UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES

MATEUS MANTOAN DE ALMEIDA

INTEGRAÇÃO DE FERRAMENTAS PARA UM FIREWALL MODULAR E ESCALÁVEL COM USO DE CONTAINERS E COMPUTADORES DE PLACA ÚNICA

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2021

MATEUS MANTOAN DE ALMEIDA

INTEGRAÇÃO DE FERRAMENTAS PARA UM FIREWALL MODULAR E ESCALÁVEL COM USO DE CONTAINERS E COMPUTADORES DE PLACA ÚNICA

Trabalho de Conclusão de Curso apresentado ao Curso Superior de Tecnologia em Sistemas de Telecomunicações, do Departamento Acadêmico de Eletrônica (DAELN), da Universidade Tecnológica Federal do Paraná (UTFPR), como requisito parcial para obtenção do Diploma de Tecnólogo em Sistemas de Telecomunicações .

Orientador: Prof. MSc. Christian Carlos de Souza Mendes

CURITIBA

2021

MATEUS MANTOAN DE ALMEIDA

INTEGRAÇÃO DE FERRAMENTAS PARA UM FIREWALL MODULAR E ESCALÁVEL COM USO DE CONTAINERS E COMPUTADORES DE PLACA ÚNICA

Trabalho de Conclusão do Curso Superior de Tecnologia em Sistemas de Telecomunicações apresentado como requisito para obtenção do título de Tecnólogo em Sistemas de Telecomunicações da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 07 de Junho de 2021

_____

Nome Daniel Fernando Pigatto
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Marcos Talau
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Leandro Batista de Almeida
Doutorado
Universidade Tecnológica Federal do Paraná

“A Folha de Aprovação assinada encontra-se na Coordenação do Curso”

CURITIBA

2021

AGRADECIMENTOS

Inicialmente agradeço as pessoas que estão comigo nesse caminho, Thaineh Souza, que me segurou nos momentos de stress, seja apenas ficando do meu lado ou fazendo uma garrafa de café para que continuasse em frente. Meus amigos Kaique, Rafael, Lucas e Felype do qual compartilhei, mesmo que a distância, todas as etapas desse processo universitário. E também minha mãe e meu irmão, que mesmo após todas as desistências anteriores ainda acreditaram que um dia seria possível.

Sobre as desistências, agradeço as faculdades que iniciei antes de conhecer a UTFPR e ao fato de ter desistido de seguir com elas, esses erros no percurso que fizeram eu chegar na cadeira da qual hoje me orgulho de ocupar. Também agradecendo a instituição UTFPR que consegue se destacar entre as outras universidades por se focar na excelência técnica.

Agradeço especificamente a duas empresas onde trabalhei, primeiro a TIM Brasil, que despertou a atenção para as Telecomunicações. E agora a Furukawa Electric, que me levou a ir além do conhecimento dos protocolos e sistemas do mercado, me dando a oportunidade de desenvolver os equipamentos que irão a campo em diversos lugares do mundo.

Em especial, e provavelmente o principal motivo de estar aqui, preciso agradecer a toda a comunidade de Software Livre, todos os fóruns, livros e entusiastas que conheci, permitindo ter me focado no aprendizado de Linux desde cedo. Em especial Ian Murdock, criador do Debian que foi a base do meu aprendizado. Linus Torvalds, que mostrou ao mundo que a qualidade técnica é mais importante do que as inúmeras discussões filosóficas que tomam o meio. E por fim ao Gleydson Mazioli da Silva que escreveu o Guia Foca Linux, sem dúvida a maior fonte de conhecimento em Linux do Brasil.

O maior bem do homem é uma mente inquieta
(Isaac Asimov)

RESUMO

ALMEIDA, Mateus Mantoan de. **Integração De Ferramentas Para Um Firewall Modular E Escalável Com Uso De Containers E Computadores De Placa Única**. 2021. 83 f. Trabalho de conclusão de curso. Tecnologia em Sistemas de Telecomunicações, Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

Firewall é o nome atribuído a um sistema de segurança para uso em redes de computadores, e possuir um *firewall* eficiente é essencial para qualquer instituição, entretanto existem poucas opções de baixo custo no mercado. Atualmente os conceitos de alta disponibilidade, escalabilidade e opções de agregação de hardware para adequar a demanda de processos e modularizar sistemas, são bastante discutidos no mercado de Tecnologia da Informação. Partindo destes princípios é interessante que se faça um estudo para a integração de ferramentas para a disponibilização de um *firewall* de baixo custo com base em serviços *open source*, usando conceitos de modularização de software e hardware e alta disponibilidade. Este trabalho tem como objetivo apresentar conceitos e ferramentas que permitam a disponibilização de um *firewall* de baixo custo, modular e com compatibilidade a ferramentas de segurança *open source*, para atendimento a pequenas e médias empresas ou instituições, sendo aplicável em um ambiente real. Os resultados apresentados demonstram que, com um investimento baixo, o sistema pode realizar a análise e tratamento do tráfego, bem como executar as aplicações escolhidas, sendo bastante eficiente em manter o sistema operando quando há perda funcional de parte do sistema. Tendo ao fim um baixo custo de hardware e custo zero a nível de licenças de software.

Palavras-chave: Firewall. alta-disponibilidade. open-source. redes de computadores. containers.

ABSTRACT

ALMEIDA, Mateus Mantoan de. **Service Integration For A Modular And Scalable Firewall With The Use Of Containers And Single Board Computers**. 2021. 83 f. Undergraduate thesis. Technology in Telecommunications Systems, Academic Department of Electronics, Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

Firewall is the name given to security systems found in computer networks, and having an efficient firewall is essential for any institution, however there are few low-cost options on the market. Currently, the concepts of high availability, scalability and hardware aggregation options to suit the demand for processes and modularize systems are widely discussed in the Information Technology market. Based on these principles, it is interesting to carry out a study for the integration of services for the provision of a low-cost firewall based on open source tools, using concepts of software and hardware modularization and high availability. This work aims to present concepts and tools that allow the provision of a low-cost, modular firewall that is compatible with open source security tools, to serve small and medium-sized companies or institutions, being applicable in a real environment. The results presented demonstrate that, with a low investment, the system can perform the analysis and treatment of traffic, as well as execute the chosen applications, being very efficient in keeping the system operating when there is functional loss of part of the system. Having at the end a low cost of hardware and zero cost in terms of software licenses.

Keywords: Firewall. high-availability. open-source. computer network. containers.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

ARM	Advanced RISC Machine
BASH	Bourne-Again SHell
CPU	Central Process Unit
DNS	Domain Name System
GB	Giba Byte
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
IPS	Internet Protocol Suite
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LAN	Local Area Network
MAC	Media Access Control
MB	Mega Byte
Mb	Megabit
MS	MiliSegundo
NAT	Network Access Translation
PING	Packet Internet Network Grouper
PPP	Point-to-Point Protocol
RAM	Random Access Memory
RFC	Request For Comments
RJ45	Registered Jack 45
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell

SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TI	Tecnologia da Informação
UDP	User Datagram Protocol
UFPR	Universidade Federal do Paraná
USB	Universal Serial Bus
VLAN	Virtual Local Area Network
WAN	Wide Area Network
WEB	World Wide Web

SUMÁRIO

1 INTRODUÇÃO

O conceito de informação é normalmente relacionado à manipulação de conhecimento, como por exemplo a noção jurídica onde “se possuem informações de um caso” ou em exemplos de onde o conhecimento é transmitido, “passando informações”. Tendo em vista as aplicações possíveis, a informação acaba por ser um conjunto de dados que pode ser tanto transmitido quanto manipulado (SEMIDÃO, 2014).

