Pós-Graduação em Ciência da Computação

EVARISTO MARCOS DE QUADROS JÚNIOR

AVALIAÇÃO DE UM MODELO PARA O GERENCIAMENTO DE RECURSOS EM UM AMBIENTE IOT USANDO VIRTUALIZAÇÃO BASEADA EM CONTÊINERES

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2017

Evaristo Marcos de Quadros Júnior

Avaliação de um modelo para o gerenciamento de recursos em um ambiente IoT usando Virtualização Baseada em Contêineres

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre Profissional em Ciência da Computação.

ORIENTADOR: **Prof. José Augusto Suruagy Monteiro**

RECIFE
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

Q1a Quadros Júnior, Evaristo Marcos de
Avaliação de um modelo para o gerenciamento de recursos em um ambiente IoT usando virtualização baseada em contêineres / Evaristo Marcos de Quadros Júnior. – 2017.
77 f.: il., fig., tab.

Orientador: José Augusto Suruagy Monteiro.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências.

1. Redes de computadores. 2. Internet das coisas. I. Monteiro, José Augusto Suruagy (orientador). II. Título.

004.6 CDD (23. ed.) UFPE- MEI 2017-252

Evaristo Marcos de Quadros Júnior

Avaliação de um modelo para o gerenciamento de recursos em um ambiente IoT usando Virtualização Baseada em Contêineres

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre Profissional em 10 de julho de 2017.

Aprovado em: 10/07/2017.

BANCA EXAMINADORA

_____

Prof. Paulo André da Silva Gonçalves
Centro de Informática / UFPE

_____

Prof. André Castelo Branco Soares
Universidade Federal do Piauí

_____

Prof. José Augusto Suruagy Monteiro
Centro de Informática / UFPE
(Orientador)

RESUMO

A Internet das Coisas (*Internet of Things – IoT*) é um paradigma onde objetos do dia a dia são equipados com capacidades como identificação, sensoriamento, conectividade e processamento. Novos dispositivos desta categoria são conectados diariamente, em redes residenciais, comerciais e acadêmicas. O provisionamento dos recursos de tais dispositivos é desafiador, sendo muitas vezes desejável, que se tenha uma arquitetura semelhante no dispositivo da ponta e no *data center*. Para contornar esta situação, este trabalho estuda a utilização de conceitos já comuns em ambientes de servidores, aplicando-os em dispositivos IoT. É sabido que a virtualização resolveu vários desafios em infraestruturas convencionais. A virtualização por contêineres é mais leve do que a virtualização usando hipervisores, tendo se mostrado uma candidata a revolucionar a IoT, assim como os hipervisores revolucionaram arquiteturas tradicionais. Desta forma, o modelo avaliado utiliza um mecanismo de contêineres em conjunto com um *switch* virtual. Com isso, busca-se tornar o gerenciamento destes dispositivos mais flexível, facilitando a realocação e otimização dos recursos em uma implementação IoT. O objetivo do trabalho foi avaliar o comportamento de rede em dispositivos comumente usados em IoT, quando submetidos ao modelo estudado. Os experimentos tiveram maior ênfase no comportamento de rede, buscando considerar métricas importantes para determinar o seu desempenho, como o atraso introduzido e a taxa de transferência efetiva do sistema. Também foram realizadas medições de uso dos recursos de processamento e memória, antes e após a introdução dos *softwares* utilizados no modelo. As medições foram feitas implementando um ambiente de testes, usando três gerações de uma plataforma utilizada frequentemente na prototipação de aplicações IoT, o Raspberry Pi. Para o mecanismo de conteinerização foi usado o Docker. O Open vSwitch foi usado como habilitador SDN, possibilitando um melhor controle no tráfego de rede entre contêineres. Os resultados experimentais referentes à implementação completa da pilha de camadas introduzidas, demonstraram que o mecanismo de contêiner teve pouca influência no desempenho de rede dos dispositivos. A introdução do *switch* virtual foi pouco significativa quando utilizado protocolos sem criptografia para fazer o tunelamento da conexão das redes virtuais. Adicionando uma camada IPsec, a degradação de desempenho ficou bastante evidente no Raspberry de primeira geração, atingindo uma taxa de transferência efetiva para recepção de dados em torno de 23% do valor medido como referência. A segunda geração apresentou desempenho superior, mas ainda em 56%. Já a terceira geração do dispositivo, mostrou desempenho aproximado de 88% do valor da taxa de referência. Mesmo com desempenho inferior no *hardware* mais restrito, a vazão resultante ainda é suficiente para muitas aplicações IoT. Tais resultados evidenciaram a evolução apresentada no *hardware* e demonstram uma real possibilidade de aplicação. Apesar da interferência no desempenho de rede com as camadas adicionadas, o uso das tecnologias envolvidas proporciona benefícios adicionais, como maior facilidade no gerenciamento em implementações de novos *softwares*, maior automação no gerenciamento de recursos, facilitação na escalabilidade das aplicações, isolamento e configuração dinâmica do tráfego de rede, e otimização no compartilhamento de recursos de *hardware*.

Palavras-chaves: Internet das coisas. Virtualização baseada em contêineres. Gerenciamento de recursos. Redes virtuais.

ABSTRACT

The Internet of Things (IoT) is a paradigm where everyday devices are equipped with capabilities such as identification, sensing, connectivity and processing. New devices in this category are connected daily in residential, commercial, and academic networks. Provisioning of those device's resources are challenging; therefore, it is often desirable to have a similar architecture at the endpoint device and data center. To overcome this situation, this paper evaluates the use of concepts already common in server environments, in IoT devices. It is well known that virtualization has solved several challenges in conventional infrastructures. Container virtualization is much lighter than virtualization using hypervisors, having proven to be a candidate to improve IoT, just as hypervisors have improved traditional architectures. In this sense, the model evaluated combines a container mechanism and a virtual switch. Thereby, the aim is to make the management of these devices more flexible, making easier reallocation and optimization of the resources in an IoT implementation. The purpose of this thesis was to measure the network behavior of devices commonly used in IoT, when submitted to the evaluated model. The experiments had focused on network behavior, seeking to consider important metrics to determine its performance, such as the delay introduced and the effective transfer rate of the system. The use of processing and memory resources were also measured, before and after the introduction of the software used in the model. The measurements were made by implementing a test environment, using three generations of a platform widely adopted for IoT prototyping, Raspberry Pi. Docker was used for the containerization mechanism. Open vSwitch was used as a SDN enabler, establishing better control over network traffic between containers. Experimental results regarding the complete implementation of the layer stack introduced, demonstrated that the use of the container mechanism had little influence on network performance. The introduction of virtual switch was insignificant when using unencrypted protocols when tunneling the virtual networks. Adding an IPsec layer, performance degradation was quite evident in the first-generation Raspberry, achieving an effective throughput for reception of 23% of the value measured as a reference. The second generation showed higher performance, but still around 56%. Already the third generation of the device, showed performance up to 88% of the transfer rate. Even with lower performance on the most limited hardware, the resulting throughput is still sufficient for many IoT applications. The results proved the evolution of hardware and demonstrate the feasibility of its use. Despite the performance interference with added layers, the use of the technologies involved provides additional benefits, such as easier management in a large-scale implementation, increased automation, more scalability, isolation and dynamic configuration of traffic network, and optimization of hardware sharing resources.

Keywords: Internet of things. Container-based virtualization. Resource management. Virtual networks.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CPU - *Central Processing Unit*

DC - *Data Center*

GPIO - *General Purpose Input Output*

GRE - *Generic Routing Encapsulation*

IoT - *Internet of Things*

IoTaaS - *IoT as a Service*

IPsec - *Internet Protocol security*

LCV - *Linux Container Virtualization*

MTU - *Maximum Transfer Unit*

NFC - *Near Field Communication*

NFV - *Network Function Virtualization*

OVS - Open vSwitch

PC - *Personal Computer*

QoS - *Quality of Service*

RAM - *Random Access Memory*

RFID - *Radio-Frequency Identification*

RPI - Raspberry Pi

SBC - *Single Board Computer*

SDN - *Software Defined Networking*

USB - *Universal Serial Bus*

VM - *Virtual Machine*

SUMÁRIO

1

INTRODUÇÃO

Neste capítulo são apresentadas as motivações e justificativas a partir das quais se idealizou este trabalho. Em seguida, é apresentada a definição dos problemas que se buscam estudar, bem como os objetivos a serem alcançados. O capítulo é finalizado com uma seção expondo a estrutura desta dissertação.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

A Internet das Coisas (IoT – *Internet of Things*) é um paradigma onde objetos do dia a dia são embarcados com capacidades computacionais de modo a torná-los "inteligentes". Estes objetos, que são conectados à Internet, deverão ser capazes de se identificar e comunicar-se entre si, com o objetivo de executar tarefas úteis. Desta forma, tais dispositivos devem ter a habilidade de coletar, processar e transmitir informações.

O uso de tecnologias como redes de sensores, *Radio-Frequency Identification* (RFID), *Near Field Communication* (NFC), redes móveis, e a própria Internet, foram os fundamentos para se chegar no conceito atual de IoT. A integração destas tecnologias em um ambiente mais amplo, torna a Internet das Coisas composta por uma vasta variedade de dispositivos, desde servidores de rede poderosos a simples etiquetas RFID. Devido a esta heterogeneidade, é certo que alguns dos dispositivos usados tenham limitações em seus poderes computacionais.

Com estas limitações em mente, muitas soluções aplicadas em redes de computadores tradicionais não são utilizadas com frequência por requererem quantidades consideráveis de recursos. A virtualização é uma delas. Contudo, Graur (2017) afirma que, conforme o número de coisas conectadas aumenta, é necessário adaptar conceitos, arquiteturas e protocolos, porém,

também é preciso preservar requerimentos majoritários de uma rede dinâmica. A IoT vai muito além de conectar coisas à Internet. A introdução destes novos dispositivos não deve colocar em risco o desempenho ou segurança dos componentes já existentes.

Tschofenig et al. (2015) salientam que o desenvolvimento de sistemas embarcados comuns, já é uma tarefa complexa em si. Projetar objetos interconectados é ainda mais desafiador. Para simplificação do desenvolvimento, diminuindo o custo para o projeto de novos produtos e protótipos, o reuso de trabalho é essencial.

Neste contexto, é sabido que a virtualização apresenta soluções para vários desafios em infraestruturas convencionais, se tornando um conceito bastante usado em ambientes com *hardware* heterogêneo. Dentre os benefícios da virtualização, pode-se citar um maior isolamento entre aplicações, melhor alocação de recursos como processamento, memória, rede, e maior facilidade na mobilidade de *software* entre diferentes tipos de *hardware*.

Recentemente, a virtualização baseada em contêineres tem se apresentado como uma alternativa aos hipervisores. Este tipo de virtualização permite a segmentação de recursos físicos da máquina, criando instâncias isoladas do espaço de usuário do sistema operacional. Nesta abordagem, não existe virtualização do *hardware*, nem é necessária a instalação de um sistema operacional completo para rodar as aplicações. Uma vez que a camada de virtualização está sobre um *kernel* comum aos sistemas hospedeiros, grande parte do *overhead* é eliminado, tornando-se uma forma de virtualização leve.

Desta forma, Celesti et al. (2016) afirmam que esta tecnologia poderá revolucionar arquiteturas IoT, trazendo as mesmas vantagens que os hipervisores trouxeram aos sistemas tradicionais em termos de gerenciamento de recursos, permitindo que gerentes possam instanciar, realocar e otimizar as capacidades de forma mais flexível.

Entretanto, faz-se necessária a compreensão de que não existe um mecanismo único para proteger e gerenciar todo um sistema. Um esquema multicamadas precisa ser desenhado de modo a prover soluções unificadas e simplistas para o gerenciamento de recursos. Ademais, em IoT, soluções precisam ser facilmente integradas em um ambiente heterogêneo, composto por nós com diferentes capacidades de processamento e comunicação (Graur, 2017).

Deste modo, este trabalho estuda o uso de uma solução para o gerenciamento da camada de comunicação com capacidades para Redes Definidas por *Software* (SDN – *Software Defined Networks*) e Redes Virtuais. SDN é uma abordagem que possui bastante afinidade com virtualização e pode oferecer soluções interessantes no âmbito de IoT.

Bizanis e Kuipers (2016) citam algumas situações que podem se beneficiar com o uso de SDN em IoT. Por exemplo, veículos interconectados em uma rodovia, trocando informações,

idealmente requerem latência próxima a zero. Uma rede de sensores em uma indústria pode requerer perdas de pacotes mínimas. Uma rede de vídeo monitoramento pode tolerar certa quantidade de erros e latência, porém requer uma largura de banda muito superior.

Para satisfazer estas necessidades distintas, ao mesmo tempo, o uso de SDN pode oferecer soluções dinâmicas em roteamento e agendamento do tráfego. Com a complexidade de uma rede consistindo de vários tipos de dispositivos e aplicações, necessitando de diferentes níveis de permissão, o uso de SDN pode ser um facilitador para trazer mais segurança e facilidade no gerenciamento. Provendo gerenciamento e coleta de dados sobre redes de forma centralizada, SDN também pode ser usada para tornar a rede mais inteligente e auto adaptativa. Decisões em tempo real podem ser tomadas de acordo com o tipo de tráfego. Redes virtuais podem ser usadas para isolar aplicações, aumentando a segurança.

Em vista das argumentações apresentadas, este trabalho propõe avaliar o desempenho de rede em um ambiente onde é usado um modelo de gerenciamento de recursos composto por um mecanismo de contêineres, para prover funcionalidades de virtualização, e um *switch* virtual, para que se possa obter um melhor controle no tráfego de rede entre contêineres. É utilizada a ferramenta Docker como mecanismo de contêineres e Open vSwitch como habilitador SDN.

1.2 DEFINIÇÃO DO PROBLEMA

Durante pesquisas iniciais, percebeu-se a existência de estudos abordando o uso do Open vSwitch e Docker em dispositivos IoT. Este autor, porém, não encontrou nenhum trabalho avaliando o desempenho de uma solução multicamada, usando estas ferramentas em conjunto. Destaca-se que as ferramentas possuem grande afinidade, sendo atraente seu uso conjunto.

Como salientado anteriormente, é esperado que redes construídas com dispositivos restritos em processamento estejam sujeitas a apresentar baixa vazão e alta latência de comunicação. A introdução destes novos componentes, certamente irá ocasionar algum distúrbio no desempenho de dispositivos restritos. Desta forma, este trabalho busca fazer uma avaliação do desempenho e comportamento de dispositivos comumente usados em IoT, quando submetidos ao modelo estudado.

Com isso, estabeleceu-se os seguintes questionamentos como norteadores deste trabalho:

1. Dispositivos IoT estão preparados para fazer uso conjunto de técnicas de virtualização por contêineres e redes virtuais, já consolidadas no gerenciamento de recursos em dispositivos de rede tradicionais?
2. Quais os impactos no desempenho de rede de dispositivos IoT quando utilizadas as soluções de virtualização selecionadas?

1.3 OBJETIVOS

1.3.1 Objetivo principal

O principal objetivo deste trabalho é avaliar o desempenho de um modelo de gerenciamento de recursos em dispositivos IoT utilizando conceitos de virtualização. Com as aferições, busca-se analisar, usando as principais métricas de desempenho de rede, o impacto das camadas de abstração adicionadas pela arquitetura estudada. Neste contexto, busca-se demonstrar a viabilidade do uso de recursos de virtualização, como facilitadores para o gerenciamento de recursos, mesmo em *hardware* restrito em desempenho.

1.3.2 Objetivos específicos

1. Implementar um ambiente de testes, buscando avaliar o desempenho em uma estrutura real;
2. Elaborar uma abordagem para avaliar o desempenho de rede em um ambiente que utiliza soluções de virtualização baseada em contêineres e *switches* virtuais em dispositivos IoT;
3. Medir as principais métricas de desempenho de rede, em um ambiente elaborado com uma seleção de *hardware* IoT com capacidades de processamento diferentes;
4. Analisar os resultados, buscando verificar quais e como as métricas são afetadas pela conteinerização e uso de um *switch* virtual em dispositivos IoT,

provendo informações do comportamento da arquitetura estudada quando aplicada em um ambiente real.

1.4 ESTRUTURA DO TRABALHO

Adicionalmente a este capítulo introdutório, esta dissertação consiste em mais 5 capítulos, os quais são descritos a seguir.

O Capítulo 2 é composto pela fundamentação teórica, descrevendo conceitos e técnicas importantes para o entendimento do trabalho.

No Capítulo 3 são apresentados trabalhos relacionados com esta dissertação, destacando algumas similaridades e diferenças presentes neste trabalho, em relação a outros projetos de pesquisa.

O Capítulo 4 apresenta a solução proposta, expondo os detalhes da implementação do ambiente, seguido pela descrição dos testes propostos para a validação.

No Capítulo 5 é exposta a avaliação e detalhes dos resultados obtidos nos testes e experimentos executados durante a implementação do ambiente, buscando evidenciar a sua aplicabilidade.

O Capítulo 6 provê um sumário conciso da dissertação, trazendo as conclusões e contribuições obtidas. Para finalizar, é apresentada uma discussão sobre ideias que podem ser usadas como ponto de partida para expandir este trabalho.

2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo busca prover uma visão geral de tópicos importantes para o entendimento do trabalho relatado nesta dissertação. Não se pretende, entretanto, detalhar tais tópicos exaustivamente, mas sim, apresentar brevemente as ideias e conceitos gerais envolvidos. Busca-se ressaltar os conceitos que possuem relação direta com o conteúdo do trabalho apresentado.

2.2 GERENCIAMENTO DE RECURSOS

Gerenciamento de recursos refere-se a alocar de forma efetiva recursos computacionais com disponibilidade limitada. Os programas de computador podem ser capazes de gerenciar seus próprios recursos, usando funcionalidades de linguagens de programação, ou podem usar todo recurso que lhe foi disponibilizado, deixando o gerenciamento por conta do sistema operacional, máquina virtual, ou outro programa.

De acordo com Drewanz (2012) um efetivo gerenciamento de recursos é importante pois aplicativos, serviços, máquinas virtuais, contêineres, possuem diferentes demandas por recursos. A combinação de cargas de trabalho distintas em um mesmo ambiente geralmente implica em diferentes acordos de nível de serviço, com necessidades particulares de vazão, tempo de resposta e disponibilidade de dados.

Entretanto, os recursos computacionais são sempre limitados, sendo compartilhados entre todos os componentes de um ambiente de TI. Portanto, é importante existir estratégias para restringir, isolar, ou pelo menos, limitar o acesso a recursos compartilhados específicos. Desta forma, é possível garantir um nível de serviço adequado para cada ambiente.

Neste contexto, a virtualização de um ambiente possibilita diferentes abordagens para garantir uma melhor distribuição dos recursos disponíveis em um ambiente computacional. Caso não existisse possibilidade de alocação, todas as cargas de trabalho seriam administradas de forma inadequada. Por exemplo, uma máquina virtual (VM – *Virtual Machine*) poderia consumir tanta memória que solicitações de outras VMs no mesmo *host* não poderiam ser atendidas, devido à falta de memória disponível. Outro exemplo, é a capacidade de determinar quais recursos devem ser exibidos ou vistos por um sistema virtualizado hospedado.

Portanto, Drewans (2012) elenca os principais objetivos de gerenciamento de recursos:

- Impedir as entidades de consumir recursos ilimitadamente;
- Ser capaz de alterar uma prioridade, com base em eventos externos;
- Garantir o equilíbrio do uso dos recursos e a otimização da utilização do sistema.

Jennings e Stadler (2015) elencam os principais tipos de recursos em um ambiente virtualizado, destacando os recursos de processamento e rede.

Os recursos de processamento são a coleção de máquinas físicas, cada uma composta por um ou mais processadores, memória, interface de rede e disco. A máquinas físicas possuem implantado nelas, um *software* de virtualização que lhes permite hospedar uma série de contêineres ou VMs isoladas umas das outras, e que podem executar diferentes sistemas operacionais, plataformas e aplicativos. Normalmente os recursos das VMs ou contêineres são limitados pela capacidade de processamento e disponibilidade de memória da máquina física. No entanto, alguns buscam destacar também o impacto da contenção em uso de caches dos processadores físicos e outros recursos micro arquitetônicos, sugerindo que o gerenciamento de recursos pode se beneficiar de modelos ainda mais detalhados.

Para Jennings e Stadler (2015) o segundo tipo de recurso a ser destacado em um ambiente virtualizado, são os recursos de rede. Para a maioria das aplicações, o gargalo de comunicação, imposto pelas tecnologias e protocolos de rede, restringe o desempenho geral. Nesse cenário, os autores citam dois aspectos críticos, sendo a topologia da rede, e como fornecer latência e vazão previsíveis.

A topologia de rede tem impacto significativo no desempenho e na tolerância a falhas de um sistema. Sendo uma prioridade, a modelagem de uma topologia escalável, na qual o aumento do número de portas, aumenta linearmente a largura de banda entregue. O segundo aspecto crítico é como fornecer latência e largura de banda previsíveis em uma rede de *data center* em face de diferentes padrões de tráfego. Tradicionalmente, a solução era provisionar banda em excesso. Porém, o alto custo para esta abordagem em ambientes de larga escala, bem

como a falta de modelos de tráfego detalhados, não demonstravam ser a melhor alternativa. Dado isso, houve um movimento para implementar a diferenciação de serviços através de políticas de QoS (*Quality of Service*) e isolamento em redes virtuais, que segregam o tráfego e permitem arquiteturas de rede mais evoluídas.

