamos uma abordagem diferente e inovadora que estende as funcionalidades da literatura em SD-WSN. A abordagem proposta possibilita a reprogramação remota da camada de aplicação de dispositivos IoT de baixo custo sem a necessidade de atualização de *firmware*.

III. A Arquitetura ESD-WSN

A arquitetura ESD-WSN visa o suporte à SD-WSN para dispositivos baseados em microcontroladores de baixo custo com RAM muito restrita e amplamente usados por desenvolvedores de aplicações de monitoramento no contexto de IoT, como a plataforma `Arduino` [22]. A ESD-WSN é uma solução *cross-layer* que estende o uso de tabelas de controle para programar a camada de aplicação dos dispositivos periféricos responsáveis pela coleta de dados. Desta forma, a aplicação alvo da rede é implementada por meio de um software que interpreta as regras descritas nas tabelas de controle dos nós sensores. O conteúdo dessas tabelas é definido pela aplicação de gerenciamento do controlador SDN, da mesma forma que as regras das tabelas de fluxo da rede.

A Fig. 1 apresenta a visão geral da organização lógica e a interação entre os componentes da arquitetura ESD-WSN. Na ESD-WSN, assumimos que o software do controlador SDN pode estar localizado na extremidade da rede (*Edge e Fog*) ou na nuvem (*Cloud*). O controlador SDN interage com uma aplicação de controle externa para produzir o conjunto de regras de configuração para os dispositivos finais. Também assumimos que o gateway é um dispositivo capaz de processar tarefas mais complexas do que os nós sensores. Nesse caso, o gateway é considerado capaz de: *i)* implementar um conjunto de protocolos de conectividade (e.g., MQTT, CoAP etc) para interoperabilidade com aplicações de IoT; *ii)* executar tarefas de segurança e otimização com base em técnicas de aprendizagem de máquina implementadas nos módulos de inteligência e aprendizado (que denominamos de módulos ML, *Machine Learning Modules*). Essas tarefas são consideradas essenciais para a próxima geração de IoT, onde adversidades podem ser tratadas de forma proativa e os dispositivos de IoT podem aprender e se adaptar dinamicamente a diferentes condições [23]; e *iii)* implementar um módulo *OF parser* para converter regras OpenFlow que podem ser criadas pelo controlador SDN para o formato adotado pela arquitetura proposta.

A operação da ESD-WSN consiste em quatro fases necessárias para a configuração e operação da rede. A primeira fase é chamada de *Descoberta de Controlador* e é responsável por descobrir as rotas de comunicação entre os nós sensores e o controlador. Na segunda fase, chamada *Relatório e Estabelecimento de Rotas de Controle*, os nós sensores enviam mensagens de relatório contendo informações de configuração e uma lista de nós vizinhos para o controlador SDN. As rotas para comunicação de controle são estabelecidas entre o controlador SDN e cada um dos nós sensores. Na terceira fase, denominada *Reprogramação Remota*, o controlador SDN envia para cada nó sensor as regras de fluxo e de programação que serão armazenadas nas tabelas de controle desses nós. Finalmente, em uma quarta fase, denominada *Transmissão de Dados*, os nós sensores passam a executar a programação definida nas tabelas de controle e iniciam a transmissão dos dados coletados. A Fig. 2 apresenta a organização interna dos componentes ESD-WSN nos nós sensores.

É importante notar que o tratamento das mensagens de configuração e operações das três primeiras fases são realizadas pelo Módulo de Controle de Rede. Por outro lado, as operações relacionadas aos fluxos de dados são realizadas por um Módulo de Processamento de Dados. Simultaneamente, a execução das tarefas agendadas é realizada por um Módulo Processador de Ações em conjunto com os módulos de Controle de Sensores e de Processamento de Dados.

O funcionamento da arquitetura, bem como a apresentação de um caso de uso utilizado como prova de conceito, são detalhados nas subseções a seguir.

A. Fase de Descoberta de Controlador

Nesta fase, os nós se comunicam entre si para determinar as rotas entre eles e o controlador SDN. Diversas abordagens na

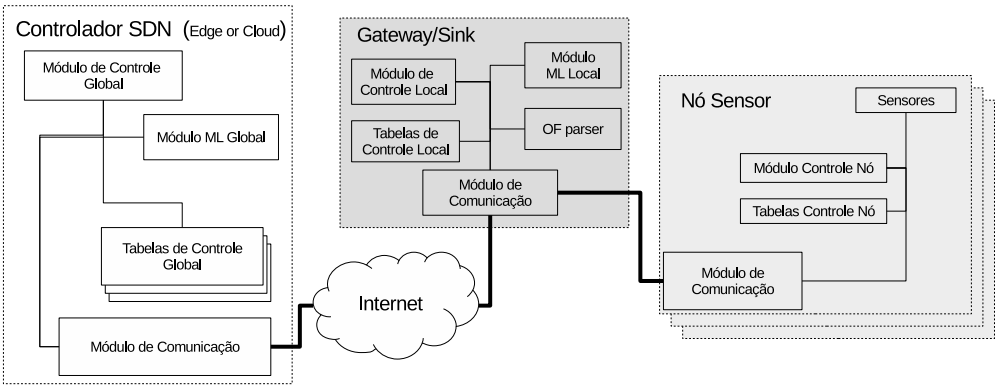


Fig. 1. Visão Geral da arquitetura ESD-WSN.

