UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LUCAS MENDES RIBEIRO ARBIZA

SDN no Contexto de IoT: Refatoração de Middleware para Monitoramento de Pacientes Crônicos Baseada em *Software-Defined Networking*

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Profa. Dra. Liane Margarida Rockenbach Tarouco

Porto Alegre
2016

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Arbiza, Lucas Mendes Ribeiro

SDN no Contexto de IoT:
Refatoração de Middleware para Monitoramento de Pacientes Crônicos Baseada em *Software-Defined Networking* / Lucas Mendes Ribeiro Arbiza. – Porto Alegre: PPGC da UFRGS, 2016.

119 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2016. Orientador: Liane Margarida Rockenbach Tarouco.

1. Software-defined networking. 2. Internet of things. 3. Gerência de redes. 4. Middleware. 5. Redes domésticas. I. Tarouco, Liane Margarida Rockenbach. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Vladimir Pinheiro do Nascimento
Diretor do Instituto de Informática: Prof. Luis da Cunha Lamb
Coordenador do PPGC: Prof. Luigi Carro
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

“There’s a way to do it better
– find it.”
— THOMAS A. EDISON

AGRADECIMENTOS

Em 2011, logo após concluir minha graduação, tive a oportunidade de participar de um projeto de pesquisa, o Projeto REMOA, que me abriu as portas para o mestrado e serviu como base para este trabalho. Agradeço à Professora Liane por esta oportunidade, mas também por toda sua orientação, paciência, insistência e desafios ao longo do período que estive sob sua orientação.

Devido ao Projeto REMOA tive a oportunidade de desenvolver minhas pesquisas no PoP-RS/RNP, onde atualmente trabalho, local onde pude ampliar, aprender a aplicar conhecimentos sobre redes de computadores que me eram tão distantes quando me formei. A experiência bastante prática e aplicada me forneceu a base para desenvolver um trabalho de pesquisa contextualizado e que leva em conta diversos detalhes e restrições presentes em redes, sejam elas domésticas, de *campus*, ou empresariais. Em especial, registro meu agradecimento a Leandro M. Bertholdo, com quem continuo aprendendo e quem, em diversos momentos, colaborou na construção desta proposta. Também sou grato a todos com quem trabalhei no PoP-RS até então porque com todos aprendi algo.

Juntamente às pessoas já citadas, agradeço ao Professor Lisandro Z. Granville e Carlos Raniery P. dos Santos que colaboraram na produção dos artigos que publiquei e aos colegas de mestrado com quem passei parte do meu tempo e realizei trabalhos nas cadeiras cursadas.

Além da contribuição nos trabalhos científicos, devo agradecer a quem colaborou dando suporte de alguma forma, a começar pela minha esposa que compreendeu quando precisei dedicar grande parte do meu tempo e minha atenção às atividades acadêmicas e foi uma grande companheira. Agradeço o apoio da minha mãe e irmão. Também agradeço à coordenação pedagógica do IFRS, campus Restinga, da época que lecionei nessa instituição pelo esforço em organizar meus horários de aulas a fim de que eu pudesse aproveitar melhor o tempo que eu dispunha.

RESUMO

Algumas palavras e definições comumente utilizadas quando se está falando de *Software-Defined Networking*, como programabilidade, flexibilidade, ou gerenciamento centralizado, parecem muito apropriadas ao contexto de um outro paradigma de rede: *Internet of Things*. Em redes domésticas já não é incomum a existência de dispositivos projetados para segurança, climatização, iluminação, monitoramento de saúde e algumas formas de automação que diferem entre si em diversos aspectos, como no modo de operar e de se comunicar. Lidar com este tipo de cenário, que pode diferir bastante daquilo que estamos acostumados na gerência de redes e serviços, fazendo uso dos recursos tradicionais como ferramentas e protocolos bem estabelecidos, pode ser difícil e, em alguns casos, inviável. Com o objetivo de possibilitar o monitoramento remoto de pacientes com doenças crônicas através de dispositivos de *healthcare* disponíveis no mercado, uma proposta de *middleware* foi desenvolvida em um projeto de pesquisa para contornar as limitações relacionadas à interoperabilidade, coleta de dados, gerência, segurança e privacidade encontradas nos dispositivos utilizados. O *middleware* foi projetado com o intuito de executar em *access points* instalados na casa dos pacientes. Contudo, as limitações de *hardware* e *software* do *access point* utilizado refletem no desenvolvimento, pois restringem o uso de linguagens de programação e recursos que poderiam agilizar e facilitar a implementação dos módulos e dos mecanismos necessários. Os contratempos encontrados no desenvolvimento motivaram a busca por alternativas, o que resultou na refatoração do *middleware* através de *Software-Defined Networking*, baseando-se em trabalhos que exploram o uso desse paradigma em redes domésticas. O objetivo deste trabalho é verificar a viabilidade da utilização de *Software-Defined Networking* no contexto de *Internet of Things*, mais especificamente, aplicado ao serviço de monitoramento de pacientes da proposta anterior e explorar os possíveis benefícios resultantes. Com a refatoração, a maior parte da carga de serviços da rede e do monitoramento foi distribuída entre servidores remotos dedicados, com isso os desenvolvedores podem ir além das restrições do *access point* e fazer uso de recursos antes não disponíveis, o que potencializa um processo de desenvolvimento mais ágil e com funcionalidades mais complexas, ampliando as possibilidades do serviço. Adicionalmente, a utilização de *Software-Defined Networking* proporcionou a entrega de mais de um serviço através de um único *access point*, escalabilidade e autonomia no gerenciamento das redes e dos dispositivos e na implantação de serviços, fazendo uso de recursos do protocolo OpenFlow, e a cooperação entre dispositivos e serviços a fim de se criar uma representação digital mais ampla do ambiente monitorado.

Palavras-chave: Software-defined networking. internet of things. gerência de redes. middleware. redes domésticas.

SDN in the IoT Context: Software-Defined Networking Based Refactoring of a Middleware for Chronic Patients Monitoring

ABSTRACT

Some words and definitions usually employed when talking about Software-Defined Networking such as programmability, frexibility, or centralized management sound very appropriate to the context of another network paradigm: Internet of Things. The presence of devices designed for security, air conditioning, lighting, health monitoring and some other automation resources have become common in home networks; those devices may be different in many ways, such as the way they operate and communicate, between others. Dealing with this kind of scenario may differ in many ways from what we are familiar regarding networking and services management; the use of traditional management tools and protocols may be hard or even unfeasible. Aiming to enable the health monitoring of patients with chronical illnesses through using off-the-shelf healthcare devices a middleware proposal was developed in a research project to circumvent interoperability, data collecting, management, security and privacy issues found in employed devices. The middleware was designed to run on access points in the homes of the patients. Although hardware and software limitations of the used access points reflect on the development process, because they restrict the use of programming languages and resources that could be employed to expedite the implementation of necessary modules and features. Development related mishaps have motivated the search for alternatives resulting in the middleware refactoring through Software-Defined Networking, based on previous works where that paradigm is used in home networks. This work aims to verify the feasability of the employment of Software-Defined Networking in the Internet of Things context, and its resulting benefits; specifically in the health monitoring of chronic patients service from the previous proposal. After refactoring most of the network and services load was distributed among remote dedicated servers allowing developers to go beyond the limitations imposed by access points constraints, and to make use of resources not available before enabling agility to the development process; it also enables the development of more complex features expanding services possibilities. Additionally Software-Defined Networking employment provides benefits such as the delivering of more than only one service through the same access point; scalability and autonomy to the network and devices monitoring, as to the service deployment through the use of OpenFlow resources; and devices and services cooperation enabling the built of a wider digital representation of the monitored environment.

Keywords: Software-Defined Networking, Internet of Things, Network Management, middleware, home networks.

LISTA DE ABREVIATURAS E SIGLAS

AS	Autonomous System
BGP	Border Gateway Protocol
DDNS	Dynamic Domain Name System
DLNA	Digital Living Network Alliance
DNS	Domain Name System
GRE	Generic Routing Encapsulation
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LAN	Local Area Network
MIB	Management Information Base
MPLS	Multiprotocol Label Switching
NAT	Network Address Translation
NFV	Network Function Virtualization
OSPF	Open Shortest Path First
PAN	Personal Area Network
PC	Personal Computer
PDA	Personal Digital Assistent
QoS	Quality of Service
REMOA	Rede-Cidadã de Monitoramento do Ambiente baseado em Conceitos da Internet das Coisas
RFID	Radio-Frequency Identification
SDK	Software Development Kit
SDN	Software-Defined Network

SNMP Simple Network Management Protocol

USB Universal Serial Bus

VLAN Virtual Local Area Network

VM Virtual Machine

WPAN Wireless Personal Area Network

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Destaca-se, comumente, em relação ao paradigma de *Software-Defined Networking* (SDN) alguns aspectos como flexibilidade, programabilidade ou centralização. SDN tornou-se conhecido por possibilitar a separação do controle da rede dos dispositivos de encaminhamento, como roteadores e switches, permitindo que o comportamento lógico fosse definido por *software*, de forma centralizada, provendo autonomia à rede através da utilização de aplicações escritas ou customizadas para cada contexto (MCKEOWN et al., 2008). Em cenários baseados em SDN, a camada de infraestrutura é configurada por um elemento central, o controlador, sobre o qual executam aplicações que implementam a inteligência da rede através de regras que definem o encaminhamento dos pacotes (SOFTWARE-DEFINED..., 2012).

Ao separar o controle da rede dos dispositivos de encaminhamento, SDN permite que diversas possibilidades sejam exploradas e implementadas independente das funcionalidades e serviços existentes nos equipamento utilizados, disponibilizados pelos seus fabricantes. Fazendo uso do protocolo OpenFlow, primeiro padrão definido especificamente para SDN (SOFTWARE-DEFINED..., 2012), pesquisadores têm desenvolvido seus trabalhos dentro do campo de redes explorando, por exemplo, virtualização de redes (SHERWOOD et al., 2009), roteamento (ROTHENBERG et al., 2012), *Quality of Service* (QoS) (EGILMEZ et al., 2012), mobilidade (YAP et al., 2010), gerência de redes (KIM; FEAMSTER, 2013), dentre muitos outros.

Dentro do campo de Internet of Things (IoT), recentemente, algumas pesquisas começaram a fazer uso de SDN. Ambientes de IoT distinguem-se das redes com as quais estamos acostumados devido à diversidade de dispositivos em diversos aspectos como formato, propósito, modo de operação e recursos, o que demanda novas formas de lidar e gerir tais cenários. Mecanismos tradicionalmente utilizados para a gerência de dispositivos, como por exemplo os baseados em ICMP e SNMP, mostram-se insuficientes e, em alguns casos, inúteis devido ao modo de operação e a funcionalidades implementadas em dispositivos com recursos limitados e que operam alimentados por bateria. Outras questões de IoT, tais como interconexão, padronização e cooperação (IOT-A..., 2013; COETZEE; EKSTEEN, 2011; TECHNOLOGY..., 2013; WU et al., 2011), endereçamento e identificação (MIORANDI et al., 2012), semântica (MIORANDI et al., 2012; HINSSEN, 2012) e segurança e privacidade (BANDYOPADHYAY; SEN, 2011; TAROUCO et al., 2012), vêm sendo pesquisadas e propostas têm sido apresentadas abordando-as.

SDN aparece como um abordagem interessante para ambientes de IoT porque traz a versatilidade do mundo do desenvolvimento de *software* para o campo de redes, onde por vezes os

administradores ficam limitados às restrições da implementação das funcionalidades e protocolos fornecida pelos fabricantes. SDN torna a rede uma plataforma para inovação possibilitando o desenvolvimento e implantação de funcionalidades abrindo as possibilidades para se lidar com as particularidades e heterogeneidade de IoT (WEE, 2014; CARAGUAY et al., 2014).

O presente trabalho explora a utilização dos recursos de SDN no contexto de IoT. O cenário no qual o trabalho se baseia é o de monitoramento remoto e autônomo de pacientes através de dispositivos de telemedicina e outros dispositivos imersos no ambiente capacitados para senti-lo. Nesse cenário foram desenvolvidas as pesquisas do projeto REMOA[1], cujo objetivo era o desenvolvimento de soluções para benefício dos cidadãos em cidades inteligentes. A proposta do projeto foi um *middleware* (TAROUCO et al., 2012), implementado em *access points*, que continha funcionalidades desenvolvidas para lidar com os problemas encontrados nos dispositivos utilizados e no cenário, são eles: interconexão, coleta de dados, gerenciamento dos dispositivos, das redes e da solução, e questões de segurança e privacidade.

Ao longo da prototipação da proposta de Tarouco et al. (2012) percebeu-se que a solução era difícil de implementar, expandir e adaptar devido às restrições que as limitações de *hardware* e do sistema utilizado nos *access points* impunham ao desenvolvimento. Apesar de haver a possibilidade de se utilizarem linguagens de alto nível e modernas para o desenvolvimento de módulos do *middleware*, a necessidade de adicionar serviços como um servidor *web*, por exemplo, para viabilizar a coleta dos dados enviados pelos dispositivos de monitoramento complicaria mais o processo de manutenção da solução que, por se tratar de uma arquitetura composta por alguns módulos configurados de acordo com o monitoramento a ser realizado, já possuía diversas questões a serem levadas em conta durante os processos de configuração e atualização. Para se obter uma solução mais robusta optou-se por utilizar linguagens como C e C++ para gerar código nativo, reduzindo o consumo de recursos e possibilitando um melhor desempenho. Tal escolha, contudo, resulta em um desenvolvimento potencialmente mais lento e complexo.

O *middleware* proposto no projeto REMOA concentrava nos *access points* as funcionalidades de encaminhamento e redirecionamento de pacotes, coleta de dados, gerência das *things* e dos *access points* e a segurança da transmissão dos dados de saúde coletados. Para cada dispositivo utilizado no monitoramento de um paciente era necessário um módulo que coletasse dados do monitoramento do paciente e para a gerência dos dispositivos, caso o dispositivo não atenda requisições SNMP um outro módulo precisaria ser utilizado para que o *access point* respondesse pelo dispositivo com base nas informações de gerência coletadas. A segurança e

[1]REMOA: Rede Cidadã de Monitoramento do Ambiente Baseado no Conceito de Internet das Coisas

privacidade, contudo, era feita através de tunelamento utilizando IPSec estabelecendo um caminho seguro entre os *access points* e os servidores de processamento e armazenamento dos dados.

Devido às questões mencionadas, propôs-se a refatoração do *middleware* fazendo uso de SDN (ARBIZA et al., 2015; ARBIZA et al., 2016), tema do presente trabalho. Refatoração, na Engenharia de Software, significa reescrever trechos de código afim de aprimorar um *software* ou função internamente, mas sem modificar seu comportamento externo (FOWLER; BECK, 1999). Apesar de se basear no conceito de refatoração, essa proposta não é restritiva em relação às alterações no comportamento externo da solução, uma vez que o objetivo é verificar a viabilidade da utilização de SDN em um contexto de IoT e quais benefícios podem ser obtidos em relação à proposta de *middleware* anterior.

Com a refatoração da proposta de *middleware* do projeto REMOA utilizando SDN moveu-se a complexidade da rede e dos serviços para fora do ambiente monitorado. Com serviços sendo providos por servidores os desenvolvedores não precisam se limitar às restrições de *hardware* e *software* dos *access points*, potencializando um desenvolvimento mais ágil de novos serviços e melhorias, bem como a adição de novos dispositivos de monitoramento. SDN provê a orquestração necessária para a comunicação das *things* com os serviços e escalabilidade para gerência das diversas redes dos ambientes monitorados, além de dinamicidade ao processo de *deployment* de novos cenários e dispositivos de monitoramento. SDN facilitou a entrega de múltiplos serviços através de um mesmo *access point* e possibilitou a construção de uma representação digital da residência do paciente monitorado mais ampla utilizando informações oriundas de todos os dispositivos conectados à rede. A proposta refatorada também simplifica a obtenção dos dados utilizados para a gerência das *things* baseando-se em recursos do próprio protocolo de SDN utilizado.

O capítulo 2 apresenta o conceito de SDN, suas aplicações e recursos utilizados para desenvolvimento e para implantá-lo, seja para fins de pesquisa ou produção. O capítulo 3 apresenta IoT através de algumas definições do paradigma e do que é considerado *thing*, seus problemas e algumas propostas que os abordam. O capítulo 4 detalha o cenário do projeto REMOA, o *middleware* e as questões que motivaram a refatoração; em seguida detalha a refatoração do *middleware* através de SDN apresentando o projeto de arquitetura e os componentes desenvolvidos: um controlador OpenFlow chamado Derailleur, uma aplicação que implementa a inteligência da solução, chamada ThingsFlow, e detalha a implantação dos recursos utilizados nos *access points*. No capítulo 5, os benefícios da proposta refatorada são apresentados e discutidos fazendo uso de comparações entre os processos de desenvolvimento, aprimoramento

e implantação de serviços e funcionalidades considerando as duas propostas: o *middleware* do projeto REMOA, proposto em Tarouco et al. (2012), e após a refatoração, proposto em Arbiza et al. (2015). Por fim, no capítulo 6, são apresentadas e discutidas as considerações sobre o trabalho, bem como os benefícios proporcionados pela utilização de SDN em um cenário de IoT.

2 SOFTWARE-DEFINED NETWORKING

Há muito tempo que o comportamento lógico das redes de computadores é definido por *software*, mais especificamente pelas funcionalidades implementadas nos sistemas operacionais em execução nos equipamentos de rede como roteadores e *switches*. Tão importante quanto as especificações de *hardware*, o sistema operacional a ser adquirido para qualquer equipamento de rede é o elemento que permite, ou não, a utilização dos recursos implementados em cada dispositivo. A configuração das funcionalidades através do sistema operacional, realizada em cada equipamento e obedecendo a sintaxe específica de cada fabricante, é crucial para o funcionamento adequado dos equipamentos e da rede como um todo.

SDN difere da forma convencional de configuração de rede por separar o plano de controle da infraestrutura física (SOFTWARE-DEFINED. . . , 2012), isto significa que o comportamento lógico da rede é definido através de um ou mais controladores que determinam como os equipamentos de rede encaminharão os pacotes. O protocolo OpenFlow foi a primeira proposta de implementação de SDN (MCKEOWN et al., 2008) e atualmente é a mais utilizada, tanto para fins de pesquisa quanto em ambientes de produção. O OpenFlow provê um padrão para a comunicação permitindo ao controlador definir o comportamento lógico da rede através da programação das tabelas de fluxo (*flow tables*) existente nos equipamentos de rede.

A separação da camada de controle da infraestrutura possibilitou o desenvolvimento de aplicações que controlam a rede programando os equipamento através do OpenFlow, com isso, diversas propostas têm sido apresentadas implementando aplicações para funcionalidades como controle e balanceamento de banda, roteamento, segurança, etc. Algumas propostas implementam através do OpenFlow funcionalidades de protocolos, como o *Spanning Tree* (STP), por exemplo. Ao retirar o controle do encaminhamento de pacotes dos equipamentos de infraestrutura e implementar a configuração lógica da rede através de aplicações, o controle e a gerência dos recursos disponíveis na rede como um todo se expandem para além das limitações de sintaxe e funcionalidades dos equipamentos e seus fabricantes.

Recentemente algumas propostas têm feito uso de SDN em cenários de IoT. As limitações de *hardware* dos dispositivos considerados *things* faz com que os métodos e mecanismos de gerenciamento, segurança e comunicação comumente utilizados em redes de computadores se mostrem ineficientes; SDN proporciona flexibilidade e dinamicidade ao ambiente de rede além do que é provido pelos protocolos de rede tradicionais, bastante engessados ou difíceis de modificar. O desenvolvimento de aplicações é outro recurso que contribui para prover funcionalidades à rede para lidar com a diversidade de IoT sem depender de soluções fornecidas por

fabricantes das *things* ou de equipamentos de rede.

Esse capítulo tem como propósito apresentar o paradigma SDN e suas aplicações dando ao leitor uma visão das possibilidades abertas pelas tecnologias que implementam este novo conceito. A seção 2.1 apresenta o protocolo OpenFlow e sua forma de funcionamento possibilitando a compreensão de como redes baseadas em SDN são implementadas e sua forma de operação; a seção 2.2 dá alguns exemplos que recursos disponíveis para desenvolvimento e utilização de SDN, em especial OpenFlow. Na seção 2.3 são citados cenários de pesquisa e produção onde SDN vem sendo utilizado para diversas finalidades. Por fim, a seção 2.4 apresenta trabalhos que se relacionam a esta proposta por abordar a utilização de SDN em cenários de IoT e redes domésticas e demonstram alguns benefícios da coexistência desses dois paradigmas de rede.

2.1 Protocolo OpenFlow

O protocolo OpenFlow foi proposto e desenvolvido por McKeown et al. (2008) que fizeram uso do protocolo NETCONF (NETWORK. . . , 2011) para programar as *flow tables* (tabelas com regras de encaminhamento) existentes nos equipamentos de rede. A Open Networking Foundation (ONF)[1] é uma organização para a promoção e adoção de SDN e atualmente responsável por coordenar o processo de desenvolvimento e padronização do protocolo OpenFlow. O OpenFlow é implementado de acordo com as especificações que padronizam o protocolo (ONF. . . , 2014), atualmente a versão 1.4 está em desenvolvimento, contudo a maior parte do *hardware* e *software* disponível no mercado está em conformidade com a versão 1.0 ou em processo de migração para a versão 1.3.

Nas redes de computadores atuais, um procedimento comum no processo de implantação da rede física e lógica é a configuração individual de cada um dos equipamentos que realizam a comutação de pacotes ou proveem serviços para a própria rede. O volume de trabalho requerido para a configuração de toda a rede varia de acordo com o número de equipamentos e a complexidade do cenário. Em redes domésticas é comum as funções de roteamento, comutação de pacotes e modulação de frequência estarem concentradas em um único *access point* que muitas vezes é entregue ao cliente pré-configurado e pronto para uso. Ao cliente é opcional alterar as configurações para que atendam necessidades específicas de roteamento, endereçamento de rede ou para utilizar serviços como DDNS, DLNA, etc. Em redes empresariais é comum haver equipamentos de rede em maior número e de maior porte onde são configurados

[1] Open Networking Foundation: https://www.opennetworking.org/about/onf-overview

mecanismos para isolamento de redes através do uso de VLAN ou divisão em sub-redes, dentre outros recursos disponíveis nos equipamentos utilizados. Em redes de provedores de serviços e de Internet o cenário é mais complexo envolvendo diversos equipamentos e diversos protocolos, como por exemplo o OSPF para roteamento interno, BGP para conexão e roteamento entre *Autonomous Systems* (AS), *Spanning Tree Protocol* para a prevenção de *loops*, além dos mecanismos de encaminhamento e isolamento, como VLANs.

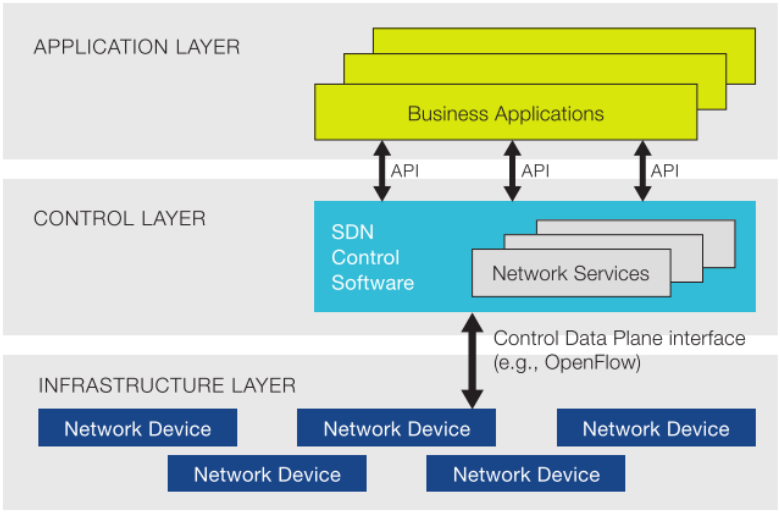


Figura 2.1 – Arquitetura do paradigma *Software-Defined Networking* (SOFTWARE-DEFINED..., 2012)

Com a separação do plano de controle da infraestrutura física, provida por SDN, como mostrado na figura 2.1 que ilustra a arquitetura desse paradigma, a configuração da forma de encaminhamento dos pacotes é feita por um ou mais controladores que instalam *flow entries* (regras de encaminhamento) nas *flow tables* dos *switches* OpenFlow. *Switches* OpenFlow também podem ser roteadores ou qualquer outro equipamento que recebe regras de encaminhamento de um controlador OpenFlow. Através da forma de funcionamento de redes SDN, o procedimento de configuração de cada equipamento, feito um a um em redes convencionais, é substituído pelo processo de desenvolvimento ou configuração de aplicações que atuarão no controlador (camada *Application Layer* na figura 2.1). As aplicações que rodam sobre o controlador é que definem o comportamento lógico da rede.

O processo de configuração dos equipamentos de rede em uma rede SDN é feito através do controlador que atende às solicitações dos *switches* por regras de encaminhamento para pacotes que o próprio *switch* não sabe como encaminhar; a forma de operação de SDN provê autonomia no processo de configuração e definição do comportamento lógico da rede. A figura 2.2, baseada no processamento de pacotes em um *switch* OpenFlow na versão 1.3.2 (OPENFLOW..., 2013), ilustra como é realizado o processamento de pacotes em SDN.

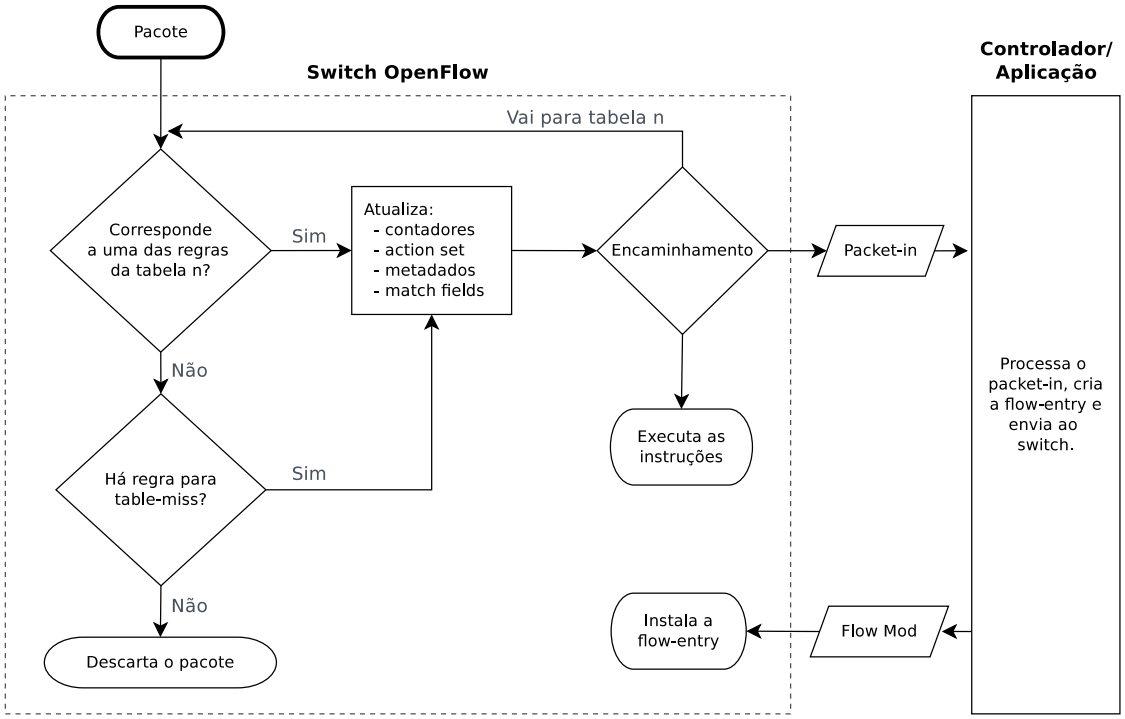


Figura 2.2 – Processamento de pacotes em *switches* OpenFlow (OPENFLOW. . . , 2013).

Quando um *switch* OpenFlow recebe um pacote uma busca é iniciada na primeira *flow table* por uma *flow entry* na qual o pacote se enquadre. *Flow tables* são compostas de *flow entries* e cada *flow entry* é formada pelos seguintes campos:

- *Match fields*: campos como *header*, porta pela qual o pacote entrou no *switch* ou metadados especificados em uma tabela anterior. Os *match fields* são utilizados na comparação que verifica se um pacote se enquadra, ou não, em uma *flow entry*;
- Prioridade: indicador de precedência da *flow entry*;
- Contadores: campo incrementado cada vez que um pacote se enquadra em uma dada *flow entry* em relação às demais *flow entries*;
- Intruções: enquanto um pacote é processado no *switch*, a ele é associado um *action set* que consiste de um conjunto de ações que serão executadas ao final do processamento do pacote. Enquanto o pacote é processado o *action set* pode ser modificado pelas *flow entries* nas quais o pacote se enquadre;
- *Timeouts*: define o tempo máximo ou tempo ocioso até que uma *flow entry* expire;
- *Cookie*: valor numérico utilizado pelo controlador para a manipulação de *flow entries*.

Caso o pacote não se enquadre em nenhuma das *flow entries*, o pacote deverá ser tratado por uma *flow entry* que trate a ocorrência de *table miss* — nome dado quando nenhuma *flow entry* existente no *switch* é capaz de processar o pacote. Toda *flow table* deve prever a ocorrência

de *table miss*, caso o contrário o pacote será descartado (OPENFLOW. . . , 2013), ao menos que seja essa a intenção. Mesmo o descarte do pacote pode ser previsto em uma *flow entry* que trate um *table miss*, bem como o envio do pacote ao controlador ou para outra tabela onde o pacote será novamente processado.

Sempre que um pacote é processado por uma *flow entry*, mesmo que seja para *table miss*, são realizadas atualizações nos contadores da *flow entry* e os dados associados ao processamento do pacote, como *action set*, metadados (informações que acompanham o pacote durante o seu processamento, são utilizados para passar informações relacionadas ao pacote de uma tabela para outra, como por exemplo, se o pacote foi recebido por uma porta lógica (GRE)) e *match fields*.

Na figura 2.2 a comunicação entre o *switch* OpenFlow e o controlador é ilustrada através de um *packet-in*. O *packet-in* contém as informações do pacote sendo processado necessárias para que o controlador e as aplicações que rodam sobre ele possam determinar qual encaminhamento o *switch* deve dar ao pacote. O caminho inverso é ilustrado por uma *flow mod* que contém os dados que vão compor a *flow entry* a ser instalada. A partir do momento que o controlador instala uma *flow entry* para tratar o pacote que gerou um *packet-in*, o *switch* apenas encaminha o pacote conforme o definido na *flow entry* sem enviar um novo *packet-in* ao controlador, ao menos que a *flow entry* expire, seja devido ao tempo ocioso, seja devido ao tempo máximo até expirar. Uma *flow entry* pode ter tempos ocioso e máximo indeterminados, ou seja, não expiram, somente serão removidas se o controlador explicitamente removê-las ou o *switch* for reinicializado.

SDN provê dinamicidade à configuração da rede através do envio de *flow entries* aos *switches*, seja como resposta a uma solicitação do *switch*, ou em um conjunto maior de *flow entries* instalados assim que o *switch* é iniciado. As regras de encaminhamento são sempre atuais em relação à política de encaminhamento da rede. Em redes baseadas em SDN a configuração, o controle e o gerenciamento da rede são centralizados em controladores e aplicações que implementam a inteligência da rede e proveem abstração dos detalhes técnicos dos equipamentos e dos protocolos.

2.2 Recursos para Desenvolvimento e Implantação de Redes SDN

A figura 2.1 ilustra SDN dividido em três camadas: aplicação, controle e infraestrutura. Cada camada representa um nicho diferente que vem sendo desenvolvido e explorado, tanto no âmbito da pesquisa quanto de mercado.

A camada de aplicação é a que acumula o maior número de propostas. Sempre que um controlador é executado, sobre ele roda uma ou mais aplicações que processam os pacotes do tipo *packet-in* que são enviados pelos *switches* ao controlador. As aplicações executadas variam de acordo com o contexto de aplicação. Kreutz et al. (2014) agrupam as aplicações nos seguintes segmentos: engenharia de tráfego, mobilidade e *wireless*, medição e monitoramento, segurança e confiabilidade e redes de *datacenter*. Algumas aplicações de alguns dos segmentos citados serão apresentadas na seção 2.3.

As aplicações precisam ser compatíveis com o controlador, ou melhor, no contexto atual, devem ser desenvolvidas sobre a API do controlador, fazendo uso de classes e chamadas de métodos providos pelo controlador. Há, contudo, propostas que são agnósticas de controlador; o NetFuse, por exemplo, roda entre os *switches* e o controlador, como um *proxy* — processa os pacotes no meio do caminho e em seguida os envia ao controlador (WANG et al., 2013).

Na camada de controle há algumas opções de controladores, alguns oferecendo um conjunto de recursos além da API para o OpenFlow. Um controlador deve fornecer uma interface (API) através da qual as aplicações possam se comunicar com os *switches* utilizando o protocolo OpenFlow, ou outro protocolo de SDN. A interface entre o controlador e as aplicações é chamada de *northbound* API; a interface entre o controlador e os *switches*, ou seja, a comunicação OpenFlow, é chamada de *southbound* API.

Grande parte dos controladores disponíveis atualmente foram implementados em linguagens de programação orientadas a objeto (OO) e fornecem uma API com a capacidade de abstrair detalhes do protocolo e possibilitar maior agilidade no desenvolvimento de aplicações. O primeiro controlador de expressão, utilizado em grande parte dos primeiros trabalhos relacionados a OpenFlow, foi o NOX[2]. O NOX foi desenvolvido em C++ e provia *binding* para Python; inicialmente implementava a versão 1.0 do OpenFlow, porém foi desenvolvido até a versão 1.3[3]. O NOX foi dividido em dois projetos: NOX, implementado em C++, e o POX, implementado em Python. O POX foi utilizado especialmente em pesquisas e educação devido à facilidade e agilidade proporcionada por Python. Atualmente o desenvolvimento de ambos os projetos se encontra estagnado.

Utilizando Java como linguagem de programação, pode-se citar alguns exemplos como Beacon[4], Floodlight[5], Maestro[6], OpenDaylight[7], ONOS[8]. Grande parte dos controladores Java

[2]NOX: http://www.noxrepo.org
[3]CPqD nox13oflib: https://github.com/CPqD/nox13oflib
[4]Beacon: https://openflow.stanford.edu/display/Beacon/Home
[5]Floodlight Project: http://www.projectfloodlight.org/floodlight
[6]Maestro Platform: https://code.google.com/p/maestro-platform
[7]OpenDaylight: http://www.opendaylight.org
[8]Open Networking Operating System — ONOS: http://onosproject.org

possui API com interface RESTful (*Representational State Transfer*). O estilo arquitetural RESTful é definido pela realização das operações para a criação, leitura, atualização e remoção de recursos através de comunicação HTTP, conjunto de operações conhecida pelo acrônico CRUD (*Create*, *Read*, *Update* e *Delete*), e atualmente é muito utilizado para implementação de *web services*. A comunicação entre cliente e servidor na arquitetura RESTful tem como característica o fraco acoplamento entre as partes devido ao uso do HTTP, desta forma o cliente pode ser implementado em qualquer linguagem com suporte a HTTP.

Dentre os controladores baseados em Java citados, o OpenDaylight tem ganho bastante destaque por ser um projeto que envolve diversos membros expressivos e por ser uma solução que agrega alguns recursos além do controlador. O projeto OpenDaylight é uma iniciativa da Linux Foundation[9] da qual participam diversas empresas, dentre elas as principais fabricantes de equipamentos de rede da atualidade (Cisco, Juniper, Brocade, Extreme, etc.), além de grandes fabricantes e desenvolvedores de servidores e sistemas utilizados na infraestrutura de redes e serviços (Microsoft, DELL, HP, IBM, Red Hat, Microsoft, etc.). O objetivo do projeto é o desenvolvimento de um *framework* padrão, modular, conectável e aberto para a programabilidade e controle da rede através de SDN e NFV (*Network Function Virtualization*). Na arquitetura do OpenDaylight, o OpenFlow coexiste na mesma camada que diversos protocolos de gerência (ex.: SNMP) e de configuração (ex.: NETCONF, BGP), abstraídos pela camada de controle para serem acessados através da API por aplicações para o gerenciamento da rede (OPENDAYLIGHT. . . , 2014).

Além de C++, Python e Java, há também controladores desenvolvidos em C e Ruby dentre os levantados por Kreutz et al. (2014).

Fabricantes de equipamentos de rede também desenvolvem suas próprias soluções para SDN. A solução da Juniper Networks, o OpenContrail[10], é baseada em protocolos padronizados (ex.: OpenFlow) e provê os recursos necessários para virtualização de rede baseada em SDN, roteamento virtual, análise estatística, programabilidade, integração com plataformas de orquestração de núvem (ex.: OpenStack, CloudStack), dentre outros recursos.

A Cisco, apesar de apoiar o desenvolvimento do *framework* aberto OpenDaylight, comercializa a solução *Application Centric Infrastructure* (ACI)[11], baseada no protocolo desenvolvido pela própria Cisco, o OpFlex[12]. A ACI é uma solução proprietária, que centraliza e

[9]Linux Foundation: http://www.linuxfoundation.org

[10]OpenContrail: http://www.opencontrail.org/about

[11]Cisco ACI: http://www.cisco.com/c/en/us/solutions/data-center-virtualization/application-centric-infrastructure/index.html

[12]OpFlex: http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html

automatiza a configuração da rede focando nos serviços que são providos. Sua utilização requer a compatibilidade dos equipamentos envolvidos com o protocolo OpFlex.