Manvi e Shyam (2014) destacam que recentemente, a proliferação de aplicações que envolvem objetos conectados à Internet, deu origem a nuvens IoT, com provedores oferecendo *IoT as a Service* (IoTaaS). Isto gerou a necessidade da criação de modelos efetivos para o gerenciamento de novos tipos de recursos, como a capacidade de sensoriamento e atuação de um aglomerado de dispositivos IoT.

2.3 VIRTUALIZAÇÃO

Virtualização é um termo comumente definido como a abstração de recursos computacionais (Sahoo et al., 2010). Isto significa basicamente, que algum componente do computador é abstraído em algum nível, para outra parte do computador, de modo a simular o ambiente em que uma aplicação é executada. A virtualização pode ser usada por várias razões, podendo-se listar como as principais:

- Compartilhamento de recursos de *hardware* como memória, rede e disco.
- Isolamento, máquinas virtuais podem ser isoladas limitando o efeito que uma aplicação pode gerar em outra.

Existem vários tipos ou níveis de virtualização, sendo que autores possuem diferentes visões quanto às classificações existentes. Sahoo et al. (2010) elencam as descrições que se seguem.

Virtualização completa: Nesta abordagem, o gerenciador de máquinas virtuais roda sobre um sistema operacional hospedeiro, como uma outra aplicação qualquer, e provê *hardware* virtual para um sistema operacional convidado. Em um ambiente totalmente virtualizado, qualquer aplicação ou sistema operacional que seja instalado, não necessitará de nenhuma modificação, nem precisará ser construído presumindo que será executado em um ambiente virtualizado. Isso cria um ambiente bastante isolado, porém poderá resultar em perda de desempenho considerável. A Figura 2.1 apresenta o formato da virtualização completa.

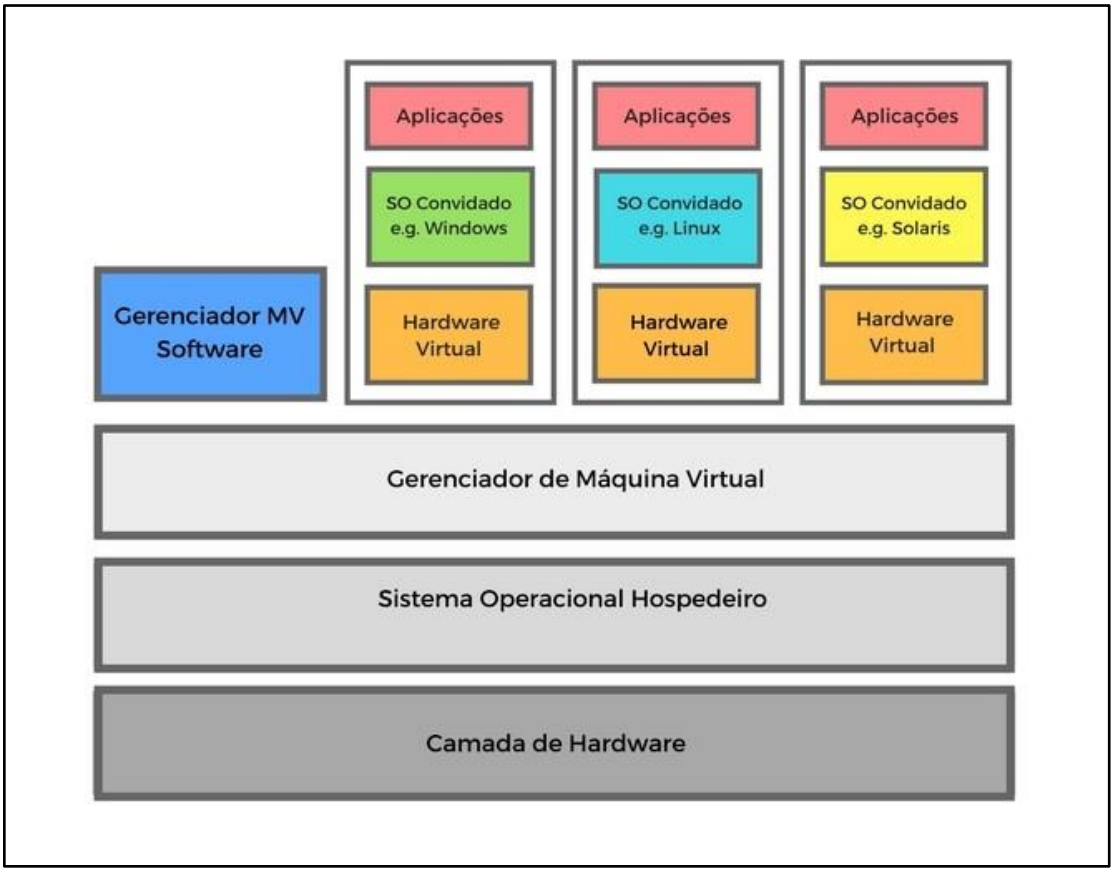

Figura 2.1 - Virtualização Completa (Adaptado de Sahoo et al. 2010)

Paravirtualização: Ao contrário da virtualização completa, o sistema operacional convidado deve ser modificado, e possui conhecimento que está rodando em um ambiente virtualizado. Na paravirtualização é feito uso de um conjunto de instruções específicas de virtualização do processador. Este tipo de virtualização possui uma camada mais leve de *software* entre o *hardware* hospedeiro e o sistema operacional convidado, obtendo desempenho mais próximo a um *hardware* nativo.

Virtualização em nível de hardware: Este formato é o mais comum em uso no mercado para servidores, possui alto nível de isolamento e desempenho. O gerente de máquinas virtuais roda diretamente sobre o *hardware*, sendo comumente chamado de hipervisor. Apesar de o hipervisor também poder ser considerado um sistema operacional, este executa apenas funções específicas necessárias para o controle de acesso ao *hardware* pelas máquinas virtuais, eliminando grande parte da sobrecarga de um sistema operacional completo. Como exemplos pode-se citar o VMware vSphere, Xen, KVM e Hyper-V. A Figura 2.2 mostra a virtualização em nível de *hardware*.

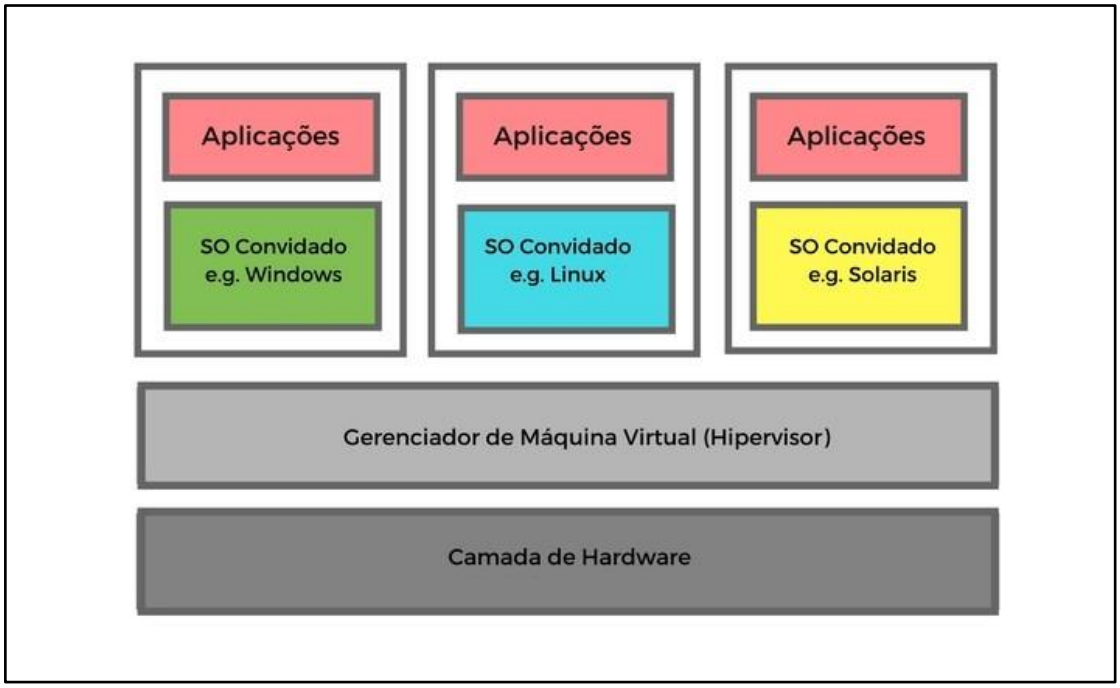

Figura 2.2 - Virtualização em nível de hardware (Adaptado de Sahoo et al. 2010)

Virtualização de aplicações: Permite que aplicações sejam executadas em um ambiente no qual não foram desenvolvidas para rodar nativamente. A camada de virtualização possui apenas os recursos necessários para a aplicação rodar, substituindo parte do ambiente de tempo de execução fornecidos pelo sistema operacional. Normalmente o usuário roda a aplicação sem a necessidade de instalação. O isolamento provido também é chamado de *sandboxing*. Um uso comum é para executar aplicações legadas em sistemas operacionais modernos. Como exemplo pode-se citar Microsoft App-V, Citrix XenApp.

Virtualização em nível de Sistema Operacional: Também chamado de Virtualização Baseada em Contêiner. Aqui não é o *hardware* que é virtualizado, mas sim o sistema operacional. Basicamente, possui uma camada de virtualização sobre um *kernel* comum aos sistemas hospedeiros. Maiores detalhes sobre este tipo de virtualização serão discutidos na próxima seção.

Virtualização de recursos: É a abstração de recursos específicos de um sistema, como espaço de armazenamento de dados, conectividade, dispositivos como impressoras, monitores, teclados, e até mesmo processador e memória. Este trabalho usará conceitos de virtualização de recursos de rede.

2.4 VIRTUALIZAÇÃO BASEADA EM CONTÊINERES

Como visto anteriormente, existem diversas abordagens para virtualização, sendo a mais popular o uso de hipervisores. Porém, para se obter recursos de virtualização em um equipamento com recursos limitados, como é o caso de dispositivos IoT, é necessária uma opção leve.

Recentemente, uma alternativa aos hipervisores que está se popularizando, é a virtualização baseada em contêineres. Este tipo de virtualização particiona os recursos físicos da máquina, criando múltiplas instâncias isoladas para o espaço de usuário. Uma limitação é que o sistema convidado deve ser o mesmo que o hospedeiro, não sendo possível por exemplo, executar um contêiner Windows em cima de um hospedeiro Linux.

A Figura 2.3 demonstra as diferenças chave entre as duas tecnologias de virtualização. A virtualização baseada em contêiner trabalha no nível do sistema operacional, provendo abstração diretamente aos processos convidados, enquanto o hipervisor fornece abstração para um sistema operacional completo. Ou seja, a tecnologia usada em hipervisores trabalha com abstração no nível de *hardware* e contêineres no nível de chamadas de sistema. Como não há um sistema operacional inteiro para cada VM, contêineres usam menos recursos de disco e memória (Celesti et al., 2016).

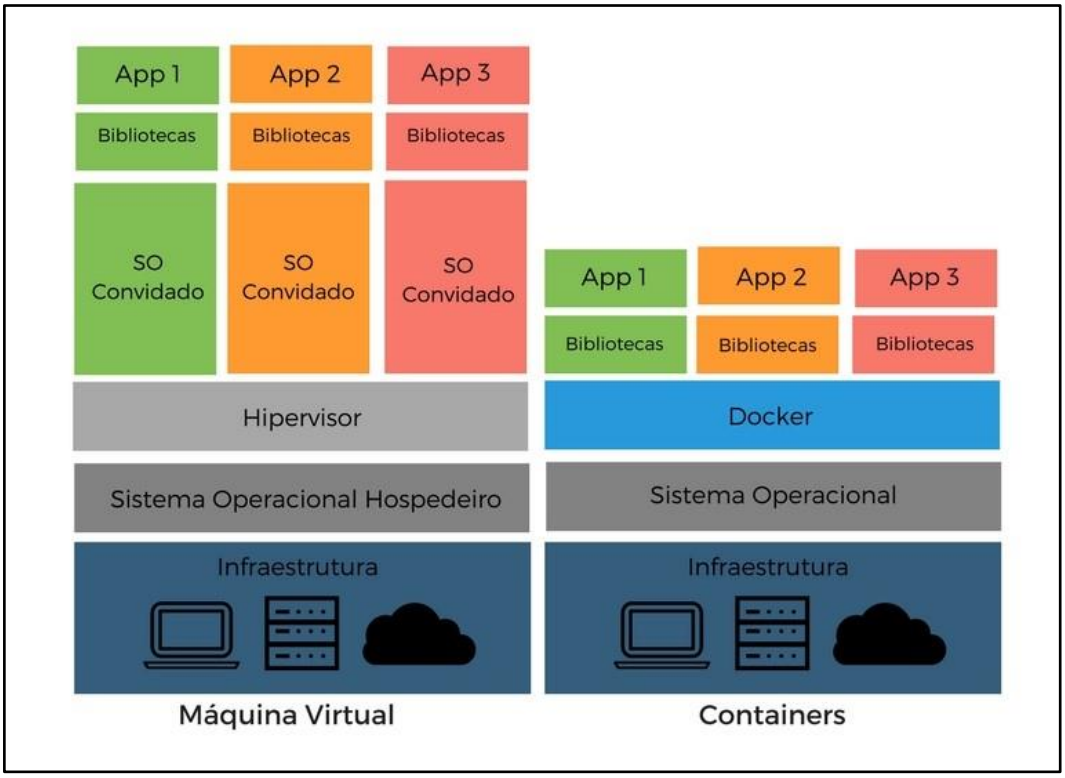

Figura 2.3 - Virtualização baseada em contêineres (Adaptado de Karim et al., 2016)

A virtualização por contêineres é implementada executando mais de uma instância do mesmo sistema operacional. Todos os contêineres compartilham um único *kernel* do sistema operacional. Desta forma, esta abordagem de virtualização teria um isolamento inferior, se comparado à abordagem de hipervisores. Entretanto, para os usuários, cada contêiner é visto como um sistema operacional independente (Xavier et al., 2013). Contêineres individuais possuem interfaces de rede virtuais independentes, espaço de processos independentes e um sistema de arquivos separado. Recursos de Sistema como memória RAM (*Random Access Memory*) podem ser alocados usando controles de grupo, provendo isolamento.

Embora não apresente o mesmo nível de isolamento, este modelo apresenta menos consumo de recursos, sendo mais performático do que as demais, uma vez que não é executado um sistema operacional completo (Xavier et al., 2013). Por possuir um *overhead* menor, consegue-se executar um número maior de contêineres do que máquinas virtuais em um mesmo *host* físico, pois usa menos recursos de disco e memória.

Computadores de placa única, ou *Single Board Computer* (SBC), são bastante populares no âmbito da Internet das Coisas. Dispositivos IoT são embarcados em um SBC equipado com um sistema operacional Linux, que roda *softwares* específicos para coletar e processar dados de sensores externos. Desenvolvimentos recentes, permitiram que a tecnologia

de virtualização baseada em contêineres seja usada em SBCs, que rodam um *kernel* Linux compatível com a camada de virtualização *Linux Container Virtualization* (LCV). Mecanismos populares de contêineres Linux são: Docker, LXC, OpenVZ.

O uso de contêineres em IoT tem se demonstrado não ser apenas um conceito teórico. No âmbito da Internet das coisas, LCV pode permitir uma configuração mais flexível, provendo uma melhor otimização e gerenciamento das capacidades de sensores e atuadores do dispositivo. O uso de contêineres pode resultar na simplificação do processo de distribuição, instalação e execução de aplicações complexas em dispositivos IoT.

A tecnologia LCV permite que múltiplas instâncias do mesmo sistema operacional sejam executadas no espaço de usuário, enquanto compartilham o mesmo kernel. Existem dois modelos gerais de uso para contêineres, o de aplicação e o de sistema. O contêiner de aplicação, como o próprio nome sugere, uma única aplicação roda no contêiner. No modelo contêiner de sistema uma instância do espaço de usuário é iniciada no contêiner. Desta maneira é permitido que múltiplas instâncias isoladas rodem ao mesmo tempo, cada uma com seus espaços de processos, sistema de arquivos e rede (Celesti et al., 2016).

Como o uso de hipervisores é o modelo mais conhecido de virtualização, busca-se fazer um comparativo entre esta tecnologia e a virtualização por contêineres. Celesti et al. (2016) comparam alguns critérios de interesse entre a abordagem tradicional por hipervisores, como tempo de inicialização, controle de tempo de execução, velocidade, isolamento, consumo de disco, canais de comunicação, atribuição ou separação de recursos e acesso direto ao *hardware*. Cada critério será explicado com mais ênfase a seguir.

Em relação ao tempo de inicialização, o contêiner geralmente é mais leve do que uma VM completa, e na maioria dos casos, o LCV será mais rápido para a inicialização inicial e da aplicação. O critério de controle dinâmico de tempo de execução demonstra que devido ao seu conceito, o LCV possui um nível de integração maior com o sistema hospedeiro. É possível, por exemplo, iniciar ou finalizar aplicações dentro de um contêiner, diretamente através do hospedeiro, sem necessariamente precisar passar por conexões virtuais, como ocorre em hipervisores. Já em relação ao critério de velocidade, o acesso direto aos dispositivos virtuais de uma VM pode apresentar uma diferença significativa de velocidade se comparada ao acesso usando o mesmo *driver* rodando no sistema hospedeiro de um contêiner. Apenas testes em um cenário específico podem demonstrar o quanto o *overhead* de comunicação se compara ao *overhead* de virtualização, afetando a latência e taxa de transferência efetiva.

Quanto ao isolamento, os contêineres não proveem do mesmo no nível de sistema. Por outro lado, o uso de máquinas virtuais, com cada VM executando seu próprio *kernel* e espaço

de usuário, permite isolamento efetivo, tanto ao nível do sistema quanto do usuário. Em relação ao consumo de disco, o LCV permite compartilhar o sistema operacional e qualquer outra parte do espaço de usuário que o desenvolvedor desejar. Enquanto o uso de hipervisores requer que as imagens de cada VM sejam armazenadas separadamente, não permitindo compartilhamento.

O comparativo dos canais de comunicação entre o ambiente virtual e físico evidencia que ambos os conceitos permitem a comunicação através de interfaces virtuais, seriais ou de rede, ou sistemas de arquivos compartilhados que podem ser usados para sinalização ou mensagens. Em relação à atribuição ou separação dinâmica de recursos, aspecto importante no gerenciamento de capacidade e de falhas, é permitido, por exemplo, atribuir recursos de processamento e memória adicionais para VMs sobrecarregadas e remover recursos das que estão com baixo uso. Ambas as soluções possuem recursos para satisfazer estes cenários.

E o último critério analisado, acesso direto ao *hardware*, mostra que uma VM rodando em um hipervisor pode ter acesso direto aos periféricos de *hardware*, usando funções específicas para este fim, disponibilizadas pelos hipervisores. Porém, o mesmo não é possível usando um contêiner LCV, pois possui acesso apenas ao espaço de usuário, requerendo que o driver correspondente esteja no *kernel* hospedeiro (Celesti et al., 2016).

2.5 REDES VIRTUAIS E REDES DEFINIDAS POR SOFTWARE

Embora as noções de SDN e redes virtuais compartilhem muitos elementos em comum e os termos sejam frequentemente usados de forma intercambiável, na verdade, são tecnologias distintas. Redes virtuais podem funcionar em combinação com SDN, geralmente com controladores e *switches* SDN atuando como orquestradores na segmentação de redes físicas. Estas tecnologias e suas combinações, podem ser usadas como facilitadores para a implementação de ambientes IoT.

A principal ideia por trás de SDN, é separar o plano de controle (onde os procedimentos lógicos que suportam os protocolos de rede são executados e todas as decisões relevantes são tomadas) do plano de dados (onde o encaminhamento de pacotes na interface mais adequada para o destino pretendido é executado). A Figura 2.4 exemplifica esta arquitetura. A principal figura nesta abordagem, é o controlador, que se comunica com as aplicações de rede que implementam funções tais como *firewall*, roteamento e engenharia de tráfego através de uma interface chamada *north bound* e distribui as regras derivadas para todos

os elementos de rede controlados pelo mesmo. O controlador também se comunica com a camada de infraestrutura através da interface *south bound*, aplicando as regras geradas para os *switches* de rede, que encaminham pacotes de acordo com tais regras. Desta forma, SDN oferece a possibilidade de encaminhar o tráfego de forma mais inteligente, por exemplo, para equilibrar a carga na rede, ou explorar recursos de rede que estejam subutilizados. (Bizanis e Kuipers, 2016).