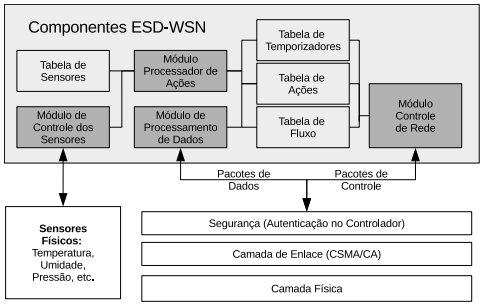


Fig. 2. Componentes da ESD-WSN nos nós sensores.

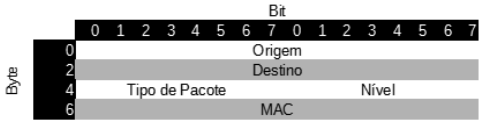


Fig. 3. Estrutura da mensagem CA.

Fig. 4. Estrutura da mensagem RPT. (a) campos comuns para todos RPTs, (b) específicos para anúncio de vizinhança, (c) específicos para anúncio de sensores e (d) específicos para estado de sensores.

literatura descrevem métodos que podem ser aplicados para descoberta de topologia de rede [24]. A arquitetura ESD-WSN é flexível o suficiente para implementar qualquer um dos métodos propostos. No entanto, como opção padrão para este estágio, optamos por utilizar um protocolo de roteamento baseado em gradiente (Gradient-Based Routing - GBR) definido em [25] e descrito a seguir.

O protocolo de descoberta de topologia baseado em GBR inicia com todos os nós sensores ativos no modo de recepção *rx*. O controlador então transmite uma mensagem de anúncio do controlador (CA) que carrega, no cabeçalho um campo denominado **nível** inicialmente definido como 1 (ver Fig. 3). Cada nó que recebe um pacote contendo uma mensagem CA com um valor de RSSI (received signal strength indication, indicador de intensidade do sinal recebido) maior do que um valor de limite mínimo (por exemplo, -98 dBm) é um salto possível em direção ao controlador que originou o CA. Assim, cada nó receptor: *i)* armazena o endereço do nó controlador, o valor do nível e o endereço do nó que retransmitiu a mensagem CA; e *ii)* retransmite a mensagem após adicionar uma unidade ao valor do nível recebido. As ações *i* e *ii* ocorrem somente se o valor do nível conhecido anteriormente pelo nó receptor for maior que o valor do nível recebido pela mensagem CA. Portanto, o endereço do nó que retransmitiu a mensagem CA que resulta no nível mais baixo é considerado o próximo salto em direção ao controlador. Em um primeiro momento, a fase de descoberta do controlador é executada por um período de tempo predefinido. No entanto, este período pode ser ajustado sob demanda durante a operação da rede.

B. Fase de Relatório e Estabelecimento de Rotas de Controle

Essa fase consiste no envio de mensagens de relatório (RPT) de cada nó sensor para o controlador SDN. A estrutura das mensagens RPT é ilustrada na Fig. 4.

As mensagens RPT podem transportar informações sobre o conjunto de vizinhos de cada nó sensor, a energia residual, o conjunto de sensores físicos disponíveis para aquisição de dados, a capacidade de memória residual, as coordenadas de posição etc. As mensagens RPT são transmitidas via unicast, por múltiplos saltos quando necessário, usando as informações de rota definidas na fase de descoberta executada inicialmente.

As mensagens RPT são necessárias para associar os nós e também para determinar as rotas, do controlador aos nós sensores, especificamente utilizadas para a transmissão de mensagens de controle. Para fazer isso, cada nó sensor retransmite as mensagens RPT para o próximo salto (conforme definido na fase de descoberta) em direção ao controlador SDN. Assim, cada nó sensor j no caminho entre um sensor i e o controlador, ao receber uma RPT originado em i, registra que está no caminho para o nó i e de qual nó recebeu o pacote de i. O controlador SDN, ao receber uma mensagem RPT, registra

o endereço da origem da mensagem (o endereço de i no caso) e o próximo salto em direção a ela (ou seja, o nó sensor que retransmitiu a RPT diretamente para o controlador).