O conceito de informatização se iniciou em meio à Segunda Guerra Mundial, em um cenário no qual se introduziu o uso de redes de computadores, alavancando a necessidade latente de se inserir a ciência da informação como uma disciplina a ser estudada academicamente. A mudança de como o mundo passou a lidar com dados no meio informatizado faz dos tempos atuais a denominada “Sociedade da Informação” (CAPURRO, 2007).

O alto fluxo e nível de informação transmitida pelas redes são suficientes para demonstrar a sensibilidade da segurança dos dados. A proteção aos estoques informacionais é imprescindível na atual sociedade, e o maior desafio na segurança da informação.

A Segurança da Informação tem por definição três pilares: a confidencialidade, a integridade e a disponibilidade. Basta que se comprometa um desses pilares para colocar em risco a segurança informacional (KLETTENBERG, 2016).

Entre os diversos riscos possíveis, podem ocorrer vazamentos de dados de fornecedores e clientes, onde algo assim poderia afetar o futuro da empresa. Por exemplo, em novembro de 2019 a empresa Prosegur sofreu um ataque em sua rede que causou a paralisação de todo o setor de TI (Tecnologia da Informação) e os sistemas para as comunicações internas foram indisponibilizados. Essa notícia foi amplamente divulgada, relacionando uma empresa de segurança de valores com uma falha de segurança digital (BLEEPING COMPUTER, 2019). Também poderiam ocorrer vazamentos de informações sigilosas de clientes, os impactando de maneira indireta, onde haveria uma redução na confiança dos clientes com a empresa. Além

disso, apenas uma notícia de um vazamento de dados já pode virar um ataque midiático à instituição.

Com o crescimento da demanda por informação também houve a necessidade do aumento do uso de redes de computadores (cada dia com mais convergência entre dados, voz e imagem), através das quais as instituições estão à mercê de ataques devido às possíveis ameaças que venham a explorar vulnerabilidades existentes no ambiente computacional. Sendo assim é de grande importância que existam ações com o objetivo de proteger estes recursos e consequentemente a informação trafegada.

1.1 PROBLEMA

Existem diversos sistemas de *firewall* para segurança da informação disponíveis, pagos ou gratuitos, proprietários ou abertos, simples ou complexos (HUAMAN; JOSEPH, 2018). Nesse meio é possível operar com aplicações básicas configuradas pelos administradores de redes, como o Netfilter que está disponível na maioria das distribuições Linux. Também existem grandes sistemas com análise de tráfego em camadas de rede mais altas (aquelas que envolvem não apenas os endereçamentos para tráfego, mas também os tipos de protocolos de aplicação utilizados, e com *appliances* (um dispositivo físico com software dedicado a uma aplicação) de *firewall* que são vendidos para serem integrados junto aos equipamentos de redes de computadores em soluções fechadas, em geral compradas de grandes fornecedores do mercado de segurança em redes. Tendo diversas opções podem existir dificuldades de escolha de aplicação e utilização por parte dos administradores de redes e outros responsáveis (YU et al, 2017).

Outro problema que é encontrado para ativar um sistema de *firewall* é a grande quantidade de aplicações disponíveis para análise e prevenção, como filtros de rede, *sniffers* de tráfego, analisadores de banda utilizada e *scanners* de portas (as definições desses conceitos serão apresentadas posteriormente na fundamentação teórica deste trabalho). Dependendo da necessidade das redes locais pode não valer a pena ter todas as aplicações disponíveis instaladas, tornando difícil escolher qual conjunto de ferramentas será utilizado.

Também existem diversos modos de configurar uma rede local em meios distintos, e cada uma delas possui requisitos de segurança específicos, adequados

e personalizados para tal. Um estudo do cenário ideal sempre é necessário, mas as escolhas geradas acabam por causar muitos problemas de adequação e configuração de ferramentas, ou a utilização de grandes serviços pagos e proprietários, que podem usar mais processamento ou mesmo ter um custo maior do que o necessário.

Nas diversas opções de *firewalls* existem impactos para que seu pleno funcionamento ocorra, entre eles o processamento dos tráfegos passantes a serem analisados, variantes de ataques, filtros que são usados como critérios de bloqueio e até mesmo *logs* (registros de histórico de eventos) que podem ser pouco eficientes.

Tendo em vista as dificuldades apresentadas para realizar a instalação e manutenção de um sistema de *firewall* adaptável a cada rede local, este acaba por ser o principal problema observado. Sendo necessário então organizar essas ferramentas de maneira eficiente, com meios para se automatizar parte dos processos, manutenção de registros e principalmente facilitar a modularização e escalabilidade (KASTNER et al, 1999). Estas acabam por ser as principais características, e mais efetivas, para mitigar os problemas de administração do sistema.

Outras questões importantes para a disponibilização de um sistema de segurança são o seu custo, que muitas vezes é alto, falta de modularidade e ausência de recursos para manter o serviço em operação em caso de falhas parciais no sistema.

1.2 JUSTIFICATIVA

Em meio a tantas aplicações de segurança e modos para se disponibilizar sistemas de *Firewall* em redes locais, podem existir várias dificuldades técnicas, por exemplo a falta de modularidade e a dificuldade em trabalhar com escalabilidade de tráfego, entre outras situações.

No mercado de TI atualmente estão em voga paradigmas como *containers* (POCINO, 2021), isolamento de processos (MAVRIDIS; KARATZA, 2021) e alta disponibilidade (LE-TRUNG, 2021). Em geral, esses métodos são levados para trabalhos com focos em serviços que operam mais próximos do usuário final, como

servidores para páginas *web*, serviços de entrega de autenticação, ou mesmo servidores de IP (endereço para acesso à rede atribuído a dispositivos), por exemplo. A grande quantidade de aplicações disponíveis levou à criação de métodos para encapsular as ferramentas em *containers* para facilitar a integração fluida em redes locais diferentes.

As ferramentas para se criar serviços de alta disponibilidade, com recursos modulares e empilháveis existem, mas normalmente não são encontradas em aplicações focadas na segurança de redes de computadores. Sendo assim, propor a integração de ambas será de grande valia para a continuidade do desenvolvimento de ferramentas encapsuladas em *containers*, e também para colaborar com novas opções de funcionalidades em sistemas de segurança nas redes de computadores.

Também existem hoje no mercado opções de computadores mais baratos e simples do que as soluções mais comuns e robustas, esses são os chamados computadores de placa-única, que possuem todos os componentes para o funcionamento básico em uma única placa lógica.

1.3 OBJETIVOS

Nesta seção será apresentado o que se pretende alcançar por meio deste trabalho de conclusão de curso.