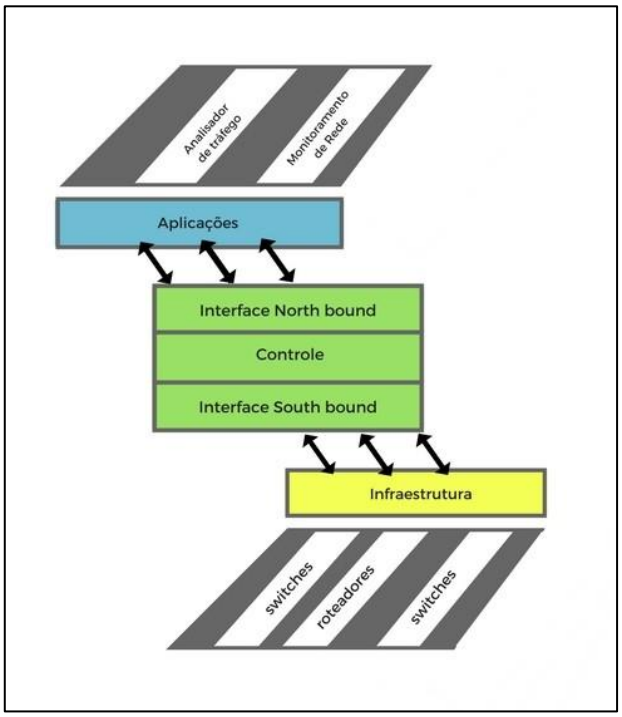

Figura 2.4 - Interfaces de controle SDN

O termo virtualização de rede diz respeito à possibilidade de vários provedores de serviços formarem redes virtuais separadas e isoladas, compartilhando os mesmos recursos físicos, fornecidos por um ou mais provedores de infraestrutura.

De acordo com Bizanis e Kuipers (2016), estas características possibilitarão criar infraestruturas que comportam múltiplos serviços IoT, permitindo QoS diversificado e rápida introdução de novos serviços e aplicativos. As técnicas de virtualização de rede oferecem custos reduzidos de despesas de capital e despesas operacionais, pois compartilham a infraestrutura de rede, melhorando o tempo de entrega de novos serviços e aplicativos ao mercado. Além disso, possibilitam maior customização e agilidade, usando conceitos de segmentação e isolamento de subredes.

Garrison (2014) exemplifica a utilização de redes virtuais com o questionamento de como máquinas virtuais ou contêineres podem ser movidas em diferentes domínios lógicos. A virtualização de rede permite criar segmentos, dividindo a rede logicamente. Uma rede virtual é formada por uma rede sobreposta, usando protocolos de tunelamento. Em vez de conectar fisicamente dois domínios em uma rede física, ela cria um túnel através de outra rede já existente. É uma maneira para os administradores segmentarem sua infraestrutura lógica, sem fazer mudanças na infraestrutura física já existente.

Desta maneira, enquanto redes virtuais, adicionam túneis que formam redes sobrepostas a uma topologia física, a abordagem SDN pode alterar dinamicamente esta topologia lógica, e ainda prover métodos de QoS complexos. Portanto, estes dois conceitos estão bastante relacionados, formando uma nova maneira para provisionar e gerenciar os recursos de redes disponíveis em uma infraestrutura. Neste trabalho, é avaliado um *switch* virtual capaz de criar redes de sobreposição, sendo programável e compatível com o conceito SDN.

3

TRABALHOS CORRELATOS

As soluções propostas nesta dissertação possuem relação com diferentes trabalhos na área de pesquisa de Internet das Coisas, que foram selecionados e agrupados neste capítulo. Embora possam estar direcionados a diferentes questões de pesquisa, ainda podem ser relacionados com o escopo desta dissertação. A importância em citá-los é que empregam técnicas que podem ser usadas em conjunto, buscando atingir as propostas deste trabalho.

3.1 DOCKER EM DISPOSITIVOS IOT

Celesti et al. (2016) em "*Characterizing Cloud Federation in IoT*" citam a virtualização baseada em contêineres, como tecnologia habilitadora para a criação de nuvens IoT federadas, entre provedores *IoT as a Service* (IoTaaS). Nuvens IoT indicam um tipo de sistema distribuído constituído por um conjunto de dispositivos inteligentes, interconectados a uma infraestrutura remota em nuvem. Os autores vislumbram que os provedores IoTaaS poderão federar seus serviços entre si, de modo a gerar economia e otimização de recursos. No cenário descrito pelos autores, cada operador de nuvem IoT, poderá compartilhar seus recursos disponíveis, como hoje já ocorre em termos de capacidade de armazenamento de dados, poder computacional em uma nuvem tradicional. Neste novo conceito, as capacidades de sensoriamento e atuação também poderão ser compartilhadas e aumentadas sob demanda. Para isso, devem adotar massivamente a virtualização baseada em contêineres.

Tso et al. (2013) descrevem um *testbed* de experimentação para computação em nuvem. Como motivação, citam que os *Data Centers* (DC) que suportam os serviços em nuvem geralmente são formados por centenas de servidores de rede, e o alto custo de investimento

constituem um grande obstáculo para a implementação e avaliação de pesquisas na área. Em contrapartida, a introdução do Raspberry Pi, um computador de baixo custo, possibilitou a construção de um DC em miniatura mais acessível. O modelo criado é composto por aglomerados de dispositivos Raspberry Pi, que emulam todas as camadas de uma pilha *cloud*, fornecendo um ambiente de experimentação em nuvem com recursos completos.

No âmbito de análise de desempenho de contêineres Linux em dispositivos IoT, pode-se citar trabalho desenvolvido por Krylovskiy (2015). Intitulado "*Internet of Things Gateways Meet Linux Containers: Performance Evaluation and Discussion*", o trabalho busca prover uma ideia inicial do comportamento das aplicações, bem como o desempenho e *overhead* introduzido pela utilização de contêineres Linux em dispositivos IoT. Como motivação para o estudo, cita os benefícios que a tecnologia pode trazer na instalação, configuração e atualização de *software* em ambientes IoT. Para as avaliações, utilizou dois modelos de dispositivos, Rapsberry Pi e Rapsberry Pi 2. Os testes de desempenho foram feitos usando *benchmarks* sintéticos para avaliar a CPU (*Central Processing Unit*), disco e rede. Também foi analisado o comportamento de duas aplicações reais usadas em IoT, o MQTT Broker, um serviço mensageiro usado para transmitir dados de sensores, e o *middleware* LinkSmart Device Gateway. Os resultados obtidos por eles demonstraram certa sobrecarga no desempenho do Raspberry Pi 1, porém, o desempenho do Raspberry Pi 2 foi promissor. Entretanto, este trabalho teve ênfase nos testes de processamento, disco e memória. Não foi feita uma avaliação aprofundada da rede e levou em consideração apenas a interface de rede do Docker, sem levar em consideração uma arquitetura de rede mais elaborada.

Morabito (2016), com o trabalho *"A performance evaluation of container technologies on Internet of Things devices"* apresenta estudos semelhantes a Krylovskiy (2015). Demonstra uma avaliação de desempenho do Docker no Raspberry Pi 2. Também usou *benchmarks* sintéticos para analisar processamento, memória e CPU, também analisou o desempenho do Apache e MySQL, buscando evidenciar testes de consumo de energia adicional causado pelo sistema de contêineres.

Em estudo mais recente, Morabito (2017) reporta uma avaliação de desempenho com os pontos fortes e fracos de dispositivos de baixa potência utilizando virtualização por contêineres. Novamente, o autor busca promover estudos focados no consumo de energia e eficiência energética, que são métricas importantes no âmbito de IoT. Neste estudo, é investigado um amplo conjunto de dispositivos *Single Based Computer* (SBC). Com o intuito de comparar os SBCs, foram utilizados na análise: Raspberry Pi 2, Raspberry Pi 3, Odroid C1+, Odroid C2, Odroid XU4. As medições feitas no estudo avaliaram o desempenho de CPU,

desempenho de memória, desempenho de disco, desempenho rede, eficiência energética e medições de temperatura da placa.

Embora existam vários estudos comparando a performance de virtualização baseada em hipervisores e contêineres, poucos foram executados em dispositivos similares ao Raspberry. Um motivo para isso, é que não é trivial executar um hipervisor nestes dispositivos. É necessário um processador com suporte a instruções de virtualização e um *kernel* compilado com funções específicas ativadas. Neste contexto, Ramalho e Neto (2016) compararam o desempenho das duas tecnologias de virtualização, rodando o hipervisor KVM e o gerenciador de contêineres Docker em um dispositivo Cubieboard2. Os resultados obtidos mostraram que, enquanto o Docker teve desempenho semelhante ao nativo, o hipervisor ocasionou uma sobrecarga que não pode ser desconsiderada em todos os testes executados (CPU, memória, disco e rede).

3.2 OPEN VSWITCH EM DISPOSITIVOS IOT

Em relação a estudos relacionados ao uso do Open vSwitch (OVS) em dispositivos Raspberry, pode-se citar o trabalho de Kim et al. (2014). No trabalho, é demonstrada a implementação de um *testbed* de baixo custo voltado ao estudo de SDN, usando o Open vSwitch. Para testar múltiplas conexões foram usadas interfaces virtuais criadas no OVS. A validação do ambiente foi realizada usando o OpenFlow 1.0, sendo que a performance apresentada foi semelhante ao uso de equipamentos NetFPGA.

Em abordagem semelhante, Han e Lee (2015) também avaliaram o uso do OVS em um *testbed* SDN de baixo custo usando Raspberry. Entretanto, diferencia-se do trabalho de Kim et al. (2014), pois não se limitaram aos testes usando interfaces virtuais do Open vSwitch. Adicionaram interfaces Ethernet USB extras, vinculadas a *bridges* do OVS, de modo a fazer com que os Raspberry se comportem de fato, como *switches* OpenFlow. A arquitetura do ambiente proposta no trabalho permite executar medições e experimentos levando em consideração variáveis de campo, como latência de propagação e erros no meio físico, ao contrário de experimentos executados em simuladores ou um emulador SDN como o Mininet.

3.3 SOLUÇÕES CONJUNTAS OVS E DOCKER EM DISPOSITIVOS IOT

A virtualização baseada em contêineres e SDN possuem muita afinidade. Tanto que diversos autores citaram em seus estudos, a conveniência de se usar conjuntamente as ferramentas Docker e o Open vSwitch, normalmente mostrando casos de uso conceituais desta combinação. Apesar disso, durante o levantamento de trabalhos relacionados, não foram encontradas pesquisas demonstrando uma análise aprofundada do desempenho de rede desta combinação específica de ferramentas, em um ambiente formado por dispositivos IoT. O que representa um diferencial no estudo reportado nesta dissertação.

4

AMBIENTE ESTUDADO E METODOLOGIA

Dada a motivação descrita no capítulo introdutório desta dissertação, buscou-se estabelecer os requisitos para a escolha das ferramentas a serem usadas no ambiente estudado. Em seguida, é apresentado o conjunto de ferramentas selecionadas, bem como a descrição da metodologia e preparação do ambiente sob o qual os testes foram executados.

4.1 APLICABILIDADE

Em relação à aplicabilidade dos conceitos de virtualização em IoT, o benefício mais promissor para o uso desta tecnologia é possibilitar o compartilhamento de recursos entre diferentes domínios IoT.

De acordo com Celesti et al. (2016) para atingir requerimentos dinâmicos de uma nuvem IoT, como balanceamento de carga de processamento e economia de energia, os serviços instalados em contêineres podem ser migrados através de dispositivos IoT pertencentes a nuvens IoT de diferentes domínios. Este cenário é exemplificado na Figura 4.1

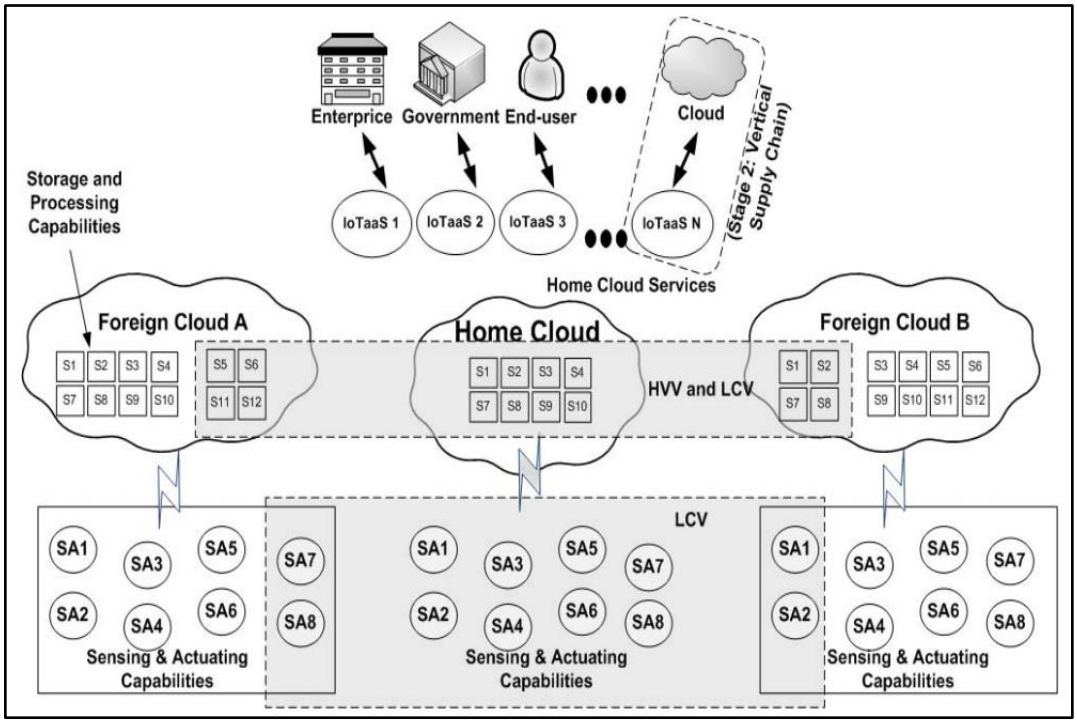

Figura 4.1 - Federação de nuvens IoT (Celesti et al. 2016)

No exemplo, uma nuvem IoT (*Home Cloud*) já saturada pode utilizar técnicas tradicionais de elasticidade que podem empregar tanto virtualização baseada em hivervisores, quanto em contêineres, para aumentar seus recursos de armazenamento e processamento (área sombreada indicada por HVV e LCV). Da mesma maneira, a capacidade de sensoriamento e atuação (área sombreada indicada por LCV) pode ser aumentada, migrando suas aplicações instaladas em contêineres, para dispositivos IoT de nuvens diferentes (*Foreign Cloud*).

Desta maneira, o provedor de serviços IoT responsável pela "*Home Cloud*" pode continuar a atender as demandas de seus clientes. Este processo pode ser totalmente transparente aos clientes. Apesar dos dispositivos IoT pertencerem fisicamente a outras nuvens, são considerados logicamente hospedados pela infraestrutura virtualizada da "*Home Cloud*".

De maneira análoga, uma nuvem IoT pode usar a virtualização não apenas para aumentar sua capacidade, mas também para aumentar seu leque de serviços, uma vez que pode rodar suas aplicações IoT em sensores de tipos específicos, que não estão disponíveis em sua própria infraestrutura física. Da mesma forma, uma infraestrutura IoT pode ser compartilhada entre vários clientes, onde cada um roda suas próprias aplicações em contêineres.

Contextualizando em aplicações IoT reais, as aplicabilidades deste conceito são inúmeras. Por exemplo, uma nuvem de sensores de controle de poluição pode disponibilizar seus dispositivos para um provedor de serviços de detecção de queimadas. Empresas

provedoras de serviços de controle de frota podem disponibilizar seus sensores de rastreamento para provedores de serviços de análise de tráfego de veículos. Um provedor de serviços meteorológico pode disponibilizar seus sensores para um provedor de serviços de agricultura de precisão, entre outras possibilidades.

4.2 ESCOLHA DO HARDWARE E SOFTWARE

Para assegurar que o ambiente de testes seja construído de forma consistente com as características esperadas em uma implementação IoT, e adequado à aplicabilidade descrita, foram levantados alguns requisitos para corroborar a escolha do conjunto de *software* e *hardware* testados.

4.2.1 Hardware

Os requisitos de uma solução IoT normalmente levantam a necessidade de um *hardware* tamanho reduzido, baixo custo e baixo consumo de energia.

O requisito de custo explica-se pelo fato de um ecossistema IoT ser composto de um grande número de dispositivos. Além disso, muitas soluções IoT usam a ideia de embarcar inteligência em objetos já comuns em nosso dia a dia. Assim como o tamanho, que deve ser proporcional ao objeto em que será embarcado, usar um *hardware* caro, pode inviabilizar economicamente a solução.

Em muitos casos, os dispositivos serão operados por baterias e dispersos em locais onde a troca da mesma pode ser inviável. Desta forma, baixo consumo de energia é um requisito chave. Porém, apesar do apelo em construir dispositivos mínimos, com baixo consumo de energia e recursos de processamento na medida para a função que exercerá, deve-se também analisar o que acontecerá caso seja necessário adicionar novas funcionalidades na aplicação.

Tais funcionalidades podem demandar recursos mais robustos, e com certeza não será interessante precisar trocar todo o *hardware* da aplicação para atender a novas demandas. Portanto, o *hardware* a ser usado não deve ser tão limitado em poder de processamento, de modo a não aceitar atualizações de *software*.

Estas limitações levaram as primeiras soluções IoT a usarem microcontroladores que rodavam apenas *softwares* especificamente desenvolvidos para eles, sem um sistema operacional. Com o desenvolvimento e miniaturização de *hardware*, atualmente já podemos contar com dispositivos que rodam um sistema operacional completo e com uma rica plataforma de desenvolvimento.

As plataformas de prototipação para IoT em alta atualmente, já usam arquiteturas comuns em aplicações de rede tradicionais, como ARM e x86. Isso torna possível disponibilizar a flexibilidade de uma máquina Linux completa, trazendo toda a confiabilidade já existente. Logicamente, ainda existem demandas onde se torna custosa e improvável a necessidade de um processador ARM ou x86 inteiro, por exemplo, simplesmente para acender uma lâmpada.

Desta forma, os dispositivos escolhidos para a criação do ambiente estudado, levou em consideração estes requisitos, procurando usar dispositivos comuns em IoT, de fácil acesso, e que o autor possuía algum conhecimento prévio.

Brown (2014, 2015 e 2016) publicou resultado de enquete conduzida pelo portal Linux.com, onde foi questionado qual o SBC preferido pelos leitores do portal, bem como qual a principal finalidade para a qual usariam tal dispositivo. As três versões do Raspberry Pi lideraram o ranking, sendo o Raspberry Pi 1 vencedor em 2014, Rasperry Pi 2 em 2015 e o Raspberry Pi 3 em 2016. Quanto à finalidade prevista para o dispositivo, a automação residencial foi a mais citada, também nos três anos.

A família de dispositivos Raspberry Pi são SBCs baseados na arquitetura ARM, apresentam baixo consumo de energia e são vendidos a um custo baixo, sendo que uma das versões (Raspberry Pi Zero) foi lançada ao preço de apenas 5 dólares. Também, conforme a pesquisa citada anteriormente, grande parte dos usuários adquirem este dispositivo com a finalidade de criar aplicações IoT.

Considerando o apresentado, foram selecionadas três versões do Raspberry Pi (1 2 e 3) para compor o ambiente estudado.

4.2.2 Software

Para a escolha dos *softwares* usados no ambiente, levou-se em consideração a existência de versões compatíveis com o *hardware* escolhido, bem como a disponibilidade de pacotes para os sistemas operacionais suportados pela plataforma de *hardware* escolhida.

Como motor para a virtualização baseada em contêineres, optou-se pelo uso do Docker. O Docker é um *software* de código aberto, lançado em 2013 e se tornou o sistema de conteinerização mais comum em uso atualmente. A abordagem do Docker primeiramente voltou-se ao apelo do desenvolvimento ágil, aumentando a velocidade no desenvolvimento e teste de aplicações, uma vez que a abordagem de contêineres facilita a portabilidade de aplicações.

Entretanto, atualmente é observada uma crescente adoção do Docker também na infraestrutura, substituindo os tradicionais hipervisores. O Docker adicionou suporte oficial ao Raspberry em julho de 2016. Neste cenário, o uso do Docker mostra-se como uma abordagem interessante para disponibilizar aplicações que rodam em contêineres neste tipo de dispositivo.

Para o *switch* virtual, foi escolhido o *software* Open vSwitch, também de código aberto. O objetivo do projeto Open vSwitch é disponibilizar um *switch* virtual com qualidade para ambientes em produção, que suportam interfaces tradicionais de gerenciamento, bem como abrir as funções de encaminhamento para extensões programáveis. O Open vSwitch é adequado para o funcionamento como *switch* virtual em ambientes virtuais Linux, como Xen, KVM, VirtualBox, suportando uma configuração distribuída entre vários dispositivos físicos.

4.3 METODOLOGIA

A metodologia usada como base deste trabalho foi adaptada da RFC2544. Nesta RFC, Bradner e McQuaid (1999) definem um conjunto de testes que podem ser usados para apresentar as características de desempenho de dispositivos de rede. Além de definir os testes, também sugerem como os resultados das medições feitas devem ser formatados e apresentados em um relatório de desempenho dos dispositivos.