Uma vez que a arquitetura é modularizada, o protocolo para descoberta de rotas de controle pode ser facilmente substituído por outro protocolo, se necessário. Ao final da segunda fase, cada nó terá enviado pelo menos uma mensagem RPT contendo informações da vizinhança, sensores disponíveis, memória disponível e energia residual. Assim, o controlador SDN possui todas as informações necessárias para associar e gerenciar os dispositivos de rede. Com essas informações, o software de controle pode determinar rotas eficientes, agendar a coleta de dados e outras operações dos nós sensores.

TABELA II
A ESTRUTURA DA TABELA DE TEMPORIZADORES.

Tamanho	Nome	Descrição
1 byte	timerID	identificação do temporizador
1 byte	actionID	Id da ação a ser acionada
2 byte	t0	tempo inicial
2 byte	countdown	intervalo entre execuções
1 byte	looping	número de repetições
1 bit	active	*flag* de ativação

TABELA III
A ESTRUTURA DA TABELA DE AÇÕES.

Tamanho	Nome	Descrição
1 byte	actionID	identificação da ação
1 byte	actionType	tipo da ação (e.g. drop, forward, broadcast, sleep, activate, deactivate etc)
1 bit	active	*flag* de ativação
2 byte	value1	valor dependente do actionType
2 byte	value2	valor dependente do actionType
2 byte	value3	valor dependente do actionType

TABELA IV
A ESTRUTURA DA TABELA DE FLUXO

Tamanho	Nome	Descrição
1 byte	flowID	identificação de Fluxo
1 byte	actionID	Id da ação a ser acionada
1 byte	timeout	tempo de expiração da regra
1 byte	counter	contador de correspondência
1 bit	active	*flag* de ativação
2 bytes	sourceAddress	endereço origem
2 bytes	sourceMask	máscara de endereço de origem
2 bytes	destinationAddress	endereço de destino
2 bytes	destinationMask	máscara de endereço de destino
1 byte	dataType	tamanho dos dados (8 ou 16 bits)
1 byte	operator	tipo operador lógico (>, <=, =, etc)
4 bytes	values	2 valores 16 bits ou a 4 de 8 bits
1 byte	offset	primeiro byte de dados para corresp.
1 byte	size	qtde de bytes de dados para corresp.

C. Fase de Reprogramação Remota

Nessa fase, o controlador transmite pacotes de controle contendo as regras a serem instaladas nas tabelas de controle locais dos nós sensores. A arquitetura proposta fornece três tipos de tabelas de controle em nós sensores: tabela de ações, tabela de temporizadores e tabela de fluxo. Combinadas, essas três tabelas permitem o controle remoto e a reprogramação da rede de sensores.

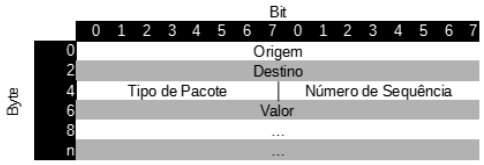


Fig. 5. Estrutura das mensagens de controle de reprogramação (PRG).

Fig. 6. Estrutura das mensagens de dados.

A tabela de temporizadores permite determinar um cronograma de atividades da aplicação, como a periodicidade das leituras dos sensores e os períodos de inatividade. A tabela de fluxo armazena as regras e filtros para o roteamento e controle do pacote de dados. A tabela de ações determina o que fazer quando um determinado fluxo de dados é detectado ou um temporizador é acionado. O formato de entrada especificado para cada uma das tabelas de controle é apresentado nas Tabelas II, III e IV.

Para configurar as tabelas de controle dos nós, o controlador SDN envia sequencialmente regras para os nós sensores por meio de mensagens de controle de reprogramação (PRG). A estrutura da mensagem PRG é apresentada na Fig. 5. A mensagem PRG possui como carga útil dados formatados conforme os campos das Tabelas II, III e IV. Ao final de um tempo predefinido, se nenhuma confirmação for recebida de um determinado nó, o controlador retransmite as mensagens PRG até receber uma confirmação (ou até atingir o máximo de tentativas).

A fase de programação remota tem uma duração determinada por um período de tempo predefinido. No entanto, essa duração pode ser reajustada em tempo de execução por meio das tabelas de temporizadores dos nós sensores.

D. Fase de Transmissão de Dados

Nessa fase, os nós sensores realizam as operações definidas nas tabelas de controle para coletar e transmitir dados. A Fig. 6 apresenta a estrutura das mensagens de dados.

E. Caso de uso para Coleta Oeriódica de Dados.

A Fig. 7 mostra um exemplo de configuração de tabelas de controle para o cenário de coleta de dados periódicos em uma rede formada por três nós sensores e um coletor central (*Sink*). Para um melhor entendimento, nesse exemplo utilizamos mnemônicos para representar o conteúdo das tabelas. Por exemplo, a primeira entrada na tabela Timer do Sink "*T1-Execute a ação A1 a cada 12 minutos*" corresponde a uma entrada conforme definido na Tabela II com os respectivos valores dos campos: timerID:=1, actionID:=1, t0:=0, contagem regressiva:=720 (segundos), looping:=255 (para sempre enquanto ativo) e ativo:=1.