1.3.1 Objetivo Geral

Realizar o estudo para disponibilização de *firewall* modular e escalável para redes locais de pequenas empresas e instituições, com o uso de *containers* e computadores de placa única de baixo custo.

1.3.2 Objetivos específicos

- Identificar ferramentas focadas em segurança para redes de computadores que possam compor um *firewall* de baixo custo;
- Sugerir uma forma de integração otimizada entre as ferramentas para uso em um sistema modular;
- Identificar topologias de rede que possam se adequar ao ambiente proposto de *firewall*;
- Analisar as maneiras de como realizar a modularidade física do sistema de *firewall* com computadores de placa única de baixo custo.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo será apresentada a base teórica dos conceitos utilizados para o desenvolvimento do trabalho.

2.1 *HARDWARE*

Hardware é a parte física dos computadores e demais conceitos de eletrônica computacional (HENNESSY; PATTERSON, 2014). Nas próximas seções serão apresentados os conceitos de hardware usados no trabalho.

2.1.1 Computadores De Placa Única

A arquitetura de Von Neumann é um dos modelos utilizados para descrever quais componentes são necessários para se definir um computador. A proposta é que o computador precise possuir pelo menos uma memória, uma unidade lógica e uma unidade de controle (VERHULST, 2003). Esse modelo é exemplificado em diversos equipamentos do cotidiano, como *desktops* e *notebooks*, onde é simples identificar os componentes descritos.

Atualmente existem os chamados computadores de placa única, que possuem todos os componentes necessários para se ter um computador em uma única placa lógica, sem que se faça necessária a inclusão de outras partes para o funcionamento básico. Existem diversas opções, com vários níveis de desempenho e capacidade de processamento. Pelo seu baixo custo e ampla gama de possibilidades de uso é uma opção que pode ser usada em larga escala para diversos projetos diferentes (RASPBERRY, [202-]).

Raspberry PI é o computador de placa única mais conhecido no meio de tecnologia e educação. O mesmo é desenvolvido por uma fundação que declara o intuito de disseminar os seus equipamentos com baixo custo para auxílio em

diversos projetos, principalmente de educação (RASPBERRY, [202-]). Na Figura 1 consta um exemplo de um dos modelos do computador de placa única.

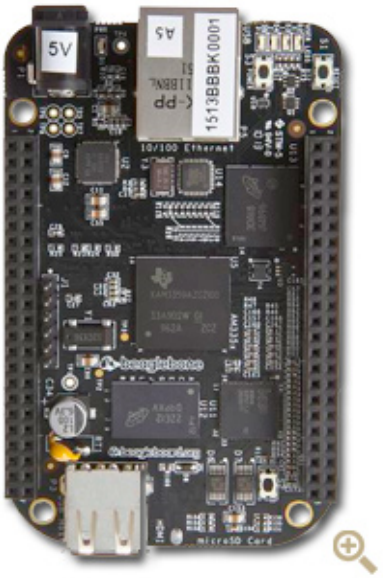
O Raspberry PI é um computador de placa única com o custo muito baixo recomendado para projetos de tecnologia e educação. Diferente de microcontroladores como o Arduíno, ele possui interfaces nativas de rede, áudio e vídeo, além de barramentos para ampliar a quantidade de periféricos na peça, trazendo muitas possibilidades de desenvolvimento com um bom desempenho e baixo custo de manutenção e energia, além do financeiro (YAMANOOR; YAMANOOR, 2017).

Figura 1 - Raspberry PI 3 model B+

Fonte: RASPBERRY, [202-]

BeagleBone é um computador de placa única desenvolvido pela *Texas Instruments*, já conhecida no mercado de componentes eletrônicos. Ele também possui compatibilidade com sistemas embarcados e possui arquitetura *ARM* em seu *chipset* recomendado para projetos de Inteligência Artificial e automação de baixo custo. Na Figura 2 o exemplo do modelo BeagleBone Black.

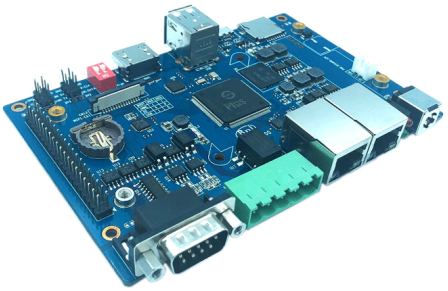
Figura 2 - BeagleBone Black



Fonte: BEAGLE BOARD, [202-]

Banana PI é uma alternativa direta ao Raspberry (como declarado pela própria desenvolvedora), completamente desenvolvido por uma empresa chinesa, a Shenzhen LeMaker Technology. O objetivo do projeto é ser uma alternativa com custo semelhante mas totalmente compatível com a peça de onde se baseia. Na Figura 3 consta o exemplo do modelo BPI-F2P.

Figura 3 - BPI-F2P



Fonte: BANANA PI, [202-]

2.2 *SOFTWARE*

Software é a parte lógica dos computadores e demais conceitos de eletrônica computacional, se constituindo de instruções detalhadas em uma linguagem específicas para ser interpretada pelo hardware suportado (LAUDON; LAUDON, 2004). Nas próximas seções serão apresentados os conceitos de hardware usados no trabalho.

2.2.1 Linux

Primariamente o Linux é um *kernel,* esse é o sistema com a função de fazer a ligação entre os *softwares* e o *hardware* que os estão suportando. Sendo também que todo sistema operacional possui um *kernel* (RUSLING, 1999). Popularmente o termo Linux também é utilizado para descrever um sistema operacional que usa esse *kernel*.

Existem sistemas Linux que são executados em diversos tipos de equipamentos diferentes, podendo ser base desde máquinas de usuários finais até equipamentos de redes como *switches* e roteadores, ou até mesmo celulares (FILHO, 2012).

Existem diversos atributos que mostram a robustez e as vantagens de se utilizar um sistema baseado em Linux, entre elas os recursos de multitarefa (que permite executar diversas aplicações simultaneamente). Possui suporte nativo a proteção de processos executados na memória RAM, que evita que um processo utilize parte da memória que está alocada para uma segunda aplicação (FILHO, 2012). Também há a modularização, que é um recurso nativo que busca liberar a memória recém utilizada imediatamente após a aplicação ser finalizada (DA SILVA, 1999). Focando em projetos de redes, sistemas Linux já possuem suporte nativo a recursos como IP, VLAN, *bridge*, *trunking* e outros protocolos e recursos úteis para telecomunicações (DA SILVA, 1999).

O Linux também possui suporte a diversas arquiteturas de processador, o que permite a instalação do sistema em diversos *hardwares*, incluindo os computadores de placa única usados de base nesse trabalho (FILHO, 2012).

Em comparação a sistemas Microsoft Windows, por exemplo, o modelo TCP/IP é mais rápido ao se utilizar Linux, pois esse recurso é constantemente melhorado e seu processamento ocorre direto no *kernel*, não sendo necessário uma camada intermediária de processamento como ocorre no *Windows* por meio do *WinSock* (DA SILVA, 1999).