Brocade e Extreme desenvolveram suas plataformas de controle baseadas no projeto OpenDaylight. A Brocade lançou o controlador Vyatta[13] que consiste de uma versão com garantia de qualidade do OpenDaylight.

A ONF lançou, em 2013, uma competição para a criação de um *driver* OpenFlow para ser utilizado por desenvolvedores de produtos no campo de SDN, que fosse desenvolvido e mantido atualizado em relação às especificações do protocolo. O projeto vencedor foi a libfluid (CPQD. . . , 2014). A libfluid[14] é dividida em duas bibliotecas com funções distintas, uma que implementa as funcionalidades relacionadas às mensagens do protocolo OpenFlow e a outra que implementa o servidor OpenFlow que ouve por mensagens dos *switches* e as trata. A libfluid foi escrita em C++, mas provê *bindings* (interfaces) para Python e Java.

As possibilidades na camada de infraestrutura vêm sendo ampliadas conforme cresce o interesse dos principais fabricantes por SDN. Fabricantes como Juniper, Brocade, Extreme, Datacom, dentre outros, desenvolvem *hardware* compatível com o OpenFlow, contudo, alguns fabricantes restringem o uso de OpenFlow exigindo a aquisição de licenças específicas que incluem o uso do protocolo. Uma solução de baixo custo por não necessitar *hardware* especializado é a utilização do Open vSwitch[15] sobre sistemas Linux ou BSD. O Open vSwitch é um *switch* virtual, com qualidade de produção, rápido, confiável e amplamente utilizado especialmente em ambientes de rede virtuais por *hypervisors* para conectar máquinas virtuais (VMs) através de *bridges* tornando-as acessíveis na rede como se fossem qualquer máquina física (GRAF, 2014). Com o Open vSwitch executando em um sistema, seja em um PC, em um *access point*, ou qualquer *hardware* rodando Linux, o dispositivo pode atuar também, ou exclusivamente, como um *switch* OpenFlow, tanto para um conjunto de VMs quanto para *hosts* físicos e virtuais existentes na rede.

2.3 Exemplos de Aplicação de SDN

As seções anteriores proporcionam uma visão do que é e de como funciona SDN, especialmente através do protocolo OpenFlow, e recursos disponíveis que possibilitam sua implantação. A proposta dessa seção é apresentar exemplos de diferentes áreas, dentro do contexto de

[13] Vyatta Controller: http://www.brocade.com/products/all/software-defined-networking/brocade-vyatta-controller/index.page

[14] libfluid: http://opennetworkingfoundation.github.io/libfluid/index.html

[15] Open vSwitch: http://openvswitch.org

redes de computadores, do uso de SDN. Os exemplos complementam a apresentação de SDN e OpenFlow provendo uma visão mais aplicada do uso dos seus recursos. O uso de OpenFlow consiste da execução de uma ou mais aplicações em um ou mais controladores, os quais proveem inteligência à rede.

2.3.1 Virtualização de Redes

O OpenFlow surgiu como uma proposta para promover inovação na área de redes de computadores através da experimentação de novos protocolos e soluções em redes de produção, mas de forma isolada, em uma rede virtual implementada através do OpenFlow (MCKEOWN et al., 2008). Redes virtuais são redes lógicas que rodam sobre roteadores e *switches*, mas com uma configuração lógica independente da disposição física dos equipamentos. Diversas redes virtuais podem coexistir sobre uma mesma infraestrutura física, compartilhando o mesmo *hardware*, mas totalmente isoladas logicamente.

OpenFlow não é o único meio de implementar uma rede virtual, outras técnicas como VLAN ou GRE, por exemplo, são utilizadas para configuração de diferentes caminhos lógicos completamente isolados. O benefício que OpenFlow provê à virtualização de redes é a forma como as redes podem ser criadas, modificadas ou eliminadas, ou seja, através do controlador e das aplicações, e não através de configuração equipamento a equipamento como é a forma de configuração tradicional. Um exemplo de ferramenta para virtualização de redes através do OpenFlow é a FlowVisor que configura *flow entries* que garantem o isolamento de cada rede virtual e os recursos mínimos para cada uma; FlowVisor é chamada de proxy OpenFlow pois realiza o encaminhamento de pacotes de cada rede virtual para seu respectivo controlador isolando, inclusive, o controle da rede (SHERWOOD et al., 2009).

2.3.2 Roteamento

No campo dos protocolos de roteamento, o projeto RouteFlow[16] utiliza tabelas de roteamento (*Routing Information Base — RIB*) e encaminhamento (*Forwarding Information Base — FIB*), estabelecidas através de protocolos como BGP ou OSPF, para a configuração de *switches* OpenFlow. O RouteFlow utiliza um conjunto de VMs Linux rodando *software*s como o Quagga[17] que implementam protocolos de roteamento; as VMs são dispostas de forma que

[16]RouteFlow Project: https://sites.google.com/site/routeflow/home
[17]Quagga Routing Software Suite: http://www.nongnu.org/quagga/index.html

espelhem a disposição dos equipamentos da rede física habilitados com OpenFlow (*switches* de baixo custo, por exemplo). Quando as rotas são estabelecidas ou modificadas na infraestrutura virtual, o RouteFlow as transforma em *flow entries* e as instala nos equipamentos da infraestrutura física.

2.3.3 Quality of Service

Na arquitetura atual e comum das redes, *Quality of Service* (QoS) é provido através de duas abordagens: 1) IntServ: faz uso do protocolo *Resource Reservation Protocol* (RSVP) (RESOURCE..., 2005) para reservar recursos para o fluxo diferenciado por todo o caminho entre as duas pontas; e 2) DiffServ: baseia-se em filas de prioridade e requer funcionalidade de classificação de pacotes nos equipamentos ao longo do caminho, o qual é definido manualmente e nos quais são feitas reservas de recursos. Em ambos os casos os caminhos são persistentes até que sejam desalocados e a reserva de recursos afeta os pacotes do tráfego não privilegiado pelo QoS. Na proposta OpenQoS o tráfego multimídia é diferenciado com base nas informações dos pacotes (ex.: *Type of Service* do IPv4, classe de tráfego de pacotes IPv6 ou MPLS, endereço de servidores multimídia como origem ou portas de serviços multimídia) os quais recebem um encaminhamento diferenciado dos demais que compõem fluxos não multimídia (EGILMEZ et al., 2012). O OpenQoS provê roteamento dinâmico encaminhando os fluxos multimídia por caminhos com melhor disponibilidade e desempenho, o que não é necessariamente o mais curto. O OpenQoS também não faz reserva de recursos evitando que os demais fluxos sejam afetados, já que o tráfego multimídia é roteado por rotas garantidamente de melhor desempenho.

2.3.4 Mobilidade

Utilizando redes sem fio enquanto se movimentam, os usuários comumente tem sua experiência afetada por problemas relacionados a *handover* (quando o usuáro é transferido de um ponto de conexão para outro). Em Yap et al. (2010) os pesquisadores apresentam o OpenRoads, uma aplicação baseada em OpenFlow que evita a perda de conexão durante o *handover* enviando o mesmo pacote para mais de um *access point* onde o usuário está conectado — o cliente precisa de mais de uma placa de rede. No vídeo demonstrativo (USING..., 2010), o OpenFlow é utilizado em *access points* Wi-Fi e WiMAX controlados pelo OpenRoads que envia os mesmos pacotes para mais de um *access point* acompanhando o movimento do usuário, desta

forma, quando a conexão migrar de um *access point* para outro, os pacotes já estarão sendo enviados.

2.3.5 Gerência de Redes

O controlador OpenFlow, sobre o qual rodam as aplicações de controle da rede, possui uma visão completa da rede física e lógica, o que abre diversas possibilidades no campo de gerência. Além da definição do comportamento lógico, o OpenFlow também fornece contadores através dos quais é possível visualizar o volume de pacotes transitando em cada fluxo. Sobre a API OpenFlow dos controladores pode-se explorar o desenvolvimento de aplicações que integrem o controle e o monitoramento da rede, inclusive, utilizando outros protocolos de gerenciamento, como o *Simple Network Management Protocol* (SNMP) e técnicas de monitoramento de conexão, como o Ping.

Kim e Feamster (2013) desenvolveram uma proposta de gerenciamento baseado em SDN. Nessa proposta os autores desenvolveram módulos que utilizam SNMP, ou mesmo informações obtidas do sistema operacional, como fontes de eventos; há um módulo que processa os eventos, seleciona as políticas que devem ser aplicadas, transforma-as em *flow entries* e as envia ao controlador para que as instale nos respectivos *switches*. Nos testes realizados em alguns prédios do Georgia Institute of Technology o sistema de autenticação da rede sem fio, que inclusive realiza verificações de segurança, foi utilizado como uma fonte de eventos para determinar como os pacotes de cada usuário autenticado seriam encaminhados, ou se seriam bloqueados.

2.3.6 Network Function Virtualization

A proposta de *Network Function Virtualization* (NFV) é separar funções e serviços de rede (ex.: NAT, DHCP, DNS) do *hardware* e de *appliances* proprietárias e entregá-las através de uma infraestrutura virtualizada (servidores, *storages*, redes) visando redução de custos, simplificação da infraestrutura e seu gerenciamento, acelerar a entrega de serviços e escalabilidade (WHAT..., 2015).

Devido a problemas relacionados ao gerenciamento de infraestrutura de redes, como por exemplo os custos de soluções proprietárias, questões de espaço físico em *datacenters* e recursos para alocação de *hardware* e manutenção especializada, empresas de telecomunicações se

uniram em 2012 e elegeram o *European Telecommunications Standards Institute* (ETSI) para trabalhar na padronização de NFV (NETWORK. . . , 2015). Dentre os desafios de padronização do ETSI para NFV estão a simplificação da operação das plataformas de virtualização, portabilidade e performance das *appliances*, coexistência com *hardware* e plataformas legadas, gerenciamento e orquestração de *appliances*, que consiste em gerenciar *appliances*, roteamento e encaminhamento a fim de atender as necessidades de cada ambiente (NETWORK. . . , 2014).

Dos desafios citados acima, o último, gerenciamento e orquestração, é o que relaciona NFV a SDN. NFV e SDN são independentes, contudo podem muito bem se complementar devido à programabilidade de SDN que pode prover, por exemplo, balanceamento de carga entre funções virtualizadas e a programação do plano de dados para atender os requisitos de NFV (ROSA et al., 2014).

2.4 Trabalhos relacionados: SDN no Contexto de IoT e de Redes Domésticas

Assim como os trabalhos mencionados anteriormente neste capítulo, os trabalhos abordados nesta seção também são exemplos da utilização de SDN, porém se relacionam de forma mais direta à proposta desenvolvida nesta dissertação por abordarem questões de IoT ou por serem soluções desenvolvidas em ambientes de redes domésticas, apesar de abordarem problemas e cenários diferentes.

A utilização dos paradigmas SDN e IoT em um mesmo contexto é recente. Os ambientes de rede de IoT possuem algumas características que fazem com que protocolos e mecanismos comumente utilizados em redes de computadores não se mostram adequados, como será apresentado no capítulo 3. A possibilidade de programar a rede através de *software* viabilizada por SDN, utilizando linguagens de alto nível, permite lidar com as especifidades dos dispositivos de IoT, dos cenários e das aplicações que se deseja desenvolver valendo-se da flexibilidade das linguagens de programação para implementar um processamento lógico, ao invés da rigidez muitas vezes presente nos protocolos de rede.

Qin et al. (2014) propõem uma arquitetura onde diferentes serviços cooperam entre si. O trabalho é baseado no cenário de uma rodovia com automóveis conectados, câmeras de monitoramento, e serviços providos pela administração da rodovia. Os automóveis recebem em tempo real notificações a respeito do tráfego, ocorrência de acidentes e sugestões de plano de viagem, principalmente para carros elétricos para os quais o plano de viagem é elaborado levando em conta a localização dos postos de recarga. Além de receberem dados dos serviços da rodovia, os automóveis também fornecem dados que permitem identificá-los. Os dados fornecidos pe-

los automóveis são armazenados no banco de dados e quando necessários são utilizados pelos serviços da rodovia. Essa proposta pode ser ilustrada através do seguinte exemplo: uma pessoa em um ponto da rodovia solicita um táxi, a informação é repassada ao sistema de controle que localiza um ou mais táxis no banco de dados; através dos dados de geolocalização, atualizados pelo próprio automóvel e enviados para o serviço de câmeras, o táxi selecionado é geograficamente e visualmente localizado e o solicitante acompanha, em tempo real, o deslocamento do veículo até o seu embarque. Outro exemplo seria descobrir quantos veículos se encontram nas estações de abastecimento, o que pode envolver o serviço de câmeras, os dados enviados pelos veículos e um processamento interno para manter esta informação atualizada.

No trabalho de Qin et al. (2014), SDN contribui na determinação de rotas e redes que serão utilizadas para alcançar os serviços apropriados. Em redes de IoT a preocupação, no que se refere ao desempenho da rede, está mais relacionada à *delays*, *jitter*, perdas de pacotes e *throughput* devido às características de comunicação dos dispositivos e seus pacotes. Através de SDN pode-se realizar a escolha do link mais adequado para cada fluxo buscando garantir um fluxo contínuo, com poucas ou nenhuma perda.

Chetty e Feamster (2012) e Yiakoumis et al. (2011) exploram SDN em cenários de rede domésticas. Mesmo que seus trabalhos não abordem diretamente as questões de IoT, servem como base por lidarem com questões específicas de redes domésticas fazendo uso dos recursos de SDN. Ambas propostas exploram as possibilidades de SDN para refatorar alguns pontos da rede.

O trabalho de Chetty e Feamster (2012) foca em mover a complexidade da rede para longe dos usuários finais. Muitos usuários não configuram suas redes adequadamente devido à falta de conhecimento para lidar com termos técnicos como tipos de pacotes, tipos de ataques, *Maximum Transmission Unit* (MTU), dentre outros. As interfaces de administração dos *access points* comumente utilizados em redes domésticas não costumam fornecer informações suficientes que permitam a um usuário sem o conhecimento técnico necessário configurar sua rede propriamente para atender suas necessidades e mantê-la segura. Os autores propõe que as opções de configuração sejam apresentadas através de uma interface que abstraia os detalhes técnicos e apresente informações facilmente compreensíveis. SDN é utilizado para traduzir as opções do usuário, realizadas através de uma interface administrativa do *access point* reformulada, em *flow entries*.

O trabalho de Yiakoumis et al. (2011) propõe a divisão da rede doméstica em diferentes "fatias" (redes virtuais), desta forma provedores de serviços, tais como banda larga, energia elétrica, gás e possivelmente outros serviços utilizariam, cada um, uma rede virtual diferente

para entregar seus serviços utilizando a mesma infraestrutura física. O objetivo dessa proposta é a redução de custos, tanto para o cliente quanto para o provedor, pois reduz e simplifica a infraestrutura necessária para que os provedores entreguem seus serviços.

Um serviço não interfere no outro, pois cada rede virtual é completamente isolada. No cenário apresentado, o isolamento é uma característica essencial. Cada "fatia" é controlada pelo provedor de serviço que a utiliza. Além do isolamento, para cada rede virtual é garantida banda, controle independente e a possibilidade de ser modificada e customizada pelo provedor de serviço.

Assim como SDN, IoT é um campo de pesquisa relativamente novo e vem evoluindo e sendo ampliado constantemente. IoT possui diversos problemas comuns a qualquer ambiente de rede, mas também problemas específicos relacionados às particularidades dos dispositivos empregados nesse paradigma. O capítulo 3 explora IoT em maior profundidade dando uma visão mais completa dos problemas existentes, parte dos quais foram encontrados no projeto REMOA e são abordados neste trabalho através de SDN.

3 INTERNET OF THINGS

O crescimento do número de uma grande variedade de dispositvos (*things*) em ambientes de redes, nos mais diversos contextos vem provocando mudanças na forma como lidamos com redes de computadores. IoT vem crescendo, não apenas em número, mas também na diversidade dos dispositivos, seja no formato, finalidade, recursos, forma de funcionamento, etc. SDN aparece como uma alternativa viável para lidar com a heterogeneidade de IoT tornando a rede uma plataforma para inovação onde serviços e funcionalidades são desenvolvidos e implantados na infraestrutura de rede de forma ágil beneficiando administradores de rede e usuários finais (WEE, 2014; CARAGUAY et al., 2014).

Este capítulo inicia apresentando a origem de IoT e percorre a literatura relacionada ao tema buscando uma definição adequada para o que se apresenta como IoT nos dias atuais. A seção 3.2 apresenta alguns exemplos de diferentes aplicações e cenários de IoT. As *things* diferem bastante dos dispositivos que estávamos acostumados a encontrar em uma rede, como PCs, servidores e notebooks, para os quais há soluções e mecanismos de gerência, segurança, interconexão, dentre outros, bem estabelecidos; a seção 3.3 aborda as questões que vem sendo consideradas como desafios e sobre as quais pesquisas têm sido desenvolvidas no campo de IoT. A seção 3.4 apresenta diversos trabalhos desenvolvidos com foco nos problemas encontrados nos dispositivos com restrições de recursos e suas redes. A seção 3.5 apresenta propostas de *middleware* desenvolvidos para contornar problemas de IoT ou prover mecanismos para compensar as limitações dos dispositivos.

3.1 Origem e Definição

Internet of Things é um termo que vem sendo utilizado em uma grande diversidade de cenários. As aplicações atuais do termo *Internet of Things* diferem muito do seu contexto original, quando Kevin Ashton o utilizou para se referir ao rastreamento de objetos em uma cadeia de produção através de *tags* RFID (ASHTON, 2009). A cunhagem do termo é atribuída a Ashton em alguns trabalhos, em outros ao Auto-ID Labs[1] (ATZORI; IERA; MORABITO, 2010; LIU; ZHOU, 2012; MATTERN; FLOERKEMEIER, 2010; GAMA; TOUSEAU; DONSEZ, 2012), o qual foi responsável pela consolidação de IoT como objeto de pesquisa.

O Auto-ID Labs consiste de uma rede de sete laboratórios de pesquisa, de sete universidades diferentes espalhadas pelo mundo, que pesquisam, propõem e desenvolvem novas

[1] Auto-ID Labs: http://www.autoidlabs.org

tecnologias e padrões trabalhando em consonância com o GS1[2], uma associação internacional, sem fins lucrativos, que desenvolve padrões e soluções que tornam cadeias de produção mais eficientes e visíveis. Até então as pesquisas do Auto-ID Labs focaram no GS1 EPCglobal, solução que permite o rastreamento de objetos em uma cadeia de produção através de um identificador único para cada objeto, o Electronic Product Code (EPC), armazenado em uma *tag* RFID (ELECTRONIC..., 2007). Cada EPC é associado em um conjunto de dados que permitem visualizar todo o trajeto de um objeto, desde sua produção até o ponto de vendas, tornando-o parte de um sistema amplo e com inteligência.

Em seu último relatório anual, o Auto-ID Labs percebeu que é necessário ir além do ponto de vendas acompanhando a expansão de IoT para novos cenários e novas tecnologias (AUTO-ID..., 2013). A proposta para os próximos dois anos é preparar a arquitetura GS1 (padrões, interfaces, definições, ferramentas e serviços) para os desafios da nova geração, proposta do GS1 Digital, produto no qual o Auto-ID focará seus trabalhos. O objetivo do Auto-ID Labs trabalhando sobre o GS1 Digital é o desenvolvimento da arquitetura visando prover visibilidade *online* aos objetos identificados através de soluções baseadas em nuvem e o uso de padrões *web*, além de garantir uma liderança inovadora no campo de IoT.

A possibilidade de utilizar diferentes tecnologias de comunicação de rede em dispositivos com restrições de recursos, e também de tamanho, expandiu o termo *Internet of Things* para outros cenários baseados em tecnologias diferentes do RFID. A utilização de RFID deixou de ser um sinônimo para IoT, deixou de ser um centro e passou a integrar soluções compostas por diferentes tecnologias. A utilização de padrões *web* e de acesso a dados bem definidos, a exemplo da proposta do GS1 Digital, possibilita que diversos sistemas baseados em diferentes tecnologias se comuniquem e integrem seus dados ampliando a visão *online* de um determinado ambiente.

De acordo com Atzori, Iera e Morabito (2010), IoT é um paradigma resultante da convergência de três diferentes visões: Internet, abrangendo os aspectos de rede; *things*, refere-se à diversidade dos dispositivos; e a visão semântica, que compreende representação, armazenamento e comunicação dos dispositivos.

Atualmente, não há uma definição, ou limites, que determinem o que pode ser considerado *thing*. *Things* são *smart objects*, ou seja, dispositivos conectados em rede que possuem as seguintes características, segundo Miorandi et al. (2012): 1) tipo físico (tamanho, forma, etc.); 2) capacidade de comunicação em rede; 3) Possui um identificador; 4) Pode ser associado a um nome ou endereço; 5) possui alguma capacidade computacional e; 6) possui capacidade de

[2]GS1: http://www.gs1.org/about/overview

sentir o ambiente onde opera.

Das características definidas por Miorandi et al. (2012), a primeira pode ser questionada se levadas em conta as diferentes aplicações e cenários que atualmente são considerados como IoT. Em cenários baseados em RFID, ou em uma rede de sensores, os aspectos físicos, em especial o tamanho, fazem sentido, pois é utilizada a mesma tecnologia ou uma determinada quantidade do mesmo dispositivo resultando em uniformidade física. Em aplicações atuais, um cenário pode ser representado digitalmente através de informações coletadas por diferentes dispositivos, possivelmente habilitados com diferentes tecnologias, com uma grande diversidade de formas, porém com dados integrados em um único sistema.

Outra característica comumente associada a *things* é o comportamento autônomo. *Machine-to-Machine* (M2M) é o termo empregado para definir a comunicação entre um grande número de dispositivos conectados ao ambiente no qual estão inseridos que se comunicam com outros dispositivos e com serviços rodando em "nuvem" (*cloud-based*), pelos quais também são gerenciados (WU et al., 2011). A autonomia pode ser total, ou seja, os dispositivos operam sem interação humana, como sensores de temperatura, por exemplo; ou parcial, quando são operados por humanos para realizar a coleta de dados, porém conectam-se automaticamente à rede e transmitem os dados coletados. Além da comunicação, há outros aspectos da autonomia que também podem estar presente nas *things* como a da tomada de decisões, porém, muitas vezes isto é papel da inteligência da solução à qual as *things* estão integradas — possivelmente um equipamento com maior capacidade computacional ou serviços executando remotamente.

Somando a comunicação autônoma às características elencadas por Miorandi et al. (2012), porém sem as restrições de tamanho, tem-se uma visão de *things* mais próxima a de *smart objects* do que *tags* RFID. O conceito atual de *things*, considerando as diversas aplicações de IoT, são *smart objects* que coletam informações sobre o ambiente no qual estão inseridos e as enviam para os serviços aos quais estão integrados.

Considerando que o termo *Internet of Things* não está mais associado de forma restrita a uma tecnologia, a um protocolo ou a uma gama limitada de aplicações, a definição mais adequada para este paradigma é representação digital do mundo físico feita através de dados enviados por dispositivos com a capacidade de sentir o ambiente (MIORANDI et al., 2012). Com o conceito de *thing* abrangendo um conjunto de dispositivos muito mais amplo, mais esforços se fazem necessários para interconectar tal diversidade e promover a adoção de padrões abertos e bem definidos para acesso aos dados.

Cabe salientar a crítica forte de Negroponte à forma como of termo *Internet of Things* é às vezes utilizado. O autor se refere às situações em que a inteligência e a interface dos dis-

positivos são transferidas para *smartphones*, *tablets* ou para a *web*, ou seja, para mais próximo do usuário e então empregam o termpo *Internet of Things*, o que Negroponte classifica como "tragicamente patético" (NEGROPONTE, 2014). O autor explica sua opinião com o seguinte exemplo do que seria a utilização adequada do termo: colocar um frango em um forno inteligente que reconhece o usuário e prepara o alimento de acordo com suas preferências. A visão de Negroponte salienta a capacidade e a importância de sentir o mundo físico, assim como apontado em Miorandi et al. (2012).

3.2 Aplicações

Nos últimos anos viu-se o surgimento e crescimento das plataformas de prototipação como Arduino[3], Raspberry Pi[4] e BeagleBone[5], dentre outras, com as quais desenvolvedores puderam implementar aplicações e soluções para diversas finalidades fazendo usos de sensores acoplados ao *hardware* na forma de módulos de baixo custo. Em paralelo, diversas empresas desenvolveram seus próprios dispositivos, com projetos de *hardware* específico para atender finalidades específicas, que compõem os serviços que comercializam. Somam-se a esses dispositivos *smartphones*, *tablets* e mesmo celulares mais simples, desde que suportem a instalação de aplicativos, através dos quais é possível capturar informações de ambientes amplos, como uma cidade, ou mesmo dados de saúde de uma pessoa que está fora de um ambiente de monitoramento controlado.

Nesta seção serão apresentadas algumas aplicações e serviços associados a IoT, acompanhados de alguns exemplos que proporcionam uma visão mais clara de como diferentes aplicações vem sendo exploradas. Serão abordadas diferentes finalidades e cenários, dispositivos completamente autônomos e com interação humana. As aplicações apresentadas abaixo reafirmam a definição de IoT como a representação digital do mundo físico, e sobre esta representação são desenvolvidos serviços. A diversidade de aplicações também demonstra que o termo *Internet of Things* vem sendo aplicado como um termo "guarda-chuva", fato propiciado por não haver um consenso a respeito da definição do paradigma e seus limites.

[3] Arduino: http://www.arduino.cc
[4] Raspberry Pi: http://www.raspberrypi.org
[5] BeagleBone: http://beagleboard.org

3.2.1 Healthcare

No campo de *healthcare*, a possibilidade de sentir o ambiente e pessoas de forma autônoma, pervasiva e não intrusiva vem sendo bastante explorada com o objetivo de prevenir doenças, acompanhar remotamente o tratamento e recuperação de pessoas e monitorar idosos. Outro aspecto que também está contido em *healthcare*, porém não associado a doenças ou limitações, é o cuidado com saúde e bem-estar, como controle de peso, alimentação e monitoramento de atividades físicas.

Os aspectos de saúde relacionados a doenças crônicas mais explorados são pressão arterial, batimento cardíaco e nível de glicose. Nos primeiros dispositivos conectados de *healthcare* o objetivo era permitir ao paciente uma coleta e armazenamento por meio digital dos seus dados e não mais uma coleta visual que deveria ser informada manualmente a um *software* para então ser enviada por meio digital a um médico. MyGlucoHealth[6] foi o primeiro dispositivo autorizado pela *U. S. Food and Drug Adminitration* (FDA) que realiza medições de glicose e as transmite via Bluetooth — os dados também podem ser coletados via USB. A conexão Bluetooth é estabelecida com um *smartphone* rodando um aplicativo disponibilizado pela empresa fabricante, ou mesmo um aparelho celular mais simples, desde que suporte a instalação de aplicativos de terceiros. Através do aplicativo, o usuário pode compartilhar seus dados com os responsáveis pelo seu tratamento que os acessam através de uma ferramenta *online*, também desenvolvida e disponibilizada pela fabricante. Além do monitor de glicose, a solução de *healthcare* desenvolvida pela fabricante do MyGlucoHealth é composta por monitor cardíaco, monitor de oximetria, monitor de atividades, monitor de pressão arterial e balança.

Na mesma linha do MyGlucoHealth, a Withings[7] desenvolveu uma solução semelhante. Os dispositivos da Withings possuem um modo de operação mais autônomo no que se refere ao envio dos dados. Uma vez configuradas as opções de conexão à rede Wi-Fi, os dispositivos enviam os dados através da rede para um servidor da empresa fabricante sempre que algum dado for coletado. A Withings disponibiliza aos seus usuários um sistema *online*, bem como aplicativos para dispositivos móveis, onde o usuário pode associar seus dispositivos à sua conta e visualizar seus dados, além de outros recursos, como por exemplo o estabelecimento de metas de perda de peso.

Ambas soluções mencionadas possibilitam a integração com serviços de terceiros, como Microsoft HealthVault[8], por exemplo, que armazena e processa em um único sistema os dados

[6]MyGlucoHealth Wireless Meter: http://www.myglucohealth.net/src/wireless.html
[7]Withings: http://www.withings.com
[8]MS HealthVault: www.healthvault.com

coletados através de diferentes soluções.

3.2.2 Cidades Inteligentes

Através da coleta e cruzamento de uma diversidade de dados coletados em uma cidade, governos e empresas passam a dispor de um grande volume de informação para embasar tomadas de decisões relacionadas à provisão de serviços públicos e privados para o benefício dos cidadãos e também para a melhoria de serviços existentes. O Waze[9], serviço de GPS colaborativo disponível gratuitamente, é um exemplo do uso das informações coletadas pervasivamente para prover melhores serviços, pois utiliza as informações de deslocamento dos usuários, coletadas através do *smartphone*, para fornecer melhores sugestões de rota, o que se aplica especialmente em ambientes urbanos onde há contratempos como congestionamentos e acidentes.

Um dos fatores importantes da mobilidade urbana, em especial no deslocamento de meios de transporte, é o estacionamento, pois é responsável por 30% do congestionamento de tráfego e poluição em grandes centros (SHOUP, 2007). Lamba (2013) relata a parceria entre IBM[10] e a Streetline, desenvolvedora do aplicativo Parker[11], em um projeto piloto desenvolvido na cidade de Birmingham, Inglaterra, para orientar motoristas sobre vagas de estacionamento disponíveis. O objetivo do projeto é reduzir o congestionamento que vem afetando a economia local, para isso foram instalados no chão sensores de baixo consumo em alguns quarteirões que detectam a presença de automóveis estacionados, os motoristas que desejam estacionar podem verificar a disponibilidade de vagas através do aplicativo. Os dados coletados pela solução também são utilizadas para análise de tempo de ocupação, identificação de áreas mais ocupadas e de tendências de ocupação em determinados locais.

Outro exemplo ainda relacionado a transporte foi a utilização pelo governo de Dublin, Irlanda, do grande volume de dados gerados por câmeras, sensores de chuva, tráfego, GPS, etc., para a identificação de fatores que prejudicam a mobilidade urbana, como congestionamentos, de forma que seja possível antecipar-se à ocorrência de tais eventos (DUBLIN. . . , 2013). Em paralelo, os dados coletados são utilizados para prover serviços ao público, como informação em tempo real do posicionamento dos ônibus, previsão de chegada ou alterações nos sinais de trânsito. Os mesmos dados são utilizados pelos controladores de trânsito para regular o fluxo dos ônibus em suas respectivas linhas.

[9] Waze: www.waze.com
[10] IBM: http://www.ibm.com
[11] Streetline Parker: http://www.streetline.com/find-parking/parker-mobile

Aplicações de IoT em ambientes urbanos vão bem além do contexto de mobilidade. A utilização de sensores em ambientes residenciais e corporativos racionaliza o consumo de recursos como água e energia. É através do uso de sensores que se pode quantificar, de forma detalhada e em tempo real, e visualizar o consumo de recursos identificando excessos e desperdícios, bem como antever alterações no padrão de demanda através do monitoramento da qualidade do ar e do clima, por exemplo.

Além dos serviços providos pelo poder público, ou por empresas privadas, novos serviços podem ser desenvolvidos por qualquer cidadão se os dados coletados em uma cidade forem disponibilizados abertamente. Em Londres, Inglaterra, dados como o número de pessoas que transita na cidade, qualidade do ar, localização de escolas, número e localização de residências em construção, taxa de criminalidade, dentre outros, são disponibilizados abertamente pelo governo municipal, o qual incentiva o desenvolvimento de aplicações que façam uso desses dados (ACCESS..., 2014).

Há também indivíduos, além do governo e de empresas, que desejam disponibilizar os dados de seus dispositivos para que outros façam uso. A Umbrellium[12] desenvolveu um mecanismo de busca para IoT onde indivíduos e empresas podem cadastrar dispositivos e serviços relacionados a IoT que estão disponíveis para uso público, o Thingful[13]. No Thingful cada dispositivo é localizado em um mapa, além da localização também estão disponíveis informações sobre o proprietário, a finalidade do dispositivo e como e porquê utilizá-lo.

3.2.3 Residências Inteligentes

Dos diferentes nichos de mercado onde o termo *Internet of Things* tem sido utilizado, o de automação residencial é o que mais tem se destacado com cada vez mais produtos. A segurança foi um dos aspectos que mais impulsionou o desenvolvimento de sensores para detecção de pessoas. Nas soluções atuais, dispositivos como portas, fechaduras e aparelhos utilizados em casa podem estar conectados e reportar seu estado através da rede, tornando-se assim, além de um dispositivo para uma determinada finalidade, um sensor.

O termostato Nest[14] é um exemplo de dispositivo, ou sistema, que vai além do funcionamento comum de soluções para controle de temperatura, pois possui a capacidade de aprender as preferências do usuário em determinados horários, economiza energia quando a casa está

[12]Umbrellium: http://umbrellium.co.uk
[13]Thingful: http://thingful.net
[14]Nest Labs: http://nest.com

vazia e se adapta às mudanças de clima, pois está sempre aprendendo através dos sensores de temperatura e das configurações realizadas. As configurações do Nest também podem ser controladas remotamente através do aplicativo para dispositivos móveis e de outros dispositivos com os quais pode ser integrado, como o monitor de atividades que detecta quando alguém acordou, o computador de bordo de um automóvel que comunica quando algum dos residentes está indo para casa e com outros dispositivos cujos fabricantes tem parceria com o Nest.

Para o monitoramento de jardins são comuns sensores de umidade que alertam para a necessidade de irrigação. Indo além, em especial por cruzar diferentes informações, o Edyn[15] se propõe a fazer análise da composição do solo sugerindo quais plantas, flores ou hortaliças são mais adequadas àquele terreno, também analisa os níveis de umidade do solo e consulta informações meteorológicas *online* para controlar a irrigação evitando desperdícios ou falta de umidade. As informações coletadas pelo Edyn, bem como suas sugestões, podem ser acessadas através de um aplicativo para dispositivos móveis. Outro aspecto interessante desta solução é que a bateria dos dispositivos é recarregada por energia solar.

Há uma grande diversidade de produtos eletrônicos para a automação residencial, tais como sistemas de iluminação, cortinas, fechaduras de portas e janelas, etc., porém muitos deles não possuem inteligência ou sequer compõem um sistema que provenha inteligência, apenas estão conectados a uma rede e podem ser controlados remotamente. Soluções como Staples Connect[16] e Ninja Sphere[17] se propõem a conectar uma diversidade de dispositivos, mesmo que tenham inteligência, provendo uma forma centralizada e integrada de administrá-los e inteligência ao ambiente residencial como um todo cruzando informações do ambiente com outras obtidas itonline (meteorológicas, por exemplo). Utilizando estas soluções, ao sair de casa o controle de temperatura é ajustado ou desligado, as janelas e portas são trancadas e as luzes e aparelhos são desligados.

Como mencionando na seção 3.2.2, IoT pode contribuir no consumo racional de recursos. Em uma residência conectada a uma *smart grid*, o sistema de automação residencial pode receber informações dos provedores de energia, água e gás a respeito da disponibilidade e do consumo; dispondo desse tipo de informação, o sistema de automação residencial pode alterar o seu funcionamento a fim de economizar recursos e alertar os moradores da residência.

[15]Edyn: http://www.edyn.com
[16]Staples Connect: http://www.staples.com/sbd/cre/marketing/staples-connect/
[17]Ninja Blocks: https://ninjablocks.com

3.2.4 Esportes e Fitness

Os *smartphones* contendo recursos como GPS e acelerômetro tornaram-se um instrumento amplamente utilizado para monitoramento do desempenho pessoal em atividades físicas. Os recursos de monitoramento nesse tipo de atividades pode ainda ser ampliado se utilizados outros dispositivos que se conectam ao *smartphone* e fazem leituras, por exemplo, de batimentos cardíacos, temperatura corporal, níveis de transpiração.

A comunicação entre o *smartphone* e um monitor é feita através de protocolos de baixo consumo como Bluetooth ou ANT+. O H7 Heart Rate Sensor[18], por exemplo, estabelece conexão Bluetooth com o *smartphone* e os dados coletados são exibidos em tempo real através de um aplicativo desenvolvido pela própria fabricante ou através de aplicativos compatíveis, como o RunKeeper[19]. O mesmo pode ser feito com outro monitor cardíaco, o Meet Tickr[20], contudo, este inclui funcionalidades como estimativa de queima de calorias e monitoramento de cadência de passos e pedaladas. Assim como o H7, a fabricante do Meet Tickr disponibiliza aplicativo para *smartphone*, mas também é compatível com aplicativos de terceiros (ex.: RunKeeper).

As fabricantes de *smartphones* estão atentas ao mercado *fitness*. Samsung e Apple lançaram *smart watches* (Apple Watch[21] e Samsung Gear Fit[22]) que, além da função de relógio, permitem acessar alguns aplicativos e funções do *smartphone*, como clientes de e-mail e atender ligações, em um dispositivo utilizado como relógio de pulso. Os dispositivos de ambas as fabricantes possuem sensores cardíacos embarcados e implementam funcionalidades que vão além do monitoramento de batimentos do coração, como estimativa de queima calórica e análise de desempenho de atividades físicas. Outras empresas como a Basis[23] e Fitbit[24] produzem dispositivos semelhantes que também utilizam *smartphones* como *gateway* para o envio e visualização dos dados coletados.

Embora todos os exemplos citados possuam algum nível de automação, o dispositivo Peaks, da Basis, é um bom exemplo onde a inteligência foi mantida no dispositivo. O Peaks detecta automaticamente se o usuário está caminhando, correndo, pedalando ou dormindo e realiza as análises de acordo com o tipo de atividade física cruzando os dados de batimento cardíacos, temperatura corporal e transpiração.