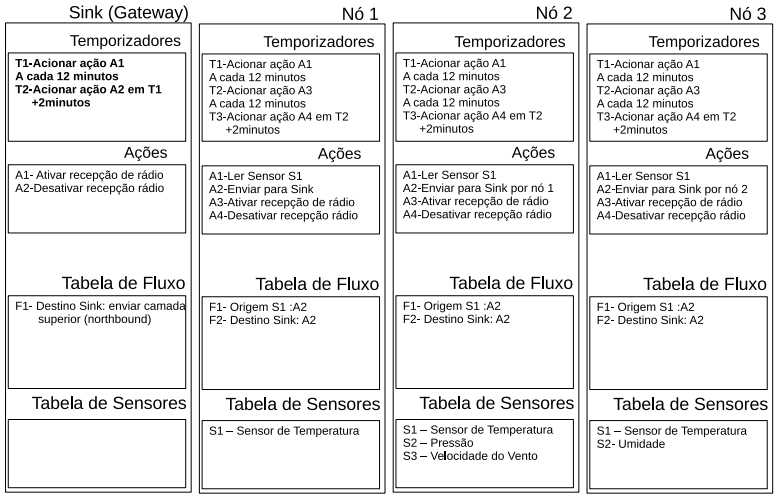


Fig. 7. Exemplo de configuração das tabelas considerando uma aplicação de coleta periódica de dados.

Esse cenário de exemplo consiste em uma topologia de rede em linha onde o *Sink* e os três nós sensores são organizados de forma que o *Sink* esteja dentro da faixa de comunicação do nó 1, o nó 2 está dentro da faixa do nó 1 e do nó 3, e o nó 3 apenas dentro da faixa de comunicação do nó 2. As entradas das tabelas de controle determinam que os módulos de transmissão de todos os nós são ativados a cada 12 minutos (temporizador T1 do *Sink* e T2 em nós sensores, ações A1 para o *Sink* e A3 para os nós sensores), dados de temperatura (S1) de todos os nós sensores são coletados (temporizador T1, ação A1) e retransmitidos para o *Sink* (fluxo F1 e F2, ação A2). Após 2 minutos os módulos de transmissão são desligados (temporizador T2 e T2, ações A2 e A4) e o ciclo se repete. Presume-se neste exemplo que o conteúdo das tabelas, exceto a tabela de sensores, foi preenchido na fase de programação pelo controlador SDN. As tabelas de sensores são configuradas antes da implantação, indicando todos os sensores físicos disponíveis instalados nos nós. As informações sobre os sensores físicos disponíveis são relatadas e usadas pelo controlador SDN para programar outras tabelas dos nós sensores.

IV. Validação

A. Simulação: Cenários, Parâmetros e Métricas

A arquitetura ESD-WSN proposta foi modelada em um ambiente de simulação Castalia [26]. Considerando uma topologia em grade, conforme ilustrado na Fig. 8, seis cenários foram definidos variando o número de nós em 16, 25, 36, 49, 64 e 81 (e identificados nos experimentos com um prefixo 'C' seguido do número de nós, e.g. C16, C25 etc). Esses cenários foram escolhidos porque são semelhantes aos cenários analisados em [21]. Na Fig. 8 os nós são representados por círculos identificados por números ou códigos. O código *S0* representa o nó *Sink* e o código *C4* representa o nó controlador. As setas representam os caminhos definidos pelo controlador para a coleta de dados.

Em todos os cenários simulados, os nós foram dispostos a 5 metros de distância um dos outros. Assim, de acordo com

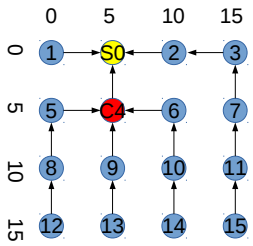


Fig. 8. Exemplo de um cenário utilizado na simulação composto por 16 nós.

o número de nós utilizados, as áreas dos cenários variaram de 225 a 1600 m^2. Os adaptadores RF foram configurados para reproduzir o consumo de energia do transceptor escolhido para o protótipo (*nRF24L01* [27]) com 2Mbps de taxa de transmissão e uma potência de saída de -7dBm para que nós a 5m de distância pudessem receber os dados com uma taxa de entrega aceitável e enquanto que os nós em distâncias maiores (e.g., na diagonal da grade) ficassem fora do alcance de comunicação. Por exemplo, no cenário representado na Fig. 8 o nó 6 pode transmitir diretamente aos nós 2, 7, 10 e $C4$, mas não aos nós $S0$, 3, 9 e 11 ou quaisquer outros nós mais distantes. Dessa forma, o protocolo de roteamento GBR foi utilizado para possibilitar a comunicação por múltiplos saltos entre os nós distantes.