Apenas o *kernel* Linux e algumas ferramentas simples não são suficientes para se ter um sistema operacional funcional. Existem diversos grupos e empresas que são responsáveis por compilar o *kernel* Linux junto com outros aplicativos como editores de texto, *firewall* e banco de dados, por exemplo. Esse trabalho resulta em distribuições Linux, que é o que será de fato instalado nos computadores e equipamentos utilizados (CAMPOS, 2006).

2.2.1.1 Raspberry PI OS

Anteriormente chamado de Raspbian, o Raspberry Pi OS é a distribuição Linux com suporte oficial pela equipe do Raspberry, é baseado na última versão estável do Debian para processadores ARM (durante a publicação desse trabalho a última versão estável é a 10.9), somado de diversos pacotes para melhor integração com o computador de placa única Raspberry PI (RASPBERRY, [202-]). O sistema base do Raspberry Pi OS (Debian) é conhecido como "sistema operacional universal". Grande parte dos pacotes e aplicações são desenvolvidas com foco no Debian, o que apresenta grande compatibilidade com as funções de segurança em redes e análise de tráfego. Como consequência essa compatibilidade acaba também ocorrendo na distribuição focada nesse parágrafo.

2.2.1.2 Alpine

Alpine é uma distribuição Linux desenvolvida com foco em segurança e principalmente simplicidade. Utilizando poucos recursos da máquina onde será instalada, sua base é muito leve. Ocupando menos de 130 MB é possível instalar o sistema completo em um computador. Ele possui também ótima integração com *containers* Linux, onde uma imagem de sua instalação ocupa apenas 8 MB, garantindo boa eficiência e uma grande quantidade de programas pré-compilados para serem instalados nele (ALPINE, [202-]).

2.2.1.3 Comparativo de distribuições Linux

No Quadro 1 é demonstrado de maneira resumida as principais características das distribuições Linux citadas anteriormente.

Quadro 1 - Comparação entre as distribuições Linux escolhidas

	Debian	Raspberry PI OS	Alpine
Quantidade de Pacotes disponíveis	Alta	Semelhante ao do debian	Menor que os outros dois exemplos, mas ainda assim alta
Compatibilidade com hardwares	Alta	Alta para Raspberries, e média com outros computadores de placa única	Alta
Recomendação de plataforma	Desktops e servidores	Computadores Raspberry	Virtualização e *containers*
Suporte	Comunidade	Empresa Raspberry	Comunidade e empresas parceiras
Última versão estável	10.9 Lançado em 27/03/2021	10 - Lançado em 04/03/2021	3.13.4 - Lançado em 31/05/2021
Custo	Gratuito	Gratuito	Gratuito

Fonte: Autoria própria

Com o Quadro 1 é possível se concluir que existem disponíveis no mercado diferentes distribuições Linux com aplicações diferentes. Se faz necessário o devido estudo para se aplicar cada distribuição no ambiente mais adequado para tal

2.2.2 *Container* Linux

Desde 2008 o *Kernel* Linux passou a possuir integrado a ele a opção de configurar "*container*", o que adicionou a possibilidade de executar processos isolados da máquina principal. O sistema possui um processo executando normalmente, mas ele não impacta os demais serviços da máquina *host*, tendo assim mais segurança para se testar recursos instáveis ou executar serviços críticos. Seu uso não foi muito popular até o momento em que o Docker foi lançado (SILVA, 2016). Depois do Docker ainda surgiram outras alternativas de configuração e gestão de *containers* como o Podman, Rocket e LXD (EMILSSON, 2020).

2.2.2.1 Docker

Docker é uma ferramenta criada em 2013 pela empresa *dotCloud* com o intuito de suportar os serviços prestados por ela de maneira mais eficiente. A proposta é rodar os processos em pequenas "máquinas" (chamadas de *containers*) dentro de um computador "hospedeiro" (chamado de *host*), é comum que o conceito se confunda com o de virtualização, onde sistemas operacionais são completamente emulados dentro de outro computador. No caso do Docker os *containers* não são máquinas virtuais completas, eles funcionam compartilhando de maneira direta os recursos do *host,* processamento de CPU, uso de memória RAM e até mesmo o *kernel* do sistema, por exemplo, são compartilhados como se fossem aplicações funcionando diretamente no *host*, isso aumenta a responsividade das aplicações.

Um ponto conceitual importante do Docker é seu gerenciamento (chamado de orquestração) onde os *containers* são tão manipuláveis quanto um processo qualquer de operação no sistema (diferente de máquinas virtuais). Os *containers* são criados com o intuito de rodar tarefas específicas e são descartados sem perdas

grandes, como aconteceria se uma máquina virtual se corrompesse. As múltiplas opções de configuração de rede do Docker permitem que as interfaces virtuais dos *containers* sejam integradas com interfaces físicas específicas do *host*, podendo receber o mesmo tráfego que nelas, também podem ser configuradas as portas abertas em cada container. Além de realizar a comunicação entre eles de maneira simples e automática (podendo ser alterado caso necessário), inclusive utilizando do conceito de *VLANs*, o que facilita na integração com topologias de rede já existentes. A opção de *docker volume* permite montar discos de armazenamento dentro dos *containers* que podem ser compartilhados simultaneamente entre eles (SILVA, 2016).

Dockerfile é um arquivo de texto com instruções de como construir um container. O arquivo possui uma série de parâmetros que a plataforma usa para realizar as devidas configurações, como a distribuição base para se construir a imagem, quais pacotes que serão instalados, se será necessário abrir alguma porta de comunicação, se é preciso copiar algum arquivo ou pasta para dentro do container e até mesmo quais usuários devem ser criados (SILVA, 2016).

Docker Swarm é uma ferramenta que permite a manipulação de *containers* espalhados por máquinas diferentes, trabalhando como *Cluster* ao criar serviços que irão carregar *containers* com réplicas em diversos hosts, podendo assim trabalhar com alta escalabilidade e sincronizar as máquinas (SOPPELSA; KAEWKASI, 2016).

2.2.3 Escalabilidade de Armazenamento

Ao se trabalhar com escalabilidade de máquinas, bem como aplicações em *cluster,* é muito comum que os dados armazenados precisem ser sincronizados junto aos demais computadores do conjunto. Sendo assim diversas ferramentas foram desenvolvidas com o intuito de realizar esse processo, tanto de forma síncrona (atualizado imediatamente entre todas as máquinas) quanto assíncrona (quando é necessária a declaração de intenção para que os dados sejam replicados).

2.2.3.1 GlusterFS

GlusterFS é uma ferramenta gratuita e *open-source* que possui a função de escalabilidade de armazenamento. Possui opções de complemento em blocos (onde se cria um diretório virtual que mostra o conteúdo de vários “nós” de um *cluster* como se estivessem em uma única máquina). A opção também pode operar com o modo de réplicas, onde ela copia em tempo real os arquivos de diferentes nós de um para o outro, criando um sistema de cópia síncrono (GLUSTER, [202-]).

2.3 REDES DE COMPUTADORES

“Redes de computadores” é o termo utilizado para qualquer topologia onde dois ou mais computadores são conectados por algum meio capaz de trocar informações a nível físico, como cabos elétricos de baixa tensão, fibra ótica ou mesmo sinais de rádio (FILHO, 2013).