[18] H7: http://www.polar.com/en/products/accessories/H7_heart_rate_sensor
[19] RunKeeper: http://runkeeper.com
[20] Meet Tickr: http://www.wahoofitness.com/devices/hr.html
[21] Apple Watch: https://www.apple.com/watch/
[22] Samsung Gear Fit: http://www.samsung.com/br/consumer/cellular-phone/convergence/rel%C3%B3gios-inteligentes/SM-R3500ZKLZTO
[23] Basis: https://www.mybasis.com
[24] Fitbit: http://www.fitbit.com

3.3 Questões em Aberto e Desafios de IoT

3.3.1 Silos

A diversidade faz parte de IoT em diversos aspectos, tais como *hardware*, finalidade, modo de operação, performance ou mesmo em padrões de dados e comunicação. Embora diferentes padrões de comunicação e de dados possam coexistir em um mesmo ambiente de rede, para que os dispositivos se comuniquem ou compartilhem dados entre diferentes soluções, é necessário que os padrões utilizados sejam abertos. Utilizando padrões abertos, mesmo que não compatíveis, é possível implementar ou utilizar mecanismos que portem os dados entre diferentes padrões para diferentes dispositivos e soluções.

Contudo, o que se observa em grande parte dos dispositivos disponíveis para compra, de aplicações com características de IoT, são dados fechados em soluções proprietárias ou compartilhados apenas com soluções de outros fabricantes, normalmente atendendo interesses de mercado. Mesmo quando os dados são compartilhados entre soluções proprietárias, a comunicação é vertical, ou seja, os dados que são coletados em um determinado ambiente são enviados ao servidor da solução a partir de onde são compartilhados com uma solução parceira. Na literatura de IoT essa forma de comunicação é chamada de *silo* (IOT-A. . . , 2013; COETZEE; EKSTEEN, 2011; TECHNOLOGY. . . , 2013; WU et al., 2011).

Os dispositivos que fazem parte de *silos* têm sua forma de operação programada *hard coded* de fábrica, ao usuário é possível apenas realizar configurações como da conexão à rede Wi-Fi, adicionar informações sobre si mesmo ou relacionadas à finalidade do dispositivo, como por exemplo, na Withings Scale[25] o usuário pode alterar a medida utilizada para o peso. Ao usuário não é permitido alterar configurações relacionadas a forma de operação e comunicação do dispositivo, tais como para qual servidor os dados serão enviados, o protocolo utilizado ou configurações de acesso ao dispositivo. Qualquer alterações nessas configurações, se realizadas, são feitas através de atualização de *firmware*.

Há, contudo, iniciativas para a padronização da comunicação de IoT de forma que as *things* consigam se interconectar. No último ano duas iniciativas surgiram, são dois consórcios com diferentes grupos de membros: a Allseen Alliance (TECHNOLOGY. . . , 2013) e o Open Interconnect Consortium[26]. Apesar da iniciativa ser interessante ainda não se sabe se os membros vão adotar os padrões em todos os seus produtos e se abrirão mão de seus padrões bem

[25] Withings Scale: http://www.withings.com/us/ws-30.html
[26] Open Interconnect Consortium: http://openinterconnect.org/

definidos e operantes. Outra questão é se esses dois consórcios serão, ou não, dois grandes *silos*.

3.3.2 Gerênciamento de Dispositivos

A seção 3.1 abordou a definição de IoT e de *things* e, como mencionado, os dispositivos deste paradigma possuem características específicas. Apesar de aspectos como forma e tamanho não serem determinantes para considerar um dispositivo como *thing*, ou não, muitos dispositivos utilizados em cenários de IoT são pequenos e seu *hardware* foi desenvolvido de forma muito sucinta, o que implica limitações de processamento, armazenamento e memória, alimentação e autonomia e restrições de recursos como os de segurança, privacidade e gerenciamento.

Em muitas redes a gerência da rede e dos serviços é feita através de *software*s cujo monitoramento é baseado em SNMP e ICMP. Ambos os protocolos requerem conectividade, o SNMP requer mais processamento e capacidade de armazenamento de objetos em uma MIB. Contudo, muitos equipamentos com restrições de bateria operam em baixo *duty cycles*, ou seja, mantem-se em *stand by* para economizar energia até que sejam utilizados, quando são reativados, transmitem os dados coletados e voltam ao modo *stand by*.

Por se manterem em *stand by*, o gerenciamento baseado em SNMP ou ICMP acusaria os dispositivos como indisponíveis em grande parte das tentativas de acesso ou testes de conectividade. IoT requer mecanismos de gerenciamento flexíveis possíveis de serem adaptados à diversidade dos dispositivos e seus modos de operação.

3.3.3 Endereçamento e Identificação

Miorandi et al. (2012) afirmam que as *things* devem ser unicamente identificadas, por exemplo, através de uma *string*, para que possam ser rastreadas ou acessadas. Contudo, em ambientes de rede onde mais de uma tecnologia de comunicação em rede está em uso, o desafio é fazer com que a identificação possa ser utilizada por todas as tecnologias.

A identificação dos dispositivos conectados a uma rede passa, primeiramente, por um endereço físico da tecnologia de comunicação utilizada. Os dispositivos conectados em uma rede não são acessíveis por dispositivos de tecnologias não compatíveis sem que um elemento os interconecte. Portanto, para a comunicação entre diferentes tecnologias, em IoT, além das camadas de rede e de seus respectivos protocolos, requer uma camada, como um *middleware*, que habilita a comunicação entre dispositivos que não podem se comunicar diretamente.

3.3.4 Semântica

As questões de acesso aos dados em IoT tem início nos problemas de interconexão para coleta em tempo real, pois há fatores dificultantes como a forma de operação com baixos *duty cycles* ou a ausência de funcionalidades que permitam que o dispositivo seja acessado para a coleta das informações armazenadas. Comumente as *things* enviam os dados através da rede assim que são coletados. Contudo, mesmo após a transmissão dos dados os problemas de semântica persistem na forma como esses dados são estruturados, armazenados e, posteriormente, acessados. Muitas *things* não possuem capacidade para armazenamento, muitas vezes mantém apenas o último dado lido. Portanto, em aplicações que requerem a persistência de dados, é necessário o uso de servidores proprietários, pessoais ou públicos onde os dados serão armazenados e disponibilizados.

Tão importante quanto a padronização da interconectividade entre dispositivos, é a padronização de estruturas para os dados e a forma de acesso, a ausência de tais padrões pode fazer com que IoT não se realize em alguns cenários (MIORANDI et al., 2012). Um exemplo da importância desse aspecto é a falta de estrutura em dados de saúde, estima-se que 80% desses dados estejam desestruturados implicando na necessidade de se realizar mineração em grandes volumes de informação para que se possa prever as futuras situações possíveis na saúde dos pacientes monitorados (HINSSEN, 2012).

3.3.5 Segurança e Privacidade

O principal problema relacionado à privacidade e segurança em IoT vem da limitação de recursos das *things* que impossibilitam que os dispositivos processem mecanismos de criptografia e autenticação (BANDYOPADHYAY; SEN, 2011). Em aplicações de IoT como *healthcare* ou automação doméstica a exposição dos dados ou a manipulação destes podem resultar em risco de danos como roubos a residências ou mesmo a morte de um paciente em tratamento. Além dos perigos, há também situações que não levam diretamente a um risco à integridade física de um cidadão, mas sim à sua integridade moral, como a exposição dos dados que alguém não desejaria compartilhar, por exemplo, seu peso, resultados de seus treinos, imagens de sua casa ou a sua localização.

Uma alternativa para prover segurança e privacidade em ambientes de IoT onde os próprios dispositivos não tem a capacidade para suportar mecanismos de criptografia e autenticação, é a utilização de equipamentos mais providos de recursos capazes de fornecer os mecanis-

mos necessários para proteger as *things*, como explorado em Tarouco et al. (2012). A utilização de um *firewall* é essencial para proteger a rede como um todo. Mesmo onde os dispositivos protegidos têm capacidade para processar listas bastante extensas de regras, em IoT a proteção de um *firewall* é ainda mais importante. Junto ao *firewall* podem ser utilizados outros recursos como túneis criptografados, acesso autenticado à rede, etc.

3.4 Tecnologias e Protocolos

Nos últimos anos, contribuições têm sido feitas por diferentes grupos tentando padronizar a comunicação de diferentes protocolos no contexto de IoT e possibilitar a interconexão entre diferentes tecnologias. No campo do RFID, a organização GS1 EPCglobal trabalha para promover a utilização do RFID através da adoção do *Electronic Product Code* (EPC) que consiste de um identificador único para identificação de objetos (ELECTRONIC..., 2007). Baseado no EPC, foi desenvolvido o *EPCglobal Architecture Framework* que consiste de padrões de *hardware*, *software* e interfaces para a troca de dados. O framework proposto padroniza a identificação e troca de objetos entre usuários finais, a troca de dados e as interfaces para captura e escrita de EPCs nos objetos (TRAUB et al., 2013).

IoT define um cenário de rede composto de uma diversidade de dispositivos gerando tráfego com características diferentes. O desafio está em interconectar diferentes tecnologias de rede à Internet ou entre si para que possam cooperar para uma determinada finalidade. O protocolo IP é utilizado para interconectar diferentes domínios de rede no mundo inteiro, trabalhos como Zorzi et al. (2010), Atzori, Iera e Morabito (2010), Tan e Wang (2010) e Miorandi et al. (2012) defendem a utilização do IP, no qual está baseada grande parte da infraestrutura de rede existente, como base para a arquitetura e comunicação de IoT. Contudo, a utilização de protocolos convencionais da Internet como TCP e IP encontra alguns problemas em cenários de IoT (SHELBY; BORMANN, 2009): 1) o protocolo IP é projetado para uso em dispositivos que permanecem conectados, o que não é a realidade de muitas *things*, como apresentado na seção 5.3; 2) o padrão IEEE 802.15.4, que serve como base para *Wireless Personal Area Network* (WPAN), não suporta multicast nativamente, esta funcionalidade é essencial em redes IPv6; 3) soluções de roteamento em redes *mesh* não são facilmente aplicadas; 4) redes de baixa potência possuem banda entre 20 a 250 Kbits e quadro de 127 bytes somados aos bytes de payload das informações de camada 2 (padrão IEEE 802.15.4), porém, o tamanho mínimo de um datagrama IPv6 é de 1280 bytes, com o compartilhamento do *link* a banda é reduzida, para a transmissão de pacotes IPv6 em quadros tão pequenos ocasiona muita fragmentação; 5) os mecanismos de

confiabilidade do TCP não distinguem as causas da perda de um pacote que pode ocorrer devido ao congestionamento de um *link*, perdas entre os *link*s causadas pela falha de um nó ou falta de energia, ou devido ao modo de operação em baixos *duty cycles*.

Contudo, a utilização do IP como base da infraestrutura de IoT proporciona algumas vantagens. Soluções e dispositivos baseados em IP podem ser facilmente conectados à infraestrutura existente em grande parte das redes sem a necessidade de *gateway* ou *middleware* intermediários; pode-se escalar uma solução com as facilidades providas por protocolos, APIs, soluções de gerenciamento e padrões bem definidos e bem documentados existentes (SHELBY; BORMANN, 2009). Alguns dispositivos disponíveis no mercado lançam mão dessa ideia desenvolvendo dispositivos com interface Wi-Fi ou Ethernet que serão facilmente conectados à rede doméstica do usuário final.

A utilização do IP em dispositivos com restrições de recursos era visto com reservas (DUNKELS; VASSEUR, 2010), há outras tecnologias e protocolos que lidam melhor com o consumo de energia e possibilitam a utilização de *hardware* de menor custo e tamanho. Bluetooth e ZigBee são exemplos de tecnologias para comunicação de curta distância utilizadas em dispositivos cuja finalidade é coletar dados de pessoas ou de um determinado espaço físico e enviá-los para dispositivos localizados no mesmo ambiente, como *gateways*, *smartphones*, etc.

Bluetooth é uma tecnologia auto-denominada simples, segura, robusta, de baixo consumo e barata (BLUETOOTH. . . , 2013) baseado no padrão IEEE 802.15.1, mas atualmente mantido pelo *Bluetooth Special Interest Group* (Bluetooth SIG)[27]. Qualquer dispositivo com suporte a Bluetooth pode se conectar a qualquer outro dispositivo que também suporte Bluetooth, o processo de conexão entre dispositivos é chamado de pareamento que consiste de um processo de troca de informações a respeito da conexão e utilização de senha (BLUETOOTH. . . , 2007). As redes Bluetooth são conhecidas como *piconets* que são redes *ad hoc* compostas por até oito dispositivos sendo que cada dispositivo pode se conectar a diversas *piconets* simultaneamente.

ZigBee é uma tecnologia desenvolvida pela ZigBee Alliance[28], assim como o Bluetooth, possui como características confiabilidade, baixo consumo e baixo custo para redes sem fio, contudo é baseada no padrão IEEE 802.15.4, o qual define as camadas física e MAC para WPANs de baixo custo e baixa frequência. O padrão 802.15.4 proporciona vantagens como facilidade de instalação, transmissão de dados confiável, operação de curto alcance, extremo baixo custo, considerável tempo de vida de bateria e uma pilha de protocolos simples e flexível (BARONTI et al., 2007). O foco de aplicação do Bluetooth é a substituição do cabo por uma tecnologia sem fio enquanto ZigBee foca em monitoramento e controle; uma rede ZigBee

[27] SIG: https://www.bluetooth.org
[28] ZigBee Alliance: http://www.zigbee.org

suporta mais de 65000 nós, a vida da bateria é mais longa que outras tecnologias sem fio e a comunicação pode acontecer até 100 metros de distância (ZIGBEE..., 2013a).

Bluetooth e ZigBee foram projetados visando cenários de rede que hoje são considerados IoT, o que não ocorreu com o protocolo IP. Contudo, trabalhos tem sido desenvolvidos propondo soluções baseadas em IP para dispositivos de IoT, o que proporciona padronização e facilita a integração dos dispositivos com a infraestrutura IP existente. A *IP for Smart Objects Alliance* (IPSO[29]) é uma associação sem fins lucrativos com mais de 60 membros que advoga o uso do protocolo IP em dispositivos de rede para aplicações de energia, consumo e industriais. O protocolo IP provou ser uma tecnologia de comunicação de vida longa, estável, escalável e com suporte a uma grande diversidade de aplicações, dispositivos e outras tecnologias de comunicação, além de proporcionar um acesso de rede padronizado, leve e independente de plataforma para *things* (DUNKELS; VASSEUR, 2010).

IPv6 over Low power WPAN (6LoWPAN) é uma proposta que permite que dispositivos construídos com base no padrão IEEE 802.15.4, caracterizados por curta distância, baixa frequência, baixo custo e baixo consumo, se beneficiem do uso de IPv6 (IPV6..., 2007). 6LoWPAN foi criado focando nos problemas encontrados na utilização de protocolos convencionais da Internet em ambientes IoT visando levar os benefícios do IPv6 para redes *Low power WPAN* (LoWPAN), tais como segurança (IPsec), *web services* (HTTP, SOAP, XML), gerenciamento (SNMP) e solucionar os problemas de tamanho mínimo de quadro (SHELBY; BORMANN, 2009), além de se beneficiar do grande número de endereços possíveis com IPv6.

Para possibilitar a transmissão de datagramas IPv6 sobre 802.15.4 em dispositivos com restrições de recursos, foi necessário implementar uma camada de adaptação que realiza a compressão dos cabeçalhos IPv6, a fragmentação e defragmentação dos pacotes, emulação de broadcast e suporte a protocolos de roteamento da camada 2 para redes *mesh* (SHELBY; BORMANN, 2009; MOREIRAS, 2013). A figura 3.1 ilustra a pilha IP comparando-a à pilha 6LoWPAN, na qual pode ser vista a camada de adaptação LoWPAN. A adaptação entre IPv6 e 6LoWPAN é feita pelos roteadores de borda das LoWPANs.

Recentemente a ZigBee Alliance anunciou a especificação *ZigBee IP* que define uma solução de rede *wireless* para dispositivos de baixo custo e baixo consumo baseada em IPv6, 6LoWPAN e outros protocolos tradicionais da Internet (ZIGBEE..., 2013b). De acordo com o anúncio, a solução baseada em IPv6 enriquece o padrão IEEE 802.15.4 adicionando camadas de rede e segurança e reduzindo a necessidade de *gateways* intermediários para a comunicação das *things*.

[29]IPSO: http://www.ipso-alliance.org

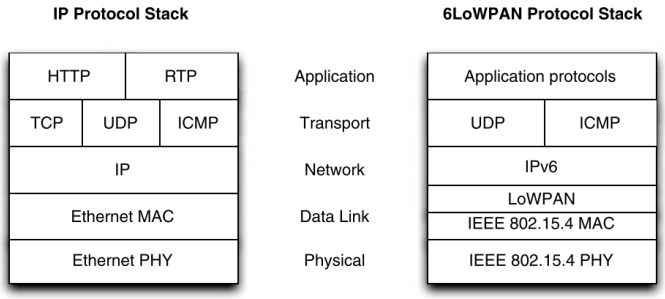


Figura 3.1 – Comparativo das pilhas dos protocolos IPv6 e 6LoWPAN (SHELBY; BORMANN, 2009)

Com foco nos problemas de roteamento encontrados em *Low power and Lossy Networks* (LLNs), onde a interconexão entre dispositivos é feita através de padrões como 802.15.1 ou 802.15.4, foi criado o grupo de trabalho *Routing over Low power and Lossy networks* (ROLL). Dentre as propostas feitas pelo ROLL está *Routing Protocol for Low-Power and Lossy Networks* (RPL) que é um protocolo de roteamento baseado em vetor distância para IPv6 que troca informações de roteamento com moderação a fim de conservar os recursos escassos dos dispositivos de LLNs (THE..., 2012).

A utilização do estilo de arquitetura *Representational State Transfer* (REST) tem se tornado cada vez mais popular na *web* por prover serviços de baixo acoplamento facilmente reutilizados que são acessados através de *Uniform Resource Identifiers* (URI) utilizando métodos HTTP (GUINARD et al., 2011). *Constrained Application Protocol* (CoAP) visa possibilitar a utilização de REST em dispositivos com restrição de recursos provendo suporte a um modelo de interação entre *endpoints* de aplicações, descoberta de recursos e serviços, URIs, multicast e tenta manter baixo *overhead* e simplicidade para evitar fragmentação (CONSTRAINED..., 2013).

Sehgal et al. (2012) estudam a utilização dos protocolos SNMP e NETCONF em dispositivos com restrições de recursos; apesar de ser possível utilizá-los, foi necessária a implementação de uma versão reduzida dos protocolos para reduzir o *overhead* sobre os dispositivos. Os testes realizados com o SNMP demonstraram uso eficiente dos recursos, o mesmo não foi constatado com NETCONF, porém os resultados podem ser melhorados com modificações na implementação. A utilização de CoAP para o transporte de mensagens de gerenciamento do NETCONF reduz o *overhead* na rede através da codificação compacta de aplicações RESTful e da utilização de *JavaScript Object Notation* (JSON) (YANG-API..., 2012). Granville et al. (2009) utilizam uma abordagem baseada em *web services* no lugar de *Script MIB* que reduz o número de mensagens de gerenciamento trocadas entre dispositivos resultando em ganho de performance.

Iniciado pelo Nest Labs, empresa adquirida pela Google em 2014, junto a parceiros

como Samsung, ARM, dentre outros, foi criado o Thread Group[30] responsável pelo desenvolvimento e promoção do protocolo Thread, projetado desde o início com foco em IoT. Thread foi desenvolvido sobre 6LowPAN e implementa mecanismos para roteamento, autenticação e economia energética. Em redes domésticas é comum os dispositivos estarem todos conectados à um roteador que acaba por se tornar um ponto de falha central da rede, o protocolo Thread foi projetado para que a rede não falhe, mesmo que o roteador venha a falhar. Em uma rede Thread, os dispositivos conectados realizam uma eleição para designar quais atuarão como roteadores e, dentre estes, um será o líder, eliminando o ponto único de falha. O protocolo também provê criptografia para uma comunicação segura. O protocolo Thread vem sendo usado em ambientes de produção nos produtos da Nest.

Outra iniciativa recente é a criação de um grupo no IEEE cujo objetivo é a padronização de um *framework* arquitetural para IoT ((STANDARD. . . , 2014). O objetivo do *framework* é desenvolver uma arquitetura de referência cobrindo abstração de dados e questões de proteção, segurança e privacidade.

3.5 Middleware

As tecnologias e protocolos apresentados na seção 3.4 abordam problemas de IoT como identificação e endereçamento, especialmente as soluções baseadas em IPv6 que reduzem a necessidade de *gateways*, pois o acesso aos dispositivos e dos dispositivos à Internet dispensa mecanismos de tradução implementados por dispositivos intermediários. Contudo, os cenários de IoT atuais ainda são repletos de *silos* e dispositivos heterogêneos onde sistemas proprietários não trocam dados entre si e os dados coletados pelos dispositivos não podem ser acessados por outros dispositivos ou serviços, exceto os do mesmo fabricante.

Para interconectar diferentes sistemas e dispositivos com diferentes tecnologias, protocolos e padrões, alguns projetos adotaram a proposta do *middleware*. *Middleware* consiste de uma ou mais camadas de *software* que interconectam sistemas e dispositivos abstraindo diferentes tecnologias e protocolos, isto permite que sejam desenvolvidas aplicações sem que o programador precise se preocupar ou conhecer os detalhes específicos dos dispositivos utilizados (ATZORI; IERA; MORABITO, 2010). Comumente o *middleware* executa em dispositivos que atuam como *gateway* e possuem as interfaces para se comunicar com as *things* e com a infraestrutura de rede conectada à Internet, o *middleware* atua viabilizando a comunicação entre as diferentes tecnologias.

[30]Thread Group: http://www.threadgroup.org/

Exemplos da utilização e propostas de middlewares podem ser encontrados em diversos trabalhos relacionados a IoT. Em Domingo (2012) são ilustrados cenários onde dispositivos móveis como *smartphones*, *tablets* e PDAs são utilizados como *gateway* e *middleware* para dispositivos de monitoramento para pessoas com deficiência e para tecnologias assistivas.

Em **??**) o *middleware* é utilizado para coletar os dados de diferentes dispositivos com diferentes tecnologias; os autores propõem uma solução chamada *RFID suite* que estende a arquitetura RFID proposta pela IBM (CHAMBERLAIN et al., 2010). O trabalho propõe a utilização de controladores de borda que, através de aplicações instaladas, coletam dados de sensores e de outros objetos e os enviam a servidores locais intermediários onde são filtrados e, em seguida, enviados a *EPC Information Services* (EPCIS) (TRAUB et al., 2013) onde são processados e armazenados.

Tarouco et al. (2012) propõem a utilização de um *middleware* em um AP para coleta de dados de saúde de pacientes com doenças crônicas e dados de gerenciamento dos dispositivos de monitoramento. Neste caso os dispositivos de telemedicina utilizam TCP/IP, portanto o *middleware* não interconecta diferentes tecnologias, o que faz é contornar problemas ocasionados pela impossibilidade de configuração dos dispositivos, tais como problemas de segurança e privacidade e a coleta dos dados.

Em IoT, especialmente antes das propostas baseada em IPv6 em WPANs, o *Middleware* foi o foco de projetos de grande escala por se tratar de um ponto central de arquitetura que interconectaria as tecnologias e, por consequência, as *things*. Abaixo são listados quatro projetos que foram de grande importância, relacionados por Miorandi et al. (2012).

- Hydra Middleware[31]: No projeto Hydra foi desenvolvido um *middleware* que permite aos desenvolvedores incorporar dispositivos com diferentes interfaces como Bluetooth, ZigBee, RFID, Wi-Fi, etc., para isso é utilizada uma abstração das interfaces físicas através de *web services*.
- RUNES Middleware[32]: O projeto RUNES foi desenvolvido em um contexto onde os dispositivos estavam diminuindo em tamanho ao passo que cresciam em poder de processamento, a preocupação se dá em interagir com dispositivos diversos. O *middleware* RUNES também aborda a abstração dos detalhes específicos dos dispositivos utilizados através de APIs e interfaces de baixo acoplamento, as quais são providas por componentes executados nos próprios dispositivos (COSTA et al., 2007).

[31] Hydra: http://www.hydramiddleware.eu/news.php
[32] RUNES: http://runesmw.sourceforge.net

- IoT-A[33]: Preocupado com os diferentes esforços para conectar dispositivos heterogêneos que acabam formando *silos*, o projeto *Internet of Things Architecture* (IoT-A) visa propor uma arquitetura para interconectar diferentes dispositivos através de uma camada de serviço. Uma atenção especial é dada a aspectos de descoberta de dispositivos e escalabilidade.
- iCore[34]: O projeto iCore também foca na abstração de objetos através de interfaces de alto nível que tornam transparente os aspectos específicos de cada dispositivo, esta abstração de objetos reais é chamada de *virtual objects*.

Em relação às questões de gerenciamento, como os empecilhos para a utilização de SNMP e NETCONF levantados por Sehgal et al. (2012), ou os relacionados ao baixo *duty cycles*, o *middleware* é utilizado para prover a gerência dos dispositivos impossibilitados de prover as informações necessárias devido às suas próprias limitações. A utilização de um *middleware* possibilita manter as informações de gerenciamento em dispositivos intermediários com melhores recursos, proposta explorada em Tarouco et al. (2012), dentro do projeto REMOA, e em Marotta et al. (2013).

Marotta et al. (2013) propõem uma solução de gerenciamento por delegação onde *things* são identificados através de protocolos convencionais, como *Link Layer Descovery Protocol* (LLDP) e *Universal Plug and Play* (UPnP), que implementam mecanismos de identificação e descoberta de vizinhança de dispositivos; uma aplicação central delega o gerenciamento ao *gateway* no qual o dispositivo está conectado que se auto configura através de *scripts* obtidos de um repositório central, com isso o *gateway* passa a se comunicar e armazenar informações de gerenciamento sobre os dispositivos monitorados.

É possível perceber nos trabalhos citados, especialmente naqueles que desenvolvem propostas de *middleware*, uma arquitetura composta por pelo menos quatro camadas, conforme ilustrado na figura 3.2: 1) *Things*: todo e qualquer dispositivo conectado que componha à solução implantada em um determinado ambiente; 2) *Edge device* (Dispositivo de borda): são *gateways* nos quais os dispositivos da camada 1 se conectam, esta camada interconecta as tecnologias de comunicação dos dispositivos da primeira camada com a camada seguinte, o papel desta camada é encarregar-se dos aspectos de rede; 3) *Middleware*: esta camada fornece uma interface de alto nível abstraindo os aspectos específicos das tecnologias utilizadas nos dispositivos da camada 1, isso permite acesso aos dados ou aos dispositivos sem a preocupação com detalhes técnicos; e 4) *Application* (Aplicação/Serviço): recebe, processa, persiste e disponibi-

[33]IoT-A: http://www.iot-a.eu
[34]iCore: http://www.iot-icore.eu

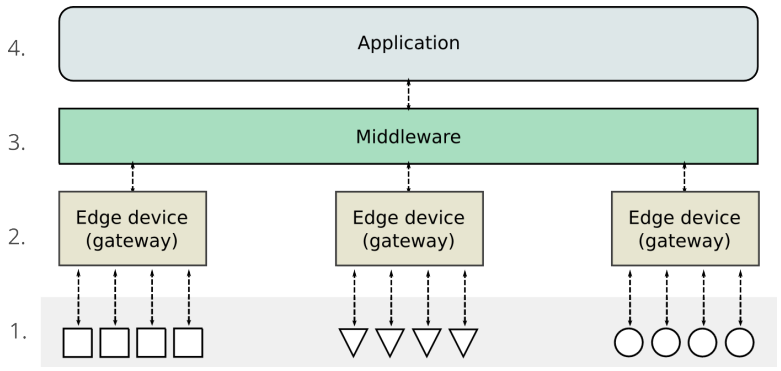


Figura 3.2 – Quatro camadas da arquitetura de IoT.

liza os dados coletados.

Em exemplos mais recentes de *middleware* constata-se que o objetivo é a integração de diferentes dispositivos em uma única solução provendo um controle centralizado através de interfaces amigáveis, muitas vezes acessíveis também através de dispositivos móveis. O Ninja Sphere, citado na seção 3.2.3, é um exemplo dessa abordagem. O Ninja Sphere contém os *drivers* nativamente implementados para conectar uma grande diversidade de dispositivos, mas por ser *Open Source* qualquer desenvolvedor pode escrever e disponibilizar *drivers* para conectar diferentes dispositivos. A camada *middleware* conecta a interface e a inteligência do sistema às diferentes *things* orquestrando seu funcionamento para que atendam à finalidade da solução de acordo com as preferências do usuário.

O *middleware* é uma peça chave em IoT, pois é a camada que interconecta diferentes *things* a sistemas que constroem uma representação digital do ambiente físico, processam os dados e, quando necessário, interferem no ambiente. Apesar da tendência do uso de tecnologias e protocolos baseados em IPv6 nas *things*, como 6LowPAN, habilitando a comunicação em rede sem a necessidade de intermediários que realizam a tradução de tecnologias, o *middleware* permanece como uma camada importante para abstração dos detalhes técnicos de acesso aos dispositivos, implementar mecanismos de gerência, segurança e privacidade e, muitas vezes, a coleta de dados de *things* fechadas em *silos*. O *middleware* é também um meio de se implementar ou executar a inteligência de uma solução em ambiente remotos.

4 REFATORAÇÃO DE MIDDLEWARE PARA IOT BASEADA EM SDN

O presente trabalho é uma continuidade das pesquisas relacionadas a *middleware* para IoT desenvolvido no projeto REMOA. O *middleware* proposto por Tarouco et al. (2012) limita seu escopo aos problemas encontrados nos dispositivos de monitoramento utilizados no projeto REMOA e não a todos os problemas de IoT citados na seção 3.3, levando em conta, inclusive, o mesmo cenário. Os problemas abordados são alguns dos problemas conhecidos de IoT. O *middleware* proposto contém mecanismos desenvolvidos para contornar os problemas encontrados que são adaptáveis a outros dispositivos de monitoramento que por ventura viessem a ser utilizados.

O que motivou a refatoração foram os problemas encontrados no próprio *middleware* que impactavam no desenvolvimento, gerenciamento e manutenção da solução, os quais serão detalhados na seção 4.1. A refatoração foi realizada implementando as mesmas funcionalidades existentes no *middleware* do projeto REMOA, mas em uma arquitetura projetada para fazer uso dos recursos providos por SDN.

A orquestração da configuração e comunicação de rede através de OpenFlow possibilitou distribuir a complexidade da rede e do *middleware*, antes restritos à arquitetura de um *access point*, em diferentes servidores dando maior poder de processamento e possibilidades de desenvolvimento à solução.

O cenário no qual esse trabalho está baseado e os problemas encontrados na proposta anterior de *middleware* são detalhados na seção 4.1. As demais seções fornecem detalhes de como foi realizada a refatoração, os benefícios almejados e faz considerações a respeito dessa proposta. A seção 4.2 fornece uma visão em alto nível da refatoração enquanto os detalhes de implementação são apresentados na seção 4.3. A seção 4.4 resume os benefícios desse trabalho, mas também apresenta alguns pontos fracos constatados apontando para possíveis soluções.

4.1 Cenário e Problemas Encontrados

O cenário inicial sobre o qual o presente trabalho foi desenvolvido consiste de redes domésticas onde dispositivos de monitoramento de aspectos de saúde estão presentes e, através dos quais, pacientes com doenças crônicas são monitorados remotamente. Todos os dispositivos de monitoramento utilizados foram adquiridos dentre os que estavam disponíveis do mercado, então foram analisados e, a partir dos problemas encontrados, foram elaborados mecanismos de contorno implementados em um *middleware* rodando sobre um *access point* (TAROUCO et

al., 2012).

Para o desenvolvimento da proposta de monitoramento dos pacientes foram utilizados a balança Withings Wireless Scale, o medidor de pressão arterial Withings Blood Pressure Monitor, a câmera, utilizada como sensor de movimento, Panasonic BL-C230 e *smartphones* através dos quais agentes responsáveis pelo acompanhamento dos pacientes monitorados atualizariam as informações sobre os tratamentos. Os problemas encontrados nos dispositivos utilizados no REMOA constam dentre as questões em aberto de IoT, apresentados na seção 3.3, contudo, os dispositivos possuem algumas particularidades relacionadas às suas formas de funcionamento e configurações *hard coded* definidas pelos fabricantes. A seguir são apresentados os problemas encontrados nos dispositivos utilizados:

- **Interconexão:** Dos dispositivos utilizados, a câmera fornece várias possibilidades de configuração, incluindo as relacionadas ao modo de operação, à conexão à rede e o envio dos dados, tais como destino, protocolo, intervalo e envio e gatilhos de coleta. Já a balança permite apenas a configuração da conexão à rede Wi-Fi; o medidor de pressão não possibilita nenhuma configuração, funciona conectado a um iPhone (até a versão 4S) ou a um iPad (até a versão 2) e através de um aplicativo disponibilizado pela fabricante é possível controlar o seu funcionamento e coletar os dados do medidor. Os dados coletados pela balança e pelo medidor de pressão são enviados para um servidor da empresa fabricante, a partir de onde podem ser acessados através de aplicativos e também de API disponibilizada pela empresa. O encaminhamento dos dados é vertical, precisa sair do ambiente onde é coletado e voltar apenas através dos meios providos pela fabricante, situação caracterizada como *silo*. O acesso aos dados deve ser autorizado por cada usuário através de mecanismo OAuth[1] implementado pela aplicação.

  No projeto REMOA havia a restrição de que os dados não poderiam ser enviados ou armazenados em servidores de terceiros, portanto era necessário se comunicar diretamente com os dispositivos para a coleta dos dados, o que não é previsto ou implementado pela fabricante da balança e do monitor de pressão arterial.

- **Coleta de dados:** A estrutura dos dados é outra questão que precisa ser contornada nos dispositivos Withings, pois recebem do servidor dados em notação JSON, contudo, enviam os dados coletados codificados como um formulário HTTP, como mostrado na figura 4.1. No trecho ilustrado são transmitidos os seguintes dados (em destaque): o identificador do dispositivo (MAC), identificação do usuário atribuída pelo sistema da fabricante, horário do registro e o valor (98,5 Kg).

[1] OAuth: http://oauth.net

```
POST /cgi-bin/measure HTTP/1.1
User-Agent: Withings UserAgent
Host: scalews.withings.net
Accept: */*
Content-Length: 233
Content-Type: application/x-www-form-urlencoded

action=store&sessionid=721-4ee232b1-50d6de38&macaddress=00:24:e4:01:36:10&userid=448489&meastim
e=1323446930&devtype=1&attribstatus=0&measures=%7B%22measures%22%3A%5B%7B%22value%22%
3A98550%2C%22type%22%3A1%2C%22unit%22%3A%2D3%7D%5D%7D
```

Figura 4.1 – Dados enviados pela balança Withings Wireless Scale à aplicação da fabricante.

- **Gerência:** Nos dispositivos utilizados é possível ver claramente a influência da forma de alimentação energética no modo de funcionamento e, por consequência, no gerenciamento dos dispositivos. A câmera é alimentada continuamente e diretamente da tomada, não possui bateria e nunca entra em modo *stand by*. Os dispositivos Withings funcionam alimentados por pilha. A balança permanece em *stand by* até que seja acionada por um usuário que a esteja utilizando — basta ao usuário subir sobre a balança para que ela saia do modo *stand by*, realize a medição e transmita os dados. O medidor de pressão fica completamente desligado, é ativado através do dispositivo ao qual está conectado (iPhone, iPad).

  A câmera pode ser monitorada através de testes de conectividade (ex.: Ping); suporta, inclusive, configuração de IP estático, contudo, não suporta monitoramento através de SNMP. Os demais dispositivos não podem ter sua conectividade monitorada por estarem a maior parte do tempo inacessíveis. As únicas informações relacionadas ao gerenciamento possíveis de serem extraídas dos dados enviados são o horário da medição e o nível de bateria.

- **Segurança e privacidade:** Todos os dados transmitidos pelos dispositivos para os servidores destino são enviados em texto plano. Mesmo a câmera que possui mais recursos implementados não possui mecanismos de criptografia. Além da exposição dos dados, não há mecanismos de autenticação que garantam a identidade de dispositivos ou servidores.

  O fato dos dispositivos Withings enviarem os dados para servidores proprietários implica outro problema, além dos já mencionados de interconexão: com os dados em servidores de terceiros não há como garantir a sua confidencialidade.

Visando contornar os problemas encontrados para viabilizar o monitoramento remoto de pacientes com doenças crônicas através dos dispositivos anteriormente mencionados, foi proposto um *middleware* no projeto REMOA, implementado em um *access point* (TAROUCO et al., 2012). A decisão de utilizar um *access point* veio do objetivo de simplificar e reduzir a

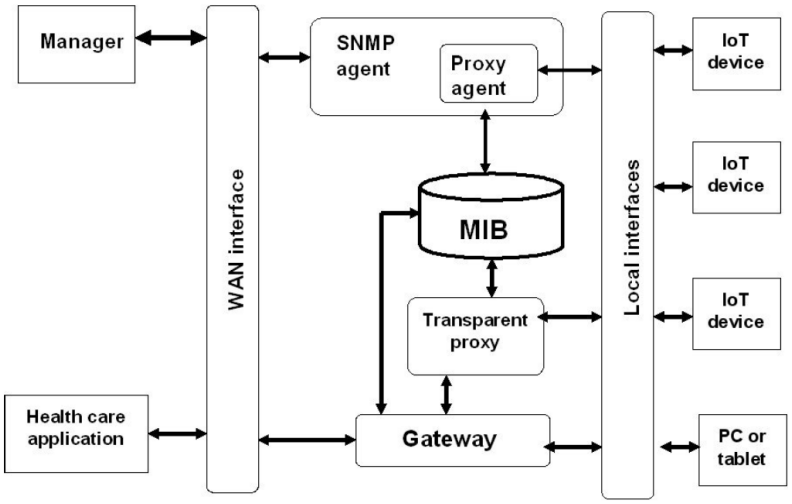


Figura 4.2 – Arquitetura do *middleware* proposto no projeto REMOA (TAROUCO et al., 2012).

infraestrutura necessária a ser instalada nas residências dos pacientes. O *access point* conectaria todos os dispositivos à Internet, tanto os de monitoramentos como os particulares pertencentes aos pacientes e seus familiares, utilizando o *link* existente na residência.