Cabe salientar que os autores buscaram criar uma metodologia genérica, que pudesse ser adaptada a diferentes equipamentos. O documento foi produzido tendo em mente que os testes possam ser executados mesmo não existindo um aparato pronto, produzido especificamente para a função de testador e com todos os testes implementados, mas que tal dispositivo possa ser construído. Com isso, é esperado que os testes possam prover resultados para que os interessados possam comparar diferentes soluções, avaliando quais se adequam melhor a cada circunstância.

Como trata-se de uma metodologia genérica, nem todos os testes descritos se aplicarão a todo dispositivo. Portanto, deve-se selecionar quais são alinhados e suportados pela tecnologia envolvida.

As próximas seções buscam compilar a metodologia usada com os pontos considerados durante a avaliação realizada neste trabalho. Questões citadas na metodologia como a inserção de modificadores, por exemplo, usar condições diferentes de endereçamento, roteamento, filtragem de pacotes, filtragem de endereços, presença de quadros de gerenciamento, não foram considerados, por serem mais aplicáveis à avaliação de dispositivos como roteadores, *firewalls*, etc.

4.3.1 Preparação dos testes

Bradner e McQuaid (1999) afirmam que a forma ideal para implementar a série de testes, é usando uma conexão ponto a ponto, onde tanto a porta de transmissão, quanto a de recepção estejam conectados diretamente no dispositivo a ser testado, conforme a Figura 4.2. Desta forma, todo o tráfego de teste irá passar apenas pelo dispositivo alvo, podendo-se determinar facilmente se todos os pacotes foram transmitidos e recebidos corretamente. Ou seja, não é recomendado que os testes sejam feitos em um ambiente em produção, pois os resultados ficariam sujeitos às interferências de outros componentes.

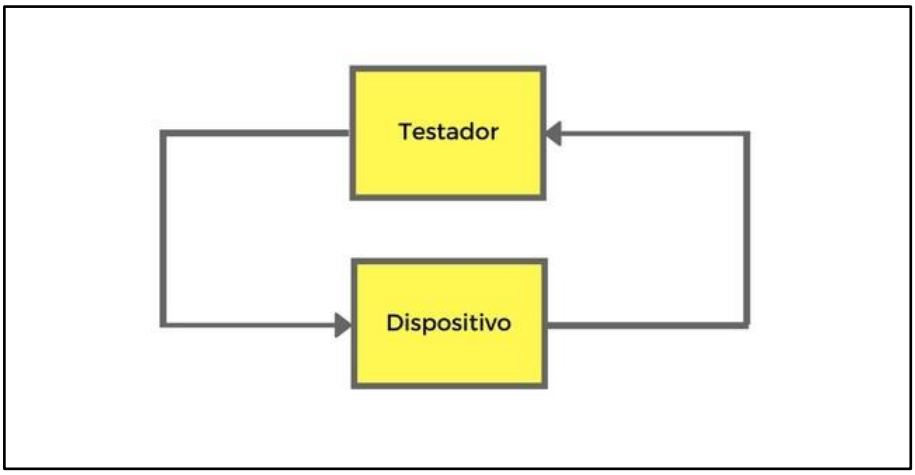

Figura 4.2 - Conexões para testes ponto-a-ponto (Adaptado de Bradner e McQuaid, 1999)

Estendendo o modelo de testes com um único dispositivo, medições considerando múltiplos dispositivos ou ambientes heterogêneos podem ser coletadas. Com isso, o dispositivo único é substituído por um sistema interconectado. Desta forma, esta metodologia pode suportar a avaliação de uma variedade de combinações de dispositivos, meios de transmissão, serviços e protocolos.

Para testar múltiplos meios, como por exemplo, simular a transição entre duas LANs conectadas em um link WAN, as conexões devem estar em série, conforme ilustrado na Figura 4.3.

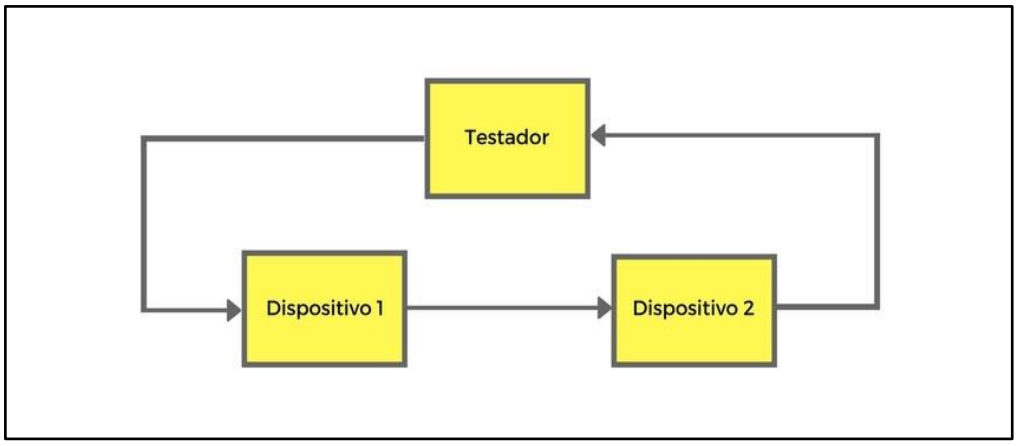

Figura 4.3 - Conexões para testes em série (Adaptado de Bradner e McQuaid, 1999)

Como neste formato múltiplos dispositivos são tratados como um sistema único, existem algumas considerações a serem feitas. Por exemplo, se forem apenas feitos testes com o conjunto completo conectado, os resultados das medições serão agregados. Esta medição sozinha, não refletirá as assimetrias existentes no comportamento de cada dispositivo. Para contornar esta situação, executam-se medições na interface entre cada dispositivo cascateado.

4.3.2 Preparação dos dispositivos a serem testados

A metodologia aplicada especifica que os objetos a serem testados devem ser configurados seguindo as instruções padrão providas ao usuário. Também deve-se ativar e configurar todos os serviços e protocolos necessários. Com isso, espera-se que todos os testes serão executados sem nenhuma alteração no dispositivo, a não ser as configurações específicas

do teste realizado. As informações de roteamento devem ser enviadas no início de cada teste, contendo todos os endereços de redes necessários para sua execução.

4.3.3 Tamanho dos quadros

Os testes devem ser realizados para um número variado de tamanhos de quadros. Ou seja, os testes devem incluir o tamanho mínimo e máximo do quadro para o protocolo sendo testado, e também uma quantidade suficiente de tamanhos entre os extremos, de forma a se obter uma caracterização completa. Bradner e McQuaid (1999) definem que no mínimo cinco tamanhos devem ser considerados.

A inclusão de tamanhos mínimos, com pouco ou nenhum dado, composto basicamente pelos cabeçalhos dos protocolos envolvidos, podem ajudar na caracterização do *overhead* de processamento por quadro. Normalmente, o tamanho máximo e mínimo serão determinados pelas limitações dos protocolos da camada de enlace. Para Ethernet, os tamanhos recomendados são 64, 128, 256, 512, 1024, 1280, 1518, porém um número maior de tamanhos pode ser usado visando aumentar a resolução dos resultados.

Quando dispositivos interconectados estiverem usando enlaces com diferentes MTUs, deve ser considerado o meio com maior MTU, reportando resultado “zero” no caso de o enlace com menor MTU não suportar fragmentação.

4.3.4 Detecção de quadros recebidos

Qualquer quadro recebido durante a execução, que não faça parte do teste envolvido deve ser descartado dos resultados. Os resultados devem considerar o número de quadros recebidos, se chegaram na ordem correta, duplicados, bem como os quadros perdidos.

4.3.5 Testes bidirecionais

Considerando que a atividade normal de uma rede envolve tanto tráfego de recepção quanto de transmissão, devem ser executados testes em ambas as direções. O testador deve ter

capacidade de oferecer a mesma taxa em ambas as direções, sendo que a soma não deve exceder o limite teórico do enlace.

4.3.6 Duração e repetição dos testes

Bradner e McQuaid (1999) sugerem que cada repetição dos testes deve durar pelo menos 60 segundos, de modo que se possa caracterizar como o dispositivo se comportará em uso contínuo. Também, cada teste foi repetido pelo menos vinte vezes, conforme sugerido na metodologia adotada.

4.3.7 Métricas

As métricas sugeridas por Bradner e McQuaid (1999) são: Vazão, latência, taxa de perdas de pacotes, quadros *back-to-back*, tempo de recuperação do sistema, tempo de *reset*.

Entretanto, neste trabalho optou-se por utilizar como métricas de desempenho de rede, a vazão, latência e *jitter* (variação do atraso).

Também foram feitos testes específicos em relação ao ambiente estudado, como o tempo de inicialização de contêineres, bem como foram executados *benchmarks* sintéticos para avaliar CPU e memória.

Maiores detalhes de como as medições foram feitas, serão descritos durante a apresentação de cada experimento conduzido, que serão expostos no Capítulo 5.

4.4 AMBIENTE DE TESTES

O ambiente de testes foi implementado buscando criar um cenário onde se pudesse simular a aplicabilidade da abordagem estudada, considerando os requisitos levantados, bem como o conjunto de *software* e *hardware* selecionados previamente.

4.4.1 Configuração de hardware

A lista de *hardware* usada no experimento refere-se a três dispositivos IoT e um PC configurado como testador. O PC usado é equipado com a seguinte configuração:

- Processador: Quad-core Intel Core i5-4590T
- Frequência base/Máxima: 2GHz / 3GHz
- Memória RAM instalada: 16GB DDR3
- Chipset de Rede: 10/100/1000 Mb/s Realtek PCIe GBE RTL8111
- Sistema Operacional: Debian Linux 8 Jessie (Software in the Public Interest Inc., 2015)
- Kernel: 3.16.0-4-amd64

Para simular os nós IoT, foram usadas três gerações do Raspberry Pi. O objetivo com isso, foi criar um conjunto de dispositivos que pudessem representar a evolução do *hardware* usado em IoT nos últimos anos. As características de *hardware* dos dispositivos usados são:

Rapsberry Pi 1:

- Arquitetura: ARMv6Z (32-bit)
- SOC: BCM2835
- CPU: 700 MHz single-core ARM1176JZF-S
- RAM: 512 MB
- Rede: 10/100

Rapsberry Pi 2:

- Arquitetura: ARMv7-A (32-bit)
- SOC: Quad-core Broadcom BCM2836
- CPU: 900 MHz 32-bit quad-core ARM Cortex-A7
- RAM: 1GB
- Rede: 10/100

Rapsberry Pi 3:

- Arquitetura: ARMv8-A (64/32-bit)
- SOC: Quad-core Broadcom BCM2837
- CPU: 1.2 GHz 64-bit quad-core ARM Cortex-A53
- RAM: 1GB
- Rede: 10/100

Os três dispositivos Raspberry usaram o sistema operacional Raspbian Jessie (Raspbian, 2015), com o kernel Linux raspberrypi 4.4.50-v7+ armv7l.

4.4.2 Configuração de software

Conforme citado anteriormente, a metodologia aplicada sugere configurar os dispositivos conforme manual provido pelos fabricantes. Desta forma, todos os sistemas operacionais e *softwares* envolvidos, foram instalados e configurados seguindo as instruções presentes nos *sites* oficiais.

O Docker (Docker Inc., 2017) foi instalado nos três dispositivos Raspberry. A versão usada nos experimentos foi a 17.05, sendo que sua instalação se tornou bastante simples desde que foi lançado oficialmente para arquitetura ARM. Antes, instalar o Docker em um dispositivo ARM era um processo bastante manual, precisando compilar o Docker a partir do código fonte. Fazer isso em um dispositivo limitado em processamento poderia demorar horas. Atualmente, um script automatizado está sendo mantido pelo projeto Docker, bastando executar o comando "curl -sSL https://get.docker.com | sh" e seguir as instruções apresentadas.

Como *switch* SDN e responsável pela criação dos túneis virtuais, foi usado o Open vSwitch 2.3.0 (The Linux Foundation, 2014) sendo que está disponível já compilado no repositório de pacotes oficiais dos sistemas operacionais usados. O OVS é usado para prover interconectividade entre os contêineres, permitindo o controle avançado do fluxo de pacotes. Maiores detalhes das configurações de rede serão apresentados na próxima seção.

Para as medições de rede foi usado o IPerf 2.0.5 (Dougan et al., 2010) e o pacote IPutils s20121221 (The Linux Foundation, 2012), ambos também disponíveis nos repositórios oficiais dos sistemas operacionais. O *software* de *benchmark* usado para os testes de CPU e memória foi o Sysbench 0.4.12 (Kopytov, A. 2009).

Para todos os testes onde se indica que foi executado um *software* rodando dentro de um contêiner, foi usado como base a imagem de sistema operacional "resin/rpi-raspbian:jessie-20160831", disponível no repositório oficial do Docker.

4.4.2 Arquitetura da rede estudada

Em uma implementação padrão do Docker, todas as interfaces virtuais dos contêineres ficam atreladas a uma *bridge*, normalmente chamada docker0, que é a entrada para a rede de contêineres que é gerenciada pelo Docker. Esta *bridge* é conectada diretamente na interface física do dispositivo, conforme demonstrado na Figura 4.4.

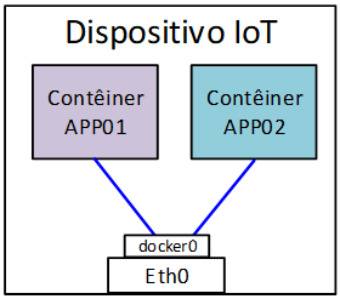

Figura 4.4 - Conexões da interface de rede de contêineres

De modo geral, uma *bridge* de rede garante conectividade entre todas as entidades de rede conectadas a ela. Porém esta configuração pode expor todo o tráfego entre um contêiner e outro, bem como a interface física. Desta forma, propõe-se o uso do Open vSwitch, atrelando as interfaces virtuais de cada contêiner em uma *bridge* do *switch* virtual, provendo isolamento para o tráfego entre diferentes contêineres.

Assim, o OVS permitirá a criação de túneis entre diferentes dispositivos rodando o Docker, colocando os contêineres de aplicação em uma mesma rede isolada e permitindo um controle avançado do fluxo de pacotes. A Figura 4.5 busca ilustrar as conexões de rede neste cenário.

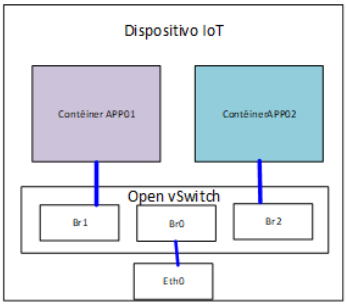

Figura 4.5 - Conexões de rede de contêineres usando um *switch* virtual

Em muitas aplicações IoT, o dispositivo sensor/atuador fica localizado remotamente, normalmente conectados usando enlaces de provedores de serviços. O Open vSwitch provê suporte nativo a uma série de protocolos de tunelamento, possibilitando levar a mesma rede do *data center*, para o dispositivo IoT usando redes *overlay*. Isso permite assegurar a conectividade entre todos os contêineres de uma aplicação, independente do *host* onde estão rodando.

A Figura 4.6 exemplifica uma infraestrutura à qual se deseja caracterizar com os experimentos.

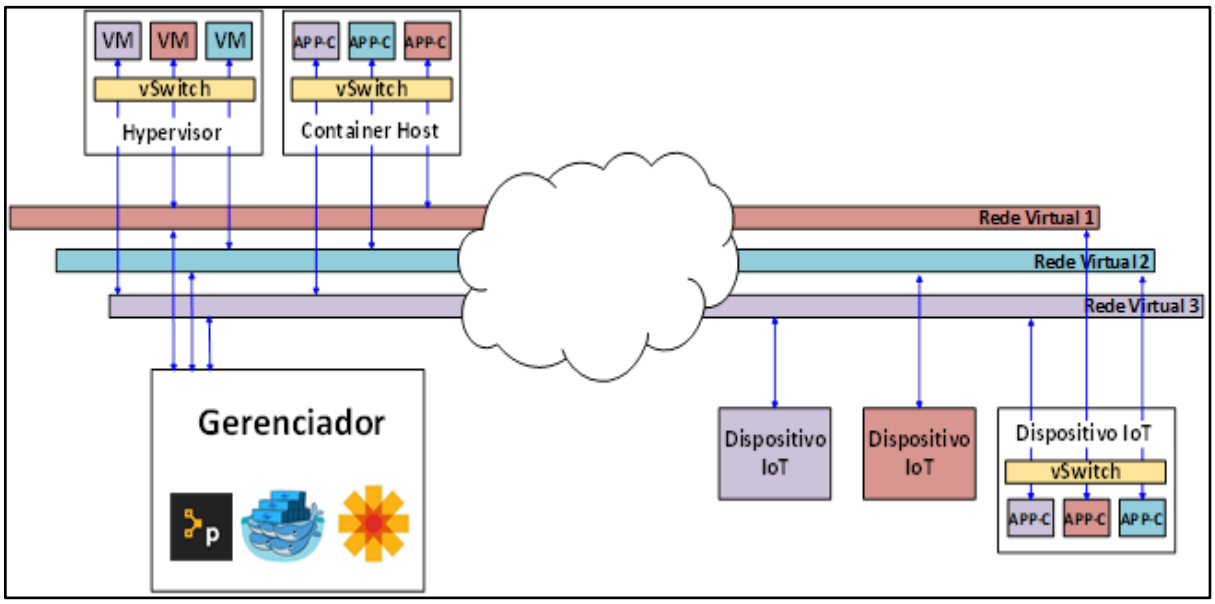

Figura 4.6 - Cenário representado pelo ambiente IoT estudado

4.4.3 Configuração de rede nas medições

Para as medições, as interfaces de rede dos dispositivos testados foram conectadas em uma rede ponto-a-ponto, conforme sugerido pela metodologia adotada e exposto no item 4.4.1. Contudo, os experimentos visam avaliar a sobrecarga em cada camada adicional introduzida no ambiente estudado. A Figura 4.7 demonstra o fluxo de informação dentro de um dispositivo IoT com a pilha de camadas introduzidas com a configuração estudada.

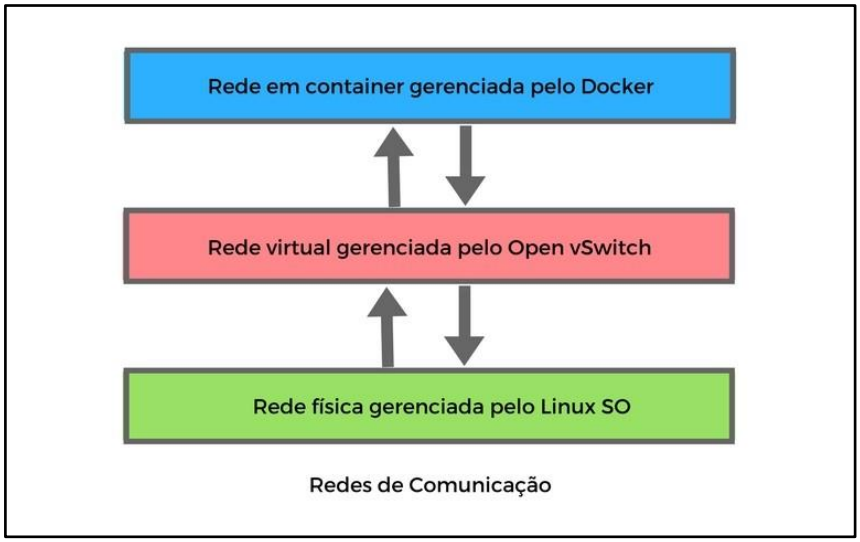

Figura 4.7 - Pilha de camadas e fluxo de informação em um dispositivo IoT

Em um ambiente real, os pacotes enviados pelo testador passam pela interface física do dispositivo. Já no dispositivo, a interface de rede encaminha os pacotes para o ambiente virtual criado pelo Open vSwitch. Finalmente o pacote é transmitido para a rede de contêineres no topo da pilha de camadas.

Desta forma, trazendo as considerações da forma de conexão de medição dos testes, que foram apresentadas na seção 4.4.1, para a arquitetura que está sendo avaliada, temos a conexão ilustrada na Figura 4.8.