Essas redes independentes se juntam para fazer parte de uma grande soma de redes que se comunicam, formando a *Internet*, uma “rede de redes”. A *internet* é uma estrutura complexa, composta de clientes, provedores, fornecedores de *links* mais robustos, servidores de grandes serviços que se integram a outras redes menores com o intuito de realizar a comunicação entre todos que que fazem parte dessa grande topologia (KUROSE; ROSS, 2006).

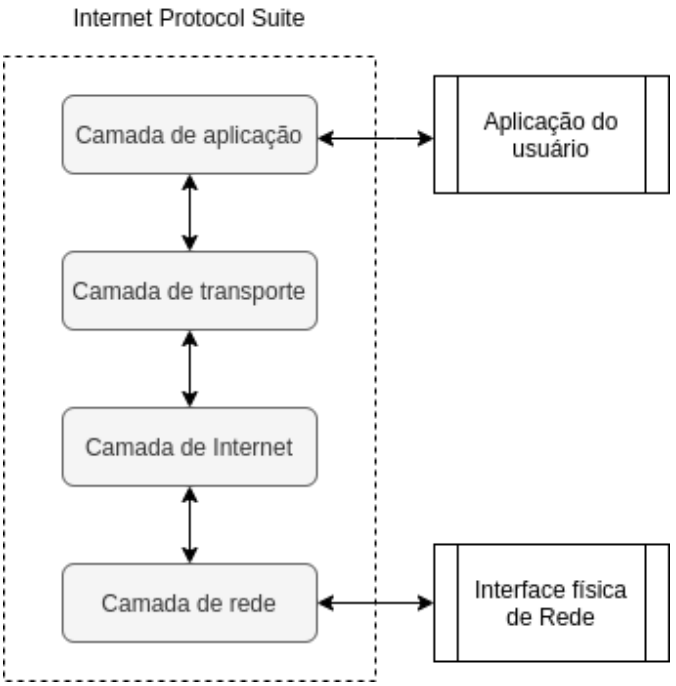
Topologia de rede é o termo usado para definir a estrutura lógica e física de uma rede de computadores, em geral usada para se referir a redes locais no que tange as suas conexões, equipamentos de borda, transmissão e clientes finais (FREUND, 2009).

2.3.1 Família de protocolos TCP/IP

Para o funcionamento corrente das redes de computadores, são usados diversos protocolos, cada um com sua função específica. Considerando que a *internet* é um destino comum na maioria das redes locais, o TCP/IP acaba por ter grande importância no meio de telecomunicações, sendo necessário entender como o mesmo funciona. Existem dois projetos diferentes, o TCP (sigla para *Transmission Control Protocol*) e o IP (*Internet Protocol*), juntos formam uma família de protocolos conhecida como TCP/IP. O principal objetivo dos projetos foi criar formas de entregar as informações por caminhos alternativos, validando as falhas da parte física para conseguir realizar seu processo de comunicação (FILHO, 2013).

O conjunto TCP/IP ainda é conhecido como IPS (*Internet* Protocol Suite), um modelo de comunicação em quatro camadas, indo desde as interfaces físicas de rede até a aplicação utilizada (FORSHAW, 2018).

Figura 4 - Descrição de camadas do *Internet Protocol Suite*



Fonte: Autoria própria, baseado no descritivo de Forshaw (p. 26, 2018)

Pelo modelo descrito na Figura 4 temos uma pilha de camadas, seguindo a partir da Interface física de rede consta:

Camada de rede: É a camada de mais baixo nível do modelo, funcionando imediatamente após as comunicações físicas. Nessa parte de processo é onde protocolos como o *Ethernet II* e o *PPP (Point to Point Protocol)* operam.

Camada de *Internet*: Nesse ponto os pacotes passam a ter endereços de rede para identificação e roteamento. É onde ficam definidos os *IP*'s (podendo ser *IPv4* ou *IPv6*).

Camada de transporte: Ponto onde ocorre a transmissão dos pacotes, fazendo a ligação entres os endereçamentos (das camadas anteriores) e as aplicações (que serão descritas na próxima camada), utilizando protocolos como UDP e TCP para criar as lógicas de transmissão utilizadas para esse transporte de dados.

Camada de aplicação: Camada mais próxima do utilizador final, onde operam os protocolos das aplicações utilizadas pela rede, como o HTTP, HTTPS, SMTP e DNS.

Seguindo o modelo TCP/IP descrito, focando em topologias locais, cada equipamento tem um endereço físico único (endereço que é lido na camada de rede) e um endereço atribuído de acordo com a rede específica, o IP (já encontrado na camada de internet). Cada grupo de IP's é uma rede específica, sendo que em uma rede local pode-se ter uma ou mais redes.

Tendo em vista o foco do trabalho, a base teórica de redes apresentada nesse tópico já é o suficiente para o entendimento do seu funcionamento. O detalhamento dos protocolos e regras utilizadas é em geral referenciada por *RFCs* (documentos onde constam informações e guias para padronização de funcionamento de protocolos e serviços). Algumas *RFCs* são as 791 (sobre o IP), a 1180 (sobre o modelo *TCP/IP*) e a 793 (sobre *TCP*).

2.3.2 Componentes Físicos de Redes

Diversos componentes podem fazer parte de uma rede local, alguns como *desktops*, câmeras, telefones e outros periféricos que ficam a vista dos usuários finais. Também fazem parte do processo os meios de transmissão como cabos metálicos, cordões de fibra ótica e sinais sem fio, que servem para conectar os outros componentes. Por fim, também existem os chamados equipamentos ativos, que são responsáveis por fazer a comunicação lógica das transmissões de dados (FILHO, 2013).

Entre os ativos, os dois principais são o *Switch* (ou comutador) e o Roteador. O primeiro é utilizado para fazer a distribuição dos pacotes transmitidos em uma rede, entregando eficiência no processo de envio de dados entre os dispositivos conectados. O comutador armazena informações referentes a cada dispositivo conectado fisicamente em suas portas, definindo assim quais caminhos devem ser utilizados para a transmissão de dados entre os dispositivos de acordo com seus endereços físicos (MAC *address*). Além de realizar vários processos para otimizar a transmissão de pacotes (KUROSE; ROSS, 2006). O Roteador é capaz de direcionar os pacotes transmitidos não apenas pelos endereços físicos dos dispositivos conectados a ele, mas também considerando os endereços de rede (IP) a eles atribuídos. Podendo diferenciar os caminhos para chegar em diferentes redes e então calcular por quais interfaces físicas os pacotes irão trafegar, fazendo assim o roteamento entre redes (KUROSE; ROSS, 2006).

2.4 *FIREWALL*

É possível afirmar que o termo *Firewall* não é um conceito fechado de sistema ou máquina que é instalada na rede local. Pode ser considerado um conjunto de esforços usados para segurança em redes, onde todos os recursos utilizados para tal fazem parte do sistema, mesmo que um desses seja uma pessoa que analisa os registros do monitoramento. Mesmo a conscientização dos usuários dos sistemas também podem fazer parte do *firewall* empregado (FILHO, 2013).

Cada rede local pode ter necessidades diferentes para o seu *Firewall*, dessa maneira uma ferramenta pronta e unificada dificilmente seria possível (BARBOSA; REIS, 2018), sendo assim poderia ser interessante uma aplicação modular, aplicando cada ferramenta e configuração necessária de acordo com cada caso.