A figura 4.2 ilustra a arquitetura do *middleware* proposto. O *access point* utilizado na prototipação foi o TP-Link WR1043ND, no qual foi instalado o sistema operacional OpenWRT[2], uma distribuição Linux para dispositivos embarcados amplamente utilizado em ambientes domésticos. O desenvolvimento do *middleware* teve como foco o ambiente e a arquitetura do *access point* e do OpenWRT. Abaixo são descritos seus principais componentes:

1. *Transparent Proxy*: componente responsável por analisar todo o tráfego que sai para a Internet. O módulo recebe dados enviados pelos dispositivos de monitoramento e os envia para a aplicação de *healthcare* do projeto REMOA. Em paralelo armazena informações sobre a atividade e comunicação dos dispositivos em uma MIB, esses dados são utilizados para finalidades de gerenciamento. Quando necessário, são aplicados filtros ou regras de redirecionamento do tráfego de rede;
2. *SNMP agent e Proxy agent*: módulos responsáveis para tratar requisições SNMP destinadas às *things* sem suporte ao protocolo SNMP. As requisições são respondidas utilizando os dados previamente armazenados na MIB pelo módulo *Transparent Proxy*.
3. *Gateway*: possibilita a comunicação transparente entre a aplicação de *healthcare* e as *things*, porém, somente quando o modo de operação e os recursos disponíveis nas *things* permite que sejam acessadas diretamente.

Tendo em conta as limitações de recursos do *access point* e do OpenWRT, optou-se pelo

[2]OpenWRT: http://openwrt.org

desenvolvimento em C e C++. Devido à maior popularidade de linguagens como Python ou Java, que dispõe de um alto nível de abstração, poderíamos tê-las utilizado para facilitar e acelerar o processo de desenvolvimento e manutenção da solução. Contudo, nos repositórios do OpenWRT não há versão de Java disponível, mas é possível instalar a JamVM, uma versão de Java projetada para dispositivos móveis. O problema de utilizar pacotes que não estão disponíveis no repositório da distribuição é sua manutenção devido o gerenciamento de dependências, ou seja, modificar manualmente pacotes em prol da atualização de um recurso pode afetar outros recursos cujas versões dos pacotes estão alinhadas e compatíveis. Quanto a Python, a versão 2.7.3 está disponível nos repositórios, contudo, o desenvolvimento não poderia ser feito com Python 3+ devido à incompatibilidade das versões 2 e 3.

A utilização de Java implica executar uma *Java Virtual Machine* em um dispositivo com restrições de recurso, mesmo que a versão possível de ser utilizada seja projetada para este ambiente. Linguagens interpretadas ou híbridas, como Java, são mais lentas e consomem mais recursos se comparados a código nativo/binário (HUNDT, 2011). O desenvolvimento em C e C++ resulta em módulos mais enxutos, leves e rápidos, prevenindo *overhead* que seria causado pela utilização de outras linguagens e, possivelmente, outros pacotes necessários para o desenvolvimento de funcionalidades específicas. Todos os pontos levantados devem ser somados ao fato de que os *access points* seriam atualizados remotamente por estarem em uso na casa de pacientes monitorados, um processo de atualização e gerenciamento de dependências não automatizado potencializa erros que, neste contexto, representaria indisponibilidade, serviço e custos de deslocamento de equipe técnica e, possivelmente, riscos aos pacientes.

Apesar dos ganhos em performance e economia de recursos, o desenvolvimento em C e C++ costuma ser mais lento e complexo que quando utilizadas linguagens que contém recursos que abstraem conversão de tipos, ponteiros, alocação de desalocação de memória, etc. Alguns recursos implementados na versão C++11, e C++14 que está em desenvolvimento, proveem a C++ um maior nível de abstração de tarefas relacionadas a tipagem, *loops*, gerenciamento de memória, dentre outros, porém, a implementação de funcionalidades como um servidor REST dentro do módulo *Transparent Proxy* para a comunicação com alguns tipos de *things* ainda implicaria um trabalho considerável e, possivelmente, significativamente mais lento do que se fosse implementado em Java, Python ou outras linguagens.

A figura 4.3 ilustra o processo de desenvolvimento e implantação de novos recursos ao sistema de monitoramento através do *middleware* do projeto REMOA. As etapas potencialmente mais demoradas são as de seleção e análise do dispositivo e implementação dos módulos *Transparent proxy*, *Gateway*, *SNMP e Proxy agents*. Sobre cada uma das etapas citadas há o

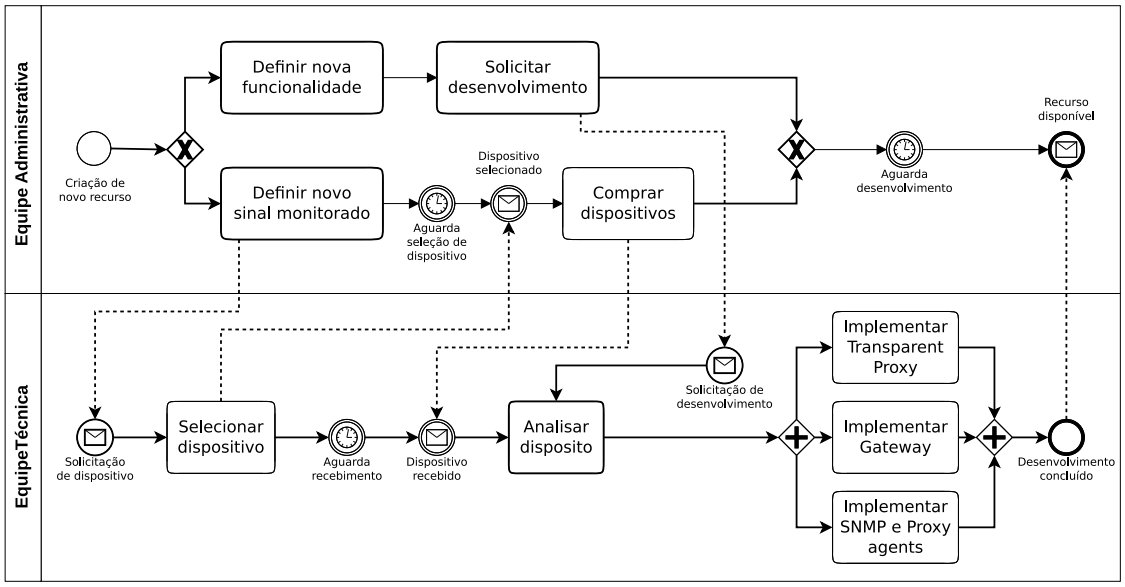


Figura 4.3 – Ilustração do processo de desenvolvimento de novos recursos no *middleware* do projeto REMOA.

impacto dos fatores relacionados ao desenvolvimento descritos nos parágrafos anteriores.

Durante a seleção do dispositivo é necessário levar em conta todos os aspectos que envolvem a sua forma de operação e seus mecanismos de comunicação em rede a fim de avaliar a viabilidade de sua utilização para monitoramento no projeto, pois há como limitante os recursos do *access point*, os recursos da linguagem utilizada para desenvolvimento e a compatibilidade com funcionalidade já implementadas, pois as novas funcionalidades deverão coexistir com as existentes.

Após a aquisição do dispositivo selecionado é necessário analisá-lo em diversos aspectos, pois muitas das informações necessárias para a implementação dos módulos não são disponibilizadas pelos fabricantes uma vez que o uso que se pretendia no projeto não é o uso projetado pela fabricante para o dispositivo. É preciso entender a forma como o dispositivo se comunica com o servidor. No caso dos dispositivos Withings, utilizados no protótipo desenvolvido, a comunicação dos dispositivos com o servidor é baseado na arquitetura RESTful, ou seja, baseada em requisições HTTP feitas a um servidor *web*. Somente depois da conclusão da análise é possível implementar os módulos. A etapa de implementação dos diferentes módulos pode ser implementada em paralelo a fim de reduzir o tempo de desenvolvimento.

O processo de desenvolvimento de novos recursos e melhorias, tais como aprimoramento dos módulos de gerenciamento, alterações na coleta e envio de dados estatísticos e de monitoramento, alterações nos mecanismos de atualização do *middleware*, ou mesmo correções de *bugs* pode requerer uma nova análise sobre dispositivos já adquiridos. O processo de desenvolvimento dos módulos não envolve necessariamente todos os módulos existentes.

O processo de implantação de um ambiente de monitoramento completamente novo ou do monitoramento de apenas um sinal de saúde ainda não monitorado em um paciente, ilustrado na figura 4.4, parte de uma solicitação de monitoramento. Quando a solicitação é para um paciente não monitorado, o processo envolve a preparação dos dispositivos necessários para monitorar todos os sinais selecionados pela equipe médica responsável, preparação do *access point* e implantação na data e no horário agendados com o paciente. Para a implantação do monitoramento de um sinal ainda não monitorado a logística do processo é mais simples, uma vez que o ambiente já conta com o *access point* e outros dispositivos, contudo, o *middleware* precisará receber os módulos atualizados que possibilitam a utilização do dispositivo sendo instalado, caso não tenha recebido no processo de atualização anterior. Por se tratar de um dispositivo em uso, o processo de atualização não deve interromper o tratamento. A atualização dos módulos é realizada remotamente, havendo qualquer erro que impeça que a atualização seja concluída, será necessário o deslocamento da equipe técnica o quanto antes para a manutenção do *access point*. Uma alternativa é a substituição do *access point* em execução por um com uma versão do *middleware* atualizada, o que implica um número maior de *access points* de reserva.

Embora a proposta do projeto REMOA possibilite o monitoramento remoto de pacientes com doenças crônicas utilizando dispositivos de monitoramento disponíveis no mercado, a abordagem concentra grande parte das funcionalidades da rede e do monitoramento sobre *access points*. As limitações físicas do *access point* e do sistema operacional restrigem os recursos disponíveis para o desenvolvimento e aprimoramento. O processo de incorporação de novos dispositivos ao sistema de monitoramento ou a um ambiente monitorado envolve, além do desenvolvimento limitado em termos de linguagem de programação, bibliotecas e *frameworks*, procedimentos de atualização remota de equipamentos em produção, o que sujeita o monito-

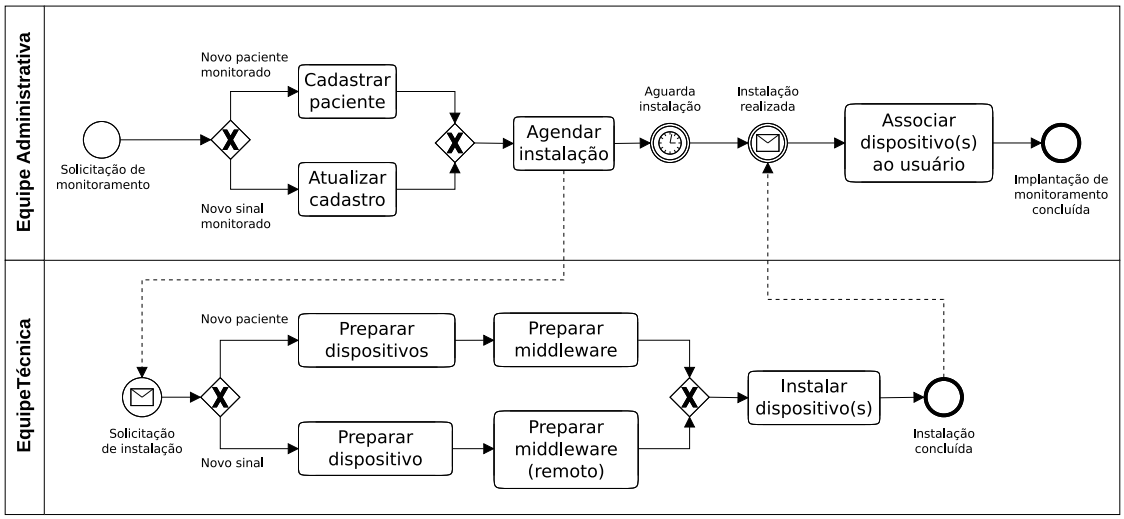


Figura 4.4 – Ilustração do processo de implantação de ambiente de monitoramento.

ramento a interrupções na ocorrência de erros. Um ponto a ser destacado é que o *middleware* proposto explora o gerenciamento dos dispositivos na rede através de uma abordagem mais adequada a IoT, onde a gerência se adapta ao dispositivo, contudo, a implementação, manutenção e aprimoramento dos recursos de gerência esbarra nas mesmas questões de desenvolvimento dos demais módulos. Outro ponto é que os dados enviados para a aplicação de *healthcare* transitam por canais seguros estabelecidos entre os *access points* e o servidor, o que também diminui os riscos relacionados à privacidade, pois pode-se garantir que os dados foram enviados através de um *access point* instalado em um ambiente de monitoramento.

4.2 Refatoração

O processo de refatoração, como descrito por Fowler e Beck (1999), consiste em reescrever trechos de código a fim de melhorar o funcionamento interno de um *software*, mas sem modificar seu comportamento externo. A refatoração do *middleware* proposto no projeto REMOA, detalhada nesta seção, visa manter as funcionalidades para o monitoramento de pacientes existentes, porém com uma arquitetura do sistema e sua comunicação baseada em SDN.

No *middleware* proposto por Tarouco et al. (2012) a complexidade da rede, a interconexão e a gerência dos dispositivos de monitoramento recai sobre o *access point* no qual roda o *middleware*. Baseado nas propostas de Chetty e Feamster (2012) e Yiakoumis et al. (2011), este trabalho se propõe a mover a complexidade de rede e dos serviços para longe da casa do usuário tornando o *access point*, onde antes rodava o *middleware*, um *switch* OpenFlow (ARBIZA et al., 2015; ARBIZA et al., 2016).

A figura 4.5 ilustra o posicionamento dos elementos que compõem a arquitetura refatorada e indica quais componentes exercem as funções dos módulos do *middleware* anterior. O *access point* concentra as funções de *switch* OpenFlow e estabelece conexões criptografadas, utilizando IPSec, com os servidores de cada serviço provido. O canal criptografado garante uma transmissão segura dos dados entre o ambiente monitorado e os servidores destino. Os módulos do *middleware* do projeto REMOA foram refatorados da seguinte forma:

- As funcionalidades do módulo *Transparent Proxy* passam a ser executadas pelo *access point* e pelos servidores remotos. Os pacotes das *things* são encaminhados pelo *access point* conforme as *flow entries* configuradas no *access point* pelo controlador. Os dados coletados pelas *things* são enviadas, através de um túnel IPSec, para os serviços remotos onde são processados e armazenados;

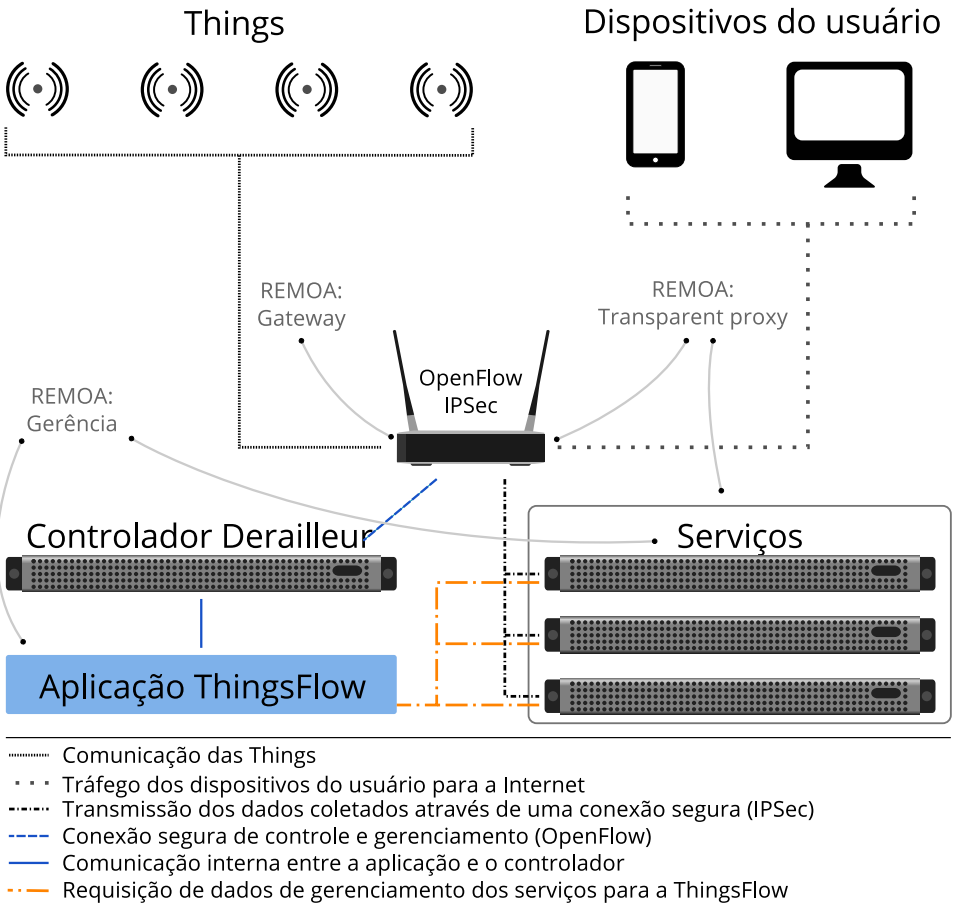


Figura 4.5 – Localização das funções dos módulos do *middleware* do projeto REMOA na arquitetura refatorada (ARBIZA et al., 2015).

- O gerenciamento das *things*, antes baseado em SNMP, é feito através dos contadores OpenFlow. Os contadores são coletados nos *access points* pela ThingsFlow que os armazena com um *timestamp* indicando quando foram coletadas. Os serviços solicitam os contadores à ThingsFlow para implementarem mecanismos de gerenciamento para as *things*;
- O módulo *Gateway* tornou-se a funcionalidade de encaminhamento de pacotes realizada pelo *access point*, o encaminhamento é definido através das regras OpenFlow.

A refatoração expandiu as possibilidades do *middleware*, em especial as que eram limitadas pela restrição de recursos do *access point* que impediam a utilização de ferramentas de desenvolvimento que requerem mais do sistema e, por consequência, do *hardware*. Com a utilização de SDN, os procedimentos de atualização dos componentes do *middleware* no *access point*, a manutenção e o processo de *deployment* (implantação) tornaram-se mais simples e, em grande parte, abstraídos pelo funcionamento padrão do OpenFlow. Abaixo estão relacionados os benefícios que essa proposta pretende trazer com a utilização de SDN em um cenário de IoT, nos seguintes aspectos:

- **Desenvolvimento**: Com a refatoração, o único processamento de pacotes realizado pelo *access point* é o encaminhamento. O desenvolvimento necessário para implementar as funcionalidades de um serviço, como a comunicação com as *things* para a coleta de dados e o processamento das informações, não é mais impactado pelas restrições do *access point*. Em servidores, significativamente menos restritos de recursos em relação ao *access point*, os desenvolvedores podem fazer uso de qualquer linguagem de programação, biblioteca, *framework* ou técnicas de programação para desenvolver as funcionalidades necessárias de cada serviço.
- **Manutenção e atualização**: Os *access points*, atuando como *switches* OpenFlow, não têm mais a necessidade de atualizações de módulos, como havia no *middleware* do REMOA. A configuração do encaminhamento de pacotes, papel desempenhado pelos *access points*, é feita sob demanda pelo controlador OpenFlow que instala as regras necessárias, as quais são fornecidas pela aplicação ThingsFlow. O processo de atualização necessário e que não é automatizado pelo OpenFlow é o das configurações dos túneis IPSec, esse processo é realizado através da atualização de um arquivo que contém o *script* de criação dos túneis.
- ***Deployment***: Para a implantação de um novo ambiente de monitoramento, ou para a execução de qualquer outro serviço provido através do *middleware* refatorado, não é mais necessária a configuração de módulos no *access point*. O *access point* recebe as configurações de encaminhamento do controlador OpenFlow assim que for inicializado ou sob demanda, quando algum pacote cujo encaminhamento não consta nas suas *flow tables*. A inserção de um novo dispositivo em um ambiente onde o *middleware* e um ou mais serviços estão em execução também foi simplificada pois nenhuma intervenção manual precisa ser realizada no *access point*, o novo dispositivo precisa apenas ser instalado no ambiente e, quando começar a transmitir, o OpenFlow instalará as *flow entries* necessárias.
- **Configuração lógica da rede**: Quando o destino dos pacotes não pode ser configurado diretamente no dispositivo em uso, o encaminhamento para o destino desejado é feito pelo *middleware*. SDN permite reescrever pacotes alterando seu destino diretamente no *access point*. As configurações e gerenciamento de rotas são feitas de forma centralizada e distribuídas em todos os *access point* conectados ao controlador. Este recurso é útil mesmo quando as *things* possibilitam configurações, pois o tráfego de um dispositivo que precisa ser desviado para outro servidor pode ser feito sem a intervenção no dispositivo que pode estar em uso em um cenário remoto.
- **Gerênciamento**: As *flow entries* instaladas em um *switch* OpenFlow possuem contadores

que são incrementados quando pacotes se enquadram nos *match fields*. As *flow entries* permitem saber quando cada *thing* e os seus respectivos serviços se comunicaram. Com as informações providas pelos contadores OpenFlow, funções de gerenciamento adaptadas ao funcionamento das *things* podem ser implementadas pelos responsáveis pelos serviços que as utilizam.

- **Cooperação entre *things***: Apesar da proposta de Tarouco et al. (2012) não abordar a entrega de mais de um serviço através do *middleware*, isso é possível na arquitetura proposta. Por não ter sido previsto, não há nenhum mecanismo para isolamento ou cooperação, por exemplo, e a utilização de cada novo dispositivo implica os mesmos custos operacionais e problemas já mencionados. Yiakoumis et al. (2011) exploram o compartilhamento de infraestrutura por diferentes serviços através de SDN, contudo, os serviços são isolados em diferentes fatias da rede. Em IoT almeja-se a colaboração das *things* para um objetivo comum (GIUSTO et al., 2010), o que não é possível quando há isolamento, como quando há *silos*. Nessa proposta os serviços compartilham dados a respeito da comunicação de suas *things* (contadores OpenFlow), ou os próprios dados coletados quando as políticas dos serviços permitirem. De posse dos dados referentes à comunicação das *things* presentes em um ambiente, um serviço pode elaborar um panorama de todo o cenário, expandindo, assim, a visão criada com base nos dados coletados pelos dispositivos de um único serviço. Um recurso também explorado através de SDN é o do reconhecimento de dispositivos, pois o controlador tem o conhecimento completo das múltiplas redes e dos dispositivos conectados a cada um dos *switches*.

A refatoração do *middleware*, através de uma implementação baseada em SDN, expande as possibilidades a serem exploradas no cenário de IoT em redes domésticas. A nova abordagem facilita a entrega de mais de um serviço através de um único *access point* e, ao mesmo tempo, possibilita que desenvolvedores lancem mão dos recursos necessários para implementação de novas funcionalidades para os serviços, uma vez que não estão mais limitados às restrições de *hardware* e de sistema do *access point*. As características do *middleware* do projeto REMOA, desenvolvidas para contornar os problemas encontrados nos dispositivos de monitoramento, foram mantidas, porém, com a utilização de SDN conseguiu-se distribuir o *middleware*, como camadas de *software*, entre diferentes equipamentos interconectados através de regras definidas via do OpenFlow, o que agiliza o processo de implantação de serviços e manutenção.

4.3 Arquitetura e Descrição Técnica

SDN possibilitou distribuir os componentes do *middleware* por diferentes sítios e equipamentos, enquanto a interconexão entre eles é orquestrada utilizando OpenFlow. A figura 4.6 ilustra a arquitetura da solução, as funções dos componentes do *middleware* refatorado e a relação entre eles. São três os componentes que compõem o *middleware* nessa proposta: os *access points* que atuam como comutadores de pacotes; o controlador Derailleur que realiza a comunicação OpenFlow entre a aplicação e os *access points*; e a aplicação ThingsFlow que fornece ao controlador as *flow entries* a serem instaladas para cada *access point* e disponibiliza os dados dos contadores para os serviços. O processamento dos dados específicos de cada serviço é executado em servidores próprios de quem os provê, os quais podem estar localizados em diferentes sítios. Os mecanismos de gerenciamento das *things* também são executados em servidores remotos e cada provedor de serviço implementa mecanismos adaptados ao funcionamento dos dispositivos. No cenário considerado, o controlador ficaria localizado em uma instituição responsável pelo projeto como um todo (ex.: universidade, prefeitura), os servidores que proveem os serviços podem ou não estar no mesmo sítio que o controlador.

A presente seção detalha o processamento dos pacotes dentro da solução proposta e descreve os recursos implementados em cada componente. A compreensão do funcionamento e da implementação dos componentes do *middleware* permite entender como os benefícios

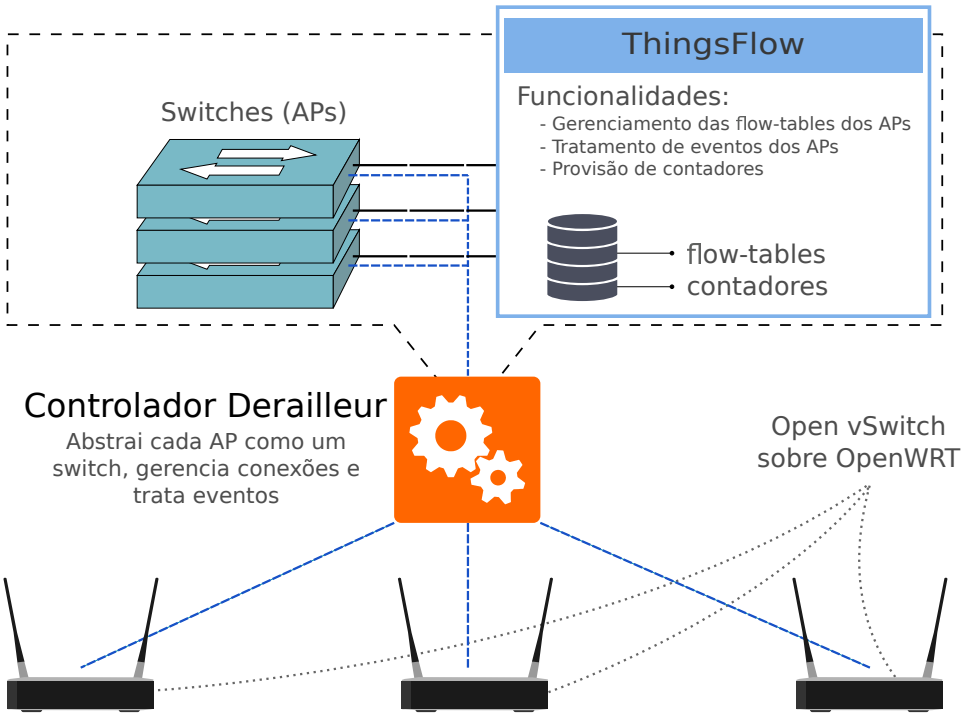


Figura 4.6 – Visão geral da arquitetura da solução após a refatoração do *middleware* (ARBIZA et al., 2015).

pretendidos, descritos na seção 4.2, foram alcançados.

4.3.1 Access Point

O funcionamento dos *access points* enquanto *switches* OpenFlow é o mesmo descrito na seção 2.1, ou seja, o processo de configuração e atualização das *flow tables* é realizado através do funcionamento padrão do protocolo OpenFlow. A proposta desta seção é ilustrar o processamento de pacotes, recebidos ou gerados por um *access point*, através de um conjunto de *flow tables* utilizadas na implementação da solução. O conjunto de *flow tables* ilustrado inclui uma específica para um serviço, neste caso foi utilizado como exemplo o serviço de monitoramento de pacientes do projeto REMOA. Cada novo serviço provido através de um *access point* implica na adição de uma ou mais *flow table*.

A fim de auxiliar a compreensão de como o processamento dos pacotes é realizado através dos componentes do *middleware*, os processos foram ilustrados na figura 4.7. O processamento ilustrado envolve quatro componentes: *access point*, controlador, aplicação e serviço (REMOA). Os detalhes de como o processamento é implementado através de regras OpenFlow nos *access points* está detalhado no apêndice D.

Todo o encaminhamento de pacotes realizado pelo *access point* é feito através do OpenFlow. Para converter os *access points* em *switches* OpenFlow, no lugar do *firmware* original do fabricante, foi instalada um imagem do OpenWRT compilada para a arquitetura do *hardware* utilizado (TP-Link WR1043ND). No processo de compilação, o Open vSwitch foi adicionado à imagem, o qual provê as funcionalidades do OpenFlow ao *access point*.

Nesta proposta, em todos os *access points* é instalado um conjunto padrão de quatro *flow tables*, às quais somam-se uma ou mais referentes aos serviços providos, o que pode variar em cada *access point*. Todos os pacotes, tanto os recebidos como os enviados pelo *access point*, são processados pelo OpenFlow. A primeira verificação é realizada na tabela 0, chamada de *Main*, a qual serve como um pré-processamento. A tabela *Main* contém regras inseridas pelos serviços que desviam o processamento dos pacotes relacionados às *things* para as respectivas tabelas. Os pacotes destinados ao *access point* são encaminhados à *flow table* 1 (Input) que verifica se o pacote pode ser aceito, ou não. Os pacotes enviados pelo *access point* são encaminhados para a *flow table* 2 (Output) que verifica se o pacote pode ser enviado pelo *access point*. Os demais pacotes que não se enquadram em nenhuma *flow entry* anterior são encaminhados à *flow table* 3 que atua como *firewall* e contém regras que filtram o tráfego, elaboradas baseadas em boas práticas e recomendações comumente utilizadas e considerando o uso da rede por dispositivos

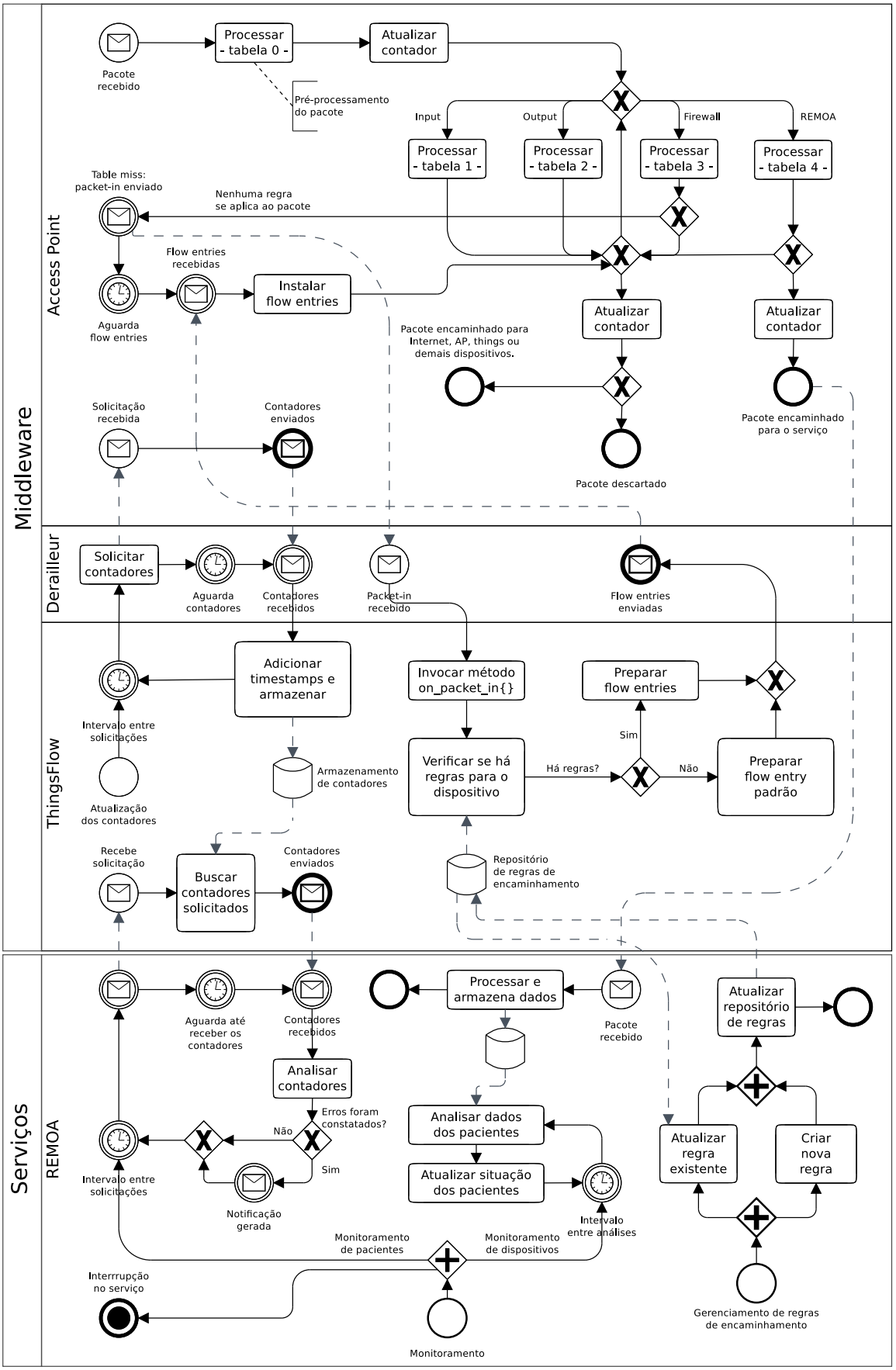


Figura 4.7 – Ilustração do processamento dos pacotes através do *middleware* e do serviço REMOA.

particulares dos residentes.

Quando um pacote é processado pelas *flow tables Input* e *Output*, apenas duas ações podem ser realizadas com um pacote: ser encaminhado ao destino, ou ser descartado. Quando um pacote é processado pela *flow table firewall*, além da possibilidade de encaminhamento e do descarte do pacote, há a possibilidade de ocorrência de *table miss*, caso nenhuma *flow entry* anterior possa processar o pacote. O *table miss* origina um *packet-in* que é enviado ao controlador, o qual deve responder enviando uma *flow entry* que defina o encaminhamento para o pacote sendo processado.

Os pacotes de dispositivos utilizados em algum dos serviços sendo providos são processados pela *flow table* específica do respectivo serviço que pode descartá-los, encaminhá-los ao destino ou encaminhá-los ao servidor da aplicação do serviço. Encaminhar um pacote para o serviço pode requerer a reescrita do pacote, como é o caso de alguns dispositivos do projeto REMOA que seriam destinados aos servidores proprietários, porém são redirecionados ao servidor do projeto.

4.3.2 Controlador Derailleur

Focando na abstração dos *access points* como *switches* e na utilização de diferentes conjuntos de *flow entries* em cada *access point*, aplicados dependendo dos serviços providos em cada ambiente, foi desenvolvido um controlador a fim de validar a proposta deste trabalho e verificar as possíveis limitações técnicas. O controlador, chamado Derailleur, foi desenvolvido na linguagem C++ utilizando as bibliotecas da libfluid, proposta campeã do *Open Networking Foundation OpenFlow Driver Competition* (CPQD. . . , 2014). Outra razão para o desenvolvimento de um controlador quando há vários é que, dos disponíveis, muitos possuem um arcabouço de funcionalidades desnecessárias para este trabalho. O controlador Derailleur foi projetado e desenvolvido de forma a ser simples e suscinto.

A libfluid é composta de duas bibliotecas: *libfluid_base* e *libfluid_msg*. A *libfluid_base* implementa o servidor que escuta e atende as conexões dos *switches* e realiza a comunicação OpenFlow entre eles e a aplicação ThingsFlow; é a *libfluid_base* que trata os eventos disparados pelos *access points* e instala e gerencia as *flow entries*. A *libfluid_msg* provê as funcionalidades para a criação, envio e *parsing* das mensagens OpenFlow. O Derailleur é implementado através de herança da classe *OFServer*, da libfluid, a qual gerencia as conexões que são recebidas e tratadas em paralelo e provê os métodos de *callback* executados na ocorrência de eventos.

Os *switches* mostrados na figura 4.6 são representações virtuais dos *access points* e con-

tém informações que possibilitam identificar cada *access point* conectado. Os objetos da classe *Switch*, que criam a abstração dos *access points*, são construídos utilizando dados requisitados pelo controlador quando a conexão com um *access point* é estabelecida. Para a coleta dos dados dos *access points*, o controlador requisita as descrições do dispositivo enviando mensagens dos tipos *OFPT_FEATURES_REQUEST* e *OFPMP_DESC*. Quando as respostas às requisições são enviadas pelos *access points*, o controlador realiza o *parsing* das mensagens e armazena os dados no objeto do tipo *Switch* correspondente ao *access point*.

Os objetos que representam os *access points* são armazenados em um *container* gerenciado pelo controlador, mas compartilhado com a ThingsFlow para que possa acessar os *access points* através dos objetos *Switch*. Cada *access point* é adicionado ou removido do *container* na ocorrência de eventos de conexão e desconexão. Cada conexão possui um número identificador que é incrementado a cada nova conexão, esse número é associado ao *Switch* e utilizado como *index* para acesso aos *access points*. O controle e a posse do *container* é exclusividade da classe que implementa o Derailleur, a classe *Controller*. As aplicações só podem acessar os *access points* através do objeto que os representa armazenado no *container*; não é possível fazer cópias de um *Switch*, desta forma evita-se a tentativa de acesso a um dispositivo que pode não estar mais conectado, mas que poderia estar representado como conectado devido ao estado do objeto copiado.