A programação dos nós sensores é definida pelo controlador SDN para que possam transmitir os dados coletados para o Sink durante a fase de transmissão dos dados. Uma transmissão confiável de dados (na fase de programação) e a agendamento (na fase de transmissão de dados) são utilizadas para reduzir a perda de pacotes e evitar colisões, respectivamente. O controlador SDN também configura as rotas com múltiplos saltos entre os nós fora de alcance direto de comunicação com o *Sink*. Finalmente, o controlador configura os nós sensores para hibernar em intervalos entre os períodos de coleta de

dados.

Em cada simulação o período de execução da rede configurado foi de duzentos segundos, que é o tempo necessário para permitir o agendamento completo da rede e o envio de 2 mensagens de dados por cada nó sensor. A simulação de cada cenário foi repetida 40 vezes para se obter um número razoável de amostras utilizadas para a análise estatística dos resultados.

Para avaliação do desempenho três métricas foram analisadas: a taxa de entrega de pacote de dados, o atraso na entrega e a sobrecarga de controle [24], [28]. Essas métricas são explicadas em detalhes a seguir.

- **Taxa de entrega de pacote de dados**: é a razão entre o número total de pacotes gerados e o número total de pacotes entregues ao *Sink*.
- **Atraso de entrega**: considerou-se como atraso de entrega o período de tempo decorrido entre a transmissão do pacote pelo nó de origem e sua entrega ao *Sink*. O atraso na entrega é calculado como uma média de todos os atrasos de entrega observados individualmente.
- **Sobrecarga de controle** foi considerada como sendo a razão entre o número total de pacotes de controle transmitidos e o número total de pacotes transmitidos na rede.

B. Resultados Das Simulações

1) Taxa de Entrega: A Fig. 9a apresenta os valores da taxa de entrega de dados resultantes em cada cenário. Como esperado, a taxa de entrega permaneceu muito próximo de 1 na maioria das simulações devido ao esquema de escalonamento adotado na fase de transmissão de dados. Em alguns casos, perdas ocorreram devido a falhas de transmissão.

2) Atraso de Entrega: A Fig. 9b apresenta o resultado para o atraso de entrega em cada cenário simulado. Pode-se notar que o atraso aumenta linearmente. Esse aumento é esperado devido ao maior fluxo de mensagens geradas na medida que o número de nós aumenta na rede. Embora a média seja inferior a 40 ms, os maiores valores de atraso estão abaixo de 140 ms, portanto, compatíveis com aplicações de IoT com baixa sensibilidade à latência.

3) Sobrecarga de Controle: A sobrecarga de controle pode ser observada nas Fig. 9d-9f. Cada gráfico apresenta um resultado para a sobrecarga de controle gerada pela abordagem proposta (valores em cor preta) e um resultado gerado por um caso base em que não se considera a etapa de reprogramação remota de nós sensores (valores em cor azul). A Fig. 9c mostra que a sobrecarga de controle observada é muito alta devido à pequena taxa de transmissão de pacotes dados considerada nesse cenário (2 pacotes por nó). No entanto, a sobrecarga de controle diminui gradualmente para níveis próximos ao caso base na medida que a taxa de transmissão de dados aumenta. As Fig. 9d, 9e e 9f mostram essa tendência considerando cenários em que aumentamos as taxas de transmissão de dados em 20, 50 e 100 pacotes por nó, respectivamente. Isso se justifica porque, como o número de nós não á alterado, a quantidade de pacotes de controle não se altera significativamente.

4) Discussão: Os resultados obtidos confirmam a eficácia da arquitetura proposta. Porém, devido ao seu nível de abstração, o modelo de simulação não é adequado para estimar o uso de memória da implementação em hardware real de baixo custo. Desse modo, a seguir, apresentamos detalhes da implementação do protótipo do ESD-WSN e sua avaliação experimental de desempenho.

V. Protótipo ESD-WSN

Implementamos um *testbed* indoor da arquitetura ESD-WSN composta por um nó controlador principal (utilizando uma placa `Arduino Due`) e quatro nós sensores (utilizando placas `Arduino Uno` com 2 KB de RAM e 32 KB de memória Flash) equipado com módulos de rádio (`nRF24L01`). Nesta implementação, abstraímos as funcionalidades do Gateway e do Controlador SDN descritas na Seção III. Portanto, implementamos uma simplificação do controlador SDN de maneira replicar operações básicas previstas pela arquitetura. Além disso, implementamos de maneira completa as funcionalidades da camada de enlace (i.e., protocolo CSMA/CA), módulos de controle e tabelas de controle, conforme definido na especificação da ESD-WSN para os nós sensores.