Para se implementar um bom sistema de *firewall* a experiência do administrador em protocolos de redes, interfaces e endereçamento, além de entender a fundo o funcionamento do modelo TCP/IP (DA SILVA, 1999).

2.4.1 Filtragem de pacotes

Filtrar pacotes é umas das principais aplicações que *firewall* pode utilizar, o processo envolve analisar o tráfego de entrada e saída da rede para avaliar com base nos parâmetros dos pacotes se eles satisfazem uma lista de regras (FILHO, 2013), que devem ser definidas previamente pelos administradores. Esses filtros e regras são criados com base em protocolos de redes, endereçamento como IP, e portas de acesso utilizadas. É importante ressaltar que todas as rotas devem encaminhar os pacotes para a máquina responsável pela filtragem dos mesmos (HERTZOG; MAS, 2019).

Netfilter é um conjunto de módulos disponíveis por padrão no *kernel* Linux para diversas aplicações de rede, o Netfilter é definido pela própria equipe que realiza sua manutenção como um “Um *framework* que permite a filtragem de pacotes, a conversão do endereço de rede [e das portas] (NAT) e outras manipulações de pacotes de rede” (NETFILTER PROJECT, [201-]). A aplicação mais conhecida do Netfilter é o Iptables.

Iptables é uma interface que o usuário do sistema tem para usar o Netfilter e realizar suas operações. A base de funcionamento do iptables é a criação de regras para análise de pacotes com base nas informações dos cabeçalhos. Ele pode ser utilizado para bloquear tráfegos específicos, forçar o encaminhamento e alterar

parâmetros dos pacotes, podendo inclusive fazer a função de um roteador (SILVA, 1999).

2.4.2 Sistema de Detecção de invasores

IDS (Intrusion Detection System, em tradução direta Sistema de Detecção de invasores) é uma aplicação que visa monitorar em tempo real o tráfego com base em regras pré-determinadas. Tendo o intuito de descobrir acessos que não possuem permissão ao acesso a essa rede, além de outras anomalias de tráfego. O conceito de IDS evoluiu para o IPS (Intrusion Prevention System, em tradução direta Sistema de Prevenção de invasores) que além de monitorar possíveis invasões também tem o intuito de agir contra elas, criando regra de acesso, bloqueando a origem da invasão e limitando tráfego por exemplo. Existem alguns programas *open source* de IDS como o Ossec, Suricata e o Snort (GUPTA; SHARMA, 2020).

2.4.3 Captura e análise de tráfego

Análise de tráfego é o processo de realizar a captura dos pacotes passantes na rede local para se avaliar a sanidade do tráfego de rede, principalmente com o intuito de buscar soluções para problemas de acesso. Tcpdump é uma ferramenta de análise de tráfego de rede que permite visualizar de forma detalhada os pacotes e acessos que acontecem nas interfaces de rede de um sistema. Existem diversos filtros para encontrar a informação necessária para a análise. O Tcpdump trabalha em diversas camadas de rede, especificando o tipo de tráfego, endereços utilizados, e conteúdo detalhado do cabeçalho dos pacotes, por exemplo. É uma ferramenta livre que opera direto no terminal da máquina host, tendo fácil integração com outras ferramentas operacionais (FILHO, 2013).

2.5 CONCEITOS PROPOSTOS PARA SE USAR COM O *FIREWALL*

Nesta seção serão apresentados os os conceitos teóricos propostos no trabalho para serem trabalhados juntamente com os conceitos já anteriormente para um sistema de *firewall*.

2.5.1 Empilhamento de *hardware*

Considerando o funcionamento padrão de um *switch* de rede, muitas vezes é comum que surja a necessidade do uso de mais portas, além das disponíveis, a solução inicial para isso é o uso de cascateamento. O processo envolve conectar dois ou mais *switches* por uma de suas portas *ethernet* para transmitir os dados entre eles, assim tendo mais interfaces disponíveis para uso, tendo que ainda configurar os dois separadamente. Alguns *switches* possuem a função de empilhamento (normalmente descritos comercialmente como “*switch stacking*”), essa função envolve adicionar um novo equipamento junto ao primeiro para ampliar a capacidade de sua função. Nesse processo os *switches* possuem uma interface disponível para ligar a outro compatível, quando isso é realizado é possível realizar a gerência dos dois como se fosse apenas um, “extendendo” a quantidade de portas disponíveis no equipamento, seria possível usar dois switches de 24 portas cada e empilhar eles sob a mesma gerência (DLTEC, 2014).

Comparando o processo de cascateamento com o de empilhamento, a principal diferença é do ponto de vista da gerência, onde fica facilitado por ter um equipamento principal, e apenas ele precisará ser configurado. Outro ponto é que nenhuma das interfaces *ethernet* ficaria indisponível por estar dedicada a transmitir os dados entre eles. Do ponto de vista de funcionalidade o empilhamento ainda possui a vantagem de não causar *loops* na rede por ter o mesmo endereço MAC aprendido em interfaces em portas diferentes, já que a tabela MAC será “conhecida” pela solução como um todo.

Por fim ainda existe o exemplo dos servidores com múltiplas máquinas, onde o provisionamento da aplicação em si é balanceada entre dois ou mais computadores, dividindo o processamento de requisições para não sobrecarregar

nenhum nó, e também servindo de redundância pro caso de indisponibilidade de um dos servidores.

2.5.2 Alta disponibilidade

Alta disponibilidade é o termo atribuído aos esforços que são somados para que serviços continuem operacionais e não fiquem indisponíveis. Sem falhas e evitando problemas de performance e queda de operação. Muitas vezes associado com a integração inteligente entre servidores e outros equipamentos ativos, posicionados para terem redundância e/ou chaveamento em caso de necessidade. Um exemplo possível é a existência de dois servidores físicos responsáveis pela disponibilização de uma página *web* e ambos ligados às saídas físicas e lógicas, e as configurar para, caso uma delas desligue ou pare de funcionar, a outra continue operando normalmente (FERREIRA et al, 2005).

3 ESTUDO DE CASO

Para a realização do estudo de caso foi proposta a construção de um pequeno ambiente que simule uma topologia de uma rede comum, se assemelhando às pequenas e médias empresas. Na proposta onde exista a rede local, seus equipamentos para comunicação interna e um *link* com a *internet* (contratado com uma operadora), os equipamentos e serviços propostos no trabalho ficam entre esse *link* externo e a rede local da instituição. Esse exemplo de arquitetura é o foco principal de aplicação deste trabalho.

3.1 METODOLOGIA

O presente estudo foi desenvolvido a partir de uma abordagem qualitativa, pois a análise da eficiência do *firewall* gera resultados descritivos dos testes. Os métodos qualitativos são identificados quando há uma a imersão do pesquisador no contexto e se assume uma perspectiva interpretativa de condução da pesquisa (KAPLAN & DUNCHON, 1988), dessa forma, pode-se entender que os dados resultantes da avaliação do *firewall* disponibilizado foram analisados por essa perspectiva interpretativa e subjetiva.