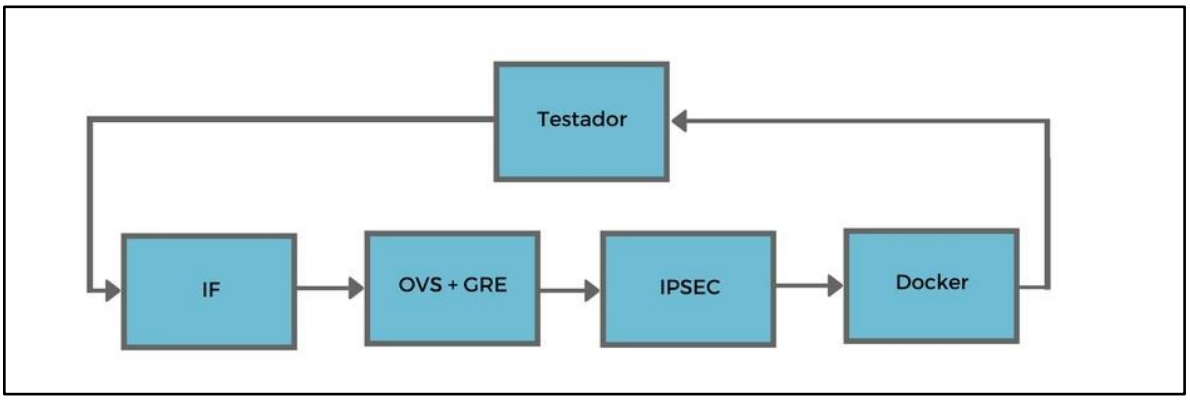

Figura 4.8 - Conexões para testes em série conforme a arquitetura estudada

Conforme exposto anteriormente, em um formato de medição ponto-a-ponto, com as medições sendo feitas com múltiplos dispositivos conectados em cascata, os resultados serão agregados como se fossem um sistema único. Esta medição sozinha, não refletirá a sobrecarga

exercida por cada camada, mas sim do sistema como um todo. Para mitigar esta sobrecarga, também devem ser executadas medições considerando as conexões intermediárias. Cada uma dessas medições, além de servir para caracterizar a implementação completa da arquitetura em estudo, também pode representar diferentes cenários de conexões rede de infraestruturas IoT.

A Figura 4.9 representa a conexão para a primeira série de medições realizadas.

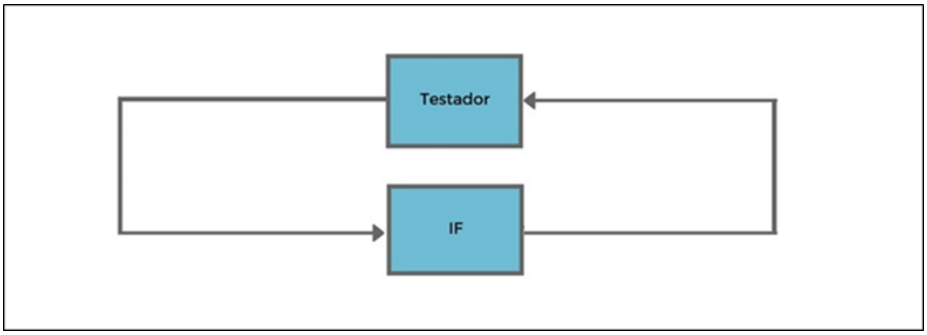

Figura 4.9 - Medição na interface física

Todas as medições foram feitas apenas na interface física Ethernet do dispositivo, uma vez que os dispositivos RPI 1 e 2 não possuem interface sem fio integrada. Para o uso em aplicações sem fio, os projetistas utilizam dispositivos extras, normalmente conectados na interface USB do Raspberry. Desta forma, podem variar de modelo e apresentar características diferentes, portanto, os resultados não poderiam ser generalizados.

A medição na interface física representa uma implantação IoT com a cenário exemplificado na Figura 4.10.

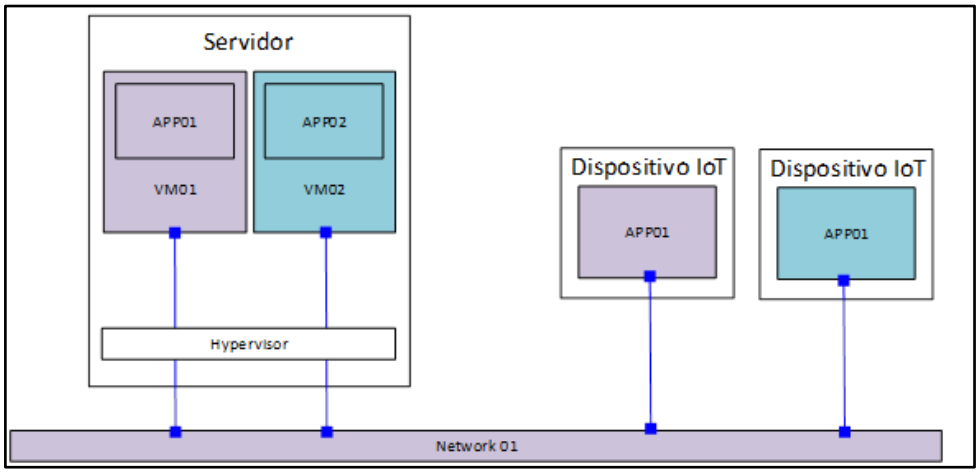

Figura 4.10 - Cenário correspondente às medições na interface física

O segundo ponto de medição foi feito na interface GRE no túnel de rede virtual criado pelo Open vSwtich, conforme mostrado na Figura 4.11.

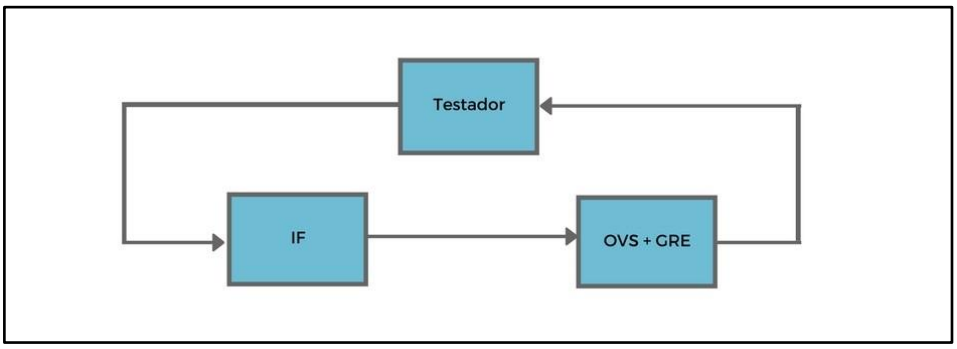

Figura 4.11 - Medição na interface GRE

A introdução desta rede *overlay* pelo *switch* virtual, pode representar o centário da Figura 4.12.

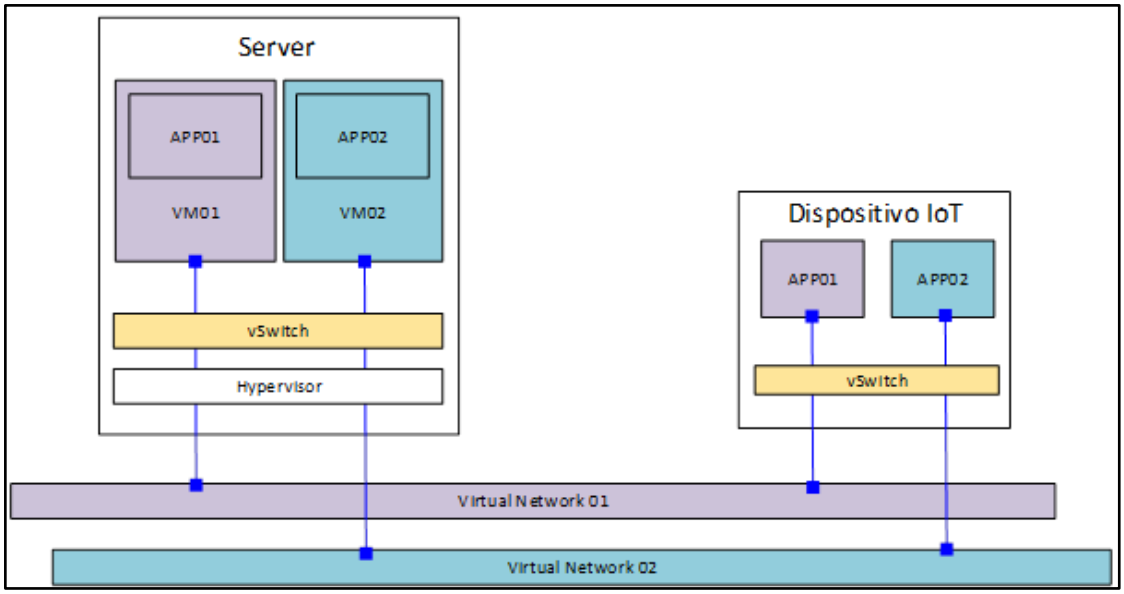

Figura 4.12 - Cenário correspondente às medições na interface GRE

A terceira medição foi feita na interface IPsec do Open vSwitch, conforme a Figura 4.13, sendo que o cenário correspondente é apresentado na Figura 4.14.

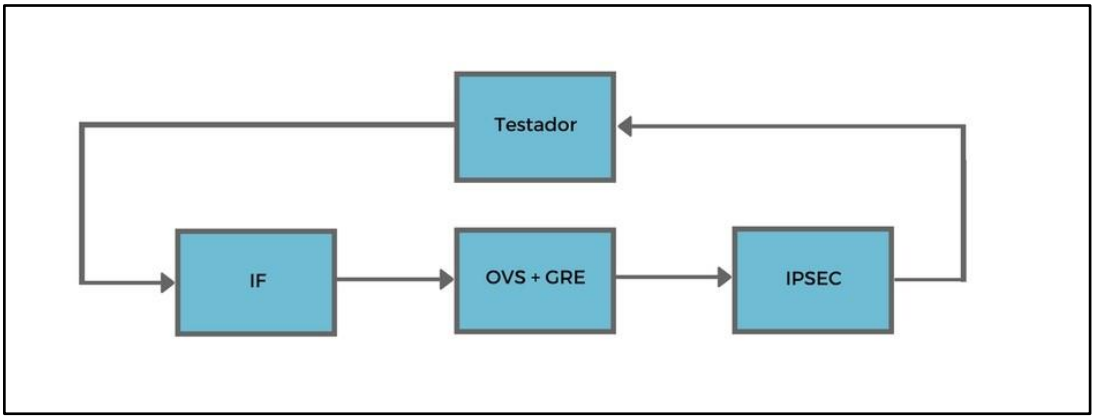

Figura 4.13 - Medição na interface IPsec

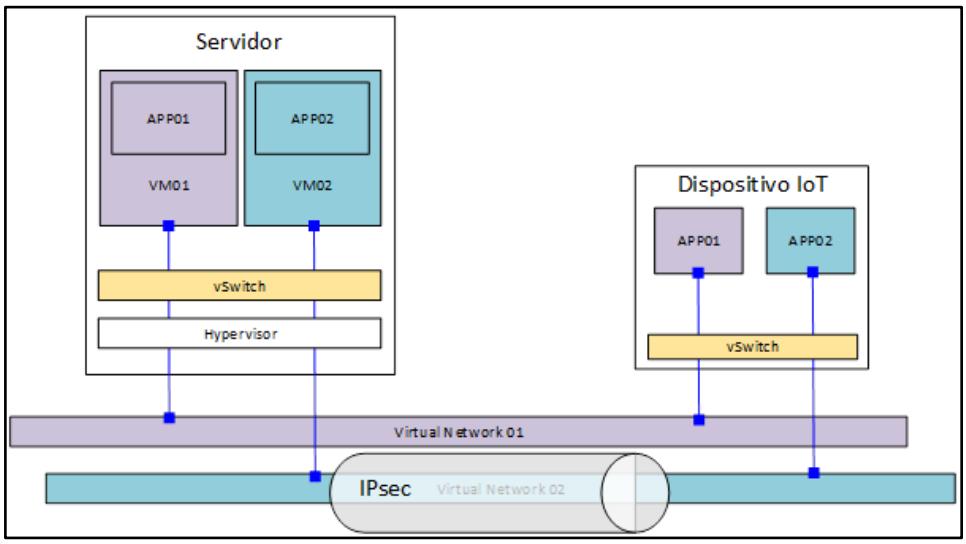

Figura 4.14 - Cenário correspondente às medições na interface IPsec

Finalmente, foi analisado o desempenho de rede na última camada do modelo estudado, com o ponto de medição na interface virtual criada pelo Docker e conectada ao testador através do túnel GRE/IPsec, conforme já exemplificado na Figura 4.8. O cenário caracterizado por esta medição é representado pela Figura 4.15.

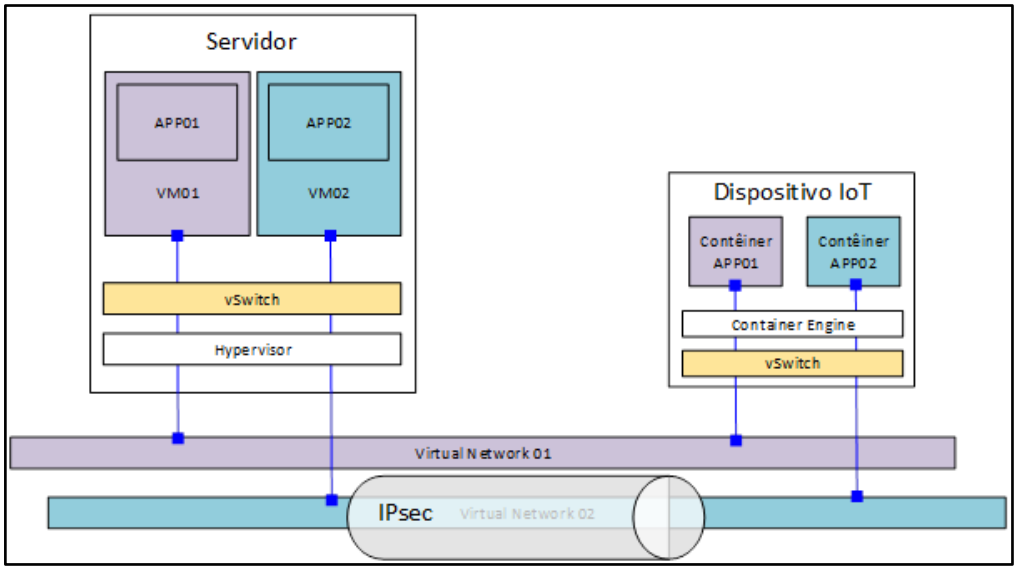

Figura 4.15 - Cenário correspondente às medições na interface Docker

5

AVALIAÇÃO DO DESEMPENHO

Este capítulo descreve os experimentos realizados expondo e discutindo os resultados obtidos. Os testes seguiram a metodologia e ambiente descritos no Capítulo 4.

5.1 MEDIÇÕES REALIZADAS

O objetivo dos experimentos foi caracterizar o comportamento de possíveis cenários em uma implementação real da arquitetura estudada. A maior ênfase foi dada nos testes de rede, de modo que com os resultados obtidos, seja possível identificar se condições como latência e taxa de transferência podem ser satisfeitas em um cenário IoT real. Buscou-se entender a diferença de desempenho de rede existente entre a interface física, e aplicando as camadas adicionais.

Como descrito no Capítulo 4, o Docker introduz interfaces virtuais que permitem a conexão entre contêineres e a rede física. Embora seja possível atribuir todos os contêineres a uma única *bridge* conectada à interface física, a arquitetura apresentada faz uso de uma camada extra usando um *switch* virtual, compatível com SDN. Isso permite um melhor gerenciamento do tráfego de rede, bem como a possibilidade de se usar redes virtuais e criptografia, isolando o tráfego em redes públicas. Para o tunelamento foi usado o protocolo GRE (*Generic Routing Encapsulation*), que se mostrou ser o mais adotado de acordo com as documentações encontradas, bem como o único método de tunelamento com suporte nativo ao IPSec pelo Open vSwitch. O custo em se adicionar estes novos recursos, é a necessidade extra de processamento, o que pode ocasionar perturbações no desempenho de rede. Os experimentos descritos nas próximas seções, buscam caracterizar tais sobrecargas.

As medições foram realizadas usando uma conexão ponto a ponto entre o “dispositivo IoT em teste” e um “dispositivo testador”. Foram abrangidos testes na interface física dos dispositivos, para obter-se os valores de referência, bem como nas interfaces das três camadas adicionais introduzidas pela arquitetura estudada. O método de testar cada camada, tem o objetivo de entender qual delas representa a maior sobrecarga acrescentada pela arquitetura, de acordo com as métricas de rede estipuladas.

Inicialmente, foram avaliadas as métricas de latência, vazão e *jitter*. Como os testes foram feitos em ambiente controlado, a métrica de perda de pacotes não apresentou resultado significativo, portanto, não foi considerada neste relatório. Uma consideração importante na utilização de contêineres em dispositivos IoT, é o tempo que um contêiner leva para ser iniciado. Embora esta métrica possa não estar relacionada diretamente ao desempenho de rede, poderá impactar fortemente por exemplo, no atraso fim a fim para disponibilização de um dado coletado por um sensor. Se houver a necessidade de inicializar um contêiner para esta operação, será introduzido atraso no sistema. De modo a validar estudos apresentados por outros autores, também foi avaliado o desempenho de processamento e memória usando o *software* de *benchmark* Sysbench.

5.2 LATÊNCIA

Este teste refere-se aos atrasos incididos durante a transmissão dos dados que trafegam na rede. Um dos tipos de atraso, pode ser causado em decorrência do tempo de propagação que o sinal leva para chegar ao destino, sendo dependente das propriedades físicas do meio de transmissão. Porém, também existem atrasos em decorrência de tarefas de processamento e enfileiramento dos pacotes sendo transmitidos, os quais se pretende medir com este experimento.

Para obter o valor de referência, que indica a latência mínima que o dispositivo pode atingir no cenário estudado, primeiramente a latência foi medida entre as interfaces físicas do dispositivo a ser testado e o testador. Para o teste, foi usado o comando “ping” do pacote IPutils. Foi usada a configuração do tamanho de pacote padrão (64 bytes).

Os resultados dos valores de referência obtidos estão expressos na Tabela 5.1:

Tabela 5.1 - Latência na Interface física (ms)

	Mínimo	Médio	Máximo	Desvio Padrão
RPI 1	1,146	1,331	1,481	0,088
RPI 2	0,968	1,194	1,341	0,124
RPI 3	0,860	1,081	1,366	0,109

Após o valor de referência ser medido, foram executados os testes nas demais camadas inseridas pelo modelo estudado, medindo a latência em cada interface.

Desta forma, o tunelamento GRE foi ativado entre duas interfaces virtuais do Open vSwitch. A latência foi então medida entre os pontos finais do túnel, cujos resultados estão dispostos na Tabela 5.2.

Tabela 5.2 - Latência na interface GRE (ms)

	Mínimo	Médio	Máximo	Desvio Padrão
RPI 1	1,397	1,773	2,377	0,185
RPI 2	1,201	1,478	1,831	0,169
RPI 3	0,931	1,346	1,457	0,125

Em seguida, a camada de criptografia foi adicionada, usando um túnel configurado como GRE/IPsec no Open vSwitch. Os resultados obtidos nesta medição estão expressos na Tabela 5.3.

Tabela 5.3 - Latência na interface IPsec (ms)

	Mínimo	Médio	Máximo	Desvio Padrão
RPI 1	2,028	2,353	2,602	0,129
RPI 2	1,704	1,917	2,277	0,128
RPI 3	1,187	1,574	1,707	0,131

Finalmente, um contêiner Docker foi ativado e sua interface virtual conectada ao ponto final do túnel GRE/IPsec, tendo a latência medida. A Tabela 5.4 demonstra os resultados desta medição.

Tabela 5.4 - Latência na interface Docker (ms)

	Mínimo	Médio	Máximo	Desvio Padrão
RPI 1	2,297	2,692	2,844	0,152
RPI 2	2,012	2,156	2,272	0,107
RPI 3	1,721	1,893	2,023	0,082

Com os dados obtidos nas medições, foram gerados os gráficos apresentados nas Figura 5.1 e Figura 5.2. Estes gráficos demonstram claramente a proporção do atraso introduzido em cada camada. O primeiro gráfico exibe a porcentagem em relação ao valor da latência total medida em cada dispositivo, enquanto que o segundo exibe os valores absolutos da latência em cada camada. A intensão com isto, é exibir de forma ilustrativa os resultados, podendo facilmente observar qual camada foi responsável por adicionar mais latência ao modelo estudado.

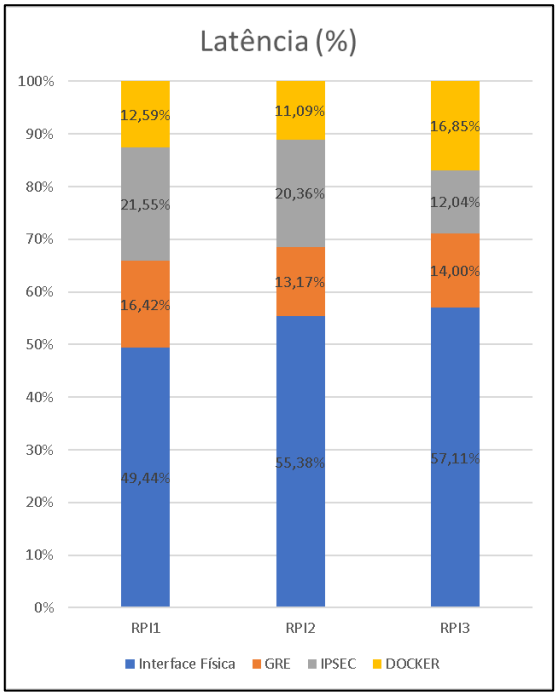

Figura 5.1 - Representação percentual em relação a latência total.