Através das informações requisitadas aos *access points* quando se conectam ao controlador é possível identificá-los. Nesse trabalho o endereço MAC foi utilizado como identificador. Comumente o endereço IP é utilizado como identificador de um dispositivo conectado à rede, contudo, os *access points* instalados em residências e conectados à Internet através de uma operadora poderão receber endereços dinamicamente, ou mesmo um endereço atrás de NAT, o que inviabiliza a utilização do endereço IP como identificador.

Dentro do controlador há a classe *Application*, com a qual o controlador compartilha o *container* de *switches* e cujos métodos são invocados na ocorrência de eventos — o controlador encaminha os eventos para a classe *Application* onde serão tratados conforme a implementação da aplicação.

A classe *Application* provê quatro métodos abstratos que devem ser implementados pelas aplicações que executarão sobre o controlador: *on_switch_up*, para quando um *access point* conecta ao controlador; *on_switch_down*, invocado quando um *access point* desconecta do controlador ou quando a conexão é interrompida; *on_packet_in*, invocado quando um *access point* envia um *packet-in*; e *message_handler*, utilizado para encaminhar à aplicação qualquer outro tipo de mensagem recebida de um *access point* para ser tratada. Dependendo do evento,

o método adequado é invocado pelo controlador. Quando o controlador invoca um método da aplicação, ele também repassa todos os dados enviados pelo *access point* quando o evento foi disparado.

4.3.3 Aplicação ThingsFlow

A ThingsFlow foi desenvolvida através de herança da classe *Application*, implementada pelo controlador Derailleur. A classe *Application* fornece quatro métodos abstratos que, obrigatoriamente, devem ser implementados dentro de uma classe filha. Na aplicação ThingsFlow cada um dos métodos foi implementado para prover determinadas funcionalidades, como descrito a seguir:

- *on_switch_up*: Os mecanismos para requisição de dados dos *access points* são implementados e executados pelo controlador assim que um *access point* estabelece conexão para a construção do objeto *Switch*. Após a criação do objeto que abstrai o *access point*, o controlador invoca o método *on_switch_up* implementado pela ThingsFlow. A ThingsFlow acessa os dados do *access point* através do objeto armazenado no *container* para identificá-lo utilizando dados de sua base de dados. Dentro da ThingsFlow cada *switch* é associado a um ou mais serviços cujas *flow tables* são enviadas e instaladas no *access point*.
- *on_switch_down*: Quando um *access point* desconecta ou a conexão é interrompida, o controlador bloqueia o acesso ao *container* de *Switches* para a remoção do objeto respectivo ao *access point* desconectado. Ao concluir, o controlador invoca o método *on_switch_down* implementado na ThingsFlow tornando-a ciente da desconexão.
- *on_packet_in*: A causa do envio de um *packet-in* por parte de um *access point* é a ocorrência de *table miss* — evento disparado quando o *switch* não encontra em suas *flow entries* uma regra que se aplique a pacote sendo processado. Ao receber um *packet-in*, o controlador invoca o método *on_packet_in* implementado na ThingsFlow, como ilustrado na figura 4.7. As regras específicas dos serviços providos através de um *access point* são instaladas quando se conectam ao controlador, portanto, os *packet-in* são enviados em função de dispositivos desconhecidos, ou seja, não associados a nenhum serviço, que tentam enviar pacotes através do *access point*. A ThingsFlow analisa os dados repassados pelo controlador com o *packet-in* e instala *flow entries* específicas para o dispositivo que tenta se comunicar encaminhando seu tráfego para a Internet. A criação de *flow entries*

específicas para o dispositivo, ao invés do uso de uma regra padrão de encaminhamento para todos os dispositivos desconhecidos, serve para possibilitar a coleta de contadores relacionados à comunicação de cada dispositivo conectado à rede. Os contadores das *flow entries* de dispositivos desconhecidos provêm os dados necessários para a construção da visão panorâmica do ambiente, descrito na seção 4.2.

- *message_handler*: Qualquer mensagem enviada pelos *access points* que não se enquadrem nos casos acima, são encaminhadas à ThingsFlow através do método *message_handler*. Sendo a ThingsFlow um protótipo para a validação dessa proposta, não foram implementados processamentos para as demais mensagens OpenFlow, senão as utilizadas nos casos anteriores.

4.4 Considerações sobre a Abordagem Proposta

Ao se optar pela refatoração do *middleware* levou-se em conta a viabilidade, baseado na literatura técnica sobre os recursos para implementação e implantação de SDN, e os benefícios que se acreditava possíveis de serem obtidos, descritos na seção 4.2. Comparado ao *middleware* proposto no projeto REMOA, esta proposta apresenta vantagens importantes em termos de escalabilidade e dinamicidade, uma vez que as regras de encaminhamento de fluxo são instaladas nos *access points* de forma automatizada, sob-demanda e através do mecanismo implementado pelo OpenFlow, dispensando intervenção nos *access points*, em especial aqueles em produção, e por consequência reduzindo o risco de erros causados por intervenção manual e necessidade de deslocamento de equipe técnica para manutenção.

Quanto aos ganhos relacionados à agilidade, é possível afirmar que ter à disposição os recursos de desenvolvimento que não estão disponíveis nos *access points* potencializa um desenvolvimento mais ágil de novas funcionalidades e melhorias. Ter mais recursos para desenvolvimento possibilita a implementação de funcionalidades mais complexas e amplas que as que seriam desenvolvidas para executar nos *access points*, o que torna a agilidade relativa àquilo que precisa ser desenvolvido; desenvolver funcionalidades complexas resulta em mais trabalho e pode tomar mais tempo que o desenvolvimento de módulos mais simples, mesmo considerando os processos de compilação cruzada e desenvolvimento em linguagem de baixo nível de abstração.

Há muita pesquisa sobre SDN e, constantemente, surgem novas propostas que podem vir a somar a esse trabalho, tais como a utilização de VLANs ou redes virtuais, caso algum serviço necessite de um maior isolamento. Para isso, contudo, é necessário implementar o suporte à

múltiplas aplicações no Derailleur, recurso não explorado até o momento.

A centralização da inteligência da rede em um controlador o torna um ponto central de falha localizado remotamente, ou seja, sua indisponibilidade comprometerá o encaminhamento dos pacotes pelos *access points* e, por consequência, a interconexão das *things* aos serviços. Na proposta do projeto REMOA o ponto central de falha eram os *access points*. Para contornar esse problema há a possibilidade de utilização de mais de um controlador e *scripts* executados nos próprios *access points* que configuram um encaminhamento padrão quando a comunicação com os controladores ficar indisponível.

A execução de diferentes serviços em um mesmo *access point* pode resultar em *flow entries* conflitantes. O OpenFlow provê um mecanismo de verificação implementado nos *switches*; quando uma *flow mod* envia uma mensagem com o *bit OFPFF_CHECK_OVERLAP* ativado, o *switch* verifica se a regra a ser instalada se aplica aos mesmos pacotes que uma regra previamente instalada e possuem a mesma prioridade, em caso positivo, a *flow entry* não será instalada e o controlador é notificado. Contudo, é necessário que os serviços sejam projetados para coexistir evitando regras que se sobreponham.

Por fim, a proposta de *middleware* apresentada nesse trabalho, não tem como objetivo abordar todas as questões em aberto de IoT, mas sim àquelas abordadas no projeto REMOA, apresentadas na seção 3.3, encontradas nos dispositivos de monitoramento utilizados. Como essa proposta amplia o *middleware* e considera a entrega de múltiplos serviços, diferentes problemas podem ser encontrados ao se utilizar diferentes dispositivos, tais problemas podem ser tratados em trabalhos futuros explorando diferentes serviços e contextos.

5 DISCUSSÃO SOBRE OS BENEFÍCIOS ALCANÇADOS

Um desafio importante deste trabalho é comprovar seus benefícios devido à natureza das suas contribuições. A principal motivação para o processo de refatoração foram as questões relacionadas ao desenvolvimento dos módulos, por meio dos quais os serviços são entregues através de *access points*, e a gerência e manutenção do *middleware* com tudo que o compõe e das *things* como elementos da rede. Na proposta do projeto REMOA, grande parte das funcionalidades dos serviços e da própria rede rodavam sobre os *access points*, com a refatoração apresentada no capítulo anterior conseguiu-se distribuir a carga dos serviços e da rede entre diferentes servidores, desta forma os serviços ficaram isolados e centralizados próximos de quem os provê.

Mover os módulos de serviços dos *access points* para servidores mais robustos potencializa o desenvolvimento mais ágil e simples dos mecanismos que compõe cada serviço. Além da disponibilidade de mais recursos para o desenvolvimento, a aplicação pode ser desenvolvida de forma mais integrada ao invés de ser composta por diferentes módulos distribuídos em servidores e *access points*.

O *middleware* do projeto REMOA foi utilizado apenas para validação e demonstração, o que não fornece dados que poderiam ser utilizados para contabilizar a quantidade de erros ocasionados por intervenção manual ou o tempo e o número de ações necessários para uma manutenção. Tais dados poderiam ser utilizados como métricas para a comparação entre as duas propostas. Em relação ao desenvolvimento, há muitos fatores que influenciam o tempo de conclusão de uma tarefa, como o nível de conhecimento do desenvolvedor, experiência com a linguagem, etc. Por estas razões, para demonstrar os benefícios da utilização da abordagem baseada em SDN, optou-se por fazer uso da comparação de processos e recursos de desenvolvimento, demonstrados na seção 5.1. A seção 5.2 demonstra, utilizando o cenário do projeto REMOA, como uma visão mais ampla do ambiente monitorado pode ser conseguida através dos recursos do OpenFlow e a cooperação entre *things* e serviços no contexto de IoT. O mesmo cenário do projeto REMOA é utilizado para demonstrar a utilização do OpenFlow para implementar a gerência de *things* como dispositivos de rede, também demonstra ganhos de escalabilidade em termos de *deployment* e gerenciamento de segurança nos diversos ambientes monitorados.

No capítulo anterior foram apresentados a arquitetura da proposta do presente trabalho e os detalhes de como foi realizada a implementação e os recursos que são utilizados, nesse capítulo alguns outros detalhes serão apresentados relativos à prototipação facilitando a com-

preensão de como os benefícios pretendidos foram alcançados.

5.1 Benefícios na Entrega de Serviços e Funcionalidades

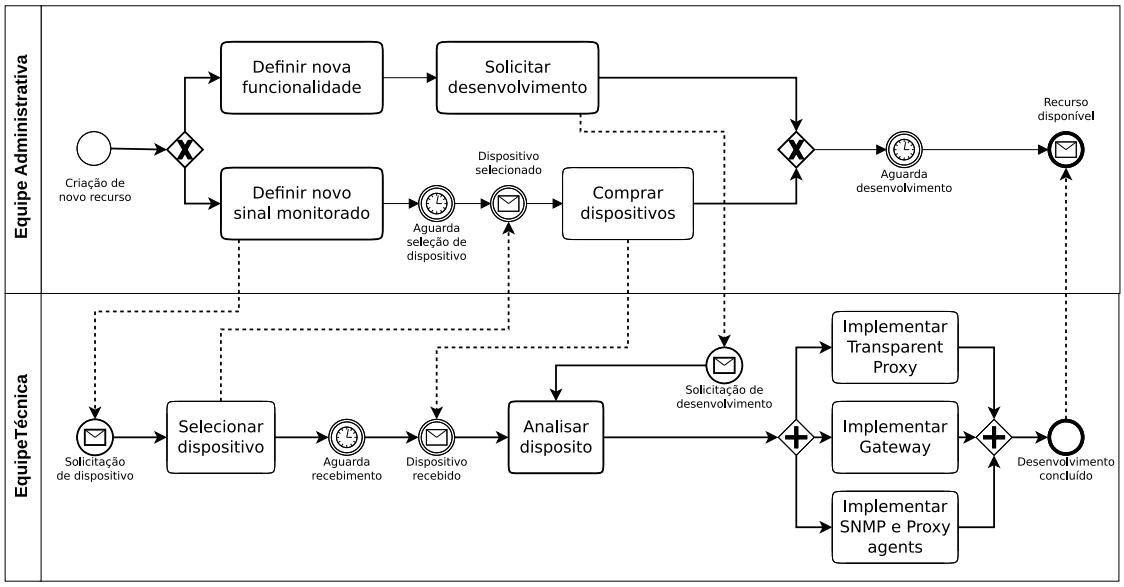
Na seção 4.1 foram apresentados dois processos (figuras 4.3 e 4.4): o primeiro demonstra as etapas para o desenvolvimento de novas funcionalidades do sistema de monitoramento de pacientes e para o monitoramento de um novo sinal de saúde; o segundo ilustra o processo para a implantação do sistema de monitoramento como um todo na residência de um paciente monitorado ou de um dispositivo para o monitoramento de algum sinal de saúde ainda não monitorado. Ambos os processos tem como contexto o monitoramento de pacientes proposto no projeto REMOA.

Nesta seção os dois processos citados serão utilizados para ilustrar o impacto da refatoração no contexto do projeto de monitoramento de pacientes. As figuras 5.1 e 5.2 serão utilizadas para ilustrar a comparação entre os processos e as etapas afetadas, destacadas com fundo cinza. Nas duas comparações são poucas etapas afetadas e menor ainda é o número de etapas reduzidas, o impacto da refatoração, contudo, está em como as etapas afetadas são desenvolvidas na abordagem baseada em SDN.

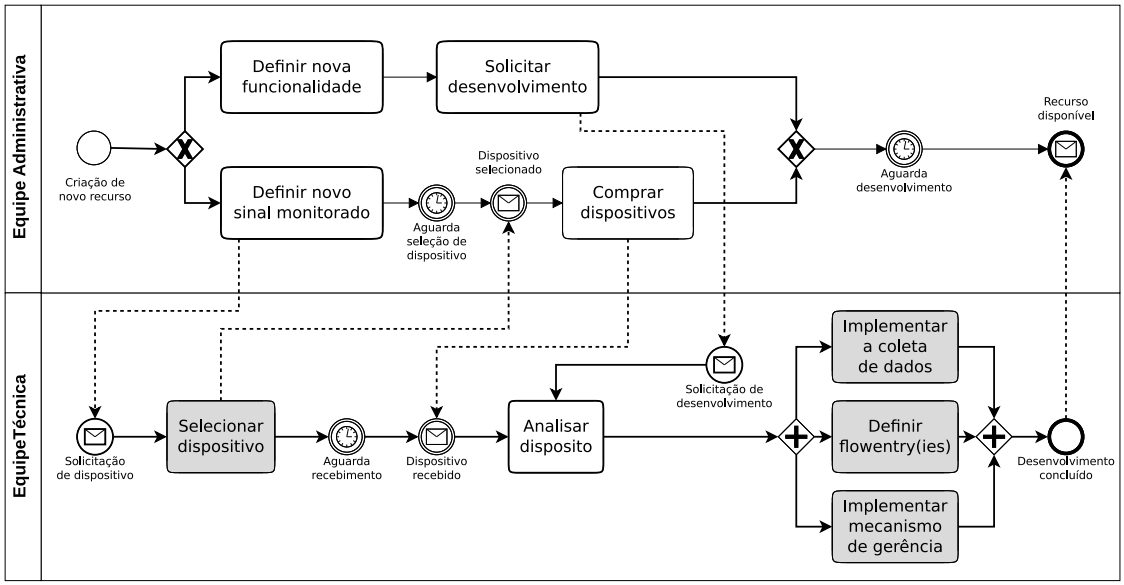
A figura 5.1 ilustra, na primeira parte, o processo de desenvolvimento utilizando o *middleware* proposto no projeto REMOA, previamente ilustrado na figura 4.3. A segunda parte ilustra o processo de desenvolvimento utilizando o *middleware* baseado em SDN, proposto nesse trabalho. A refatoração ocasionou mudanças nas etapas do processo relacionadas à seleção dos dispositivos a serem utilizados no monitoramento de sinais de saúde dos pacientes e no desenvolvimento e definição dos mecanismos empregados para viabilizar o monitoramento através dos dispositivos selecionados.

Na versão anterior do *middleware*, a seleção dos dispositivos a serem utilizados para o monitoramento deveria, necessariamente, levar em conta as limitações do *access point* (*hardware* e sistema) e do *middleware*, pois as características da comunicação de rede de alguns dispositivos poderiam ser impedimentos para utilizá-los no monitoramento devido ao custo operacional para desenvolver os mecanismos necessários, mantê-los e gerenciá-los.

Os dispositivos da fabricante Withings utilizados na prototipação do projeto REMOA, por exemplo, empregam uma comunicação baseada em operações GET e POST do protocolo HTTP, enviadas a programas em CGI (*Common Gateway Interface*) rodando em servidores proprietários. Nos repositórios do OpenWRT estão disponíveis módulos CGI para as linguagens PHP, Ruby e Perl, além das bibliotecas CGI e FastCGI que podem ser utilizadas para desenvol-



a) Processo de desenvolvimento utilizando o *middleware* do projeto REMOA.



b) Processo de desenvolvimento utilizando o *middleware* baseado em SDN.

Figura 5.1 – Comparativo entre os processos de desenvolvimento utilizando o *middleware* do projeto REMOA e o *middleware* baseado em SDN.

vimento em C, C++ e *scripts* Shell. Contudo, a comunicação através de CGI requer a instalação de um servidor *web* nos *access points*, ou seja, utilizar linguagens com maior nível de abstração para implementar a comunicação via CGI implica mais pacotes para manutenção, mais carga sobre o *access point*, diversos cuidados de segurança por haver um servidor *web* ativo, além da manutenção dos diferentes programas para atender as requisições CGI dos diferentes dispositivos que venham a ser utilizados que empregam este tipo de comunicação. A implementação do módulo para os dispositivos Withings em C ou C++ também tem seu custo, especialmente no desenvolvimento do módulo em si e dos mecanismos para encaminhar a comunicação dos dispositivos para servidores e portas corretas para serem atendidos pelos programas apropriados

(não proprietários), funcionalidade que costuma ser desempenhada pelo servidor *web*.

A câmera Panasonic, utilizada como sensor de movimento, assim como os dispositivos Withings, emprega comunicação baseada em HTTP, mas também pode enviar as imagens capturadas por e-mail ou para um servidor FTP. As dificuldades encontradas com a câmera Panasonic se assemelham às encontradas com os dispositivos Withings, pois o desenvolvimento dos módulos de comunicação dos dispositivos com o *middleware* implica mais pacotes instalados e execução de serviços nos *access points*. A implementação com linguagens com menor nível de abstração, por sua vez, implicam restrições no desenvolvimento.

As demais etapas afetadas no processo de desenvolvimento pela refatoração são as que compreendem a implementação dos módulos do *middleware*. As etapas "Implementar Transparent Proxy", "Implementar Gateway" e "Implementar SNMP e Proxy agents" foram substituídas, respectivamente, pelas etapas "Implementar a coleta de dados", "Definir *flow entry*(ies)" e "Implementar mecanismo de gerência". Essas etapas de desenvolvimento, quando realizadas no contexto do projeto REMOA, sofrem as restrições mencionadas nos parágrafos anteriores.

A etapa "Implementar Transparent Proxy" compreendia o desenvolvimento de um módulo para se comunicar com o dispositivo de monitoramento e coletar/receber os dados. Sendo que os dispositivos foram projetados pelas fabricantes para serem utilizados por usuários finais e integrados a serviços também providos pela fabricante, sequer havia a possibilidade de configurar alguns dos dispositivos para que se comunicassem com o *middleware*, portanto, o módulo precisava se comportar tal qual o servidor proprietário para viabilizar a comunicação com o dispositivo e assim coletar os dados lidos. O desenvolvimento deveria ser feito considerando as diversas limitações já mencionadas do *hardware*, sistema operacional e do próprio *middleware*. Além da comunicação para a coleta de dados, o *Transparent Proxy* também coletava e armazenava informações utilizadas na gerência. Na abordagem baseada em SDN esta etapa foi substituída por "Implementar a coleta de dados" que, dependendo dos dispositivos a serem utilizados, também precisa se comportar como o servidor proprietário da fabricante, contudo, agora o desenvolvimento pode lançar mão dos recursos antes não disponíveis. Nesta abordagem toda a aplicação pode ser desenvolvida de forma mais integrada, o que também impacta na manutenção uma vez que não é necessário gerenciar módulos em diversos *access points*, bem como filtros e outros possíveis mecanismos necessários para o redirecionamentos dos pacotes destinados a servidores remotos.

A etapa "Definir *flow entry*(ies)", que substitui "Implementar Gateway", consiste em elaborar as regras de encaminhamento necessárias para viabilizar a comunicação do dispositivo em questão com o servidor provedor do serviço. As regras de um serviço não podem se so-

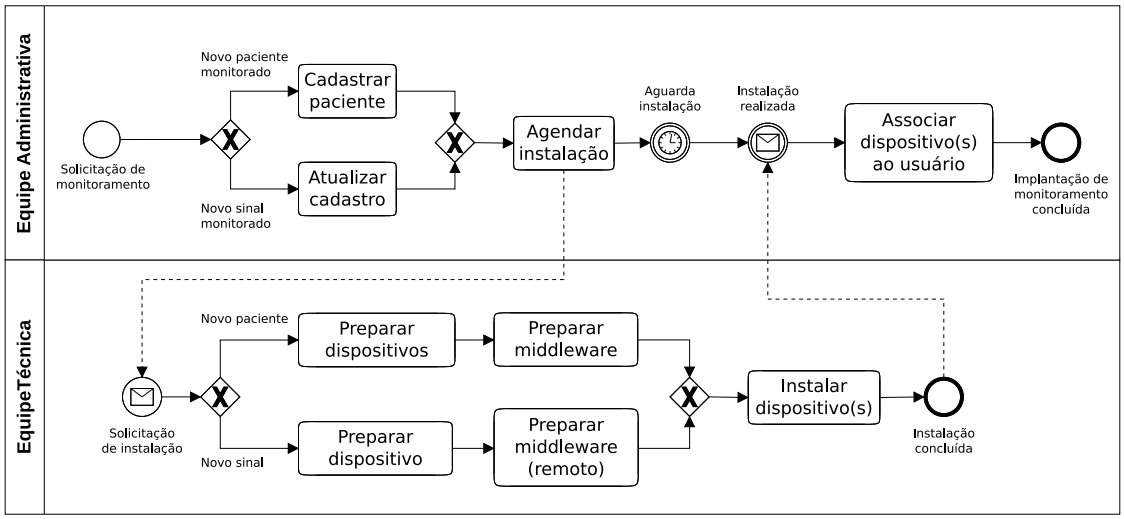
brepor a de outros serviços. O OpenFlow dispõe de um mecanismo de verificação que impede a instalação de regras que se sobreponham a regras existentes. Havendo sobreposição, os administradores precisarão reelaborar algumas regras garantindo a coexistência dos serviços. As regras são instaladas sob demanda e, se necessário, através do modo de operação do OpenFlow.

A gerência dos dispositivos, antes baseada em SNMP e cujas informações eram coletadas pelo módulo *Transparent Proxy*, foi substituída por mecanismos desenvolvidos com base nos contadores das *flow entries*. Estes mecanismos, assim como a coleta de dados, foi movida para servidores daqueles que proveem os serviços, não executam mais nos *access points*. Na etapa "Implementar SNMP e Proxy agents" era necessário implementar os mecanismos para que as informações de gerência pudessem ser coletadas, o que era feito através de configuração do servidor SNMP para que executasse *scripts* quando fosse solicitado um determinado OID privado, o *script* retornaria as informações coletadas e armazenadas pelo *Transparent Proxy*. Na abordagem baseada em SDN, fazendo uso dos contadores OpenFlow que proveem a mesma informação que o modulo baseado em SNMP do *middleware* anterior, os desenvolvedores dispõe de mais recursos para implementar mecanismos de gerência adequados às particularidades de cada dispositivo utilizado.

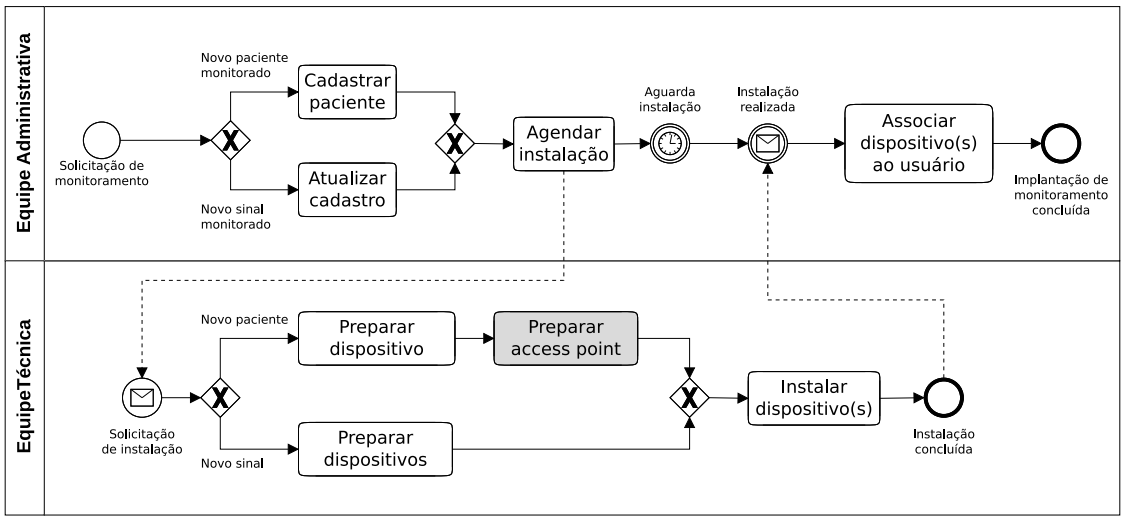
No processo de implantação, ilustrado pela figura 5.2, uma etapa foi eliminada e outra sofreu alterações, estas mudanças, contudo, representam ganhos em termos de escalabilidade. No processo anterior, quando um novo paciente passasse a ser monitorado, era necessário preparar os dispositivos de monitoramento e instalar o *middleware* contendo os módulos que viabilizassem o monitoramento de todos os sinais necessários, dependendo do tipo de tratamento. Após a refatoração, a etapa "Preparar *access point*" foi reduzida a uma verificação de funcionamento do *access point* a ser utilizado e, se necessária, alguma configuração nos parâmetros de rede. A etapa "Preparar *middleware* (remoto)" foi eliminada, esta etapa compreendia a atualização do *middleware* no *access point* da residência onde um novo sinal de saúde passaria a ser monitorado. Agora, a atualização necessária nos *access points* é automatizada pelo OpenFlow, portanto, para monitorar um novo sinal, basta que o dispositivo de monitoramento seja instalado na casa do paciente e que sejam feitas as associações necessárias no sistema de monitoramento, porém, nenhuma intervenção no *access point* é necessária.

5.2 Cooperação em IoT e Entrega de Múltiplos Serviços

Um dos principais problemas encontrados em IoT, e que também foi encontrado no projeto de monitoramento de pacientes, são os *silos*, ou seja, dispositivos que fazem parte de



a) Processo de implantação utilizando o *middleware* do projeto REMOA.

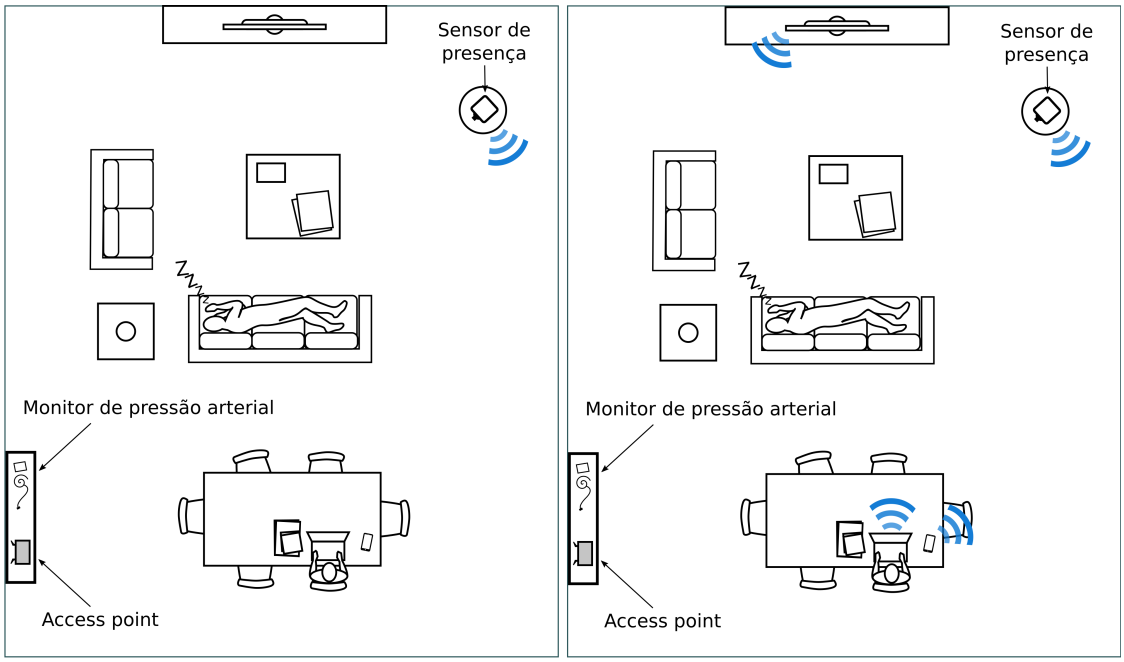


b) Processo de implantação utilizando o *middleware* baseado em SDN.

Figura 5.2 – Comparativo entre os processos de implantação do monitoramento utilizando o *middleware* do projeto REMOA e o *middleware* baseado em SDN.

uma solução que emprega um formato de comunicação vertical onde os dados coletados de um determinado ambiente ficam fechados na solução ao invés de serem compartilhados. Em cenários onde serviços serão providos com a restrição de que os dados não podem ficar em posse de terceiros, como no projeto de monitoramento de pacientes, e onde há dispositivos de diferentes fabricantes, é de grande importância uma abordagem que permita a cooperação entre dispositivos e serviços através do compartilhamento de informações.

Utilizando SDN foi possível construir uma representação digital do ambiente monitorado mais ampla do que na abordagem anterior. A figura 5.3 ilustra um paciente monitorado que adormeceu enquanto assistia televisão e não realizou uma medição de pressão arterial no horário programado. No lado esquerdo da figura o único dispositivo de monitoramento ativo no momento representado é o sensor de presença, o qual não detecta movimentos do paciente, nem da outra pessoa presente na sala, pois os movimentos são muito discretos ou distantes que



a) Representação do ambiente baseada nos dispositivos de monitoramento

b) Representação do ambiente baseada em todos os dispositivos conectados

Figura 5.3 – Comparativo entre as representações digitais de um mesmo ambiente utilizando o *middleware* do projeto REMOA e o *middleware* baseado em SDN (ARBIZA et al., 2016).

não disparam uma captura por não excederem o *threshold* configurado. O monitor de pressão arterial, mesmo presente no ambiente, permanece em *stand by* até ser utilizado. Os únicos dados recebidos pela equipe de monitoramento vem da detecção de movimentos ocorrida há algum tempo. Devido à falta de informação sobre o estado atual do paciente, uma ação precisa ser tomada para verificar se não se trata de uma emergência médica. Supondo que há uma quantidade considerável de pacientes sendo monitorados, a equipe de monitoramento possivelmente terá que agir, à princípio para verificação, em diversos momentos. Sendo que as fontes de informações sobre os pacientes e os ambientes monitorados são apenas os dispositivos de monitoramento utilizados no projeto, são poucos os elementos a serem considerados para determinar quais casos tem prioridade na tomada de ações.

O lado direito da figura ilustra o mesmo ambiente, no mesmo momento, com informações sendo providas por outros dispositivos conectados à rede, além dos dispositivos de monitoramento, permitindo a construção de uma representação digital do ambiente mais ampla, com mais dados para auxiliar a equipe de monitoramento na tomada de decisões. Os dados dos dispositivos que não fazem parte de algum serviço são fornecidos através de contadores OpenFlow, contabilizados através das *flow entries* criadas e instaladas para encaminhar o tráfego destes dispositivos, conforme detalhado na seção 4.3.3. Informações como o endereço MAC permitem supor ou concluir qual tipo de dispositivo está conectado no ambiente monitorado, alguns dispositivos podem estar cadastrados como pertencentes aos membros da família do pa-

ciente monitorado e associados a um tipo (ex.: *smartphone*, *notebook*). No caso do cenário ilustrado, apesar do atraso na medição de pressão arterial, a equipe de monitoramento sabe que a televisão está ligada e que um dos familiares do paciente está em casa utilizando seu *notebook* e seu *smartphone*. Neste caso, o contato que seria para uma verificação do estado de saúde do paciente passa a ser apenas para lembrá-lo da medição não realizada. Tendo em mãos um conjunto de informações mais amplo, a equipe de monitoramento tem recursos para concluir que o paciente não está sozinho e pode supor que não se trata de uma emergência, o contato com o paciente neste caso tem menor prioridade que outros onde não há tanta informação.

A construção da representação digital ilustrada no lado direito da figura 5.3 é feita com base nos pacotes transmitidos pelos dispositivos conectados à rede no ambiente monitorado, cujos dados das leituras realizadas em um intervalo de uma hora estão representados na figura 5.4. A leitura do número de pacotes de cada dispositivo é feita a cada 5 minutos. O gráfico ilustra atividade de apenas um dos dispositivos de monitoramento, o sensor de presença, na primeira e na terceira leitura. Contudo, os demais dispositivos conectados à rede, pertencentes aos residentes, transmitiram dados em todas as leituras, em alguns momentos com picos indicando uma atividade maior, como um acesso a uma página ou serviço na Internet, em outros momentos o volume de pacotes indica pouca atividade além da transmissão de pacotes de controle, o que sinaliza que o dispositivo está em uso, ou apenas ligado.

Os contadores de um dispositivo podem ser utilizados por outros serviços de forma que todos os dispositivos conectados cooperem para um ou mais objetivos, como o monitoramento de pacientes. No cenário utilizado como exemplo há apenas um serviço, o qual utiliza os dados dos seus próprios dispositivos (dados coletados para o monitoramento de saúde) e dos demais dispositivos conectados à rede, através dos contadores OpenFlow, para construção da representação digital do mundo real, indo ao encontro da definição de Miorandi et al. (2012) para IoT.

Além de fornecer mais dados para a construção de uma representação mais ampla dos ambientes monitorados, a utilização de SDN em ambiantes de IoT, contando com escalabilidade provida pelo OpenFlow para definir o comportamento lógico da rede, facilita a entrega de diferentes serviços, utilizando a mesma infraestrutura física (*access point*), compartilhando informações dos mesmos dispositivos. É pouco provável que um projeto de monitoramento, ou um serviço para caracterizar perfis de usuários de Internet com base nos sites acessados pelos usuários, compartilhem os dados coletados com outros serviços, afinal, ambos os serviços citados requerem cuidados legais e autorizações dos envolvidos para que possam ser realizados. Contudo, os contadores dos dispositivos utilizados em um serviço podem servir para outros

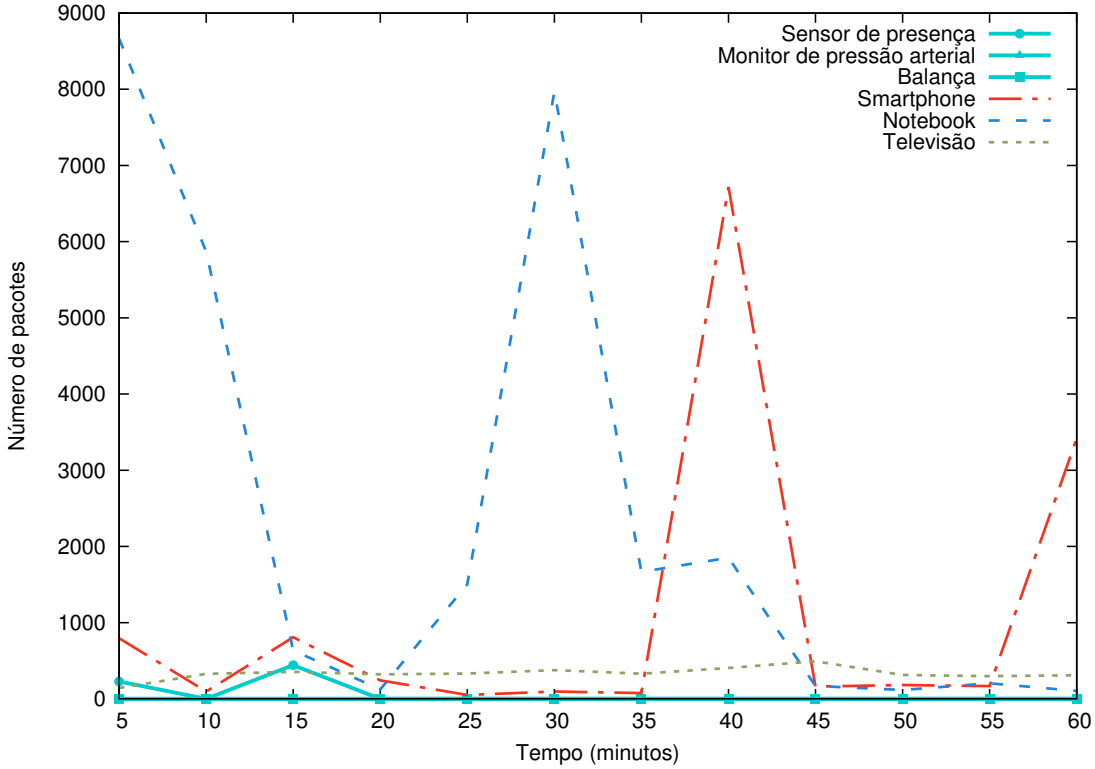


Figura 5.4 – Pacotes transmitidos ou recebidos pelos dispositivos conectados à rede no ambiente monitorado.

serviços. Na figura 5.3, por exemplo, o *notebook* e o *smartphone* poderiam ser utilizados para caracterizar o perfil do usuário, semelhante aos estudos realizados pelo Cetic.br[1], atualmente feitos utilizando formulários impressos. Neste caso os dados da *flow table* de pré-processamento servem para que o serviço de monitoramento de pacientes construa a representação digital do ambiente.