A natureza deste estudo tratou-se de uma pesquisa aplicada, cujos conhecimentos gerados servem de base para a disponibilização de um *firewall* modular e empilhável para redes locais de pequenas empresas e instituições.

Os objetivos visam uma pesquisa exploratória, na qual pretende-se a consolidação dos estudos das ferramentas de segurança para decidir quais serão integradas ao sistema final de acordo com sua eficiência e integração com as demais aplicações e recursos utilizados. Uma pesquisa exploratória é definida por se pretender gerar uma teoria, mais do que verificar; explicar um processo, ação ou interação, exigir um procedimento sistematizado passo a passo para o estudo e ser uma pesquisa orientada para os dados obtidos (GASQUE, 2007).

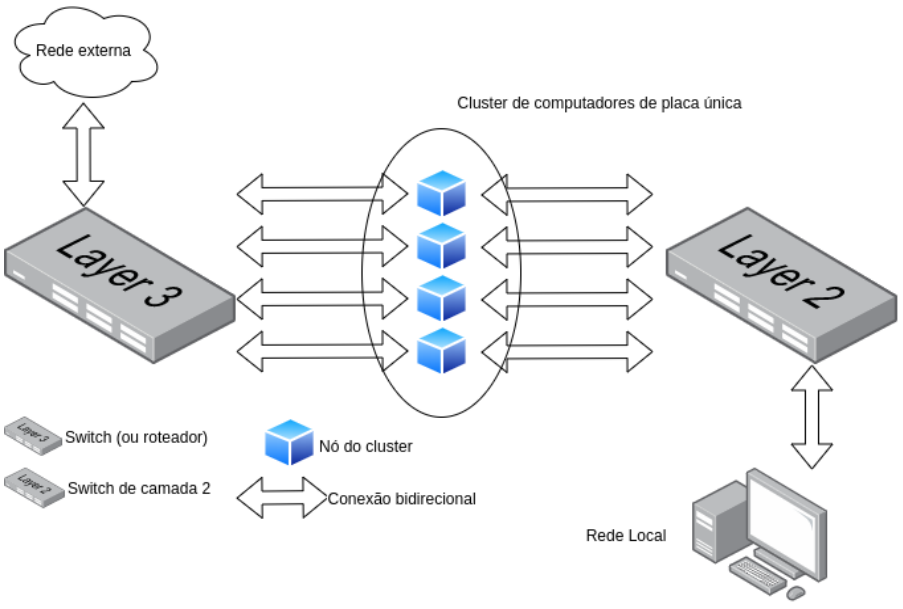
O presente estudo se define como uma pesquisa experimental, ao qual se incluíram uma fase de levantamento das ferramentas utilizadas para integrar o sistema, seguida da construção dos *containers* base para testes, na sequência foi realizada a pesquisa e configuração das aplicações de segurança, além da construção dos processos de orquestração dos *containers* e por fim, a construção do *cluster* múltiplos computadores.

Dessa forma, através dos métodos utilizados e da análise da eficiência dos mesmos, pretendia-se atingir os objetivos principais de disponibilização do *firewall* modular de baixo custo, e por fim conseguir avaliar sua eficiência e aplicação por meio de testes.

3.2 TOPOLOGIA PROPOSTA

Para realizar a implementação sugerida, os equipamentos que compõem a função de *firewall* serão inseridos entre as redes LAN e WAN, interligando a conexão da rede externa com as máquinas da rede local, a topologia está descrita na Figura 5.

Figura 5 - Infográfico da topologia proposta para o trabalho



Fonte: Autoria própria

A Figura 5 apresenta a utilização de uma conexão de rede externa, coexistindo com uma conexão bidirecional com um equipamento ativo, podendo ser um Roteador ou um *Switch* de camada 3 (que consiga manipular e direcionar pacotes com base em endereçamento IP, e não apenas endereços físicos como os *switches* de camada 2), sendo responsável por agregar os pacotes vindos da rede externa e encaminhar os mesmos para o *firewall* em *cluster*.

O *cluster* é formado de computadores de placa única, para se ter duas placas de rede em cada nó, foi usado o Raspberry PI adicionado de um adaptador RJ45-USB. A interface de rede RJ45 nativa do equipamento é a interface de entrada e a do adaptador é a de saída. As interfaces de saída dos Raspberries são encaminhadas para um *switch* padrão, que irá direcionar o tráfego para as interfaces de entrada da rede local, onde a conexão será distribuída de forma a atender os computadores e demais equipamentos.

3.3 PREPARO DOS NÓS

Cada nó do *cluster* será um computador de placa única, as peças disponíveis são *raspberries* de modelos diferentes (um do modelo 3B+ e dois são do modelo 1B+) devido a disponibilidade de componentes. O passo a passo detalhado dos pontos apresentados nesta seção estão no APÊNDICE A

3.3.1 Aplicações de suporte

A base do sistema operacional utilizado no Raspberry PI é o Debian. Tendo em vista as aplicações que serão utilizadas é essencial instalar alguns *softwares* que já estão disponíveis nos repositórios do próprio sistema operacional. Além dos pacotes que são instalados por padrão na versão básica do Raspberry Pi OS é necessário instalar alguns softwares:

- *curl*, usado para manipular serviços em *HTTP* e HTTPS por linha de comando.
- *apt-transport-HTTPS*, para habilitar o download de repositórios localizados em repositórios que usam protocolo *ssl*.
- O *iproute2* que possui uma série de utilitários para redes e roteamento em sistemas com *kernel* linux.
- O *iptables-persistent*, aplicação para habilitar a persistência de configurações do iptables após a re-inicialização (ou desligamento) do computador utilizado.
- O *wget*, que é usado para fazer o download de arquivos via linha de comando, é muito utilizado para baixar chaves de licença de utilização de programas.
- E por fim o *gnupg* que é utilizado para a criptografia com chaves assimétricas.

3.3.2 Conexão ssh

Para realizar a comunicação entre os nós, a opção escolhida foi o ssh (*secure shell*) devido ao grande número de configurações possíveis com ele, sua segurança de transmissão de dados e a possibilidade de se configurar chaves de acesso entre computadores. Para os testes realizados todos os comandos aplicados foram realizados utilizando o usuário *root*, que possui privilégios totais no sistema. É necessário habilitar o acesso ssh pelo usuário *root* nas configurações dos servidores ssh. A princípio, a cada conexão aberta via ssh é necessário digitar a senha de acesso ao nó. Para facilitar esse processo mas mantendo a segurança, foram geradas chaves privadas de acesso e compartilhadas entre os nós.

3.3.3 Docker *cluster*

A própria empresa responsável pela aplicação Docker disponibiliza em seu *site* um *script* de instalação que detecta a distribuição usada e faz todo o processo de instalação automaticamente. O *script* está localizado no endereço *https://get.docker.com* e pode ser executado pelo software *curl* (SILVA, 2016).