Uma imagem do *testbed* é apresentada na Fig. 10. O protótipo dos nós sensores consumiu 14.558 bytes, sendo 45% do espaço de armazenamento total de 32Kb. As variáveis consumiram 855 bytes, sendo 41% dos 2.048 bytes disponíveis de RAM. Os códigos fontes da implementação estão disponíveis em: *github.com/LuisEduardoUnifesp*.

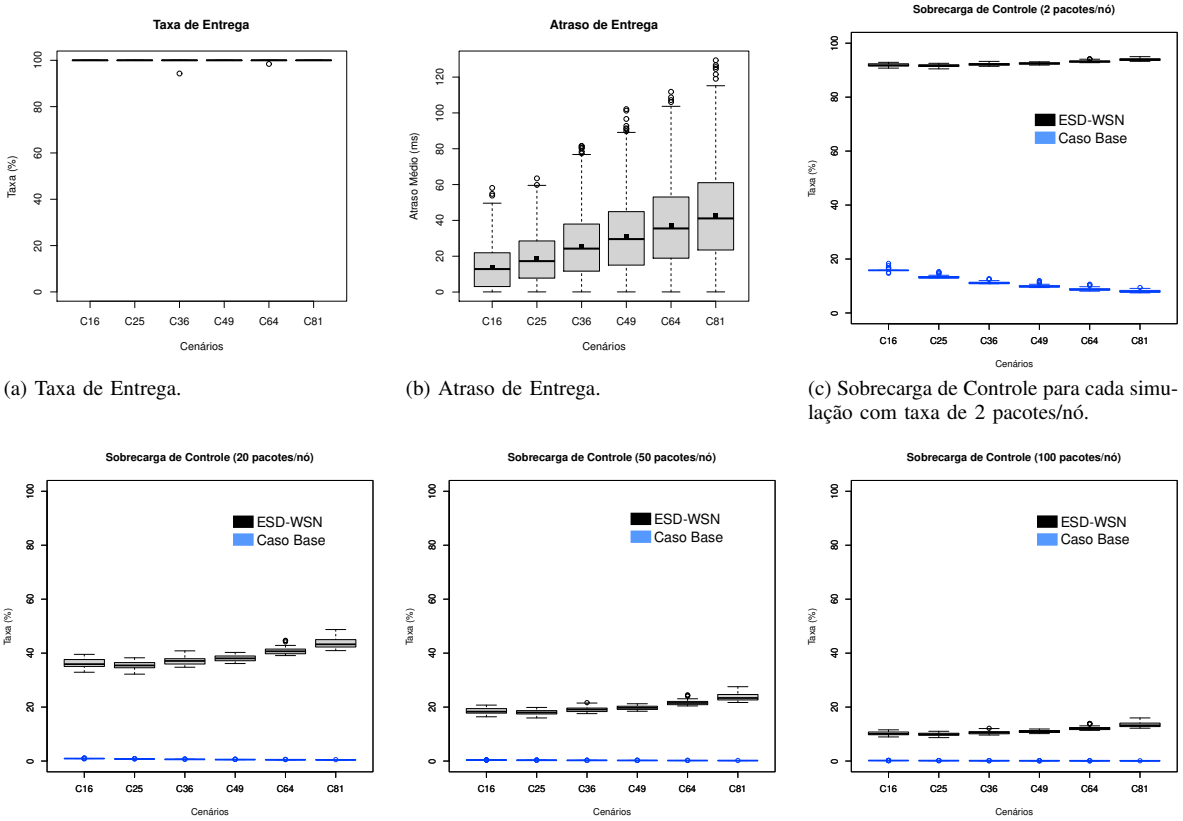
A. Avaliação Experimental de Desempenho

Para avaliar o desempenho da arquitetura ESD-WSN implementada, consideramos uma rede composta por 4 nós conectados em uma topologia em estrela, onde cada nó é conectado ao controlador/*gateway* da rede. Avaliamos o desempenho por meio das métricas descritas na Seção IV. A configuração e os parâmetros de rede são apresentados na Tabela V.

TABELA V
Parâmetros dos Experimentos

Parâmetros	Valor
Taxa de Transmissão (Mbps)	2
Tamanho de Pacote (bytes)	32
Número de Nós	4
Topologia	Estrela
Intervalo de Geração de Pacotes (s)	360, 180, 60, 30, 1
Duração Total Experimento (s)	600
Duração Fase de Prog. Remota (s)	180
Duração Fase Trans. de Dados (s)	420
Execuções	10
Tipo de Aplicação	Coleta Periódica

No conjunto de experimentos, os nós implementaram uma aplicação de coleta periódica de dados em que se transmitem mensagens de dados em intervalos distintos de 1, 30, 60, 180 e 360 segundos, configurando-se um cenário específico para cada uma dessas taxas de geração de pacotes. Além disso, para cada cenário, a execução foi repetida 10 vezes. A Fig. 11 apresenta os resultados observados para, respectivamente, taxa de entrega, atraso e sobrecarga de controle incluindo os resultados para o caso base sem a fase de reprogramação remota de sensores (valores em cor azul).