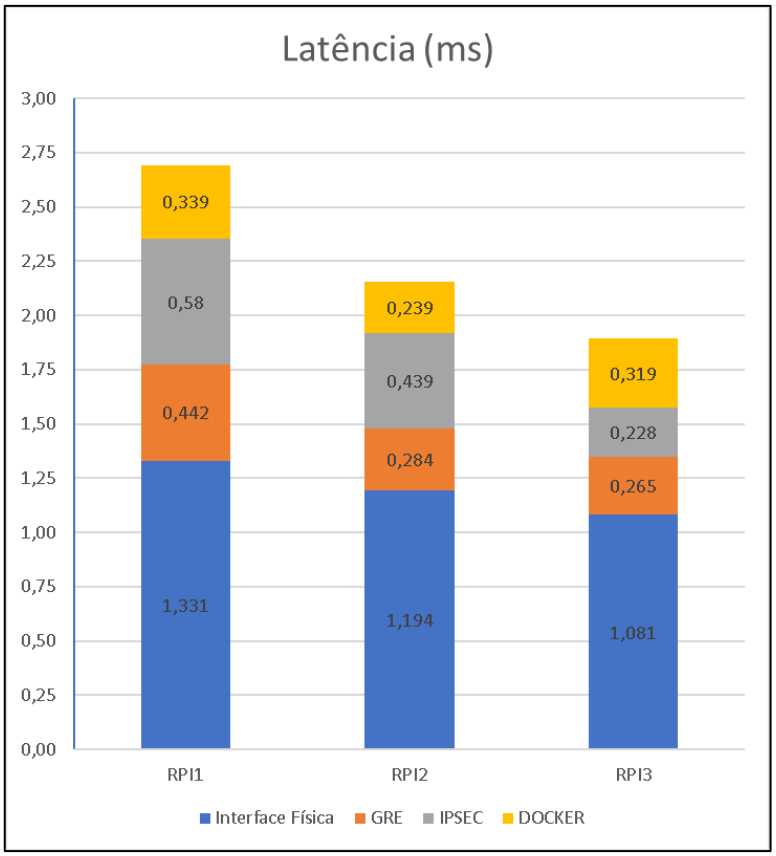

Figura 5.2 - Componentes da latência em cada interface (ms)

Pode-se perceber através dos gráficos, que a latência na interface física é responsável pela maior parte do atraso no ambiente estudado. Entretanto, objetivamos mensurar os atrasos adicionais.

Observando o gráfico percentual, a camada IPsec é a maior responsável pelo aumento da latência, cerca de 20% do total, nos dispositivos RPI 1 e 2. Entretanto, no Raspberry Pi 3 ocorreu justamente o contrário. A latência na interface IPsec foi a menor dentre as 4 camadas, sendo a interface do Docker, a maior responsável pelo atraso adicional no sistema.

Bradner e McQuaid (1999) recomendam executar testes utilizando vários tamanhos de pacotes. Isto é necessário para se obter uma caracterização completa do desempenho do dispositivo. Os pontos sugeridos para Ethernet são 64, 128, 256, 512, 1024, 1280, 1518. Porém, nesta medição também foram considerados pontos intermediários a estes, de modo a aumentar a resolução dos resultados, e com isso observar mais claramente o comportamento do sistema.

A Figura 5.3, Figura 5.4 e Figura 5.5, representam os gráficos formados a partir dos resultados obtidos nesta medição. Para este teste também foi usado o comando “ping” do pacote

IPutils do Linux. Para se obter o conjunto de dados necessários para gerar o gráfico, foi criado um *script* que testa a latência média de cada ponto, considerando vinte amostras.

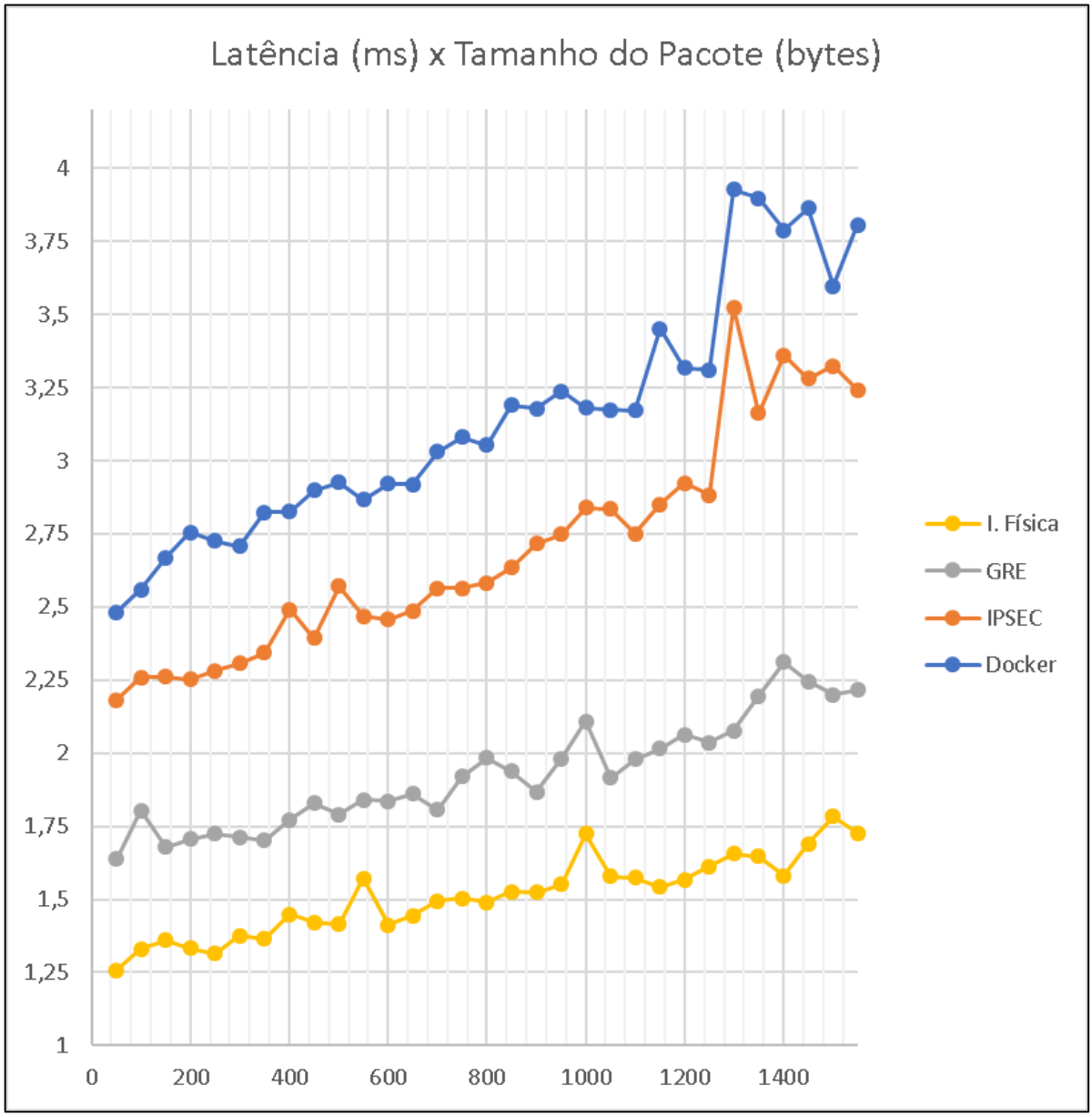

Figura 5.3 - Latência x Tamanho do pacote (RPI 1)

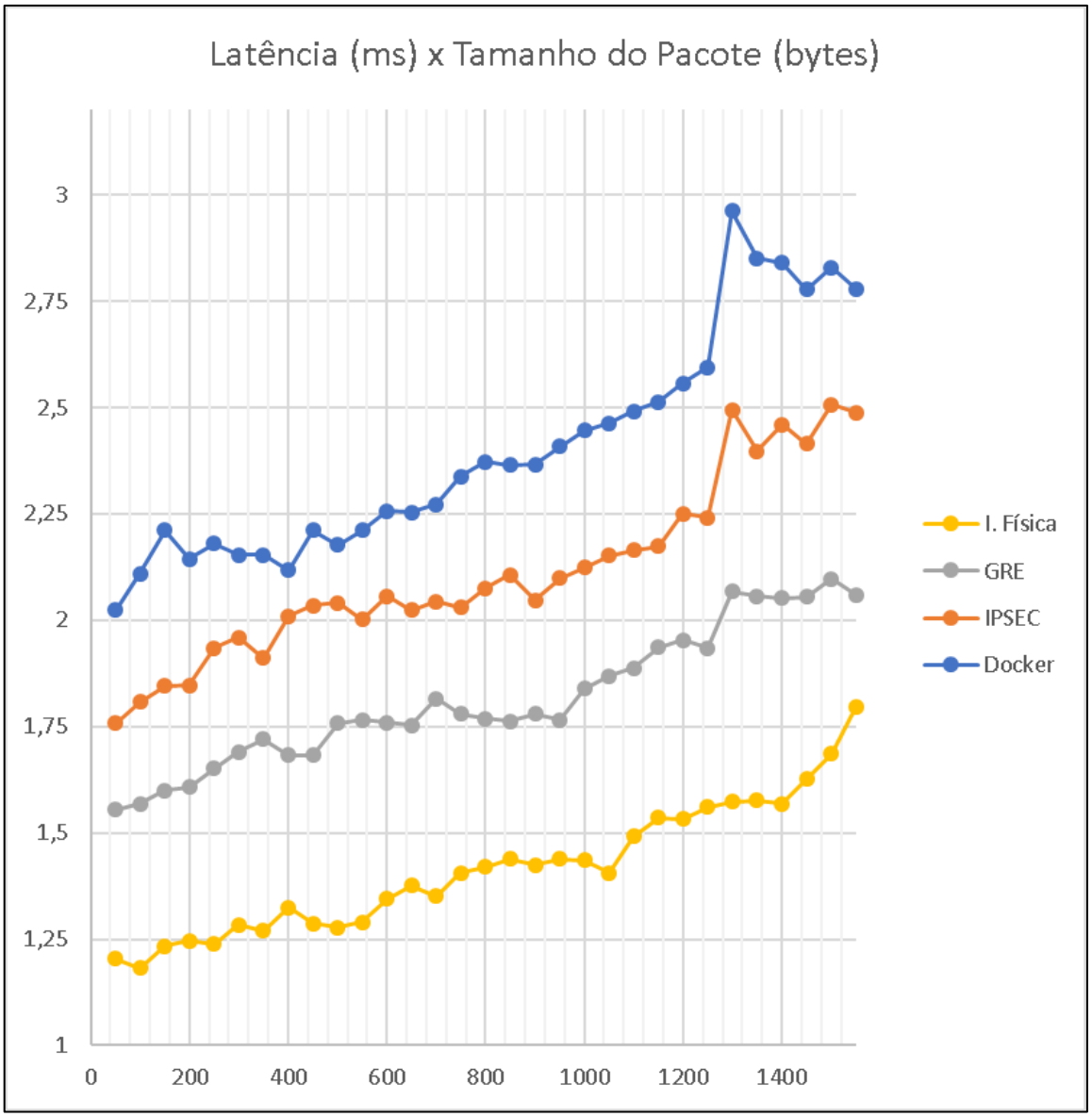

Figura 5.4 - Latência x Tamanho do pacote (RPI 2)

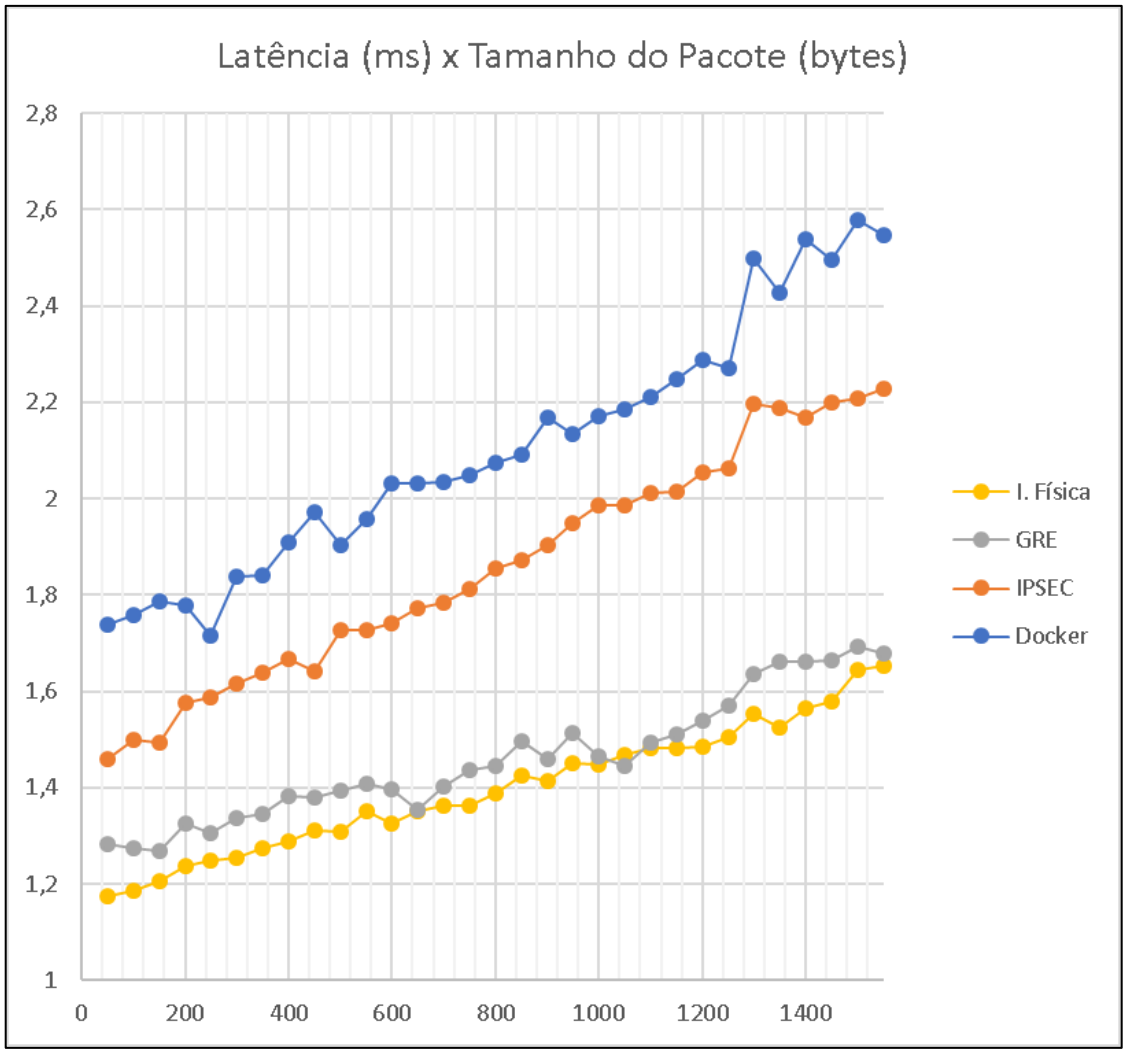

Figura 5.5 - Latência x Tamanho do Pacote (RPI 3)

Conforme já afirmado anteriormente, Bradner e McQuaid (1999) sugerem que os testes envolvam tamanhos de pacotes diferentes. No caso de divergências entre o tamanho de MTU em diferentes interfaces, o teste deve levar em consideração um intervalo de tamanhos que inclua o maior MTU do sistema. Com isso é possível perceber alguns aspectos relevantes sobre o sistema.

Observando os gráficos, nas interfaces físicas, pode-se notar um crescimento quase linear da latência, conforme o tamanho do pacote transmitido é aumentado. Entretanto, nas camadas adicionais, principalmente IPsec e Docker, observa-se um salto abrupto na latência de pacotes com tamanho acima de 1300 Bytes. Isso ocorre devido à discrepância entre os MTUs da interface física e da interface GRE, que deve ter seu tamanho reduzido para que pacotes encapsulados possam ser transmitidos em um frame Ethernet com MTU padrão.

5.3 VAZÃO DE DADOS

Este teste busca demonstrar a taxa de transferência efetiva dos dispositivos, no ambiente estudado. Nem sempre a taxa de transferência fica próxima ao valor máximo teórico informado na documentação do dispositivo. Dependerá também de vários fatores, como a capacidade do enlace de transmissão, perdas e atrasos no sistema, bem como do poder de processamento do dispositivo.

Os testes de vazão foram feitos usando a ferramenta IPerf, com suas configurações padrão, exceto o tempo de duração do teste, que foi alterado para 60 segundos, conforme metodologia adotada. Para se obter o valor de referência, que indica a vazão máxima que dispositivo pode atingir no cenário estudado, primeiramente foi medida a vazão entre as interfaces físicas do dispositivo em teste e o testador. Em seguida, foram medidas as interfaces das interfaces GRE, IPsec e Docker.

Foram conduzidos testes unidirecionais, onde o dispositivo IoT foi responsável por transmitir ou receber o fluxo de pacotes, e também testes bidirecionais, onde transmissão e recepção ocorrem simultaneamente.

Os testes de vazão, para recepção de dados, apresentaram os resultados expostos na Tabela 5.5, sendo graficamente ilustrados na Figura 5.6.

Tabela 5.5 - Vazão de recepção (Mbps)

RX	I. Física	GRE	IPSEC	Docker
RPI 1	69,9	51,1	16,5	9,73
RPI 2	94,1	91,4	52,9	43,9
RPI 3	95,2	91,6	84,1	73,4

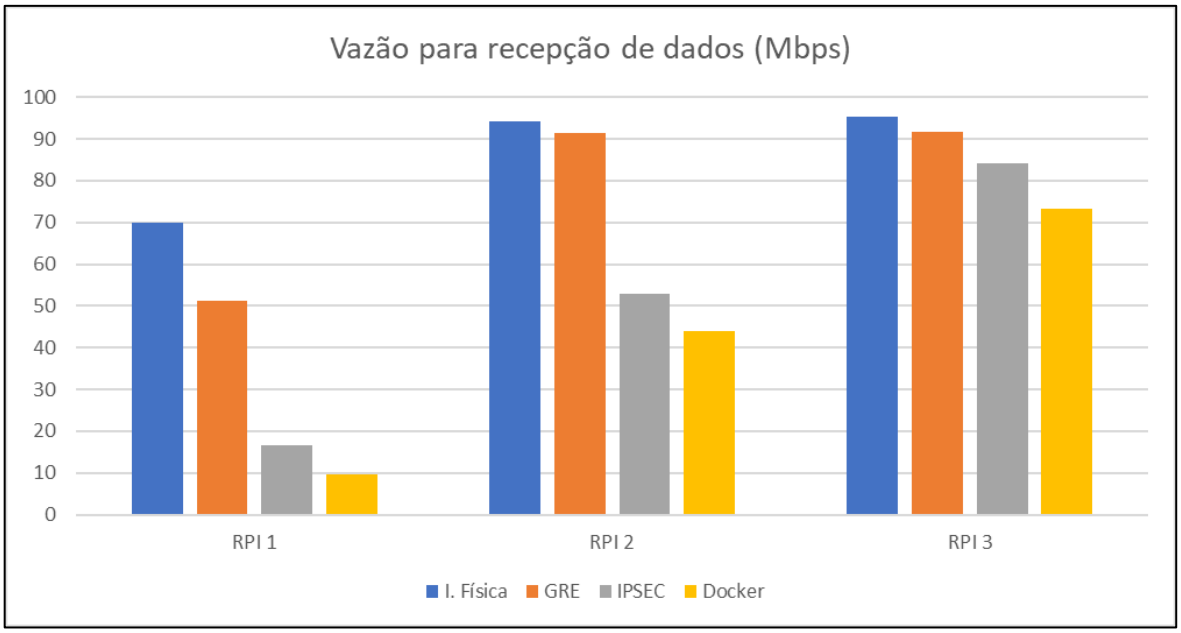

Figura 5.6 - Vazão de recepção (Mbps)

Os testes demonstram que o RPI 1 apresenta um desempenho abaixo do esperado até mesmo na interface física. Enquanto o máximo teórico informado é 100Mpbs, as medições demonstraram uma capacidade de apenas 70% deste valor, com um valor de vazão medido de 69.9Mbps. O RPI 2 e 3 apresentaram um resultado esperado, ficando próximo ao valor máximo teórico, com uma perda de apenas 5%.

Na interface GRE o desempenho do RPI 1 ficou em torno de 70% do valor medido na interface física. O desempenho dos RPI 2 e 3 foram semelhantes, com uma queda de desempenho na ordem de 5% em relação ao valor da interface física.

A interface IPsec apresentou-se como maior gargalo da arquitetura estudada. O RPI 3 ainda conseguiu apresentar boa vazão para recepção, mantendo um desempenho em torno de 88% do valor medido na interface física. O RPI 2 teve um valor medido de 56% da taxa de referência, e o RPI 1 por sua vez, teve uma queda abrupta de desempenho, mantendo 23% da vazão em relação à interface física.

Considerando todas as camadas da arquitetura estudada (GRE+IPsec+Docker), para recepção de dados, o RPI 1 manteve apenas 13% da vazão de referência medida na interface física. O RPI 2 manteve 46% enquanto que o RPI 3 manteve 77%.

Os testes de vazão, para transmissão de dados estão expostos na Tabela 5.6 e Figura 5.7.