A seção 4.3.1 detalhou o processamento dos pacotes através do *middleware*, o qual conta com quatro *flow tables* comuns a todos os *access points*, mais as tabelas específicas de cada serviço. A primeira tabela realiza o pré-processamento, pois contém as regras para encaminhar os pacotes dos dispositivos para as tabelas correspondentes. Embora as *flow entries* das tabelas dos serviços possam conter dados particulares de cada serviço, as *flow entries* da primeira tabela são regras de encaminhamento baseadas em informações dos dispositivos (ex.: MAC) e não do destino, o que não expõe a privacidade dos usuários, e cujos contadores podem ser acessados por qualquer serviço.

[1]Cetic.br: http://cetic.br

5.3 Gerência de Redes para IoT

Na proposta de Chetty e Feamster (2012), que foi uma das que serviu como base para este trabalho, SDN é utilizado para mover a complexidade da rede para fora do ambiente doméstico. A seção 5.1 demonstrou que distribuir a estrutura do *middleware* e dos serviços, antes concentrada quase que por inteiro nos *access points*, entre servidores dedicados a coletar os dados e processá-los beneficiou o processo de desenvolvimento em diversos aspectos. Também foram demonstrados benefícios no processo de implantação de serviços. Contudo, SDN não é essencial para realizar o encaminhamento de pacotes para servidores remotos, isso poderia ser feito, por exemplo, utilizando regras de *firewall*. Apesar de não ser essencial, a utilização de SDN é conveniente, especialmente em se tratando de IoT, com sua diversidade de cenários e dispositivos, pois oferece uma abordagem centralizada, em nível de *software*, para definição e orquestração do comportamento lógico da rede, provendo escalabilidade e flexibilidade à gerência e manutenção dos serviços, do *middleware* e das redes.

As regras de *firewall* utilizadas nos *access points* na abordagem anterior, além do encaminhamento dos pacotes oriundos dos dispositivos de monitoramento realizado para possibilitar a coleta de dados, deveria prover também a segurança para a rede doméstica da residência do paciente monitorado. Contudo, em cada residência, por haver um conjunto de dispositivos elaborado para a realização do monitoramento do paciente que ali reside, o *firewall* deveria conter regras que permitissem a comunicação de cada dispositivo e ao mesmo tempo permitisse a utilização dos demais dispositivos da rede com segurança.

Na proposta apresentada neste trabalho, a segurança para a rede dos pacientes é tratada como um serviço, possuindo uma *flow table* específica. Há também *flow tables* com regras para proteção do *access point*. As *flow entries* que proveem segurança às *things* devem constar nas *flow tables* dos serviços que as utilizam.

Com a definição do *firewall* das redes domésticas através de SDN ganha-se escalabilidade na gerência da segurança, pois além das regras comuns definidas para serem utilizadas em todas as residências, a ThingsFlow instalará as *flow tables* dos serviços sendo providos em cada residência, as quais implementam a segurança para as *things* utilizadas. As *flow tables* são instaladas através do OpenFlow quando os *access points* conectam ao controlador, algumas *flow entries* são instaladas posteriormente, conforme os dispositivos tentam transmitir algum pacote fazendo com que os *access points* solicitem regras de encaminhamento ao controlador. Com a automatização da configuração dos *access points* evita-se o risco da ocorrência de erros de processos realizados manualmente; mesmo quando automatizado a quantidade de detalhes pode

Tabela 5.1 – Políticas de segurança implementadas nos *access points*

Protocolo	Origem	Destino	Ação
	access point: *input*, *output*		
Todos	Administração	*access point*	Aceitar
SSH	LAN	*access point*	Negar
Telnet	LAN	*access point*	Negar
HTTP	LAN	*access point*	Negar
HTTPS	LAN	*access point*	Negar
Todos	LAN	*access point*	Aceitar
Todos	WAN	*access point*	Negar
Todos	*access point*	Administração	Aceitar
Todos	*access point*	LAN	Aceitar
Todos	*access point*	Todos	Negar
	Dispositivos de monitoramento: *encaminhamento*		
HTTP	*things* Withings	Serviço	Aceitar
HTTP	Serviço	*things* Withings	Aceitar
FTP	Sensor de presença	Serviço	Aceitar
FTP	Serviço	Sensor de presença	Aceitar
Todos	LAN	WAN	Negar
	Rede doméstica: *encaminhamento*		
SSH	WAN	LAN	Negar
Telnet	WAN	LAN	Negar
HTTP	WAN	LAN	Negar
HTTPS	WAN	LAN	Negar
Todos	LAN	Todos	Aceitar
Todos	Todas	LAN	Aceitar

ocasionar erros no momento que o mecanismo de automatização é definido ou configurado.

Há uma limitação técnica para a implementação completa do *firewall* utilizando SDN, o Open vSwitch, em sua versão estável atual (2.3), não suporta *tracking* de conexões (*conntrack*), recurso necessário para implementação *stateful* do *firewall*, ou seja, onde regras são aplicadas também levando em conta o estado de uma conexão (nova, estabelecida, relacionada). Mesmo já sendo possível utilizar a versão 2.4 que contém suporte a *tracking*, o conceito de *stateful* no OpenFlow ainda não está maduro. Neste trabalho a abordagem empregada para a implantação das políticas de segurança é a *stateless*. Para contornar as limitações técnicas, seria possível utilizar o iptables[2] em paralelo, instalado por padrão no OpenWRT, contudo, optou-se por utilizar exclusivamente o OpenFlow valendo-se dos benefícios de escalabilidade proporcionados.

A tabela 5.1 ilustra as políticas de segurança empregadas nos ambientes monitorados. A implementação das regras nos *access points* está disponível no apêndice D, apresentadas utilizando da sintaxe do Open vSwitch.

Uma outra conveniência de se utilizar SDN para definir o encaminhamento dos pacotes realizado pelos *access points* é que os contadores do OpenFlow fornecem as mesmas informa-

[2]Iptables: http://www.netfilter.org/projects/iptables

ções para a gerência dos dispositivos que o mecanismo antigo, composto de “SNMP Agent”, “Proxy Agent” e MIB, ou seja, informações sobre quando as *things* se comunicaram. Apesar dessa informação estar implícita na coleta de dados, é interessante haver mecanismos independentes de gerência, pois a informação implícita na coleta de dados é que houve comunicação entre as duas pontas, *thing* e servidor. Os contadores indicam quantos pacotes foram encaminhados, podem ser utilizados por outros serviços e também para fins de *debug*. O fato dos dados de alguma coleta não terem sido transmitidos não significa, necessariamente, que a medição não foi realizada, ou que o dispositivo de monitoramento esteja com problemas no funcionamento, pois podem ter ocorridos problemas ao longo do caminho que tenham impedido que os dados chegassem ao servidor. O modo de operação das *things* não será alterado enquanto a bateria for um limitante, portanto é importante que a gerência forneça meios de se adaptar ao funcionamento dos dispositivos.

A gerência baseada em SDN proposta neste trabalho é desenvolvida através de aplicações, implementadas em cada serviço, que fazem uso dos contadores coletados dos *access points* pela ThingsFlow. Os desenvolvedores podem agregar à gerência das *things* quaisquer outros mecanismos de gerenciamento disponíveis, tal como SNMP e ICMP, dentre outros, e os transmitidos pelas *things* durante a transmissão dos dados, como o nível de bateria.

6 CONSIDERAÇÕES FINAIS

Por refatoração, conceito oriundo da Engenharia de Software, entendemos a reescrita de trechos de código com o objetivo de aprimorar um programa internamente, mantendo, porém, seu comportamento externo. Neste trabalho apresentamos a refatoração de uma proposta de *middleware* para *Internet of Things* utilizando *Software-Defined Networking*. Na proposta anterior, a qual refatoramos, as funcionalidades do *middleware* para viabilizar o monitoramento de pacientes com doenças crônicas era implementado, quase que por inteiro, em *access points*, os quais acabavam por concentrar os serviços da rede, monitoramento de pacientes, segurança e a gerência dos dispositivos.

Através da refatoração, as funcionalidades do *middleware* foram distribuídas entre diferentes servidores restando aos *access points* a função de encaminhar os pacotes de rede e estabelecer um canal seguro para o envio dos dados coletados. SDN possibilitou a orquestração da comunicação entre as *things* e seus respectivos serviços através da manipulação das tabelas de encaminhamento dos *access points* por uma aplicação desenvolvida para prover às diversas redes domésticas a inteligência necessária para a coexistência de diferentes serviços e o uso da rede pelos residentes. A refatoração proporcionou escalabilidade, dinamicidade e autonomia no que se refere ao gerenciamento de rede em ambientes distribuídos com serviços baseados em IoT.

O protótipo para verificação da viabilidade da proposta utilizou recursos que vem sendo desenvolvidos e testados por comunidades de desenvolvedores e empresas, como o OpenWRT, o Open vSwitch e a libfluid. Os *access points*, rodando OpenWRT, receberam as funcionalidades do Open vSwitch para atuarem como *switches* OpenFlow enquanto as funcionalidades de cada serviço, tais como a coleta e processamento dos dados lidos pelas *things* e funcionalidades relacionadas à gerência dos dispositivos, passaram a executar em servidores dedicados dos serviços sendo prestados. A coexistência de diferentes serviços foi possibilitada pela separação do tráfego baseada nas características dos pacotes que passam pelos *access points*, os quais são analisados pelo *switch* OpenFlow e encaminhados de acordo com regras definidas para os dispositivos pelos provedores dos serviços.

O funcionamento da solução foi automatizado, em partes, pela utilização do OpenFlow e o restante pelo controlador Derailleur e a aplicação ThingsFlow. A configuração dos *access points* é feita pela ThingsFlow no momento em que os *access points* são inicializados, as regras de encaminhamento são instaladas pelo controlador nos *access points* de acordo com os serviços que são prestados em cada ambiente. Devido ao funcionamento padrão do OpenFlow, no qual

os *switches* solicitam ao controlador regras de encaminhamento para os pacotes cujas características não são contempladas por nenhuma das regras existentes nas *flow tables*, o conjunto de regras relacionadas aos serviços e à segurança das redes são mantidas atualizadas e sempre contemplam os dispositivos existentes no ambiente. SDN automatizou a configuração dos *access points* proporcionando ganhos em escalabilidade e eliminando a necessidade de configuração manual dos equipamentos para atender as necessidades específicas de cada ambiente.

A presente proposta foi avaliada tendo como base o contexto do monitoramento de pacientes do projeto REMOA, cenário no qual foi desenvolvida a proposta de *middleware* anterior, onde o *middleware* era composto por módulos implementados em *access points*. A comparação para demonstrar os benefícios foi feita levando em conta os processos de desenvolvimento e implantação dos serviços nas duas propostas. Quando comparado à proposta anterior, a abordagem baseada em SDN proporcionou benefícios ao potencializar o desenvolvimento e manutenção de serviços e funcionalidades de forma mais ágil e mais completa em termos de funcionalidades, uma vez que os desenvolvedores passam a poder lançar mão de recursos indisponíveis quando o alvo do desenvolvimento é o *hardware* e o sistema repletos de restrições dos *access points*.

A utilização de contadores do OpenFlow, coletados através das regras de encaminhamento instaladas nos *access points* para cada dispositivo presente na rede, coletados pelos *access points* e disponibilizados pela ThingsFlow aos serviços, possibilitou a construção de uma visão mais ampla do ambiente monitorado ao fazer uso dos dados de utilização da rede por parte dos dispositivos não associados ao monitoramento de saúde, tais como *notebooks*, smartphones, televisores e outros. A utilização dos dados dos dispositivos presentes no ambiente por qualquer serviço habilita uma forma de cooperação para que se alcance um ou mais objetivos, aspecto almejado no contexto de IoT.

Na proposta refatorada foram obtidas as mesmas informações utilizadas para a gerência dos dispositivos que na proposta anterior, ou seja, quando se comunicaram através da rede e dados de bateria, quando aplicável, informação extraída dos dados enviados pelo dispositivo ao serviço. Contudo, com SDN, simplificou-se consideravelmente o mecanismo para obtenção das informações de gerência dos pontos de vista da arquitetura da solução, desenvolvimento e manutenção.

Com este trabalho demonstrou-se que a utilização de SDN em cenários de IoT é viável. Além de demonstrar a viabilidade, também demonstrou ao longo de suas seções os benefícios obtidos pela da utilização de SDN em relação à proposta anterior. A performance dos *access points* não foi avaliada no trabalho por não ter sido um limitante em nenhum momento, porém esta questão foi levada em conta tendo em mente as limitações de recursos do modelo utilizado.

Uma proposta interessante, como complementação do presente trabalho, é determinar os recursos mínimos para a implantação da solução proposta, mesmo que os componentes que compõe o *access point* possam ser instanciados facilmente em dispositivos mais capacitados. Um problema conhecido em abordagens centralizadas é que o nodo central, neste caso o controlador, é um ponto de falha que pode comprometer o funcionamento dos demais elementos que compõem a solução. Esta proposta não prevê a utilização de mais de um controlador, em especial próximo aos *access points*, o que também pode ser tema de um trabalho futuro.

REFERÊNCIAS

ACCESS to open data. **Greater London Authority**, 2014. Disponível em: <http://www.london.gov.uk/priorities/business-economy/vision-and-strategy/smart-london/access-to-open-data>. Acesso em: 19 abr. 2014.

ARBIZA, L. M. R. et al. Refactoring internet of things middleware through software-defined network. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, 30., 2015, Salamanca, Spain. **Proceedings...** New York, NY, USA: ACM, 2015. p. 640–645.

ARBIZA, L. M. R. et al. SDN-Based Service Delivery in Smart Environments. In: NOVAIS, P. et al. (Ed.). **Intelligent Distributed Computing IX**. Guimarães, Portugal: Springer International Publishing, 2016. cap. 11, p. 475–484.

ASHTON, K. That ”internet of things” thing. **RFID Journal**, 2009. Disponível em: <http://www.rfidjournal.com/article/view/4986>. Acesso em: 07 abr. 2012.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A survey. **Computer Networks**, v. 54, p. 2787–2805, 2010.

AUTO-ID Labs - Annual Report II. **Auto-ID Labs**, 2013. Disponível em: <http://www.autoidlabs.org/uploads/media/Auto_ID_Labs_Annual_report_2013v18-final.pdf>. Acesso em: 10 abr. 2014.

BANDYOPADHYAY, D.; SEN, J. Internet of Things: Applications and Challenges in Technology and Standardization. **Wireless Personal Communications**, Springer US, v. 58, n. 1, p. 49–69, 2011.

BARONTI, P. et al. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards. **Computer Communications**, v. 30, n. 7, p. 1655–1695, 2007. ISSN 0140-3664.

BLUETOOTH Basics. **Bluetooth Special Interest Group**, 2013. Disponível em: <http://www.bluetooth.com/Pages/Basics.aspx>. Acesso em: 03 ago. 2013.

BLUETOOTH User Interface Flow Diagrams For Bluetooth Secure Simple Pairing Devices. **Bluetooth Special Interest Group**, p. 17–18, 2007. Disponível em: <https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=86173>. Acesso em: 03 ago. 2013.

CARAGUAY, A. L. V. et al. Sdn: Evolution and opportunities in the development iot applications. **International Journal of Distributed Sensor Networks**, 2014.

CHAMBERLAIN, J. et al. Ibm websphere rfid handbook: A solution guide. **IBM Redbooks**, 2010. Disponível em: <http://www.redbooks.ibm.com/redbooks/pdfs/sg247147.pdf>. Acesso em: 16 nov. 2013.

CHETTY, M.; FEAMSTER, N. Refactoring network infrastructure to improve manageability: a case study of home networking. **ACM SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 42, n. 3, p. 54–61, 2012. ISSN 0146-4833.

COETZEE, L.; EKSTEEN, J. The internet of things - promise for the future? an introduction. In: IST-AFRICA CONFERENCE PROCEEDINGS, 2011, Gaborone, Botswana. **Proceedings...** Danvers, MA, USA: IEEE, 2011. p. 1–9.

CONSTRAINED Application Protocol (CoAP). **IETF Working Draft**, 2013. Disponível em: <http://tools.ietf.org/html/draft-ietf-core-coap-18>.

COSTA, P. et al. Reconfigurable Component-based Middleware for Networked Embedded Systems. **International Journal of Wireless Information Networks**, Kluwer Academic Publishers-Plenum Publishers, v. 14, n. 2, p. 149–162, 2007.

CPQD Selected as Winner and Recipient of $50.000 Grand Prize. **Open Networking Foundation**, 2014. Disponível em: <https://www.opennetworking.org/component/content/article/26-news-and-events/press-releases/1431-open-networking-foundation-announces-openflow-driver-contest-winner>. Acesso em: 02 out. 2014.

DOMINGO, M. C. An overview of the Internet of Things for people with disabilities. **Journal of Network and Computer Applications**, Academic Press Ltd., London, UK, UK, v. 35, n. 2, p. 584–596, 2012.

DUBLIN City Council. **IBM Smarter Planet**, Dublin, Ireland, 2013. Disponível em: <http://public.dhe.ibm.com/common/ssi/ecm/en/imc14829ieen/IMC14829IEEN.PDF>. Acesso em: 2 jul. 2014.

DUNKELS, A.; VASSEUR, J.-P. IP for Smart Objects. **IPSO Alliance Whitepapers**, 2010. Disponível em: <http://dunkels.com/adam/dunkels08ipso.pdf>.

EGILMEZ, H. E. et al. OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks. In: SIGNAL & INFORMATION PROCESSING ASSOCIATION ANNUAL SUMMIT AND CONFERENCE (APSIPA ASC), 2012 ASIA-PACIFIC, 2012, Koc Univ., Istanbul, Turkey. **Proceedings...** Danvers, MA, USA: IEEE, 2012. p. 1–8.

ELECTRONIC Product Code (EPC): An Overview. **EPCglobal INC**, 2007. Disponível em: <http://www.gs1.org/docs/epcglobal/an_overview_of_EPC.pdf>. Acesso em: 23 ago. 2013.

FOWLER, M.; BECK, K. **Refactoring: Improving the Design of Existing Code**. 431. ed. Boston, MA, USA: Addison-Wesley, Boston, 1999. 25 p.

GAMA, K.; TOUSEAU, L.; DONSEZ, D. Combining heterogeneous service technologies for building an Internet of Things middleware. **Computer Communications**, v. 35, n. 4, p. 405–417, 2012.

GIUSTO, D. et al. (Ed.). **The Internet of Things**. New York, US: Springer-Verlag, 2010.

GRAF, T. Why Open vSwitch? **Open vSwitch page on Github**, 2014. Disponível em: <https://github.com/openvswitch/ovs/blob/master/WHY-OVS.md>. Acesso em: 05 jan. 2015.

GRANVILLE, L. Z. et al. On the management performance of networked environments using web services technologies. In: LEE, I. (Ed.). **Handbook of Research on Telecommunications Planning and Management for Business**. [S.l.]: IGI Global, 2009. cap. 95, p. 1768–1785.

GUINARD, D. et al. From the internet of things to the web of things: Resource-oriented architecture and best practices. In: UCKELMANN, D.; HARRISON, M.; MICHAHELLES, F. (Ed.). **Architecting the Internet of Things**. Heidelberg, Germany: Springer Berlin, 2011. cap. 5, p. 97–129.

HINSSEN, P. The age of data-driven medicine. **Across Technology**, 2012. Disponível em: <http://datascienceseries.com/assets/blog/The_Age_of_Data-Driven_Medicine.pdf>. Acesso em: 13 nov. 2013.

HUNDT, R. Loop recognition in c++/java/go/scala. In: **Proceedings...** [S.l.: s.n.], 2011.

IOT-A: Internet of Things Architecture. **Internet of Things Architecture (IoT-A)**, 2013. Disponível em: <http://www.iot-a.eu/public>. Acesso em: 25 jul. 2013.

IPV6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. **IETF Requests for Comments**, n. 4919, 2007. Disponível em: <http://datatracker.ietf.org/doc/rfc4919/>.

KIM, H.; FEAMSTER, N. Improving network management with software defined networking. **Communications Magazine**, IEEE, Danvers, MA, USA, v. 51, n. 2, p. 114–119, February 2013.

KREUTZ, D. et al. Software-Defined Networking: A Comprehensive Survey. **CoRR**, 2014. Disponível em: <http://arxiv.org/abs/1406.0440>.

LAMBA, N. Innovative parking plan could help clear birmingham's traffic and skies « a smarter planet blog a smarter planet blog. **A Smarter Planet Blog**, 2013. Disponível em: <http://asmarterplanet.com/blog/2013/01/22902.html>. Acesso em: 30 ago. 2013.

LIU, Y.; ZHOU, G. Key technologies and applications of internet of things. In: INTELLIGENT COMPUTATION TECHNOLOGY AND AUTOMATION, 5., 2012, Zhangjiajie, Hunan. **Proceedings...** Danvers, MA, USA: IEEE, 2012. p. 197–200.

MAROTTA, M. A. et al. Through the internet of things - a management by delegation smart object aware system (mbdsas). In: COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 37., 2013, Kyoto, Japan. **Proceedings...** Danvers, MA, USA: IEEE, 2013. p. 732–741.

MATTERN, F.; FLOERKEMEIER, C. From the Internet of Computers to the Internet of Things. In: SACHS, K.; PETROV, I.; GUERRERO, P. (Ed.). **From Active Data Management to Event-Based Systems and More**. Heidelberg, Germany: Springer Berlin, 2010, (Lecture Notes in Computer Science, v. 6462). p. 242–259.

MCKEOWN, N. et al. OpenFlow: enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, 2008.

MIORANDI, D. et al. Internet of things: Vision, applications and research challenges. **Ad Hoc Networks**, v. 10, n. 7, p. 1497–1516, 2012.

MOREIRAS, A. M. Zigbee uses 6lowpan now! will your next lamp have ipv6? (*ZigBee usa agora 6loWPAN! Sua próxima lâmpada terá IPv6?*). **Centro de Estudos e Pesquisas em Tecnologia de Redes e Operações - ceptro.br**, 2013. Disponível em: <http://ipv6.br/zigbee-usa-agora-6lowpan-sua-proxima-lampada-tera-ipv6/>. Acesso em: 06 ago. 2013.

NEGROPONTE, N. A 30-year history of the future. **TED.com**, 2014. Disponível em: <http://www.ted.com/talks/nicholas_negroponte_a_30_year_history_of_the_future?utm_source=newsletter_daily&utm_campaign=daily&utm_medium=email&utm_content=image__2014-07-08\#t-650082>. Acesso em: 10 jul. 2014.

NETWORK Configuration Protocol - NETCONF. **IETF Request for Comments**, n. 6241, 2011. Disponível em: <https://tools.ietf.org/html/rfc6241>.

NETWORK Functions Virtualisation. **European Telecommunications Standards Institute**, 2015. Disponível em: <http://www.etsi.org/technologies-clusters/technologies/nfv>. Acesso em: 06 jan. 2015.

NETWORK Functions Virtualization Technology Leaflet. **European Telecommunications Standards Institute**, 2014. Disponível em: <http://www.etsi.org/images/files/ETSITechnologyLeaflets/NetworkFunctionsVirtualization.pdf>. Acesso em: 06 jan. 2015.

ONF Technical Library. **Open Networking Foundation**, 2014. Disponível em: <https://www.opennetworking.org/sdn-resources/technical-library#tech-spec>. Acesso em: ago. 2014.

OPENDAYLIGHT Technical Overview. **OpenDaylight**, 2014. Disponível em: <http://www.opendaylight.org/project/technical-overview>. Acesso em: 16 dez. 2014.

OPENFLOW Switch Specification: Version 1.3.2. **Open Networking Foundation**, 2013. Disponível em: <https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf>. Acesso em: 10 fev. 2014.

QIN, Z. et al. A software defined networking architecture for the internet-of-things. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, 2014, Krakow, Poland. **Proceedings...** Danvers, MA, USA: IEEE, 2014. p. 1–9.

RESOURCE ReSerVation Protocol (RSVP). **IETF Request for Comments**, n. 2205, 2005. Disponível em: <https://tools.ietf.org/html/rfc2205>.

ROSA, R. et al. Network function virtualization: Perspectivas, realidades e desafios. In: MINICURSOS – SIMPóSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS, 32., 2014, Florianópolis, Brazil. **Proceedings...** Florianópolis, Brazil: SBC, 2014.

ROTHENBERG, C. E. et al. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In: WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKS, 1., 2012. **Proceedings...** New York, NY, USA: ACM, 2012. p. 13–18.

SEHGAL, A. et al. Management of resource constrained devices in the internet of things. **Communications Magazine**, IEEE, Danvers, MA, USA, v. 50, n. 12, p. 144–149, 2012. ISSN 0163-6804.

SHELBY, Z.; BORMANN, C. **6LoWPAN: The Wireless Embedded Internet**. [S.l.]: John Wiley & Sons, Ltd, 2009. v. 43. 244 p.

SHERWOOD, R. et al. Flowvisor: A network virtualization layer. **OpenFlow.org Archive**, 2009. Disponível em: <http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf>. Acesso em: 02 mar. 2014.

SHOUP, D. Cruising for parking. **Access Magazine**, p. 16–22, 2007. Disponível em: <http://shoup.bol.ucla.edu/CruisingForParkingAccess.pdf>. Acesso em: 30 ago. 2013.

SOFTWARE-DEFINED Networking: The New Norm for Networks. **Open Networking Foundation**, p. 12, 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>. Acesso em: 31 jul. 2013.

STANDARD for an Architectural Framework for the Internet of Things. **IEEE Standards Association – P2413 Group**, 2014. Disponível em: <http://grouper.ieee.org/groups/2413/>. Acesso em: 23 set. 2014.

TAN, L.; WANG, N. Future internet: The internet of things. In: INTERNATIONAL CONFERENCE ON ADVANCED COMPUTER THEORY AND ENGINEERING, 3., 2010, Chengdu, China. **Proceedings...** Danvers, MA, USA: IEEE, 2010. p. 376–380.

TAROUCO, L. M. R. et al. Internet of things in healthcare : Interoperatibility and security issues. In: IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS, INTERNATIONAL WORKSHOP ON MOBILE CONSUMER HEALTH CARE NETWORKS, SYSTEMS AND SERVICES, 1., 2012, Ottawa, Canada. **Proceedings...** Danvers, MA, USA: IEEE, 2012. p. 6121–6125.

TECHNOLOGY Leaders Establish the AllSeen Alliance to Advance the 'Internet of Everything'. **Allseen Alliance**, 2013. Disponível em: <https://allseenalliance.org/announcement/technology-leaders-establish-allseen-alliance-advance-internet-everything>. Acesso em: 26 out. 2014.

THE Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams. **IETF Requests for Comments**, n. 6553, 2012. Disponível em: <http://tools.ietf.org/html/rfc6553>.

TRAUB, K. et al. **The GS1 EPCglobal Architecture Framework, v.1.5**. 2013. Disponível em: <http://www.gs1.org/gsmp/kc/epcglobal/architecture>. Acesso em: 9 mai. 2013.

USING all wireless networks around me. **Stanford's OpenFlow Channel**, 2010. Disponível em: <https://www.youtube.com/watch?v=ov1DZYINg3Y>. Acesso em: 14 dez. 2014.

WANG, Y. et al. Netfuse: Short-circuiting traffic surges in the cloud. In: INTERNATIONAL CONFERENCE ON COMMUNICATIONS, 2013, Budapest, Hungary. **Proceedings...** nvers, MA, USA: IEEE, 2013. p. 3514–3518.

WEE, S. The next generation of networked experiences. In: INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE DIGEST OF TECHNICAL PAPERS, 2014, San Francisco, CA, USA. **Proceedings...** Danvers, MA, USA: IEEE, 2014. p. 29–35.

WHAT is NFV – Network Functions Virtualization? **SDxCentral**, 2015. Disponível em: <https://www.sdxcentral.com/resources/nfv/whats-network-functions-virtualization-nfv/>. Acesso em: 07 jan. 2015.

WU, G. et al. M2m: From mobile to embedded internet. **Communications Magazine**, IEEE, Danvers, MA, USA, v. 49, n. 4, p. 36–43, 2011.

YANG-API Protocol. **IETF Working Draft**, 2012. Disponível em: <http://tools.ietf.org/html/draft-bierman-netconf-yang-api-01>.

YAP, K.-K. et al. Openroads: Empowering research in mobile networks. **ACM SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 40, n. 1, p. 125–126, 2010.

YIAKOUMIS, Y. et al. Slicing home networks. In: ACM SIGCOMM WORKSHOP ON HOME NETWORKS, 2., 2011, Toronto, ON, Canada. **Proceedings...** New York, NY, USA: ACM, 2011. p. 1–6.

ZIGBEE Alliance FAQ. **ZigBee Alliance**, 2013. Disponível em: <http://www.zigbee.org/About/FAQ.aspx>. Acesso em: 04 ago. 2013.

ZIGBEE IP: The First Open Standard for IPv6-Based Wireless Mesh Networks. **ZigBee Alliance**, 2013. Disponível em: <http://www.zigbee.org/default.aspx?Contenttype=ArticleDet&tabID=332&moduleId=&Aid=446&PR=PR>. Acesso em: 06 ago. 2013.

ZORZI, M. et al. From today's INTRAnet of things to a future INTERnet of things: a wireless- and mobility-related view. **Wireless Communications**, IEEE, Danvers, MA, USA, v. 17, n. 6, p. 44–51, 2010.

ANEXO A — ARTIGO PUBLICADO NO ICC'12 MOBICHESS

- Título: Internet of Things in Healthcare: Interoperatibility and Security Issues
- Conferência: IEEE International Conference on Communications (ICC'12), International Workshop on Mobile Consumer Health Care Networks, Systems and Services (MobiCHeSS)
- URL: http://www.cms.livjm.ac.uk/mobichess
- Data: 10-15 de junho de 2012
- Local: Ottawa, Canadá

Internet of Things in Healthcare : Interoperatibility and Security Issues

Liane Margarida Rockenbach Tarouco, Leandro Márcio Bertholdo, Lisandro Zambenedetti Granville, Lucas Mendes Ribeiro Arbiza, Felipe Carbone, Marcelo Marotta, José Jair Cardoso de Santanna
Institute of Informatics
Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil
liane@penta.ufrgs.br, berthold@penta.ufrgs.br, granville@inf.ufrgs.br, lucas@pop-rs.rnp.br, felipe.carbon@inf.ufrgs.br, mamarotta@inf.ufrgs.br, jjcsantana@inf.ufrgs.br

***Abstract*—Internet of Things devices being used now expose limitations that prevent their proper use in healthcare systems. Interoperability and security are especially impacted by such limitations. In this paper, we discuss today´s issues, including benefits and difficulties, as well as approaches to circumvent the problems of employing and integrating Internet of Things devices in healthcare systems. We present this discussion in the context of the REMOA project, which targets a solution for home care/telemonitoring for patients with chronic illnesses.**

Keywords-Internet of Things (IoT); Healthcare; Security

I. INTRODUCTION

Internet of Things (IoT) encompasses a set of technologies that enable a wide range of appliances, devices, and objects (or simply "things") to interact and communicate among themselves using networking technologies. Human beings supply most of the contents and information found on Internet so far, whereas in IoT, small devices are frequently the active element that provides the information. There are many applications for IoT; including healthcare systems, which are the main focus of this paper. Healthcare systems use a set of interconnected devices to create an IoT network devoted to healthcare assessment, including monitoring patients and automatically detecting situations where medical interventions are required.

It is generally recognized that the chronically ill, such as those with hearth failure, hypertension, respiratory diseases or diabetes require medical, hospital, and emergency services more often than regular patients [1]. Information and communication technologies are amongst the tools that could help mitigate some of the problems associated with aging populations, increased rates of chronic illnesses, and shortage of health professionals, and, at the same time, facilitate service reorganization. Modern measuring devices, such as, blood pressure, weight, and movement sensors, incorporate communication capabilities. They can create IoT networks implemented for home telemonitoring. These same devices are those usually employed by health workers to check the overall condition of patients with chronic illnesses.

REMOA[1] is a project which targets home solutions for care/telemonitoring of patients with chronic illnesses. Different strategies and protocols for general data exchange among monitoring devices like blood pressure monitors and movement sensors are considered. The project also encompasses the design and implementation of a healthcare-devoted middleware. The general system collects information from different sensing devices through a middleware that provides interoperability and security needed in the context of Internet of Things for healthcare. Monitoring devices are connected via wireless networking technology. The monitoring application frontend, which is functionally similar to a network manager, is responsible for storing, aggregating, consolidating, and comparing the collected information against historical series so that when thresholds are crossed, the frontend system may issue alerts or execute specific procedures. When limits are crossed, depending on threshold policies, alarms are triggered to enable health workers to promptly react to health-related events.

The topology of the typical home telemonitoring healthcare network includes an intermediary processing proxy that forwards sensing data to the remote server responsible for data analysis, consolidation, and critical events detection. In this way, surrounded by smarter environment that employs communication and information technologies, citizens (especially elders) and medical teams find better conditions to proceed with proper domestic-based treatments.

This paper is organized as follows: In Section 2 we list and describe the sensing devices being used. In Section 3 we present solutions to deal with interoperability issues in healthcare systems. In Section 4 we deal with security problems in telemonitoring healthcare systems, while in Section 5 we describe strategies and workarounds for solving current problems. Conclusions are presented in Section 6.

[1] REMOA: *Rede Cidadã de Monitoramento do Ambiente Baseado no Conceito de Internet das Coisas* (Citizen Network for Environment Monitoring Based on the Concept of Internet of Things).

II. DEVICES TO BE CONNECTED

Based on requirements of a home telemonitoring system for patients with chronic diseases (e.g., blood pressure, patient recovery of hospitalization), some devices have been selected (Figure 1) to be used in the REMOA project, in order to monitor some aspects of patients' health. These devices have Wi-Fi interfaces and features that enable interoperability and data transmission. Devices selected for use in the project are:

- A Panasonic BL-C230A Wi-Fi IP camera to detect the movements of monitored patients - The lack of movement in periods of the day when movements are expected may indicate some critical problem. The selected device is able to detect movements based on three different sources of data: 1) image; 2) sounds; and 3) body heat. Every movement detected sends forth an alert message that carries the images which are sent to an e-mail address or uploaded to an FTP (File Transport Protocol) server;
- A wireless body scale fitted with a Wi-Fi interface manufactured by Withings fitted with software which calculates the patients' percentage of fat, muscle mass, and body mass index. The standard configuration of this device connects the wireless network immediately after weighing and sends the data to an URL internally set to the manufacturer's website;
- A Withings blood pressure device with the peculiarity that its operation depends on the connection to an Apple device (iPad, iPhone. or iPod Touch). The operation is similar to the scale (both products are from the same manufacturer) with the difference that this device is operated via an application installed in an Apple device. The application is also responsible for sending measurement data to the remote server and informing the patient about the progress of the measurements, warning of possible errors in handling the device, such as poor posture or sudden movements that affect correct measurement of the patient's blood pressure.

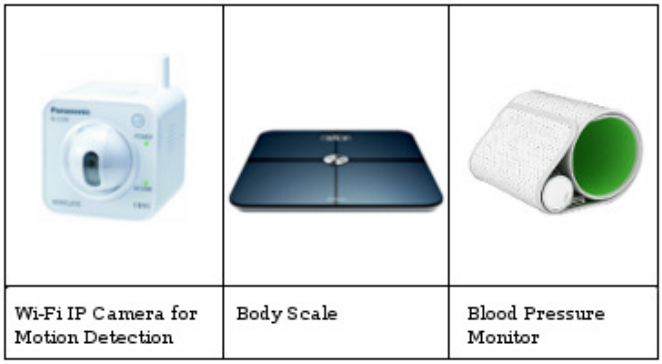


Figure 1. Devices being connected for home healthcare telemonitoring

The reading data sent from monitoring devices and patients' information will be accessed through a web browser on a regular computer and in Apps on smartphones and tablets. These devices are used by the medical staff to monitor patients' health and by patients who can access information about their own treatment and may interact with medical staff and other patients doing social networking and other activities typical of the context of Patient 2.0 where patients participate actively in their own treatment [2].

The device responsible for centralizing the data transmission of all selected device is a wireless Access Point (AP) that supports OpenWRT or DD-WRT. This approach enables the development and deployment of additional software, and supports protocols such as IPv4, IPv6, NAT, SNMP proxy and serves as a gateway to other monitoring devices.