(a) Taxa de Entrega.

(b) Atraso de Entrega.

(c) Sobrecarga de Controle para cada simulação com taxa de 2 pacotes/nó.

(d) Sobrecarga de Controle para cada simulação com taxa de 20 pacotes/nó.

(e) Sobrecarga de Controle para cada simulação com taxa de 50 pacotes/nó.

(f) Sobrecarga de Controle para cada simulação com taxa de 100 pacotes/nó.

Fig. 9. Resultados das Simulações. Nos gráficos (c) a (f) os valores na cor preta correspondem aos resultados da proposta e os valores em cor azul correspondem ao caso base (sem reprogramação remota).

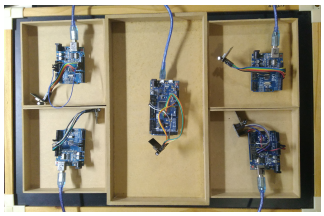


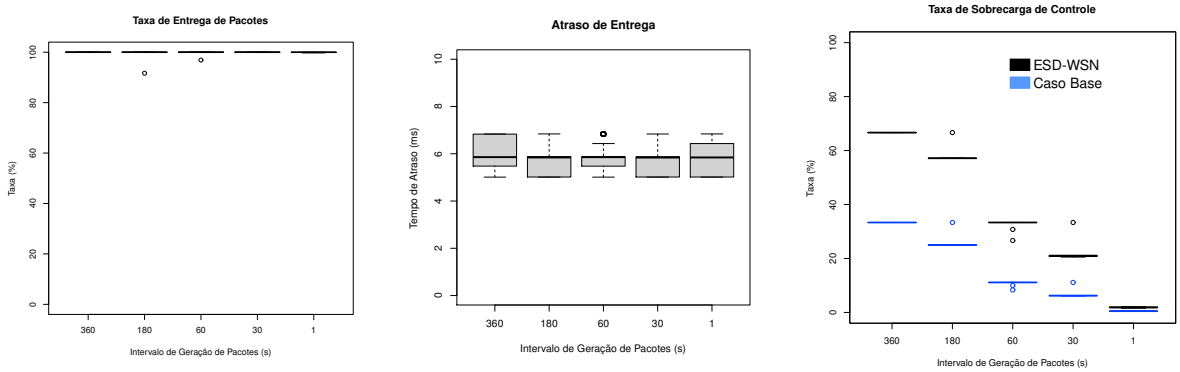
Fig. 10. *Testbed* da arquitetura ESD-WSN.

1) Discussão:: analisando os resultados apresentados na Fig. 11 pode-se notar a estabilidade do protótipo com relação à taxa de entrega e o atraso na entrega. Os resultados são esperados, uma vez que o *testbed* é composto de poucos nós e dispostos muito próximos uns dos outros. No entanto, é importante mencionar que como os nós não possuem relógios sincronizados e, devido às limitações de processamento dos dispositivos `Arduino`, a latência da arquitetura pode aumentar substancialmente com o aumento do número de nós. Já a sobrecarga de controle diminui com aumento do número de pacotes de dados (ocasionado pela diminuição do intervalo de geração de pacotes) e se aproxima dos valores apresentados pelo caso base. Esse comportamento é esperado uma vez que: *i*) quantidade de nós é fixa e por isso número de mensagens de controle varia muito pouco; e *ii*) devido a baixa taxa de erros nas transmissões.

VI. Considerações Finais

Neste artigo propusemos uma arquitetura estendida que permite o gerenciamento para além das camadas de rede e enlace de dispositivos IoT de baixo custo, proporcionando maior flexibilidade de programação nas camadas de aplicação: chamada de Extended Software-Defined Wireless Sensor Network (ESD-WSN).

Desenvolvemos um protótipo e implantamos um ambiente de teste *indoor* utilizando hardware de baixo custo com capacidades de armazenamento e processamento muito restritas. Validamos a proposta por meio de simulação e uma análise experimental, e apresentamos os resultados que indicaram a eficácia de operação, flexibilidade e eficiência da arquitetura proposta.As próximas etapas desta pesquisa incluem o desenvolvimento do módulo *Parser OF*, módulos de ML e uma expansão do *testbed* com implantação de dispositivos indoor/outdoor em campo.



(a) Taxa de Entrega. (b) Atraso de Entrega. (c) Sobrecarga de Controle.

Fig. 11. Resultados Experimentais.

REFERÊNCIAS

[1] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling sdns," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, pp. 1–6, IEEE, 2012.

[2] 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, pp. 594–600, IEEE, 2011.

[3] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A survey on software-defined wireless sensor networks: Challenges and design requirements," *IEEE Access*, vol. 5, pp. 1872–1899, 2017.