Tabela 5.6 - Vazão de transmissão (Mbps)

TX	I. Física	GRE	IPSEC	Docker
RPI 1	45,7	45,7	16,7	11,7
RPI 2	94,2	90,7	43,2	42,0
RPI 3	95,1	91,7	83,9	81,7

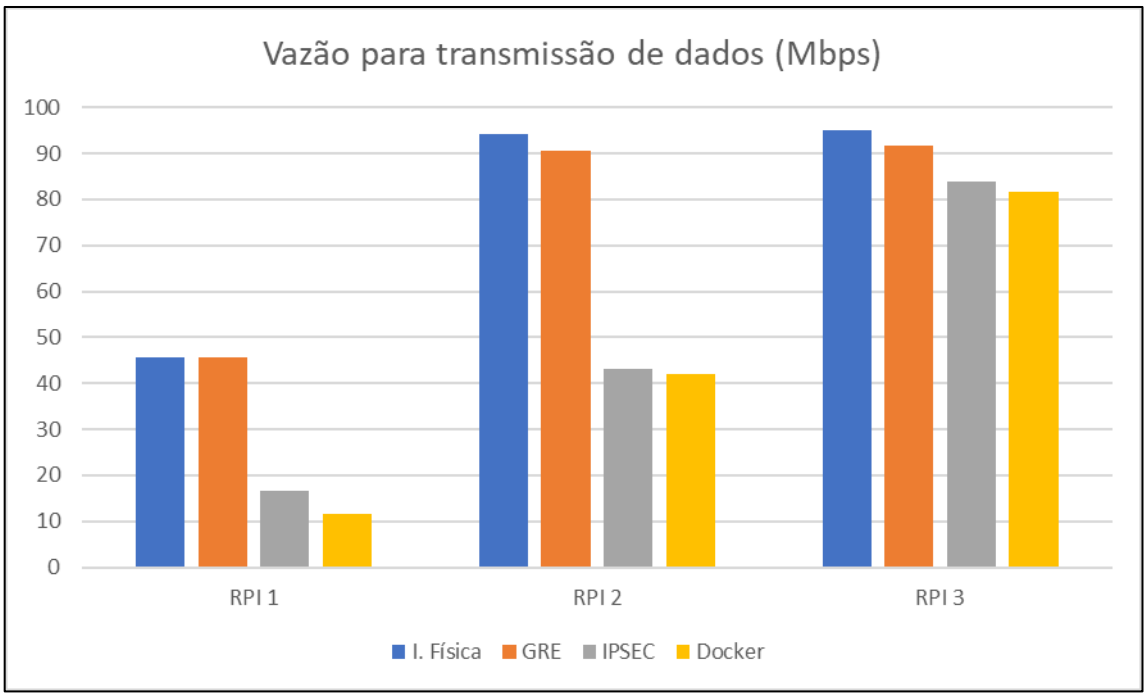

Figura 5.7 - Vazão de transmissão (Mbps)

Os resultados obtidos na medição da vazão de transmissão foram bastante semelhantes aos testes de recepção. Os destaques a serem comentados são que o RPI 1 apresentou um valor ainda mais baixo na interface física, apenas 45% do valor teórico, e o RPI 2 apresentou 20% menos vazão na interface IPsec, em comparação com os testes de recepção.

Outro resultado relevante é que o RPI 1 e 3 apresentaram uma vazão de transmissão um na ordem de 10% maior do que a medida para recepção, na interface GRE+IPSEC+Docker. As demais medições tiveram resultados praticamente iguais.

O teste de vazão bidirecional indica a capacidade do dispositivo em transmitir e receber dados simultaneamente. A Tabela 5.7 e Figura 5.8 apresentam os resultados obtidos nesta medição.

Tabela 5.7 - Vazão bidirecional (Mbps)

	I. Física	GRE	IPSEC	Docker
RPI 1 RX	49,4	45,3	14,9	4,4
RPI 2 RX	91,4	87,9	41,1	29,4
RPI 3 RX	91,4	88,5	78,4	62,7
RPI 1 TX	9,25	7,44	4,7	1,17
RPI 2 TX	50,9	43,6	28,0	9,2
RPI 3 TX	56,2	43,1	27,3	12,9

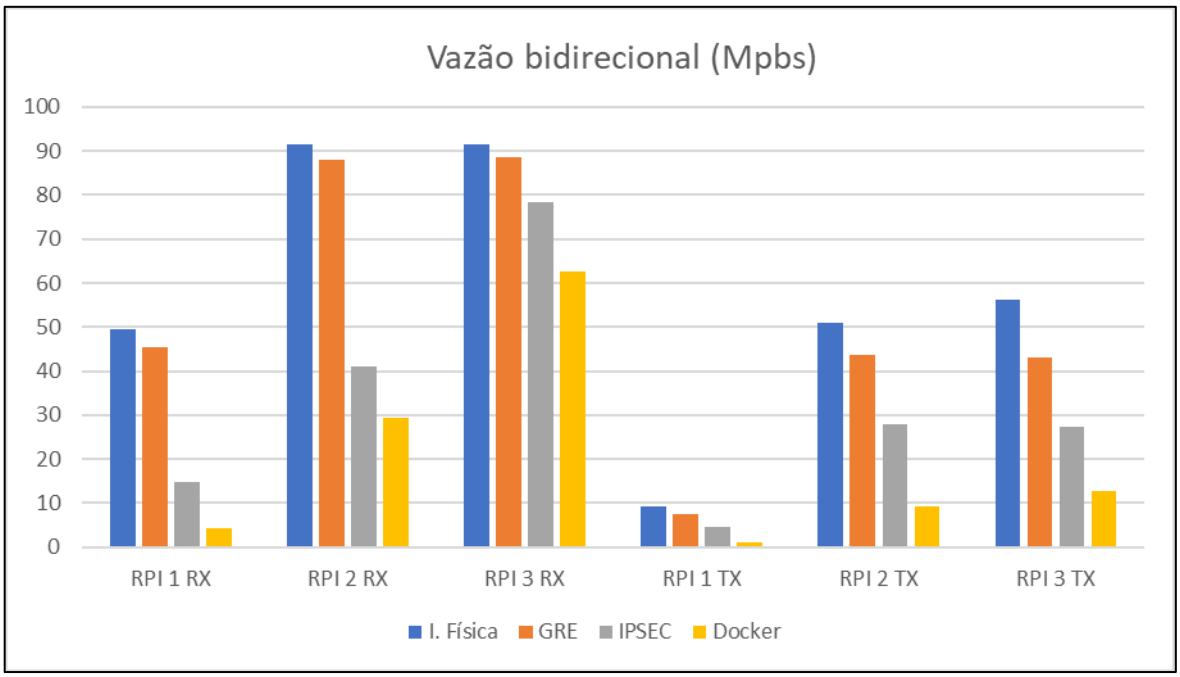

Figura 5.8 - Vazão bidirecional (Mpbs)

As medições bidirecionais indicam que os dispositivos tiveram seu desempenho prejudicado principalmente para transmissão de dados.

Os valores medidos na camada física indicam que o RPI 3 e 2 mantiveram mais de 95% da vazão de recepção, enquanto que a transmissão caiu para quase metade dos valores medidos nos testes em apenas uma via. O RPI 1 apresentou uma vazão de 70% e 20% para transmissão e recepção respectivamente, em relação aos testes unidirecionais.

Na interface GRE, novamente resultados semelhantes, o RPI 2 e 3 mantiveram cerca de 95% do desempenho para recepção, e em torno de 48% para transmissão. O RPI 1 manteve 88% para recepção e 16% para transmissão.

Quanto à interface IPsec, o RPI 1, 2 e 3 mantiveram 90%, 77% e 93%, respectivamente, do valor medido no teste unidirecional para recepção, e 28%, 64% e 32% para transmissão.

A medição no topo da pilha, representada pela interface do contêiner Docker, a vazão medida manteve 45%, 66% e 85% da taxa de transmissão e 10%, 21% e 15% da taxa de recepção, para o RPI 1, 2 e 3, respectivamente.

5.4 *JITTER*

Basicamente, o *jitter* é definido como a variação no atraso do recebimento de pacotes. Se o transmissor enviar pacotes em uma cadência contínua e com espaçamento igual, devido ao congestionamento da rede, enfileiramento de pacotes ou erros de configuração, o tempo entre o recebimento dos pacotes pode variar em vez de permanecer constante. O *jitter* não necessariamente está relacionado ao valor absoluto da latência. Pode-se ter um tempo de resposta alto, mas um baixo *jitter*. Esta métrica é importante em sistemas onde os pacotes devem chegar em um ritmo regular ao destino, como por exemplo, aplicações de voz.

Nesta medição, os resultados foram obtidos gerando um fluxo de pacotes UDP com o IPerf. Os resultados obtidos estão expressos na

Tabela 5.8 e na Figura 5.9. A indicação TX ou RX refere-se que o dispositivo foi responsável por enviar (TX) ou receber (RX) o fluxo gerado.

Tabela 5.8 - Jitter

	I. FISICA	**GRE**	**GRE+IPSEC**	**GRE+IPSEC+DOCKER**
RPI 1 TX	0,058	0,116	0,142	0,205
RPI 1 RX	0,056	0,090	0,073	0,100
RPI 2 TX	0,046	0,126	0,156	0,139
RPI 2 RX	0,036	0,038	0,056	0,042
RPI 3 TX	0,035	0,070	0,067	0,082
RPI 3 RX	0,030	0,025	0,040	0,059

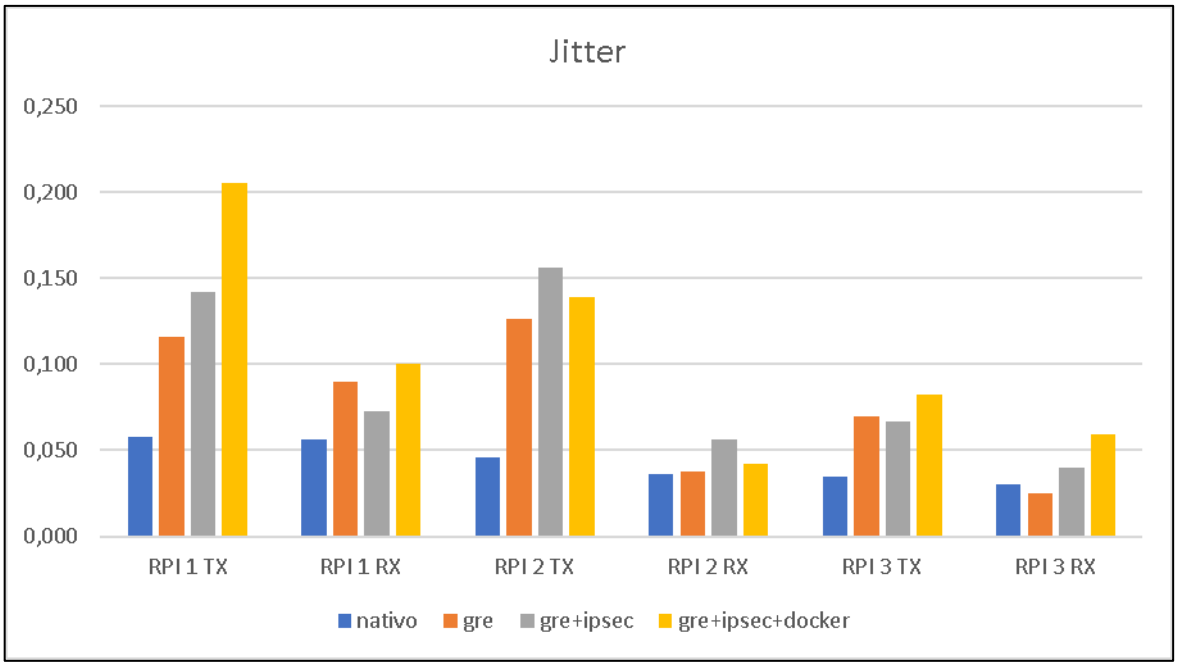

Figura 5.9 - Jitter

Os resultados evidenciam que o aumento mais visível nas medições de *jitter*, ocorre no tunelamento GRE, principalmente quando o dispositivo é responsável pelo envio do fluxo de pacotes. Nas camadas superiores (IPsec e Docker), o *jitter* continua aumentando no RPI 1, a cada camada adicionada. Por outro lado, no RPI 2 e 3, o *jitter* permanece praticamente estável nas camadas IPsec e Docker.

Observa-se que o Raspberry Pi 3 é o que apresenta menor *jitter* em todas as camadas. Com isso, conclui-se que a capacidade de processamento influencia não somente na latência, mas também, causa variações no atraso dos pacotes transmitidos e recebidos, caracterizando o aumento no *jitter*.

5.5 TEMPO DE INICIALIZAÇÃO DO CONTÊINER

Pode-se relacionar este tempo, com o tempo de *boot* de uma VM tradicional. Em aplicações de sensoriamento e automação, isso pode impactar diretamente por exemplo, no instante de tempo que o dado será coletado, ou que um atuador será ativado. Se o *software* não estiver rodando previamente, o contêiner terá que ser iniciado, podendo ocasionar atrasos na

coleta do sensor ou na ativação de comandos para um servomecanismo acoplado ao dispositivo IoT.

Para mensurar este efeito, foi executada a biblioteca WiringPI (Wiring Pi, 2016) dentro de um contêiner. Esta biblioteca possui um utilitário em linha de comando que pode ser usado para configurar e programar os pinos da interface GPIO (*General Purpose Input Output*), podendo ser executado através de scripts *shell*. Em seguida, foi calculado o tempo necessário para executar um programa básico que faz uso da porta GPIO. Foram executados comandos de leitura e escrita dos pinos. Os resultados das medições foram obtidos invocando os comandos de inicialização do contêiner juntamente com o comando "time" do Linux.

Este experimento pode ser relacionado com o tempo mínimo para ativação de um recurso, como um atuador ou leitura de um sensor, caso o contêiner não esteja em execução. Os tempos médios coletados para iniciar o contêiner e efetuar uma operação de leitura e escrita estão descritos na Tabela 5.9. Percebeu-se que com o contêiner já carregado, o tempo de acesso à porta GPIO foi praticamente em tempo real, tendo os mesmos resultados do acesso nativo.

Tabela 5.9 - Tempo de inicialização do contêiner (s)

	Teste de Escrita	**Teste de Leitura**
RPI 1	1,295	1,338
RPI 2	1,784	1,860
RPI 3	4,130	4,422

A Figura 5.10 representa graficamente os resultados obtidos.

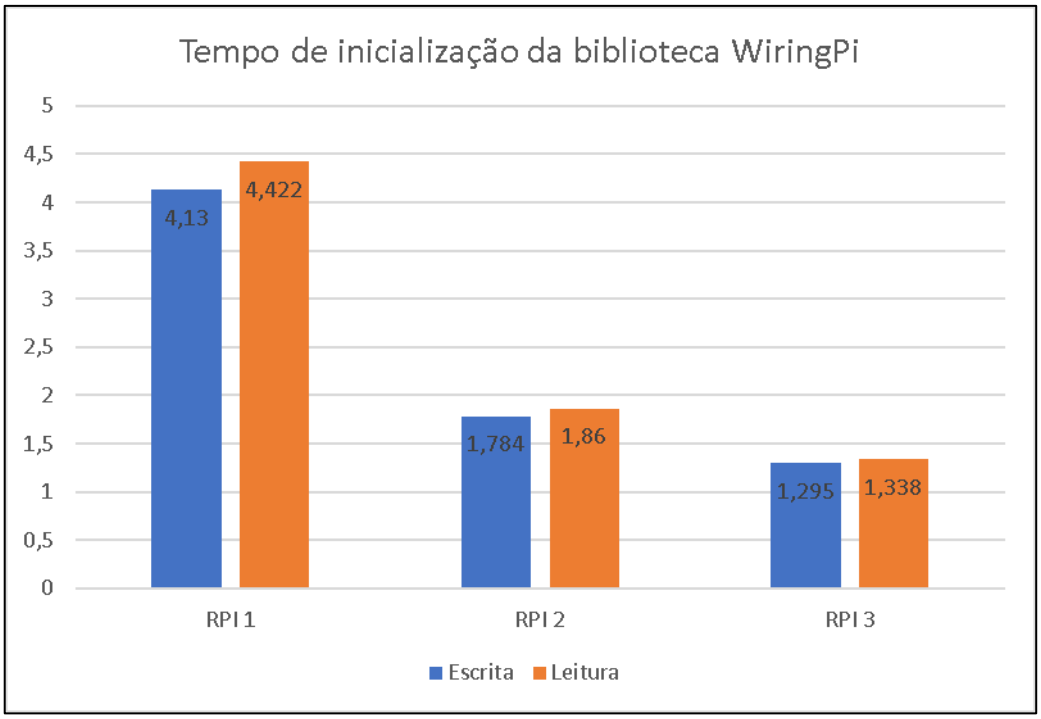

Figura 5.10 - Tempo de inicialização do Contêiner (s)

Em soluções de virtualização mais complexas, por exemplo hipervisores, a inicialização de uma VM pode impactar fortemente no tempo para ativação de um serviço, uma vez que *boot* de uma VM pode durar minutos. Os resultados apresentados demonstram um baixo tempo de ativação, menos de 1,5 segundo no dispositivo mais recente estudado. Considerando todos os resultados, o tempo de inicialização dos contêineres ainda foi abaixo de 5 segundos, o que ainda é rápido comparando com o *boot* de uma VM tradicional, ainda mais considerando as capacidades de *hardware* de um Raspberry Pi 1.

5.6 PROCESSAMENTO

Para estes testes foi escolhida a ferramenta Sysbench (2009) na versão 0.4.12. O Sysbench é um *benchmark* sintético que possui testes para avaliar CPU, memória, I/O de disco, e até mesmo recursos de bancos de dados.

Quando é rodado usando o teste de CPU, basicamente, o Sysbench irá verificar o tempo levado para encontrar números primos até o máximo de operações informado na inicialização do teste. Em cada operação, é feita a divisão padrão do número a ser calculado,

por todos os números existentes entre 2 e a raiz quadrada do número calculado. Se qualquer destas divisões restar 0, o próximo número é calculado. O teste pode ser executado com *threads* simultâneas, de modo a usar mais de um núcleo do processador avaliado. Como tratam-se de operações matemáticas básicas, este teste coloca algum estresse no CPU, podendo oferecer uma ideia de seu comportamento.

Os testes foram executados calculando o tempo de execução para 5000 operações. Como os Raspberry Pi 2 e 3 possuem processadores de quatro núcleos, foram executados testes com até quatro *threads* simultâneos.

Os resultados obtidos estão expressos na Tabela 5.10 e Figura 5.11.

Tabela 5.10 - Tempo de execução Sysbench (s)

	Threads	Nativo	Docker
RPI 1	1	209,668	210,452
	2	209,467	209,032
	3	209,716	210,205
	4	209,657	210,434
RPI 2	1	111,934	111,974
	2	55,736	55,745
	3	37,236	37,451
	4	28,104	28,262
RPI 3	1	70,687	70,743
	2	34,940	34,960
	3	23,360	23,340
	4	17,666	18,190

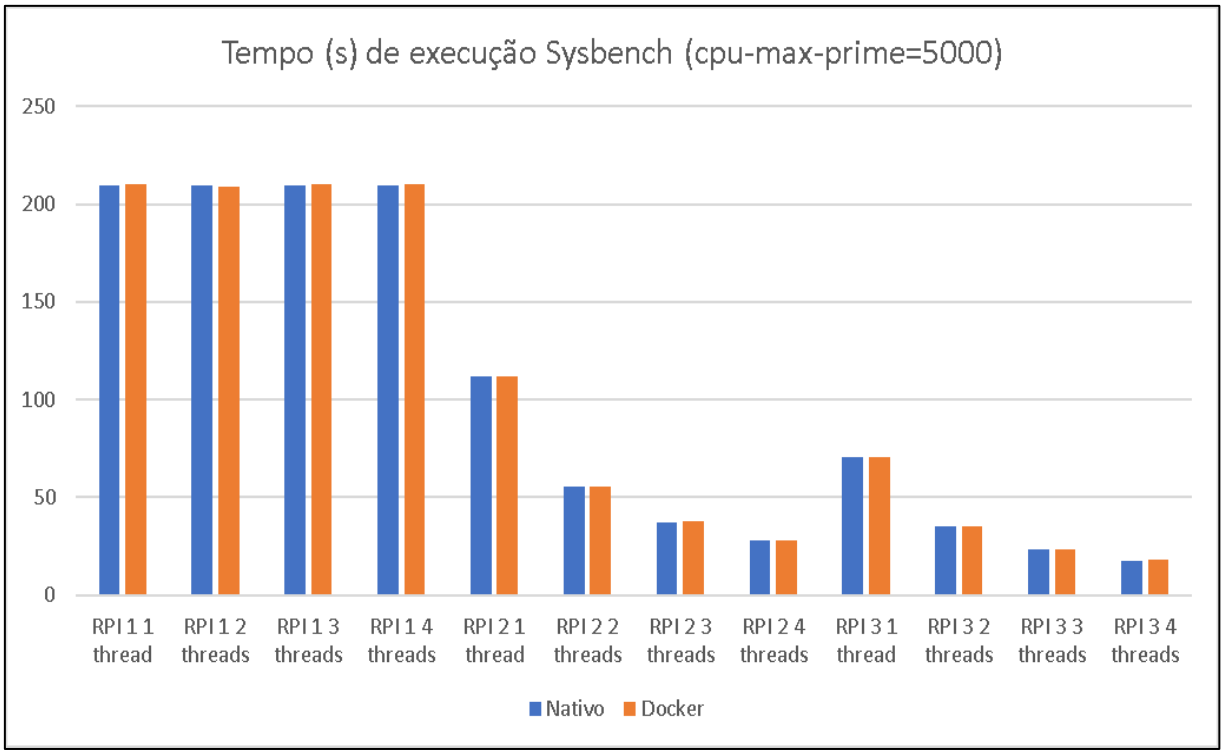

Figura 5.11 - Tempo de execução Sysbench (s)

Com os resultados obtidos, pode-se observar que é praticamente negligenciável a diferença no desempenho rodando o *benchmark* dentro de um contêiner ou nativamente, para um único thread. Como o processador do RPI1 possui apenas um núcleo, aumentar o número de *threads* não surte diferença no tempo de execução do teste.