III. INTEROPERABILITY ISSUES

The REMOA project was designed to deploy a monitoring environment based on Wi-Fi (802.11). At first, we expected that the various monitoring devices would be able to communicate with remote servers just through a Wi-Fi access point, which would serve as a gateway between Internet and our monitored environment. This approach, although applicable, has restricted the diversity of medical devices available, mainly because of the tendency to use Bluetooth technology in healthcare devices. This trend is due to the low cost and low power consumption of Bluetooth-enabled devices. Despite its major availability, we have only found a few solutions in the market which enable the interconnection between Bluetooth (802.14)/ZigBee (802.15.4) networks and Internet. This point was a challenging one adding difficulties in sending data collected by healthcare devices to a remote server over the network

In a fully Wi-Fi-based environment, the interoperability problem could be solved through a Wi-Fi/Bluetooth gateway, providing all management capabilities desired, either by using the Simple Network Management Protocol (SNMP) or Web services. However, even healthcare devices built with Wi-Fi interfaces aren't fully suitable for use in a flexible monitoring environment. We found that most of such devices operate by communicating with servers and proprietary system, as is the case of the Withings body scale. This problem required a circumvent solution to be implemented in the communication infrastructure. This workaround is centered in a Access Point (AP) running a Linux (OpenWRT / DD-WRT), and thus allowing the addition of the software to the AP. Although the transmitted data layout is proprietary, in most cases, like the Withings body scale, a traffic analysis revealed that the data is transmitted in plain text and using the JSON format, which represents a security weakness because it allows eavesdropping during the connection

IV. SECURITY ISSUES

The fact that personal private data will be collected through telemonitoring implies the need for strategies and mechanisms to ensure adequate security and privacy. As highlighted in [3], "having every 'thing' connected, new security and privacy problems arise, e.g., confidentiality, authenticity, and integrity of data sensed and exchanged by 'things'." This author lists the standard security requirements:

- Resilience to attacks - The system has to avoid single points of failure and should adjust itself to node failures;
- Data authentication - As a principle, retrieved addresses and object information must be authenticated;
- Access control - Information providers must be able to implement access control on the data provided;
- Client privacy - Measures need to be taken that only the information provider is able to infer from observing the use of the lookup system.

Nevertheless, in view of the mobility context where health agents will access patient's data when visiting their homes, it implies the need for "privacy enhancing technologies," as mentioned in [4], to ensure that the patient's personal data is protected against non-authorized access.

The solution for interoperability and management of home care network has been projected to use amplified network management concepts in order to include Internet of Things management separating the capabilities and functions of an object from the implementing technology such as RFID or WLAN as proposed in [5]. In this way, the management and security architectures must be oriented to monitoring events to ensure the security of medical devices and mobile devices and further use a unified authentication design for the whole set of systems, i.e., medical and infrastructures.

The planned authentication services use Shibboleth, a middleware layer for authentication and access control developed as an Internet2 project [6]. By using Shibboleth it is possible to take advantage of a Federated Authentication service already implanted by RNP (Rede Nacional de Ensino e Pesquisa - the Brazilian Research and Education Network). CAFe ("Comunidade Acadêmica Federada") , a federation of Brazilian research and education institutions acting as identity and service provider. CAFe allows every user to have just one user account in his/her origin institution that is valid for all services offered by the federation, thus eliminating the need of multiple passwords and multiple registration processes (Single Sign-On SSO). However, in the IoT context other security issues arise, to data integrity and user privacy because mobile devices are potential targets to malicious attacks, which requires further advancements in terms of studies and security countermeasures. In the case of healthcare environments, avoiding successful attacks to the system or at least mitigating them are primary goals because failures or information leaks can represent damages to the patients. In contrast to other domains that can absorb some costs of system abuse, healthcare systems cannot. Once sensitive information about an individual's health problems is uncovered and social damage is done, it is impossible to revoke the information [7].

Attackers have well defined goals when focusing on mobile devices. Usually, attacks aim at stealing users' information, attacking devices resources, or even shutting down some applications [8]. There are many threats surrounding mobile devices; the majority of these threats are inherited from conventional computing systems and adapted for mobile devices. However, some threats get more attention because of the potential problems they can cause to the systems. Examples of these threads are:

i) Man-in-the-Middle;

ii) Routing diversion attack;

iii) Denial of service aiming to cause a Battery Exhaustion in the device.

These attacks, when applied exclusively on mobile devices, expedite given factors such as: a) natural use of broadcast for communication; b) lack of certification sources; c) use of batteries as power source; and d) mobility. Therefore, to manage such a large range of mobile devices mapping and mitigating the most relevant threads in a given context is an essential task.

In this context, devices that operate in an wake-up and input data approach, as the Withings Body Scale used in this project, are less vulnerable to battery exhaustion attacks because the exposure time boils down a few seconds in which the device is connected through Wi-Fi interfaces and there is no service to be attacked. Moreover, interception and connection issues persist and the lack of any server/service authentication infrastructure to where the data will be sent and the lack of obscurity of content are security gaps that demanded a circumvent solution in the REMOA project.

V. CIRCUMVENT APPROACH

Given the problems described in the previous section, adjustments became necessary in the communication infrastructure initially planned, in order to enable using the concept of homecare IoT with a safer approach. In REMOA, an access point is used to provide several features and additional security functions (Figure 2):

i) A gateway connecting the devices BT / ZigBee to the Internet with encryption;

ii) Transparent HTTP proxy, to manipulate the transient data and account information from cloud IoT to Internet;

iii) IoT device management;

iv) Authorization, Authentication and Accounting for Internet access and systems used by patient and health agents.

The role of the gateway is to translate messages from one technology to another, for example, a computer connected via Ethernet, trying to query a device connected in a PAN (IEEE802.15.4). In addition, one of the requirements of this project is the aggregation of additional protection mechanisms for the transmission and this effect magnified the role of the gateway, which may works also as a transparent proxy.

The transparent proxy analyzes the data flow between client and server. This data stream can be modified e.g., by filtering, changing origin and destination, and may be blocked. In addition to modifications, the proxy provides extra features like exchanged-information storage, audititing through system logs,

traffic accounting, and SNMP MIB updating (Management Information Base) that will be used for remote management.

The management of proxy clients can be accomplished through two approaches: Web services and SNMP proxy agent. Web services allow retrieving information from devices that operate as services [9]. Services are routines that perform requests to the devices according to the technology supported by them (e.g., IEEE802.15.4, Bluetooth, Wi-Fi). These routines are accessed through remote requests. These requests are normally held on widespread protocols (e.g., HTTP, SOAP) [10]. Each of the protocols is associated with an approach to Web Services, such as Service Oriented Architecture (SOA) or Resource Oriented Architecture (ROA). The SNMP proxy agent is a conceptual database fulfilled by relevant information of management, i.e., one MIB [11]. The purpose of a MIB is to enable the management of devices that do not have SNMP agent installed. This management is accomplished through the proxy agent that accesses the managed devices using proprietary technologies. For example, a manager can send requests to an SNMP proxy agent, which retrieves the MIB site or collects the requested information directly from the device using the communication technology that has the same ability to use (Wi-Fi, Bluetooth).

The device used in the project is an access point that already has an SNMP agent. The SNMP agent is the necessary infrastructure for the SNMP proxy agent. With this existing infrastructure, the use of proxy agent solution becomes more interesting than build up a Web service. To meet the objectives of the project, the proxy agent requires modifications to allow recovery of information on each of the connected devices. This new information includes: (i) the type of device found, (ii) the unique identifier of the device, (iii) the energy level from the battery, (iv) a number on the measurement performed by the device, (v) time of the last request, (vi) the total time of operation of the device, (vii) the number of managed devices. Figure 2 shows the structure of the modules involved in communicating with IoT devices. The received data from the IoT devices are initially treated by the transparent proxy for changing the destination IP address (pre-configured by the manufacturer on some devices). Additionally, the communication is encrypted and data generated by the devices is extracted and update to the MIB (counters, status).

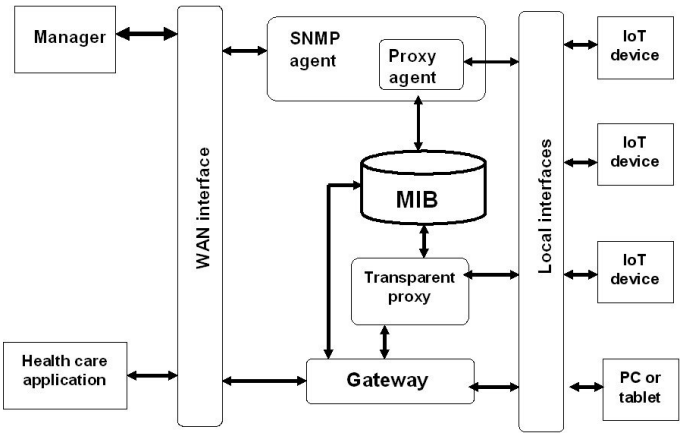


Figure 2. Infrastructure for IoT access via proxy agent

Figure 2 also shows the management requests for the IoT devices being manipulated by an SNMP Proxy Agent. When the manager sends commands to the agent, trying to get information about status or behavioral aspects of a device, the Proxy Agent will use any possible means to provide the information. Sometimes, this course of action could lead to implement translators to proprietary protocols or even store data from any previous communication that this device has done using the network and consequently, always keeping the MIB updated. This way, the information is forwarded back to the manager as an SNMP response message.

Any simple SNMP request must contain a unique identifier targeting managed objects. This unique identifier must be generated/configured during the activation process of the device. The activation process establishes an association with the device and the patient. This association is generated by combining a unique device identifier and the base unique identifier. Each patient uses his/her monitoring devices connected to a specific access point. This combination is registered in the telemonitoring system. Every time a new device is added, the external system records the access point used and stores the complete information in the MIB. Using this architecture it is possible to access and manage this kind of information just by using standards SNMP access. Managed objects hold the full data obtained from any device.

The transparent proxy is also the component responsible for identifying the originator device of the data, time stamping it, and applying encryption mechanisms to protect the transmission against unauthorized access.

Furthermore, considering that health workers should be allowed to access data of visited patients but only under policies regarding access and use of health data, an authentication system based was defined on a set of parameters including user id, password identification, time, device identification and geographic location of the device. This authentication system is used also to control access to patient own data. End-user devices like tablets and smartphones with GPS are being used for this extended security mechanism. An health worker should be able to access data from a patient using his/her mobile device while in the patient's home on a scheduled visit. But access is not permitted from other places or during periods of inactivity specifically linked to the patient in order to protect patient privacy and considering ethics considerations as discussed in [4]. These strategies aim to answer to security requirements of data authentication, access control, and client privacy. The requirement of resilience to attacks may be handled with more than one WAN connection using different kinds of communication like cell phones, cable modem, or even dial up and the switching from one to another may be managed by the gateway component. But this approach is still being incorporated to the solution because of operational characteristics of the access point selected to be used as infrastructure for this project, which intended to provide home healthcare services using devices already available on the market in contrast with other proposals using phone calls or implantable devices as described in [12].

The healthcare application runs in a remote server and manages telemonitoring applying filters on the data to prevent

reception of data from some health monitoring device that may be still active out of the follow-up period. Devices that were transferred from a home to another place without proper registration may send data to the system but this data should not be considered if they are no longer associated with a patient under monitoring. Many devices that comprise the landscape of Internet of Things do not have an unique identification and validation of data coming from these devices must be made by applying appropriate rules specific to each context.

VI. CONCLUSIONS

The experiment reported on using off-the-shelf IoT devices for an application of health telemonitoring at home showed that although feasible, the emergent market still does not offer flexible products that may be easily adapted for use in contexts other than the offered by the manufacturer and that allows only access to pre-configured servers in some cases. This points out the fact that IoT interoperability issues are still incipient despite not being considered a problem to develop a data transfer system connecting health care providers with patients [13] and the use of closed solutions may become a limitation when it comes to integrating IoT devices in a broader context. Some middleware proposals use Service-Oriented Architectures (SOA) mechanisms as basis for a middleware architecture in embedded networks [14] but in any case there is a need for standards to improve interoperability of devices especially in the case of healthcare devices. Needed standards should encompass open APIs, choice of interconnection interfaces, and configuration options of the operating mode of the monitoring/control device, including the aggregation of additional security mechanisms.

The embedded middleware proposed in this work offers a solution for interoperability enhancement and security management in a context using a special type of Internet of Thing devices for health monitoring with Wi-Fi interfaces. These kinds of devices cannot be connected directly to Internet for interoperability and security reasons, as it was discussed in this study. With the middleware it will be possible to provide an enhanced AAA (Authentication, Authorization and Accounting) service that it is especially important in this context as upheld in [4]. In this case, it was considered easier to create the middleware to manage and intermediate the communication between the "things" than to search for a solution that would modify the "things" due to the availability of the access point supporting the middleware software. Since the software used is free, this solution may be released as a product at no extra cost.

ACKNOWLEDGEMENTS

We thank RNP (Rede Nacional de Ensino e Pesquisa) and CTIC (Centro de Pesquisa e Desenvolvimento de Tecnologias Digitais para Informação e Comunicação) for financial support on the REMOA project.

REFERENCES

[1] G. Paré, K. Moqadem, G. Pineau, C. St-Hilaire, "Clinical effects of home telemonitoring in the context of diabetes, asthma, heart failure and hypertension: a systematic review," J Med Internet Research, June 2010. Available at: http://www.jmir.org/2010/2/e21/ doi: 10.2196/jmir.1357

[2] L. Bos, A, Marsh, D. Carroll, S. Gupta, M. Rees, "Patient 2.0 empowerment," International Conference on Semantic Web and Web Services, pp. 164–167, 2008.

[3] S. Babar, P. Mahalle, A. Stango, N. Prasad, R. Prasad, "Proposed security model and threat taxonomy for the Internet of Things (IoT)," Communications in Computer and Information Science, 89 CCIS, pp. 420-429, 2010.

[4] R.H. Weber, "Internet of Things - New security and privacy challenges," Computer Law and Security Report, 26 (1), pp. 23-30, 2010.

[5] M. Sedlmayr, H. Prokosch, U. Münch, "Towards smart environments using smart objects," Paper presented at. Studies in Health Technology and Informatics, vol. 169, pp. 315-319, 2011.

[6] Internet2, "Shibboleth." Available at: http://shibboleth.internet2.edu/

[7] S. Katzenbeisse, M. Petković, "Privacy-preserving recommendation systems for consumer healthcare services," Paper presented at the ARES 08 - 3rd International Conference on Availability, Security and Reliability, Proceedings, pp. 889-895, 2008.

[8] G. Delac, M. Silic, J. Krolo, "Emerging security threats for mobile platforms," MIPRO, 2011 Proceedings of the 34th International Convention, pp.1468-1473, May 2011.

[9] T. Riedel, N. Fantana, A. Genaid, D. Yordanov. H.R. Schmidtke, M. Beigl, "Using web service gateways and code generation for sustainable IoT system development," Internet of Things (IOT), pp.1-8, November-December 2010.

[10] N. Lim; S. Majumdar, B. Nandy, "Providing interoperability for resource access using web services," Communication Networks and Services Research Conference (CNSR), 2010 Eighth Annual, pp. 236-243, 2010.

[11] S.S., Chavan, R. Madanagopal, "Generic SNMP proxy agent framework for management of heterogeneous network elements," In Proceedings of the First international conference on Communication Systems and Networks (COMSNETS'09), pp. 1-6, 2009.

[12] Anh L. Bui, Gregg C. Fonarow, "Home Monitoring for Heart Failure Management", Journal of the American College of Cardiology, vol. 59, Issue 2, 10 January 2012.

[13] Ladyzynski, P., Wojcicki, J.M., Foltynski, P. "Effectiveness of the telecare systems". In IFMBE Proceedings, 37, pp. 937-940, 2011.

[14] Jesús Salceda, Iván Díaz, Juan Touriño, Ramón Doallo, "A middleware architecture for distributed systems management", Journal of Parallel and Distributed Computing, vol. 64, Issue 6, pp. 759-766, June 2004.

ANEXO B — ARTIGO PUBLICADO NO ACM SAC'15

- Título: Refactoring Internet of Things middleware through Software-Defined Network
- Conferência: 30th ACM/SIGAPP Symposium On Applied Computing
- URL: http://www.acm.org/conferences/sac/sac2015
- Data: 13-17 de abril de 2015
- Local: Salamanca, Espanha

Refactoring Internet of Things middleware through Software-Defined Network

Lucas M. R. Arbiza, Leandro M. Bertholdo, Carlos Raniery P. dos Santos,
Lisandro Z. Granville, Liane M. R. Tarouco
Informatics Institute
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
{lmrarbiza, berthold, crpsantos, granville, liane}@inf.ufrgs.br

ABSTRACT

Refactoring means to rewrite pieces of code aiming to improve it internally but keeping the expected software behavior. In this paper we present the refactoring of an Internet of Things middleware based on Software-Defined Network. In a previous work we proposed a middleware to address issues we found in healthcare devices used to monitor patients with chronic illnesses in their homes. Software-Defined Network allowed the redesign of the middleware architecture to improve *things* management, its interconnection with services, and the deployment process of new monitoring scenarios. Refactoring process also extended the middleware to support multiple services in a single home network sharing the same network infrastructure. This work details an OpenFlow controller and an application developed to achieve our goals; we also present sample scenarios where our approach can be applied showing different services delivered in the same home network environment, and using data from all connected devices to build a digital representation of the physical realm.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Internet of Things middleware; C.2.3 [**Network Operations**]: Network management; C.2.4 [**Distributed Systems**]: Network operating system—*Software-Defined Network*

General Terms

Design, Experimentation, Management

Keywords

Software-Defined Network, Internet of Things, Middleware, Refactoring

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695861

1. INTRODUCTION

Modern communication networks are demanding the use of new and sophisticated management solutions. While such networks used to be composed of traditional devices (*e.g.,* routers, servers, PCs), today, the number and variety of connected elements has increased. Such elements, usually referenced to as *things*, can vary in many aspects, such as size, computing resources, operating mode, network technologies and protocols. This scenario is known as Internet of Things (IoT), a term originally defined in 1999 to describe the RFID tags used to track objects in supply chains. Nowadays, the term means the digital representation of physical environment, created using data sent by multiple sensing devices [14].

The importance of research in IoT is evidenced by the growth in the number of heterogeneous devices connected to the Internet — the industry predicts about 15 billions by 2015 and 50 billions by 2020 [7]. These devices are used for many and diverse applications, for example healthcare, home automation, fitness tracking, environment monitoring, and in smart cities. Integrating heterogeneous devices into existing network infrastructure is a complex task. In a previous effort to develop IoT services for smart cities [5], we have identified a set of important aspects to be considered when employing IoT devices as part of a service. These aspects are presented in the following:

- Interconnection: most of the Internet-enabled devices (*i.e.*, *things*) available today employ proprietary communication methods to connect with the manufacturer servers or with other devices from the same manufacturer. Such operation mode is know as *silo* [1,11,13,17]. The main problem with *silos* is that data are closed in a proprietary solution; even when accessible through APIs (Application Programming Interface), it is first sent to proprietary servers;
- Security and privacy: data collected by the devices are usually transmitted in plain text and there is no authentication between device and destination server;
- Management: IoT devices usually employ sleep schedule mechanisms to reduce the power consumption, avoiding traditional network management solutions (*e.g.,* SNMP, NETCONF, ICMP-based connectivity monitoring) to be employed;
- Data structure: according to [12], 80% of healthcare data is unstructured, thus requiring data-mining in

large volumes of data in order to find relevant information to foresee future outcomes in patients health.

In order to tackle such issues, and considering the working scenario of REMOA project (health monitoring of patients with chronic illnesses), we developed a middleware to execute on Access Points (APs) [16]. According to [9], middleware is one or more software layers employed to abstract technological details to the application level. The middleware development allowed us to identify that resource constraints of target environment (APs) slow down development of required modules, consequently making the monitoring solution hard to scale, deploy, and improve.

In this work we refactor the REMOA middleware in order to support Software-Defined Network (SDN) technologies as underlying network substrate. Specifically, we refactored two important features provided by the middleware: *things* management and interconnection. Inspired by [10], the middleware functions implemented to run in APs were moved to remote servers, except the switching and routing tasks, now performed as an OpenFlow switch. In the refactoring process we also extended middleware to support more than one service in the same AP.

In SDN-based networks, forwarding tables of switches are configured by controllers, which are entities responsible for storing and distributing flow rules and actions. Once an unmatched packet arrives at a switch, it contacts the network controller, which can decide to modify the current flow-table rules, or to deploy new rules. This approach presents advantages for IoT environments since the APs can be updated by the SDN controller once new monitoring devices are added to the network and start sending packets.

In our refactored middleware, services run on remote servers and APs forward network packets based on flow rules received from the SDN controller. Since the servers do not present the same hardware limitations as the APs have, more sophisticated services (*e.g.*, management tasks, data receiving and processing) can be created and deployed to the IoT environment in a fast and dynamic fashion; particularly because developers can make use of development resources not possible to be used when all the middleware was being developed targeting APs. Another benefit enabled by employing SDN in IoT context is the view of the networked environment as a whole; services may use information about communication of every connected device, even *things* belonging to other services, to expand the digital representation of the environment beyond that provided by data from their devices.

This paper is structured as follows: In section 2 we provide an overview of the SDN paradigm and present other efforts where the SDN paradigm is investigated in the IoT scenario. Section 3 depicts the REMOA middleware, its components, and development decisions. Middleware refactoring is detailed in section 4 where we present the architecture of our approach and its components: Derailleur OpenFlow controller and ThingsFlow application (sections 4.1 and 4.2). In section 5 we provide some examples of services and scenarios where this approach can be deployed. Finally we present our considerations and point out next steps in our work.

2. RELATED WORK

SDN is a network paradigm that has been largely explored in the last years, both in industry and academia. The main feature of SDN is decoupling the network control plane from the forwarding devices, allowing to define the network logic through applications that configure forwarding rules on SDN switches [6].

In IoT field, SDN started to be explored more recently. Qin *et al.* [15] propose an architecture where SDN is used to enable different services to cooperate. The authors exemplify their proposal in a road scenario where, for example: vehicles receive notifications about traffic and accidents; vehicles communicate with each other; travel planning of electric cars is optimized based on the location of recharging stations; cabs can be located. In that work, vehicles are recognized based in data themselves provide. Such data are crossed with data stored in databases; once a vehicle is recognized, its data can be sent to other services, such as a network of cameras, that will visually locate a vehicle (*e.g.*, cab) based on geo-location data.

The work of Chetty and Feamster [10] and Yiakoumis *et al.* [18] does not address IoT explicitly, but their efforts tackle home networks issues and use SDN to develop their refactoring proposals. Chetty and Feamster [10] succeeded in moving network complexity away from the home end-users. End-users often do not properly configure their private networks because of the specificity of technical information presented to them, such as protocols, packet types, attack types, and Maximum Transmission Unit (MTU). The authors propose that network configuration possibilities are presented in a high level abstraction configuration interface; SDN is used to translate from configuration interface to technical OpenFlow settings.

Yiakoumis *et al.* [18] propose to slice home network allowing different service providers (*e.g.,* broadband, electric energy, gas, NetFlix, GoogleTV) to use different slices to deliver their services sharing the same network infrastructure, aiming at costs reduction. Each service cannot interfere in other services so that slices isolation is an essential feature. Each slice is controlled by a service provider. For each slice, it is guaranteed isolation of traffic and broadband, independent control and the possibility to be changed and/or customized by the service provider.

The target scenario of this paper is also home networks, but with the characteristics of the monitoring environment of the REMOA project. Laying over the ideas of the previously cited work, we extended the REMOA middleware through a refactoring process, but we kept focus on IoT issues. In the following sections we describe how refactoring has been carried out, as well the redesigned architecture and the implemented components.

3. REMOA MIDDLEWARE

The REMOA middleware was developed to address common issues found in IoT environments, mainly focused on our health monitoring scenario: interconnection, security and privacy, management, and data structure [16]. The middleware was developed considering the TP-Link WR1043ND AP, with architecture presented in Figure 1, and the main components discussed in the following:

1. Transparent Proxy: this component is responsible for analyzing all the outgoing traffic to the Internet. It receives health data from *things*, sends to the healthcare application, and records in the MIB (Management

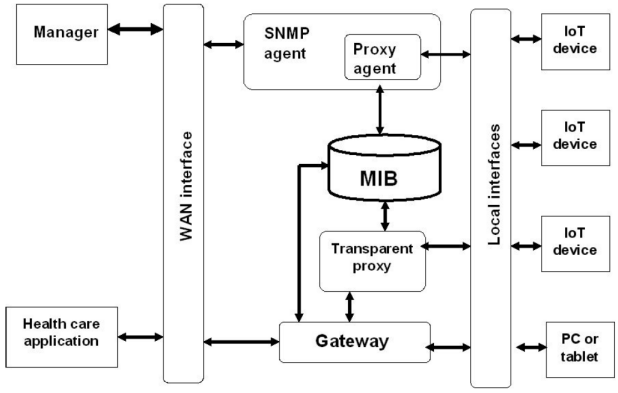


Figure 1: REMOA Middleware Architecture [16].

Information Base) information about *things* operation to be used for management purposes; when necessary it applies filtering or redirecting rules to the network traffic;

2. SNMP agent and Proxy agent: responsible for handling SNMP requests made to *things* with no SNMP support, it replies to requests with data previously stored in the MIB by Transparent Proxy module;

3. Gateway: enables transparent communication between the healthcare application and *things*, but only when *things* operating mode allows them to be accessed. When necessary, Gateway translates messages from one technology to another (*e.g.*, Ethernet to 802.15.4-based technology).

The middleware was designed to execute over the OpenWRT [4], a Linux-based operating system for embedded devices, which is widely employed in domestic APs. In order to reduce resources consumption of APs (*e.g.*, memory and processing), the middleware was developed using the C++ instead of other resources demanding programming languages, such as Java and Python.

With C++ there is a gain in performance and resources saving running native code, but developing in C++ use to be slower than developing with languages featured to abstract type conversions, pointers, memory allocation/deallocation, etc. C++11/C++14 features make development easier, but still demands lots of work, for example, to implement a Transparent Proxy module to communicate with *things* in a REST (Representational State Transfer) style.

The deployment of new middleware features, or to employ new monitoring devices in REMOA approach is a process that requires much working efforts, *i.e.*, the development of specific modules having as target a resource constrained system and architecture of the AP followed by the update process. If AP is already running in the house of a patient it is updated remotely; if some error occur the monitoring is compromised, in addition technical staff will need to move to the patient house for maintenance.

4. MIDDLEWARE REFACTORING

Figure 2 illustrates the middleware refactoring. The complexity of network services provided by modules earlier implemented over the AP, now is distributed among servers provided with more computing power. AP now acts as an OpenFlow switch and establishes IPSec connection with the servers to where collected data are sent. The three modules of the REMOA middleware were refactored as the following:

- Transparent Proxy module tasks are performed by AP and service servers. AP forwards packets based on flow rules sent by ThingsFlow application through the controller; data collected by *things* are sent through an IPSec tunnel to the services where are parsed and processed.

- *Things* management that was mostly based on SNMP in REMOA middleware, now is based on OpenFlow counters. Counters are retrieved from APs by ThingsFlow that makes it available with a timestamp indicating when it was retrieved; services retrieve counters stored in ThingsFlow to implement management mechanisms to its *things*.

- Gateway now is the packet forwarding feature realized by AP. In this approach forwarding is defined by OpenFlow rules. In current IoT scenario there is a trend to use IP-based approaches in *things*, such as 6LowPAN and CoAP, for example. If *things* communication is not based on IP, usually there is a device such as a gateway/hub that connects *things* in an IP network, so that packets are forwarded as any IP packet.

After the refactoring, the development needed for each service (communication with *things*, data collecting, processing, etc.) is not more impacted by AP constraints; developers can use any programming language, library, framework or programming techniques to develop features for services, specially those complex or even impossible to be developed to run on AP.

In this work we extended middleware capabilities enabling it to provide multiple services in a single AP. Yiakoumis et al. [18] also explore the sharing of network infrastructure by

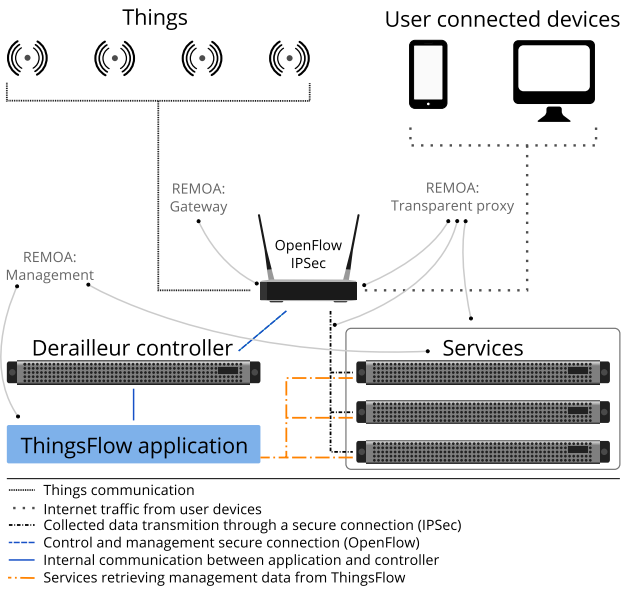


Figure 2: Middleware architecture refactoring.

different service providers, but in their work the network is sliced and each slice is isolated and is controlled by a service provider. Slicing does not apply to our scenario because we focus on IoT; in our approach services share data from their *things* enlarging the digital panorama of the physical environment. Due to privacy questions some data must not be shared, such as personal and health data of a patient; although data about *things* communication can be shared to be used by any service, even from those which collected health data.

Modules updates in APs in use in patients houses are no longer needed; AP, as an OpenFlow switch, receives forwarding rules on demand from the controller. IPSec settings are updated in each AP configuring them to create tunnels with servers running the services they are providing; it is done by a script that retrieves proper settings from the same server where controller and application are running.

The benefits provided by this approach are: the support to multiple services; the development of services and its features in a not constrained environment; and automated update of forwarding settings based on OpenFlow. In addition, SDN-based refatoring allows to recognize connected devices providing vision of network scenario and to implement management features suitable for IoT.

4.1 Derailleur Controller

The OpenFlow controller we developed in this work, called Derailleur, has been built on top of `libfluid` [2], the winner of Open Networking Foundation OpenFlow Driver Competition [8]. `libfluid` is divided in two libraries: `libfluid_base`, that implements a server to listen to switches connections and to handle switches events; and `libfluid_msg`, which creates, parses and sends OpenFlow messages. Derailleur controller was implemented extending `libfluid_base OFServer` class through inheritance.

Derailleur was designed to be lean and to abstract APs as switches objects, as shown in Figure 3. Derailleur retrieves information from each connected AP sending `OFPT_FEATURES_REQUEST` and `OFP_MULTIPART_REQUEST` (`OFPMP_DESC` to request description) messages; APs replies are parsed and stored in the corresponding switch object stored in a virtual stack (array), and managed by controller. Each AP may be recognized through the information retrieved; in our scenario we used MAC (Media Access Control) as an identifier; we can not use IP addresses because APs in home networks will receive IP address dynamically, additionally they may be behind NAT (Network Address Translation).

Derailleur runs an application (a child object of `Application` class) over their; Derailleur forwards events from APs to the application that will handle then according to the service, as described in Section 4.2. `Application` class provides four abstract methods: `on_switch_up`, `on_switch_down`, `on_packet_in`, and `message_handler`; controller triggers the suitable method depending on the AP event. When an event occurs, the controller sends to the application data related to the event triggered by switch; the application handles the event and communicate with the AP through a switch object stored in the switches stack. Switches stack is managed by `Controller` class and shared with the `Application` class. Through the switches objects, the application can contact APs to retrieve data (*e.g.*, flows, counters, information about connected devices) and to manage flow-tables.

To deploy OpenFlow in APs we used the virtual switch Open vSwitch (OVS) [3] on OpenWRT. We built an OpenWRT image with OVS, the resulting image also contains scripts to configure OVS bridges and network interfaces when AP boots up.

4.2 ThingsFlow Application

ThingsFlow extends Application class from Derailleur controller. ThingsFlow implements the inherited abstract methods `on_switch_up`, `on_switch_down`, `on_packet_in`, and `message_handler`; these methods are triggered by controller when AP events occur. ThingsFlow also provides methods to manage flow-tables in APs, independent of APs events.

`on_switch_up` runs always when an AP connect to the controller; ThingsFlow identifies the AP through the switch object stacked by the controller and queries database for flow-tables of services being provided on that home network and installs them on AP. The method `on_switch_down` is triggered when any AP disconnects from the controller; Derailleur removes the corresponding switch object from the stack; through this method ThingsFlow knows which AP disconnected and will not try to access it.

Flow-tables sent by ThingsFlow do not contain default rule to process packets from unknown devices (*e.g.*, devices not specified in any service). When a packet does not match any rule on the flow-table of a given AP, it causes a `table-miss` event making controller to trigger `on_packet_in` method; ThingsFlow installs flows to forward packets from that specific unknown device to the Internet or inside the network. Flows defined specifically for unknown devices enables the AP to implement flow counters for each connected device; the counters may be used by any service, as shown in Section 5, and provide a view of the network scenario. ThingsFlow also makes available to the services flows counters retrieved from APs; counters show if and when devices communicate and allow services to monitor their *things*.

Table 1 provides a basic example of flow-tables arrangement and packet processing in a scenario where a health monitoring service (REMOA) is provided. Beyond REMOA flow-tables, there is one table for packets pre-processing and

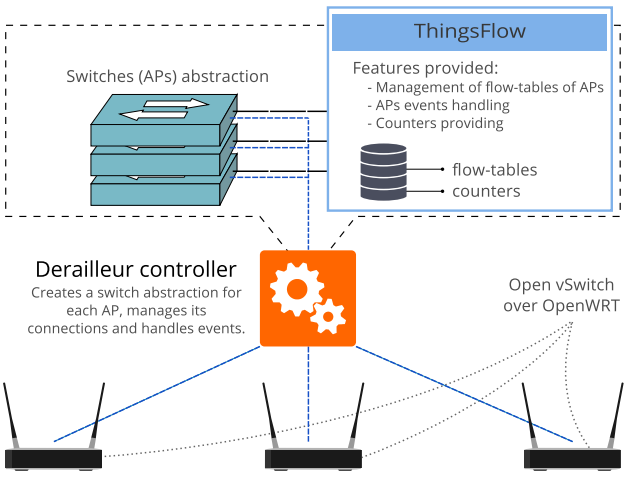


Figure 3: Overview of middleware components.

Main (0)	Input (1)	Output (2)	Firewall (3)	REMOA (4)
* Packets addressed to AP: go to table 1; * Packets sent by AP: go to table 2; * Packets from healthcare devices: go to table 4; * Packets to healthcare devices: go to table 4; * Packets addressed to healthcare manufacturer servers: go to table 4; * Any other packets: go to table 3.	* Check if the incoming packet is allowed or not in this device.	* Check if the outgoing packet is allowed to be sent by this AP.	* Check if packet is allowed according to the network and services policies; * Packets that are not explicitly refused causes a *table-miss* event; a *packet-in* message is sent to the controller.	* HTTP packets from healthcare devices are redirected to REMOA service; * Packets from REMOA service to healthcare devices are allowed; * HTTP(s) packets to healthcare devices' manufacturer sent by other devices are normally forwarded; * ICMP and SNMP packets to devices that support this protocols are allowed; * Any other packet is dropped.

Table 1: Example of packet processing among flow-tables.

three other providing filtering rules to protect AP and home network.

5. EXAMPLE SCENARIO

In this section we present some examples of services that may be provided in the same home network through the same AP. In the example scenario APs are supplied by local government to be used in social and/or well being projects provided, for example, by hospitals or universities. APs and ThingsFlow are managed by government staff that work together with services managers when services are deployed, configured or updated; for example, flows from different services may match the same packet, so managers need to handle this kind of issues, otherwise OpenFlow switches will not install overlapping flows.

This section describes three example services being delivered in the same home network. The following depicts how services could be deployed based on our approach:

Health monitoring (REMOA):

Provided by a hospital, this service monitors patients with chronic illnesses (*e.g.*, diabetes, hypertension) that do not need to be hospitalized. Patients are monitored through healthcare devices connected to the Internet through an AP using the existing Internet link in patient house. Healthcare devices and the AP are provided by the hospital.

Healthcare devices we used are hard-coded configured to send reading data to its manufacturer servers, so traffic from these devices are redirected to REMOA by OpenFlow that rewrites destination IP in packet header.

Data sent by healthcare devices provide a vision of patient health. ThingsFlow knows every device connected to the AP, it makes available flow counter of every device enabling healthcare application to draw a panorama of the entire networked environment and to infer about what is happening on patient's surroundings. This view of the environment as a whole is useful, for example, if a sensor does not detect patient presence for a large period of time, what could trigger an alarm, but their smartphone and notebook are connected and TV just started to transmit a streaming; probably patient is watching TV making soft movements that are not detected by sensor, although they are not unconscious.

Most of healthcare devices only connect when they are used for measurements and this must be done in pre-defined periods of time. Based on OpenFlow counters healthcare application may implement management mechanisms to suit treatment particularities, such as measurement schedule.

Access to smart city services

City government offers websites and mobile applications to be used by people; information available refers, for example, to transit, bus itinerary and current location, touristic places, events schedule. APs providing services in homes closer to the street, in special near bars, cafes or squares, broadcasts a SSID (Service Set IDentifier) to be used by anyone to access the city services; access attempts to any other content are dropped. The traffic of this service is entirely isolated from home network.

Research to characterize Internet usage

It is a research developed by an university to characterize Internet usage considering aspects such as people age, gender, education. OpenFlow counters allow to distinguish traffic from each connected device; it is made through specific flows that forward packets from devices of people participating of study to analyzed destinations, such as social networks, news, mail, web access, chat. Researches also consider time spent in each destination and period of the day, among others. Crossing people information with data from counters researches can define different usage profiles.

In the examples above we showed how different services could be implemented sharing the same network infrastructure, configured through a central application, the ThingsFlow. We emphasize that connected devices are not closed

in a single service; data from OpenFlow counters related to communication of networked devices allow that even *things* closed in manufacturer silos be used by different services for different purposes. Note that devices used by the third service example are the same that provide the panorama of surroundings of patients monitored; connections using the second service allows to estimate average waiting time in a bus stop or people concentration.

6. FINAL CONSIDERATIONS AND FUTURE WORK

In this paper we presented the refactoring process based on SDN of the REMOA middleware; we redesigned middleware architecture aiming to speed up development and deployment process of services, features and *things*. Our work focus on IoT services delivering in home networks.

Through the refactoring, network and service complexity that were concentrated on APs were distributed among different servers. APs were turned in OpenFlow switches in charge of packet forwarding according flows rules received from controller; they also provide a encrypted connection through which sensed data are sent.