[4] J. Kipongo, E. Esenegho, and T. G. Swart, "Efficient topology discovery protocol using it-sdn for software-defined wireless sensor network," *Bulletin of Electrical Engineering and Informatics*, vol. 11, no. 1, 2022.

[5] B. T. De Oliveira, L. B. Gabriel, and C. B. Margi, "Tinysdn: Enabling multiple controllers for software-defined wireless sensor networks," *IEEE Latin America Transactions*, vol. 13, no. 11, pp. 3690–3696, 2015.

[6] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pp. 513–521, IEEE, 2015.

[7] T. Luo, H.-P. Tan, and T. Q. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1896–1899, 2012.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[9] L. E. Lima, B. Y. L. Kimura, and V. Rosset, "Experimental environments for the internet of things: A review," *IEEE Sensors Journal*, vol. 19, no. 9, pp. 3203–3211, 2019.

[10] E. Baccelli, J. Doerr, S. Kikuchi, F. A. Padilla, K. Schleiser, and I. Thomas, "Scripting over-the-air: Towards containers on low-end devices in the internet of things," in *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 504–507, IEEE, 2018.

[11] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, "A review of low-end, middle-end, and high-end iot devices," *IEEE Access*, vol. 6, pp. 70528–70554, 2018.

[12] A.-C. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Sd-wise: A software-defined wireless sensor network," *Computer Networks*, vol. 159, pp. 84 – 95, 2019.

[13] R. C. Alves, D. A. Oliveira, G. Nez, and C. B. Margi, "It-sdn: Improved architecture for sdwsn," in *XXXV Brazilian Symposium on Computer Networks and Distributed Systems*, 2017.

[14] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "Openstate: Programming platform-independent stateful openflow applications inside the switch," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 44–51, Apr. 2014.

[15] A. Varga, "Discrete event simulation system," in *Proc. of the European Simulation Multiconference (ESM'2001)*, pp. 1–7, 2001.

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

[17] I. Memsic, "Telosb," in *Telosb Mote Platform datasheet*, 2022.

[18] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Local computer networks, proceedings 2006 31st IEEE conference on*, pp. 641–648, IEEE, 2006.

[19] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pp. 455–462, IEEE, 2004.

[20] T. Instruments, "Simplelink sensortag," in *https://www.ti.com/tool/TIDC-CC2650STK-SENSORTAG*, 2022.

[21] C. B. Margi, R. C. A. Alves, and J. Sepulveda, "Sensing as a service: Secure wireless sensor network infrastructure sharing for the internet of things," *Open Journal of Internet Of Things (OJIOT)*, vol. 3, no. 1, pp. 91–102, 2017. Special Issue: Proceedings of the International Workshop on Very Large Internet of Things (VLIoT 2017) in conjunction with the VLDB 2017 Conference in Munich, Germany.

[22] A. AG, "Arduino," in *https://www.arduino.cc*, 2022.

[23] F. Restuccia, S. D'Oro, and T. Melodia, "Securing the internet of things in the age of machine learning and software-defined networking," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4829–4842, 2018.

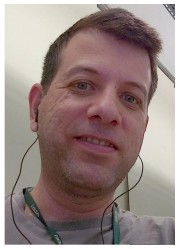
[24] T. Theodorou and L. Mamatas, "Software defined topology control strategies for the internet of things," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 236–241, 2017.

[25] C. Schurgers and M. B. Srivastava, "Energy efficient routing in wireless sensor networks," in *2001 MILCOM Proceedings Communications for Network-Centric Operations: Creating the Information Force (Cat. No.01CH37277)*, vol. 1, pp. 357–361 vol.1, Oct 2001.

[26] A. Boulis, *Castalia, A simulator for Wireless Sensor Networks and Body Area Networks. User's Manual.* NICTA, version 3.2 ed., 2011.

[27] S. Nordic, "nrf24 series," in *https://www.nordicsemi.com/Products/nRF24-series*, 2022.

[28] T. Sayjari, R. M. Silveira, and C. B. Margi, "Control and data traffic isolation in sdwsn using ieee 802.15. 4e tsch," in *2021 IEEE Statistical Signal Processing Workshop (SSP)*, pp. 126–130, IEEE, 2021.



Luis Eduardo Lima received his M.Sc. Degree in Computer Science from the Federal University of São Paulo - UNIFESP (2014).

He is currently a PhD candidate at UNIFESP where he develops research in the areas of wireless sensor networks and software defined networks.

Valério Rosset received his PhD degree in Electrical and Computer Engineering from University of Porto (Portugal) in 2009.

He is an associate professor at the Federal University of São Paulo (UNIFESP). His current research topics of interest are: cognitive networks, adaptive routing and fault tolerant protocols for network communications (WSNs, WSANs, DTNs and M2M) and computer/network security protocols.