Entretanto, para o Raspberry Pi 2 e 3, quando o número de *threads* é aumentado, o comportamento continua praticamente o mesmo do aplicativo rodando nativamente. É possível observar que os *threads* são distribuídos entre os núcleos do processador em uma maneira efetiva. Ou seja, o tempo de execução decresce proporcionalmente ao aumento do número de *threads*. O tempo de execução usando dois *threads* equivale a aproximadamente 1/2 do tempo gasto com um único thread, bem como o tempo para três *threads* equivale a 1/3, e assim sucessivamente, até que o número de *threads* seja igual ao número de núcleos.

Vários estudos relacionados a esta dissertação apresentaram testes semelhantes, buscando evidenciar algum possível *overhead* de processamento ocasionado pelo sistema de contêineres. Portanto os resultados já eram esperados. Ainda assim, o experimento foi conduzido de modo a verificar se o comportamento também se repetia no ambiente estudado, de modo a validar os resultados obtidos por outros autores.

Um diferencial das medições feitas neste experimento, é apresentar resultados para múltiplos *threads*, o que foi pouco explorado em outros trabalhos.

Cabe salientar, que *benchmarks* sintéticos geralmente testam apenas algumas operações do processador, podendo divergir dos resultados que uma aplicação real pode ter.

5.7 MEMÓRIA

Para avaliar a desempenho de leitura e escrita de memória, também foi usado o Sysbench. No modo de teste de memória, o Sysbench aloca um buffer, com tamanho determinado de acordo com parâmetro informado na inicialização do teste, e executa operações de escrita e leitura, de maneira sequencial ou randômica. Os testes foram executados com um tamanho total transferido de 2GB, em blocos no tamanho de 1KB.

Tabela 5.11 - Taxa de transferência da memória RAM (MB/s)

	Escrita sequencial	**Leitura sequencial**	**Escrita randomica**	**Leitura randomica**
RPI 1	96,69	143,19	116,22	127,87
RPI 1 Docker	96,76	142,17	116,72	127,23
RPI 2	195,90	238,38	220,10	227,30
RPI 2 Docker	195,26	235,89	219,82	226,01
RPI 3	315,98	373,00	351,37	355,14
RPI 3 Docker	316,80	372,92	351,64	355,96

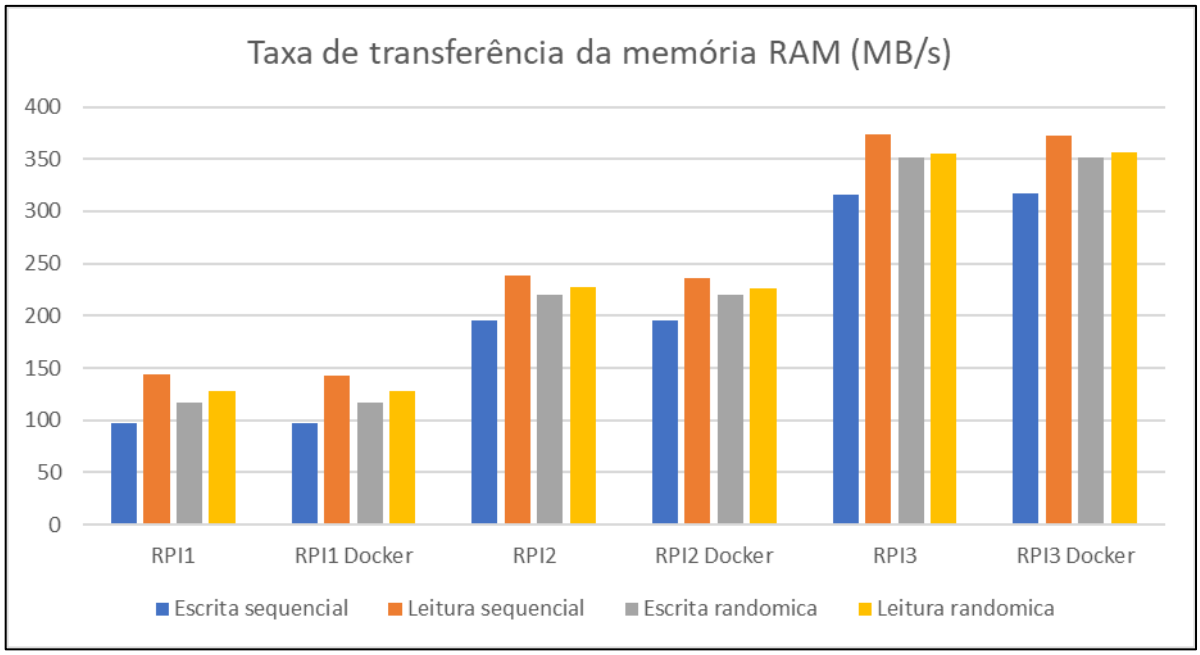

Figura 5.12 - Taxa de transferência da memória RAM (MB/s)

Assim como os resultados obtidos nos testes de processamento, os resultados para os testes de memória demonstraram desempenho muito parecido entre o *benchmark* rodando nativamente ou dentro de um contêiner Docker, conforme resultados exibidos na Tabela 5.11 e Figura 5.12.

6

CONCLUSÃO

Este capítulo apresenta as considerações finais em relação ao exposto nesta dissertação. Em seguida, busca-se elencar as contribuições obtidas com a realização deste trabalho, finalizando com sugestões para possíveis trabalhos futuros que possam se desdobrar a partir deste.

6.1 CONSIDERAÇÕES FINAIS

O objetivo principal deste trabalho foi avaliar se dispositivos comuns em IoT estão preparados para o uso conjunto de abordagens de virtualização por contêineres e virtualização de redes, e se o desempenho de rede destes dispositivos pode ser satisfatório em um ambiente real.

Para isso, foram levantados requerimentos para fomentar a base da implementação e testes subsequentes. Foi percebido que a virtualização baseada em contêineres é uma alternativa que demanda poucos recursos computacionais em relação aos tradicionais hipervisores, e consequentemente, pode ser adotada para aperfeiçoar o gerenciamento de recursos em IoT.

Neste contexto, foram apresentados uma arquitetura e um conjunto de *softwares* que podem ser usados em implementações IoT. Em seguida, foram conduzidos experimentos que podem servir de base para a caracterização do comportamento de rede desta solução, em relação ao desempenho nativo.

Os resultados mostram que o Docker como mecanismo de contêiner, provê uma interface de virtualização leve, preparada para o uso em dispositivos restritos, sem grandes

implicações no desempenho. O impacto em termos de desempenho do mecanismo de contêiner foi pequeno em relação às execuções nativas.

Em relação ao Open vSwitch, também se mostrou adequado ao uso em dispositivos IoT, trazendo flexibilidade de configuração de rede, carente em uma implementação pura do Docker. Ele oferece capacidade de segmentação do tráfego entre contêineres, usando configurações de *bridges* e túneis, bem como pode aplicar regras de filtragem e QoS dinâmicas usando um controlador SDN.

A criptografia embarcada do Open vSwitch ocasionou uma considerável perda de desempenho de rede em todos os dispositivos estudados, porém ofereceu ainda, vazão suficiente para cobrir muitas aplicações IoT. Particularmente, o maior impacto foi observado no Raspberry de primeira geração, sendo que na terceira geração, a vantagem em usar um protocolo sem criptografia, visando apenas aumento de desempenho e desconsiderando a segurança, passa a ser muito baixa. Isso revela a evolução na capacidade de processamento destes dispositivos, demonstrando que em gerações futuras, muito provavelmente, a sobrecarga introduzida pela criptografia, poderá ser negligenciada.

Em relação ao *hardware*, o Rapsberry PI 3 possui o melhor desempenho. Porém, a primeira geração, embora com certa limitação, foi capaz de rodar todos os testes.

De maneira geral, foram observados distúrbios em todas as métricas de rede estudadas, vazão, latência e *jitter*, de modo que devem ser ponderados os benefícios que a introdução destas novas camadas irão gerar na aplicação que está rodando no dispositivo. Portanto, para cada tipo de aplicação deve-se avaliar se a arquitetura em questão pode suprir seus requisitos.

Os parágrafos a seguir buscam fazer uma sumarização sobre as conclusões em relação às questões de pesquisa e objetivos estabelecidos no capítulo introdutório.

O objetivo principal da dissertação foi atingido pois se avaliou o desempenho do modelo selecionado no estudo de gerenciamento de recursos em dispositivos IoT, utilizando conceitos de virtualização e verificou-se a viabilidade através de medições de desempenho de rede. Para realizar o objetivo, foram selecionados *software* e *hardware* em conformidade com a proposta. Em seguida, foi implementado o ambiente de testes para avaliação de desempenho. O próximo passo foi a aferição das métricas de desempenho de rede nos diferentes tipos de *hardware*, e por último, a análise dos resultados do comportamento da arquitetura estudada quando aplicada em ambiente real.

A primeira questão levantada foi sobre a utilização de técnicas de virtualização por contêineres para o gerenciamento de recursos em dispositivos IoT. A resposta com o presente estudo é que sim, elas podem funcionar de maneira eficaz. Basicamente, capacidade de

memória e do processamento determinam se o *hardware* pode usar uma distribuição Linux embarcada, que suporte os *softwares* usados no ambiente estudado. Os experimentos demonstraram que a virtualização baseada em contêineres é bastante leve, não necessitando de capacidades muito superiores do que as requeridas para rodar o próprio Linux.

A segunda questão de pesquisa, em relação aos impactos no desempenho da rede quando utilizadas as soluções de virtualização e SDN, é respondida conduzindo experimentos apresentados no capítulo 5. Os experimentos buscaram fazer um comparativo do comportamento de todas as camadas de abstração introduzidas com a estrutura estudada, evidenciando as interferências no desempenho de rede. A introdução do *switch* virtual foi pouco significativa quando da utilização de protocolos sem criptografia para fazer o tunelamento da conexão das redes virtuais. Com a adição de uma camada IPsec, a degradação de desempenho ficou bastante evidente no Raspberry de primeira geração, atingindo uma taxa de transferência efetiva para recepção de dados em torno de 23% do valor medido como referência. A segunda geração apresentou desempenho superior, mas ainda em 56%. Já a terceira geração do dispositivo, mostrou desempenho aproximado de 88% do valor da taxa de referência. Mesmo com desempenho inferior no *hardware* mais restrito, a vazão resultante ainda é suficiente para muitas aplicações IoT. Tais resultados evidenciaram a evolução apresentada no *hardware* e demonstram uma real possibilidade de aplicação dos conceitos estudados.

Apesar da interferência no desempenho com as camadas adicionadas, o uso das tecnologias envolvidas proporciona benefícios adicionais, como maior facilidade no gerenciamento em implementações de novos *softwares*, maior automação no gerenciamento de recursos, maior escalabilidade das aplicações, isolamento e configuração dinâmica do tráfego de rede, e otimização no compartilhamento de recursos de *hardware*.

6.2 CONTRIBUIÇÕES

As principais contribuições atingidas com este trabalho foram:

- Exposição da possibilidade de manter a compatibilidade com soluções que fazem uso de virtualização e SDN, mesmo usando dispositivos restritos em desempenho.

- Um modelo para o gerenciamento de recursos, baseado em conceitos de contêineres e SDN, que permite a integração de redes de dispositivos IoT com aplicações em nuvem e datacenter.
- Uma implementação para testes, baseada na arquitetura proposta, que permite fazer uma avaliação do comportamento de aplicações rodando sobre esta infraestrutura.
- Uma avaliação caracterizando o comportamento de rede desta solução, em uma das principais famílias de dispositivos usados na prototipação de soluções IoT.
- Resultados experimentais corroborando com a possibilidade de aplicação real dos conceitos envolvidos.

6.3 TRABALHOS FUTUROS

De forma alguma a implementação apresentada neste trabalho contempla as infinitas possibilidades para esta área, nem que está adequada a todo tipo de *hardware* e aplicações. Portanto, maiores experimentações e investigações devem ser feitas em relação a este tópico. Esta seção busca discutir algumas sugestões de possíveis desdobramentos para este trabalho, as quais serão expostas nos parágrafos seguintes.

Acesso a dispositivos externos que usam interface USB, ou a própria interface GPIO do Raspberry, normalmente requer que o contêiner seja executado com privilégios elevados. Também, não está clara a interferência do sistema de contêineres no uso destas interfaces. Recomenda-se testes avaliando tanto a segurança, quanto o desempenho das interfaces físicas do *hardware* estudado.

Os contêineres usados neste trabalho foram montados usando imagens base do Raspbian, adicionando os utilitários necessários para os experimentos. Alguns experimentos foram feitos com imagens já disponíveis no repositório do Docker. Alguns recursos que possam estar habilitados por padrão nestas imagens base, podem causar certo impacto no desempenho. Quando se usa dispositivos com recursos computacionais limitados, os contêineres devem ter apenas o suficiente para que a aplicação funcione. O mesmo pode ser dito do sistema Linux hospedeiro. Esforços na otimização de um sistema específico para uso das soluções

apresentadas, e a conseguinte avaliação de seu desempenho, podem produzir resultados relevantes.

A exemplo, o CoreOS (CoreOS Inc., 2017) busca criar uma distribuição Linux específica para o uso como motor de contêineres, podendo-se fazer uma analogia aos hipervisores de VMs. Porém até o momento da conclusão deste trabalho, não foi percebido algum sistema operacional com esta abordagem, adequado ao uso com o *hardware* estudado. Cita-se a existência do HypriotOS (Hypriot, 2017), mantido por desenvolvedores que iniciaram a portabilidade do Docker para a plataforma Raspberry. Porém, este sistema é baseado no Raspbian, com as funcionalidades do Docker integradas por padrão, não tendo necessariamente, a mesma abordagem para desempenho do CoreOS.

O Open vSwitch possui compatibilidade com outros protocolos de tunelamento, como o VXLAN, STT, Geneve. Porém este trabalho limitou-se ao uso do GRE, por demonstrar-se ser o mais usado, sendo o único protocolo com compatibilidade nativa do Open vSwitch, a uma camada de criptografia. Testes envolvendo outros protocolos de tunelamento podem produzir resultados interessantes.

Recomenda-se a implementação e experimentos em outros tipos de *hardware* para ampliar o escopo do estudo, particularmente, em outras arquiteturas de CPU, como as plataformas x86 voltadas a IoT providas pela Intel.

REFERÊNCIAS

BIZANIS, N; KUIPERS, F. A. SDN and Virtualization Solutions for the Internet of Things: A Survey. **IEEE Access**, vol. 4, pp. 5591-5606, The Netherlands, 2016.

BRADNER, S.; MCQUAID, J. Benchmarking Methodology for Network Interconnect Devices. **RFC 2544**, March 1999.

BROWN, E. **Top 10 Open Source Linux and Android SBCs** (2014). Disponível em: <https://www.linux.com/news/top-10-open-source-linux-and-android-sbcs> Acesso em: 10/04/2017.

BROWN, E. **Top 10 Linux and Android Hacker SBCs of 2015** (2015). Disponível em: <https://www.linux.com/news/top-10-linux-and-android-hacker-sbcs-2015> Acesso em: 10/04/2017.

BROWN, E. **Top 10 Best Open-Spec Hacker SBCs** (2016). Disponível em: <https://www.linux.com/news/raspberry-pi-stays-top-survey-81-open-spec-sbcs> Acesso em: 10/04/2017.

CELESTI, A. et al. Exploring Container Virtualization in IoT Clouds**. IEEE International Conference on Smart Computing**, St. Louis, MO, pp. 1-6, 2016.

CELESTI, A. et al. Characterizing Cloud Federation in IoT. **2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)**, Crans-Montana, pp. 93-98, 2016.

COREOS INC. **CoreOS**, 2017. Disponível em: <https://coreos.com/> Acesso em 10/04/2017.

SOFTWARE IN THE PUBLIC INTEREST INC. **Debian 8**, 2015. Disponível em: <https://www.debian.org/> Acesso em 10/04/2017.

DOCKER INC. **Docker**, 2017. Disponível em <https://www.docker.com/> Acesso em 10/04/2017.

DREWANZ, D. **Resource Management as an Enabling Technology for Virtualization** (2012). Disponível em: <http://www.oracle.com/technetwork/articles/servers-storage-admin/resource-mgmt-for-virtualization-1890711.html> Acesso em: 10/04/2017.

GARRISON, S. **Understanding the differences between Software Defined Networking, network virtualization and Network Functions Virtualization** (2014). Disponível em: <https://www.networkworld.com/article/2174268/tech-primers/understanding-the-differences-between-software-defined-networking-network-virtualizati.html> Acesso em: 10/04/2017.

GRAUR, F. Dynamic network configuration in the Internet of Things. **5th International Symposium on Digital Forensic and Security** (ISDFS), Tirgu Mures, pp. 1-4, 2017.

HAN, S.; LEE, S. Implementing SDN and network-hypervisor based programmable network using Pi stack switch. **2015 International Conference on Information and Communication Technology Convergence (ICTC)**, Jeju, pp. 579-581, 2015.

HYPRIOT, 2017. **HypriotOS**. Disponível em: <https://blog.hypriot.com/> Acesso em 10/04/2017.

DOUGAN, J. et al. **IPerf**, 2010. Disponível em: <https://iperf.fr/> Acesso em 10/04/2017.

THE LINUX FOUNDATION. **IPutils**, 2012. Disponível em: <https://wiki.linuxfoundation.org/networking/iputils> Acesso em 10/04/2017.

JENNINGS, B.; STADLER, R. Resource Management in Clouds: Survey and Research Challenges. **Journal of Network and Systems Management**, v. 23, issue 3, New York, pp 567–619, 2015.

KARIM, M. B. A. et al. Extending Cloud Resources to the Edge: Possible Scenarios, Challenges, and Experiments. **2016 International Conference on Cloud Computing Research and Innovations** (ICCCRI), Singapore, pp. 78-85, 2016.

KIM, H.; KIM, J.; KO, Y. B. Developing a cost-effective OpenFlow testbed for small-scale Software Defined Networking, **16th International Conference on Advanced Communication Technology**, Pyeongchang, pp. 758-761, 2014.

KRYLOVSKIY A. Internet of Things gateways meet linux containers: Performance evaluation and discussion. **2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)**, Milan, pp. 222-227, 2015.

MANVI, S.; SHYAM, G.K. Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. **Journal of Network and Computer Applications**, v. 41, Pages 424-440, Oklahoma, 2014.

MORABITO, R. A performance evaluation of container technologies on Internet of Things devices. **2016 IEEE Conference on Computer Communications Workshops** (INFOCOM WKSHPS), San Francisco, CA, pp. 999-1000, 2016.

MORABITO, R. Virtualization on Internet of Things Edge Devices with Container Technologies: A Performance Evaluation. **IEEE Access**, vol. 5, pp. 8835-8850, Finland, 2017.

THE LINUX FOUNDATION. **Open vSwitch**, 2014. Disponível em: <http://openvswitch.org/> Acesso em 10/04/2017.

RASPBIAN. **Raspbian Jessie**, 2015. Disponível em: <https://www.raspbian.org/> Acesso em 10/04/2017.

SAHOO, J.; MOHAPATRA, S.; LATH, R. Virtualization: A Survey on Concepts, Taxonomy and Associated Security Issues. **2010 Second International Conference on Computer and Network Technology**, Bangkok, pp. 222-226, 2010.

KOPYTOV, A. **Sysbench**, 2009. Disponível em: <https://github.com/akopytov/sysbench)> Acesso em 10/04/2017.

TSCHOFENIG, H.; ARKKO, J.; THALER, D.; MCPHERSON, D. Architectural Considerations in Smart Object Networking, **RFC 7452**, March 2015.

TSO, F. P. et al. The Glasgow Raspberry Pi Cloud: A Scale Model for Cloud Computing Infrastructures. **2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops**, Philadelphia, PA, pp. 108-112, 2013.

WIRING PI, **Wiring Pi**, 2016. Disponível em: <http://wiringpi.com/> Acesso em 10/04/2017.

XAVIER, M.G. et al. Performance Evaluation of Container-Based Virtualization for High Desempenho Computing Environments, 2013 **21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing**, Belfast, 2013, pp. 233-240.