The middleware were extended to provide multiple services through the same AP, in the same home network. As this work focus on IoT scenarios, *things* belonging to a service may provide useful data to other services; while users consume a service their devices connections may transparently supply data to the service itself, or any other service.

In our approach, all services run in remote servers provided with more computing resources than APs; there is no need to develop modules to run on APs. Developers can make use of any programming language, libraries, frameworks, and many other development resources that potentially make services deployment, improvements, and maintenance faster. Flow-tables of each AP are updated by controller when *things* and other connected devices communicate; no manual intervention is required.

Management of *things* must be designed considering *things* operating mode and their usage; OpenFlow counters allow to know when and how *things* are working. ThingsFlow periodically retrieves counters from APs and provide them to the services that implement management features to suit particularities of *things* they use.

Even running different services in the same AP we did not evaluated the impact of flow-tables processing in AP; it is one of the next steps in our work, as well a circumvent to deal with possible controller faults. Derailleur and ThingsFlow are both under development to finish some features; we intent to continue developing then to investigate other ways SDN can be explored in IoT scenarios and services delivering.

7. REFERENCES

[1] IOT-A: Internet of Things Architecture. `http://www.iot-a.eu/public`.

[2] libfluid - The ONF OpenFlow driver. `http://opennetworkingfoundation.github.io/libfluid/index.html`.

[3] Open vSwitch. `http://openvswitch.org/`.

[4] OpenWRT. `http://openwrt.org`.

[5] REMOA - Rede Cidadã de Monitoramento de Ambiente Baseado no Conceito de Internet das Coisas. `http://remoa.tche.br`.

[6] Software-Defined Networking: The New Norm for Networks. `https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf`, 2012.

[7] Silicon Labs 2013 Annual Report. `http://files.shareholder.com/downloads/ABEA-39NRLI/3514356491x0x730508/7FEFCB63-3F64-4D05-86BE-A9724E4AA891/SLAB\_32633\_proof\_rev2.pdf`, 2013.

[8] CPqD Selected as Winner and Recipient of $50.000 Grand Prize. `https://www.opennetworking.org/component/content/article/26-news-and-events/press-releases/1431-open-networking-foundation-announces-openflow-driver-contest-winner`, 2014.

[9] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[10] M. Chetty and N. Feamster. Refactoring network infrastructure to improve manageability: a case study of home networking. *ACM SIGCOMM Computer Communication Review*, 42(3):54–61, 2012.

[11] L. Coetzee and J. Eksteen. The Internet of Things - promise for the future? An introduction. In *IST-Africa Conference Proceedings, 2011*, pages 1–9, 2011.

[12] P. Hinssen. The age of data-driven medicine. `http://datascienceseries.com/assets/blog/The_Age_of_Data-Driven_Medicine.pdf`, 2012.

[13] Linux Foundation. Technology Leaders Establish the AllSeen Alliance to Advance the 'Internet of Everything'. `https://allseenalliance.org/announcement/technology-leaders-establish-allseen-alliance-advance-internet-everything`, 2013.

[14] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[15] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian. A Software Defined Networking architecture for the Internet-of-Things. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9, 2014.

[16] L. M. R. Tarouco, L. M. Bertholdo, L. Z. Granville, L. M. R. Arbiza, F. Carbone, M. Marotta, and J. J. C. de Santanna. Internet of Things in Healthcare : Interoperatibility and Security Issues. In *IEEE International Conference on Communications, International Workshop on Mobile Consumer Health Care Networks, Systems and Services*, pages 6121–6125, Ottawa, 2012.

[17] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. D. Johnson. M2M: From mobile to embedded internet. *Communications Magazine, IEEE*, 49(4):36–43, 2011.

[18] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown. Slicing Home Networks. In *Proceedings of the 2Nd ACM SIGCOMM Workshop on Home Networks*, HomeNets '11, pages 1–6, New York, NY, USA, 2011. ACM.

ANEXO C — ARTIGO PUBLICADO NO IDC'15 FI&SN

- Título: SDN-Based Service Delivery in Smart Environments
- Conferência: 9th International Symposium on Intelligent Distributed Computing (IDC'15), International Workshop on Future Internet and Smart Networks
- URL: http://marco.uminho.pt/conferences/fi-sn-2015/
- Data: 7-9 de outubro de 2015
- Local: Guimarães, Portugal

SDN-Based Service Delivery in Smart Environments

Lucas Mendes Ribeiro Arbiza, Liane Margarida Rockenbach Tarouco, Leandro Márcio Bertholdo, and Lisandro Zambenedetti Granville

Abstract Internet of Things scenarios demand adaptability to hold the heterogeneity of systems and devices employed; gateway-based solutions are a common answer to the issues of smart environments. The development of software for gateways, using a middleware to handle the different devices demands, is possible in our working scenario, but the maintenance cost of such a solution is high because of software development constraints imposed by hardware and system limitations of the gateway. This paper depicts an SDN approach for smart environments resulted from a refactoring of a previous middleware proposal. Through an instantiation of the refactored middleware in a home network to deliver a health monitoring service the benefits are demonstrated; the benefits regard the digital representation of the physical realm, deployment and maintenance of services, and management of devices and networks.

1 Introduction

The rapid growth of population impose to cities a major challenge to their sustainability, as well as a threat to the infrastructure of main services provided to the population. A solution is expected to show up with the advent of smart cities. As discussed in [1], more than 150 cities can be documented around the world as smart.

Smart cities require IT services to capture, integrate, analyze, plan, inform, and act intelligently on city activities, resulting in a better place to live. This implies services to make life easier for people and businesses. A smart community is a community that carries out conscious efforts to use information technology to trans-

Lucas Arbiza · Liane Tarouco · Leandro Bertholdo · Lisandro Granville
Informatics Institute, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil.
e-mail: {lmrarbiza,granville}@inf.ufrgs.br,{berthold,liane}@penta.ufrgs.br

form significantly and fundamentally the way of life and work within its territory, instead of following an incremental way.

From this perspective, a smart city refers to a physical environment in which communication and information technology, and sensor systems that are part of it, disappear as they become embedded into physical objects and the environments in which people live, travel, and work [2]. Smart cities should use smart computing technologies to build critical infrastructure components and services of a city more intelligent, interconnected, and efficient [3].

Homes have becoming increasingly smarter embedding many devices able to sense their surroundings; usually those devices are part of a solution for home automation, security, healthcare, among other purposes. Smart solutions employed in citizens homes may be part of broad system, such as a smart city solution for intelligent use and distribution of energy. Smartness in home environments bring some issues for network management, security and orchestration due to the demands to handle the heterogeneity of smart devices when they are employed in services delivering.

Software-Defined Networking (SDN) is a paradigm where network intelligence and control are taken from forwarding devices and are deployed in central controllers where network logic behavior is defined by software, developed or customized to fit the needs of each network environment. SDN provides to the network the flexibility of software, allowing the development of features not available in network hardware; using SDN enables the development or the use of applications for network orchestration and management, suitable to deal with heterogeneity of smart environments.

This paper presents the use of a SDN-based middleware [4] in a health monitoring environment aiming to achieve a simpler and easier to manage solution to monitor patients in their own homes, when compared to a previous middleware where SDN was not used. In the same scenario, we also exploit the use of SDN resources for network management for smart environments. The presented approach also enables the delivering of multiple services for smart environments, sharing the same network infrastructure.

The rest of this paper is organized as follow. Section 2 presents some concepts of smart environment, focusing specially in healthcare. Section 3 details what is SDN paradigm, how it works, and its resources. In section 4 we discuss how SDN can be used to empower smart environments, we present the SDN-based middleware used detailing its architecture, implementation and workflow. Section 5 demonstrates the use of the SDN-based middleware for health monitoring and benefits achieved by using the SDN approach. In section 6 we present our final considerations.

2 Smart Environments

Smart cities are composed of smart devices. Currently, a large number of smart objects and different types of devices are interconnected and communicate via the

Internet Protocol, which creates a worldwide ubiquitous and pervasive network referred to as the Internet of Things (IoT) [5, 6]. With the inception of IoT, the Internet is further extended to connect things, such as power meters, heartbeat monitors, temperature meters, and many powerful operations, such as health care units, green energy services, and smart farming utilities, that can be made available to people for enhanced quality of life.

IoT is becoming the Internet of Everything (IoE) [7]. Most of smart devices, used in the context of Internet of Things (IoT), do not employ generic/open standards when communicating. One of the challenges in this context derives from the need to find a form of automated communication and to integrate the various devices and protocols, making it possible to obtain information about the scenario being monitored.

Data collected by devices on a given environment are combined to provide services (*e.g.*, smart homes, and healthcare) [8]). The combination can also be made through mashups [9], where data from different sources and using various resources are handled to make possible creating a vision of a whole.

3 Software-Defined Networks

SDN is mainly known for decoupling network control plane from the forwarding devices, usually combined in the same device, such as routers. Decoupling enables the network logic to be defined at the software level and to implement features that may not be available in the network hardware in use [10]. In general, the network logic behavior is defined configuring every network device individually, using vendor specific syntax and limited to the features available according to the licenses acquired. SDN allows to centralize network intelligence in a controller that sends forwarding rules, called *flow entries*, to the switches and routers defining logic behavior of the network [11].

Figure 1 illustrates the three layers SDN architecture: 1) Infrastructure layer: comprised of forwarding devices (switches and routers) enabled with an SDN standard, such as OpenFlow; 2) Control layer: one or more devices running a controller software enabling the communication between application and infrastructure layers; and 3) Application layer: one or more applications by which network intelligence is implemented; this layer makes use of the control layer to configure forwarding devices. Control layer communicates with infrastructure layer through OpenFlow messages or other SDN standard.

When a forwarding device, also called OpenFlow switch, receives a packet the switch looks up in its *flow tables* for a *flow entry* matching the received packet. If none of the existing *flow entries* match the packet, the switch sends a *packet-in* to the controller. *packet-in* is processed at the application layer and a *flow entry* is sent to the switch through the controller containing forwarding rules for the packet.

The controller has an entire view of the network; it is aware of every switch connected to it and of every device connected to the switches. That privileged view is a

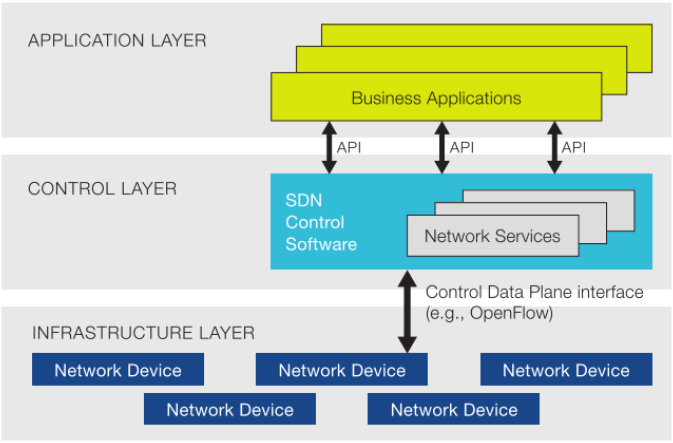


Fig. 1 Software-Defined Networking Architecture [10].

valuable resource exploited in some works that propose SDN-based approaches in different network related fields, for example: network virtualization: SDN is used to create isolated slices in the network over the same physical hardware [12, 13, 14]; routing: routes established by BGP and OSPF are reflected to low cost switches not enabled with routing protocols [15]; QoS: network paths are dynamically allocated assuring the best performance and availability for differentiated traffic [16]; mobility: aiming to prevent Wi-Fi handover from occurring when users are moving the traffic is sent to more than one access point simultaneously [17]; network management: SDN is used combined with SNMP and others protocols and security and authentication mechanisms existing in a campus, events arising from different sources and network policies are translated to *flow entries* to be installed on the forwarding devices [18].

Taking network control to the application level empowers a low cost network with capabilities not available in the existing hardware, as seen in [15]. By using SDN, network administrators do not dependent on features provided by vendors anymore, since new features can be developed using high level abstraction programming languages and management solutions may be combined with control applications to improve network intelligence, management, and automation.

4 Empowering Smart Environments with SDN

Recently, SDN started to be exploited in smart environments. In research efforts such as [19, 20], the authors made use of SDN in home networks, as we do in our work as well. In [19], the authors provide a simplified interface by which non technical users are able to configure their home networks properly. Those users

usually expose themselves to security and privacy risks just because they do not know technical information required to configure most of access points. OpenFlow is used to translate network settings configured by users to *flow entries* to be installed in the SDN empowered access point. In [20], SDN is used to slice the network in isolated virtual networks allowing different service providers to share the same network infrastructure to deliver their services at users' homes. Each provider have control over its slice to deliver the service as needed.

Based on the works mentioned above, in [4] we used SDN to refactor an IoT middleware designed to enable health monitoring of patients with chronic illnesses in their own homes [21]. In an environment empowered with SDN, we are able to provide a wider view of a sensed environment. In [22], IoT is defined as a digital representation of the physical realm built from data collected by devices enabled to sense the environment they are in. Although off-the-self sensing devices usually employ a vertical communication called *silos* [23, 24, 25], in this case retrieved data are sent to proprietary servers and cannot be retrieved directly from devices. OpenFlow provides resources by which one can build a representation of an environment based on the communication of all connected devices in a home network. SDN also benefits IoT in network orchestration and management. The following subsections present our approach to achieve our goals in IoT environments through SDN.

4.1 Architecture and Workflow

The architecture depicted in figure 2 is split into three layers: the access points, the controller called Derailleur, and the application called ThingsFlow. APs act as OpenFlow switches forwarding packets according to the *flow entries* received. Derailleur controller listens to AP connections; when an AP establishes a connection, Derailleur uses OpenFlow messages to retrieve information from the AP to build an abstraction as a switch object. Each AP may be identified and accessed individually. Derailleur owns and manages switches objects that are created or destroyed reflecting APs status. The controller triggers events to be handled by the application when APs connect, disconnect, or send OpenFlow messages. ThingsFlow is the application where network intelligence is implemented. Each AP may provide a different set of services, so ThingsFlow installs in a given AP only the *flow tables* of services that AP provides. The ThingsFlow also collects counters from APs and make it available to the services, thus to be used for different purposes, for example, management and services features.

4.2 Implementation Details

APs run OpenWRT firmware, a Linux-based operating system for embedded devices, built containing the virtual switch Open vSwitch which provides OpenFlow

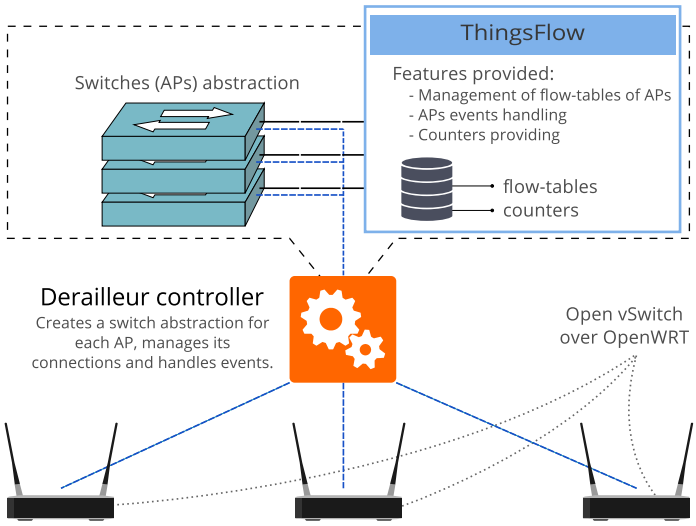


Fig. 2 Architecture overview and components roles [4].

capabilities to APs. APs can be replaced by any other hardware and operating system able to run Open vSwitch.

The Derailleur controller was developed in C++ using libfluid [26], the winner of Open Networking Foundation OpenFlow Driver Competition. libfluid is composed of two libraries: *libfluid_base*, that provides server features such as listening loop and events handling; and *libfluid_msg*, that provides mechanisms to build and parse OpenFlow messages.

ThingsFlow was developed inheriting an abstract application class provided by Derailleur. Through the abstract application class, the controller shares and provides access to the switches objects to ThingsFlow.

5 SND-based Healthcare

To demonstrate the benefits achieved employing our SDN-based approach in a smart environment, we use the same health monitoring example that we were working when we designed the previous middleware, in the REMOA[1] project. REMOA is a project that targets home solutions for care/telemonitoring of patients with chronic illnesses. The project also encompasses the design and implementation of a middleware to address issues found in monitoring devices used in the project, enabling

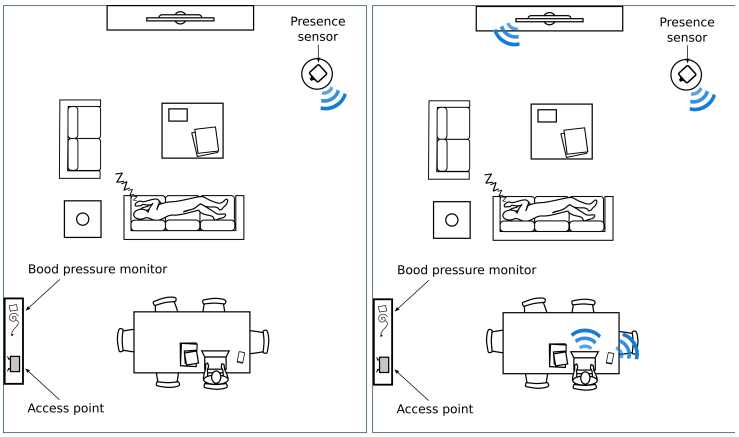
[1] *Rede Cidadã de Monitoramento do Ambiente Baseado no Conceito de Internet das Coisas* (Citizen Network for Environment Monitoring Based on the Concept of Internet of Things)

interoperability and security needed in the context of IoT for healthcare. The issues addressed are the following:

- Interoperability: used devices do not reply requests; they just send the data they read to vendor servers employing proprietary standards to communicate, what characterizes a *silo*. Settings, such as destination server or protocols to be used when transmitting data, cannot be changed.
- Security and privacy: data transmission is neither encrypted nor authenticated.
- Management: traditional management mechanisms, such as those based on ICMP or SNMP, cannot be used because of the sleeping scheduled mechanism employed by devices to save battery.
- Data structure: proprietary standards are also used to structure the data sent. It forces every data received to be parsed according to vendor standards. Usually vendors of devices designed for end-users do not provide the documentation required for parsing. The scheme used to parse the messages results from the analysis of the communication of the devices.

Figure 3 illustrates two different views of the same environment at the same moment. The hypertensive patient has fallen asleep while was watching TV; the patient is late with blood pressure measurement. Movements of patient are not captured by presence sensor because do no exceed thresholds. The left side illustrates the view provided by the REMOA middleware, where digital representation of the environment is based only on data provided by monitoring devices. The right side illustrates a view built combining data from monitoring devices and all other devices transmitting at that moment.

Monitoring based on health devices and presence sensor would trigger an emergence event if data from OpenFlow counters were not taken into account. The view



a) Environment sensed by monitoring devices b) Environment sensed using all connected devices

Fig. 3 Comparison between digital representations of the same environment with and without using OpenFlow counters.

illustrated in the right side of the figure is built using OpenFlow counters from *flow entries* installed on APs to forward traffic of devices used in the house of the monitored patient. These counters allow monitoring staff to know that the patient is not alone because a smartphone and a notebook belonging to one of the residents are in use at that moment. It is also possible to know that the TV is on. Monitoring staff can suppose that the patient is watching TV, making short movements and forgetting to do measurement. The action to be taken could be a phone call to remember the patient about the missing measurements.

In [21], the management of health monitoring devices was based in SNMP. Information about devices communication were stored in a Management Information Base (MIB) in the AP by a middleware module. In the SDN-based approach, this information is provided by OpenFlow counters, collected from APs by ThingsFlow, that makes counters available to the service provider that implements management mechanisms suitable for each monitoring device. Counters are combined with other data sources, such as the battery level transmitted by monitoring devices along collected data. Another important benefit achieved is the scalability regarding deployment of APs and monitoring devices. The configuration of the network behavior in the patient's residence is completely automated by OpenFlow; ThingsFlow provides the suitable *flow entries* for each AP. Through the orchestration of the network provided by SDN the network complexity was moved from APs to remote servers allowing developers to use of any development resources available in services features development instead of getting limited by hardware and system constraints of the APs. OpenFlow also rewrites packets headers to prevent health data of patients from being sent to proprietary serves; those packets are so forwarded to the monitoring server where patient data are processed.

6 Final Considerations

Designing a network solution for an smart environment is not an easy task because of the different demands of the smart devices. Solutions are usually based on local gateways that not only are in charge of related communication tasks but also must provide security, compatibility, and network management. To keep a gateway-based solution simple and less expensive, it is necessary to use an approach that enables the flexibility and functionality demanded, without high complexity and cost. In this sense, transferring more complex functions to a remote server and leaving the gateway to take care of only the task of switching and routing is the proposed solution presented in this paper.

Moving network control to the application level empowers a low cost network with capabilities usually not available in the gateway hardware. By using SDN, network administrators are neither dependent nor limited to the features provided by gateway vendors anymore. New features can be developed using high level abstraction programming languages and management solutions can be combined with control applications to improve network intelligence and automation.

The environment presented in this work provides health care service at home level to patients with chronic illness, but can also be used in other smart city environments as well. This work demonstrated how SDN can be used to build a wider view of an smart environment combining data from sensing devices with counters of network communication of all connected devices. SDN is also a good choice for smart environments because it enables the development of more appropriate management mechanisms and improves flexibility in network orchestration to fit the diversity of smart devices. As future work, we will deploy more complex scenarios with a wider range of services, aiming to exploit deeper SDN resources and its benefits for this kind of environment.

References

[1] L. Anthopoulos, P. Fitsilis, in *Advanced Communication Technology (ICACT), 2014 16th International Conference on* (2014), pp. 190–195. DOI 10.1109/ICACT.2014.6778947

[2] A. Steventon, S. Wright (eds.), *Intelligent Spaces: The Application of Pervasive ICT*, 1st edn. (Springer, London, 2006). DOI 10.1007/978-1-84628-429-8

[3] D. Washburn, U. Sindhu, S. Balaouras, R.A. Dines, N. Hayes, L.E. Nelson, Helping CIOs Understand "Smart City" Initiatives. Tech. rep., Forrester Research Inc. (2010)

[4] L.M.R. Arbiza, L.M. Bertholdo, C.R.d.P. Santos, L.G. Granville, L.M.R. Tarouco, in *30th ACM/SIGAPP Symposium On Applied Computing* (ACM, Salamanca, Spain, 2015), pp. 640–645. DOI 10.1145/2695664.2695861

[5] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Future Gener. Comput. Syst. **29**(7), 1645 (2013). DOI 10.1016/j.future.2013.01.010

[6] I. Mashal, O. Alsaryrah, T.Y. Chung, C.Z. Yang, W.H. Kuo, D.P. Agrawal, Ad Hoc Networks **28**(0), 68 (2015). DOI http://dx.doi.org/10.1016/j.adhoc.2014.12.006

[7] K.S. Yeo, M. Chian, T. Ng, D.A. Tuan, in *Integrated Circuits (ISIC), 2014 14th International Symposium on* (2014), pp. 568–571. DOI 10.1109/ISICIR.2014.7029523

[8] R. Blasco, A. Marco, R. Casas, D. Cirujano, R. Picking, Sensors **14**(1), 1629 (2014). DOI 10.3390/s140101629

[9] C.R.P.d. Santos, R.S. Bezerra, J.M. Ceron, L.Z. Granville, L.M.R. Tarouco, IEEE Communications Magazine **48**(12), 112 (2010)

[10] Software-Defined Networking: The New Norm for Networks (2012). URL `https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf`

[11] J.A. Wickboldt, W.P.d. Jesus, P.H. Iolani, C.B. Both, J. Rochol, L.Z. Granville, IEEE Communications Magazine **53**(1), 278 (2015)

[12] N. Mckeown, T. Anderson, L. Peterson, J. Rexford, S. Shenker, S. Louis, (2008)

[13] R. Sherwood, G. Gibb, K.K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. Parulkar, (2009). URL http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf

[14] R.B. Rafael Pereira Esteves, Lisandro Zambenedetti Granville, IEEE Communications Magazine **51**(7), 80 (2013)

[15] RouteFlow Project. URL https://sites.google.com/site/routeflow/home

[16] H.E. Egilmez, S.T. Dane, K.T. Bagci, A.M. Tekalp, in *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*. Koc Univ., Istanbul, Turkey (IEEE, 2012), pp. 1–8

[17] K.K. Yap, M. Kobayashi, R. Sherwood, T.Y. Huang, M. Chan, N. Handigol, N. McKeown, ACM SIGCOMM Computer Communication Review **40**(1), 125 (2010)

[18] H. Kim, N. Feamster, Communications Magazine, IEEE **51**(2), 114 (2013). DOI 10.1109/MCOM.2013.6461195

[19] M. Chetty, N. Feamster, SIGCOMM Comput. Commun. Rev. **42**(3), 54 (2012). DOI 10.1145/2317307.2317318

[20] Y. Yiakoumis, K.K. Yap, S. Katti, G. Parulkar, N. McKeown, in *Proceedings of the 2Nd ACM SIGCOMM Workshop on Home Networks* (ACM, New York, NY, USA, 2011), HomeNets '11, pp. 1–6. DOI 10.1145/2018567.2018569

[21] L.M.R. Tarouco, L.M. Bertholdo, L.Z. Granville, L.M.R. Arbiza, F. Carbone, M. Marotta, J.J.C. de Santanna, in *IEEE International Conference on Communications, International Workshop on Mobile Consumer Health Care Networks, Systems and Services* (Ottawa, 2012), pp. 6121–6125. DOI 10.1109/ICC.2012.6364830

[22] D. Miorandi, S. Sicari, F.D. Pellegrini, I. Chlamtac, Ad Hoc Networks **10**(7), 1497 (2012). DOI 10.1016/j.adhoc.2012.02.016

[23] IOT-A: Internet of Things Architecture. URL http://www.iot-a.eu/public

[24] G. Wu, S. Talwar, K. Johnsson, N. Himayat, K.D. Johnson, Communications Magazine, IEEE **49**(4), 36 (2011). DOI 10.1109/MCOM.2011.5741144

[25] L. Coetzee, J. Eksteen, in *IST-Africa Conference Proceedings, 2011* (2011), pp. 1–9

[26] libfluid - The ONF OpenFlow driver. http://opennetworkingfoundation.github.io/libfluid/index.html

APÊNDICE D — *FLOW ENTRIES* INSTALADAS NOS *ACCESS POINTS* UTILIZADOS NOS AMBIENTES MONITORADOS

Este apêndice apresenta e detalha as *flow entries* instaladas nos *access points*, tendo como base o cenário do monitoramento de pacientes que foi utilizado para avaliar a proposta do presente trabalho. Como apresentado na seção 4.3.1, o processamento dos pacotes dentro do *access point* é feito utilizando tabelas, as *flow entries* apresentadas a seguir estão separadas por tabelas e acompanhadas de uma explicação que detalha as suas finalidades.

Para facilitar a compreensão das *flow entries*, foram omitidos os campos *cookie*, *duration*, *n_packets* e *n_bytes*. Por serem campos de controle e estatística, não são necessários para o entendimento do que as *flow entries* fazem e ao que se aplicam. Os endereços IP e MAC foram substituídos por textos que identificam a qual dispositivo ou servidor a *flow entry* se aplica.

D.1 Tabela 0 — Pré-processamento

A tabela 0 realiza o pré-processamento, é a primeira tabela a processar qualquer pacote recebido pelo *switch* OpenFlow. Esta tabela encaminha os pacotes, dependendo de sua origem ou destino, à tabela responsável por processá-lo. Na tabela 0 as regras incluem os endereços MAC dos dispositivos de monitoramento para contabilização de pacotes e porque é uma forma de direcionar o tráfego apenas dos dispositivos pertencentes ao serviço REMOA para a tabela apropriada. O pré-processamento é feito da seguinte forma:

- Pacotes ARP são encaminhados normalmente, assim é possível utilizar regras L3 uma vez que a tabela de vizinhança pode ser preenchida;
- Pacotes endereçados ao *access point* são encaminhados à tabela 1 (Input);
- Pacotes enviados pelo *access point* são encaminhados à tabela 2 (Output);
- Pacotes relacionados aos dispositivos do serviço de *healthcare* são encaminhados à tabela 4 (REMOA);
- Demais pacotes são encaminhados à tabela 3 (Firewall).

Abaixo estão as *flow entries* que implementam as políticas descritas acima:

```
# Encaminha pacotes ARP
table=0, priority=100,arp actions=NORMAL

# Access Point: Input e Output
table=0, priority=10,dl_dst=<MAC access point> actions=resubmit(,1)
```

```
table=0, priority=10,dl_src=<MAC access point> actions=resubmit(,2)

# Dispositivos de monitoramento (REMOA)
table=0, priority=10,dl_dst=<MAC balanca> actions=resubmit(,4)
table=0, priority=10,dl_src=<MAC balanca> actions=resubmit(,4)
table=0, priority=10,dl_dst=<MAC medidor de pressao arterial> actions=resubmit(,4)
table=0, priority=10,dl_src=<MAC medidor de pressao arterial> actions=resubmit(,4)
table=0, priority=10,dl_dst=<MAC sensor de presenca> actions=resubmit(,4)
table=0, priority=10,dl_src=<MAC sensor de presenca> actions=resubmit(,4)

# Demais dispositivos
table=0, priority=0 actions=resubmit(,3)
```

D.2 Tabela 1 — Input

A tabela 1 atua como um *firewall* determinando quais pacotes destinados ao *access point* serão aceitos. O processamento realizado é o seguinte:

- Pacotes originados pela administração dos serviços são aceitos em qualquer porta;
- Pacotes com destino às portas dos serviços SSH (porta 22), Telnet (porta 23) e *web* (portas 80 e 443), não originados pela administração, são descartados;
- Aceita qualquer outro pacote oriundo da rede interna (importante para serviços como DHCP e DNS);
- Os demais pacotes são negados.

Abaixo estão as *flow entries* que implementam as políticas descritas acima:

```
# Pacotes oriundos da administracao do servico
table=1, priority=100,ip,nw_src=<IP servidor REMOA> actions=NORMAL

# Descarta pacotes destinados as portas 22, 23, 80 e 443
table=1, priority=90,tcp,tp_dst=22 actions=drop
table=1, priority=90,tcp,tp_dst=23 actions=drop
table=1, priority=90,tcp,tp_dst=80 actions=drop
table=1, priority=90,tcp,tp_dst=443 actions=drop

# Aceita pacotes com origem na rede interna
table=1, priority=10,ip,nw_src=<rede interna/mascara> actions=NORMAL

# Descarta os demais pacotes
table=1, priority=0 actions=drop
```

D.3 Tabela 2 — Output

Assim como a tabela 1, a tabela 2 também analisa os pacotes relacionados à comunicação do *access point*, contudo, a filtragem é realizada considerando os pacotes enviados pelo *access point*. As políticas aplicadas estão descritas a seguir:

- Pacotes destinados à administração dos serviços são aceitos;
- Pacotes destinados à rede interna são aceitos;
- Os demais pacotes são negados.

Abaixo estão as *flow entries* que implementam as políticas descritas acima:

```
# Pacotes destinados a administracao do servico
table=2, priority=100,ip,nw_dst=<IP servidor REMOA> actions=NORMAL

# Pacotes destinado a rede interna
table=2, priority=10,ip,nw_dst=<rede interna/mascara> actions=NORMAL

# Demais pacotes sao descartados
table=2, priority=0 actions=drop
```

D.4 Tabela 3 — Firewall

Enquanto as tabelas 1 e 2 filtram o tráfego originado ou destinado ao *access point*, a tabela 3 processa o tráfego que passa pelo *switch*, oriundo ou destinado à rede interna do ambiente monitorado ou para a Internet. As políticas utilizadas na tabela 3 são as seguintes:

- Para os dispositivos que já geraram *table miss*, há regras que habilitam o encaminhamento normal, através das quais é possível obter contadores de tráfego, desde que os pacotes sejam originados na rede interna;
- Pacotes com destino à rede interna, cujo alvo são portas que dão acesso a sistemas ou serviços de compartilhamento em LAN, são descartados. A lista abaixo ilustra alguns dos serviços bloqueados:
    - SSH (porta 22)
    - Telnet (porta 23)
    - SMTP (porta 25)
    - Netbios (135-139)
    - SMB (445)

    - Socks (1080)
    - Outros serviços...
- Pacotes destinados aos dispositivos desconhecidos, ou dos residentes, são encaminhados normalmente quando oriundos da rede externa.
- Pacotes de dispositivos desconhecidos geram *table miss* (um *packet-in* é enviado ao controlador).

Abaixo estão as *flow entries* que implementam as políticas descritas acima:

```
# Regras instaladas para encaminhar o trafego de dispositivos
# desconhecidos que ja geraram table miss, desde que originados
# na rede interna.
table=3, priority=90,ip,dl_dst=<MAC televisao>,nw_src=<rede interna/mascara> actions=NORMAL
table=3, priority=90,dl_src=<MAC televisao> actions=NORMAL
table=3, priority=90,ip,dl_dst=<MAC smartphone>,nw_src=<rede interna/mascara> actions=NORMAL
table=3, priority=90,dl_src=<MAC smartphone> actions=NORMAL
table=3, priority=90,ip,dl_dst=<MAC smartphone 2>,nw_src=<rede interna/mascara> actions=NORMAL
table=3, priority=90,dl_src=<MAC smartphone 2> actions=NORMAL
table=3, priority=90,ip,dl_dst=<MAC notebook>,nw_src=<rede interna/mascara> actions=NORMAL
table=3, priority=90,dl_src=<MAC notebook> actions=NORMAL

# SSH
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=22 actions=drop

# Telnet
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=23 actions=drop

# SMTP
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=25 actions=drop

# Netbios
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=135 actions=drop
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=136 actions=drop
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=137 actions=drop
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=138 actions=drop
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=139 actions=drop
table=3, priority=10,udp,nw_dst=<rede interna/mascara>,tp_dst=135 actions=drop
table=3, priority=10,udp,nw_dst=<rede interna/mascara>,tp_dst=136 actions=drop
table=3, priority=10,udp,nw_dst=<rede interna/mascara>,tp_dst=137 actions=drop
table=3, priority=10,udp,nw_dst=<rede interna/mascara>,tp_dst=138 actions=drop
table=3, priority=10,udp,nw_dst=<rede interna/mascara>,tp_dst=139 actions=drop

# SNMP
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=161 actions=drop
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=162 actions=drop
table=3, priority=10,udp,nw_dst=<rede interna/mascara>,tp_dst=161 actions=drop
table=3, priority=10,udp,nw_dst=<rede interna/mascara>,tp_dst=162 actions=drop

# SMB
```

```
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=445 actions=drop
table=3, priority=10,udp,nw_dst=<rede interna/mascara>,tp_dst=445 actions=drop

# Socks
table=3, priority=10,tcp,nw_dst=<rede interna/mascara>,tp_dst=1080 actions=drop
table=3, priority=10,udp,nw_dst=<rede interna/mascara>,tp_dst=1080 actions=drop

# Outros servicos...

# Encaminha normalmente os demais pacotes aos dispositivos que ja
# geraram table-miss
table=3, priority=5,dl_dst=<MAC televisao> actions=NORMAL
table=3, priority=5,dl_dst=<MAC smartphone> actions=NORMAL
table=3, priority=5,dl_dst=<MAC smartphone 2> actions=NORMAL
table=3, priority=5,dl_dst=<MAC notebook> actions=NORMAL

# Pacotes de dispositivos desconhecidos geram table miss
table=3, priority=0 actions=CONTROLLER:65535
```

D.5 Tabela 4 — REMOA

A última tabela do conjunto apresentado processa os pacotes oriundos ou destinados aos dispositivos de monitoramento de sinais de saúde dos pacientes. As políticas utilizadas são as seguintes:

- Pacotes originados pelas *things* da marca Withings, destinados aos servidores da fabricante, são redirecionados ao servidor do REMOA;
- Pacotes originados pelas *things*, destinados ao servidor do projeto REMOA, são encaminhados;
- Pacotes originados pelo servidor do REMOA, destinados às *things*, são encaminhados;

Abaixo estão as *flow entries* que implementam as políticas descritas acima:

```
# Redireciona pacotes destinados ao servidor Withings, originados pelos
# dispositivos de monitoramento, para o servidor do projeto REMOA.
table=4, priority=100,tcp,dl_src=<MAC balanca>,nw_dst=<IP servidor Withings>,tp_dst=80 actions
    =set_field:<IP servidor REMOA>->ip_dst
table=4, priority=100,tcp,dl_src=<MAC medidor de pressao arterial>,nw_dst=<IP servidor
    Withings>,tp_dst=80 actions=set_field:<IP servidor REMOA>->ip_dst

# Encaminha os pacotes originados pelos dispositivos de monitoramento
# destinados ao servidor do REMOA
table=4, priority=100,ip,dl_src=<MAC balanca>,nw_dst=<IP servidor REMOA> actions=NORMAL
table=4, priority=100,ip,dl_src=<MAC medidor de pressao arterial>,nw_dst=<IP servidor REMOA>
    actions=NORMAL
table=4, priority=100,ip,dl_src=<MAC sensor de presenca>,nw_dst=<IP servidor REMOA> actions=
```

```
    NORMAL

# Encaminha os pacotes originados pelo servidor do REMOA para os dispositivos
# de monitoramento.
table=4, priority=100,tcp,dl_dst=<MAC balanca>,nw_src=<IP servidor REMOA> actions=NORMAL
table=4, priority=100,tcp,dl_dst=<MAC medidor de pressao arterial>,nw_src=<IP servidor REMOA>
    actions=NORMAL
table=4, priority=100,tcp,dl_dst=<MAC sensor de presenca>,nw_src=<IP servidor REMOA> actions=
    NORMAL
```