Para montar o sistema de *cluster* com o *Docker* é preciso iniciar o serviço *swarm*. Ao realizar esse processo em um dos computadores ele se torna o *manager* (computador responsável por distribuir as tarefas aos demais nós) de um novo *cluster*. Também é gerada uma chave de segurança que possui encriptação conforme o protocolo ssh, assegurando que caso os dados sejam interceptados, eles não possam ser lidos sem a chave. Para adicionar demais computadores no processo, basta copiar essa linha de comando e executar a mesma em todos os nós. Abaixo um exemplo do processo de inicialização gerando o comando com a chave encriptada:

```
# docker swarm init

Swarm initialized: current node (gbxn97eofsppgyjb8lq0prt0j) is now a manager.

To add a worker to this swarm, run the following command:

      docker swarm join --token
SWMTKN-1-064k04om7vy032enr2ow2w2d9ptqcrwwimghdyfe9rmdm5oov4-6m28afwq4cdf4yyvv
ywpn6vfi 192.168.15.7:2377
```

Os detalhes de configuração do cenário com o Docker estão no APÊNDICE B.

3.3.4 Gluster *cluster*

A instalação do *GlusterFS* é realizada de maneira simples, basta baixar as chaves de licença (gratuitamente) e adicionar os repositórios na base do sistema. Tendo ele instalado, é necessário configurar os nós para replicarem os diretórios que serão compartilhados, o processo envolve “sondar” os computadores disponíveis na mesma rede. Em seguida foi necessário criar um “volume” que é montado em todas as máquinas. Dentre as opções de configuração possíveis será configurado como

“réplica”, deste modo todos os dados que são criados e manipulados em qualquer nó são imediatamente replicados para os demais, se comportando como um diretório único montado em todos os computadores simultaneamente. Desta maneira é criada a escalabilidade de armazenamento para guardar as configurações das aplicações e salvar os registros necessários durante a operação. O processo completo está detalhado no APÊNDICE C.

3.4 ENCAPSULAMENTO DAS APLICAÇÕES EM *CONTAINERS*

Para que as aplicações escolhidas funcionem com o *Docker* conforme proposto, é necessário que elas estejam previamente instaladas em “imagens”. Para tal é necessário programar um *Dockerfile* que irá construir essa imagem.

Para o desenvolvimento das imagens contendo o Tcpdump e Snort foi usado o mesmo modelo de *Dockerfile* (apresentado no APÊNDICE D). Primeiro foi necessário escolher a distribuição base do *container*. Visando o menor consumo possível de recursos físicos foi escolhido o Alpine Linux, cuja sua base usa menos de 10 MB de armazenamento da máquina hospedeira. Outro motivo da escolha do Alpine foi o fato de que todos os pacotes previamente escolhidos já são pré-compilados para o sistema, facilitando suas instalações.

Para os *containers* das aplicações do Tcpdump e Snort o processo descrito no último parágrafo foi o suficiente para se utilizar as ferramentas, entretanto para se usar o netfilter (e iptables) houve um problema de permissões: o netfilter opera a nível do *kernel* Linux, ou seja, a baixo nível sistêmico, onde a operação do Docker não funciona. Por esse motivo, a criação e manipulação das regras com o Iptables precisarão de um segundo método fora do Docker para serem operadas e sincronizadas entre os nós.

3.5 CONFIGURAÇÕES DAS APLICAÇÕES UTILIZADAS

Nos próximos tópicos são brevemente descritas as configurações necessárias para cada ponto do sistema de *firewall*.

3.5.1 Raspberry PI OS

Para o sistema base dos nós, além da instalação das aplicações já descritas anteriormente, foram alterados os *hostnames* (o nome dado a cada computador, ou nó do *cluster*). Em sistemas Linux é possível associar um IP a um nome utilizando o arquivo “/etc/hosts”, dessa maneira a comunicação e escrita de *scripts* é simplificada ao se usar um nome e não com um endereço numérico. Os nós foram “batizados” como *node0*, *node1* e assim por diante. E entre eles os nomes e IPs dos demais nós também foram cadastrados.

Outro ponto foi a criação de um diretório para fazer a já citada escalabilidade de armazenamento, uma pasta dentro do sistema para guardar as configurações das aplicações e seus *logs*.

Para implementação dos pontos descritos nessa seção, os comandos usados estão no APÊNDICE A.

3.5.2 Tcpdump

As configurações básicas do tcpdump para serem executadas dentro em *containers* isolados são os mesmos da própria aplicação, com a diferença de que, na necessidade de gerar os *logs* das capturas de pacotes, eles foram enviados para um subdiretório daquele criado para ser montado em *cluster*, outro ponto é que na criação do container do tcpdump foi utilizado o parâmetro “*--network*” com o argumento *local*, dessa maneira ele compartilha os dados e informações das placas de rede com o nó em questão, fazendo a análise dos dados que passam diretamente pelas interfaces de rede onde constam. Mais detalhes das configurações encontram-se no APÊNDICE F.

3.5.3 Snort

O Snort é um sistema de detecção de intrusão (IDS) que depende de uma série de parâmetros carregados quando o serviço é iniciado. Por padrão, a aplicação utiliza configurações pré-definidas contidas em arquivos específicos para isso, armazenados no computador que a hospeda (SNORT, [202-]). Para adequar esses pontos ao sistema em desenvolvimento os arquivos de configuração devem estar já preparados dentro do diretório compartilhado, e esse deve ser montado no *container* ao ser iniciado. Detalhes dos arquivos usados para a configuração presentes no APÊNDICE E.

3.5.4 Iptables

Como citado anteriormente não foi possível a encapsular o Iptables em *containers*, entretanto o iptables opera como uma sequência de regras fixas nos computadores, que não depende de processos em alto nível. Bastando aplicar as regras do iptables conforme sua documentação em cada nó.

3.6 ORQUESTRAÇÃO DAS APLICAÇÕES EM CLUSTER

Para trabalhar com as aplicações em *cluster* são necessárias ações específicas de “administração” dos serviços utilizados, nos próximos tópicos essas ações serão descritas.

3.6.1 Iptables

Como o Iptables não pode ser encapsulado para poder realizar as configurações em *cluster*, a opção foi aplicar as regras em todos os nós

simultaneamente e usar a aplicação *iptables-persistent* para salvá-las. Para garantir a aplicação em todos os nós como esperado foi utilizado *script* simples em um laço *for* conforme exemplo abaixo:

```
for server in $(cat /HOTSFILE)
      do
            ssh root@$server "${IPTABLE_RULE}"
            ssh root@$server "iptables-save"
      done
```

No código descrito, o laço percorre a quantidade de nós existentes no *cluster* preenchidos em um arquivo, e executa em todos eles a regra específica. Mais detalhes da configuração do iptables em *cluster* estão descritas no APÊNDICE H.

3.6.2 Criação dos Docker Services

A ferramenta Docker Swarm permite a manipulação de “serviços” que são criados com base em parâmetros muito parecidos com os de criação de um container isolado. Ele replica a ação a ser executada para os demais nós do *cluster*. Fazendo a gerência da operação como, por exemplo, reiniciar quando algum container é finalizado por algum motivo não esperado. Também fazendo o balanceamento de carga e recurso com base na utilização de cada nó.

Para o caso do tcpdump e snort, a lógica utilizada é a mesma do caso do container isolado. É importante observar que as configurações em escalabilidade de armazenamento devem estar configuradas para que todos os nós operem da mesma maneira, com as mesmas regras e realizando o devido registro em todas as máquinas corretamente.

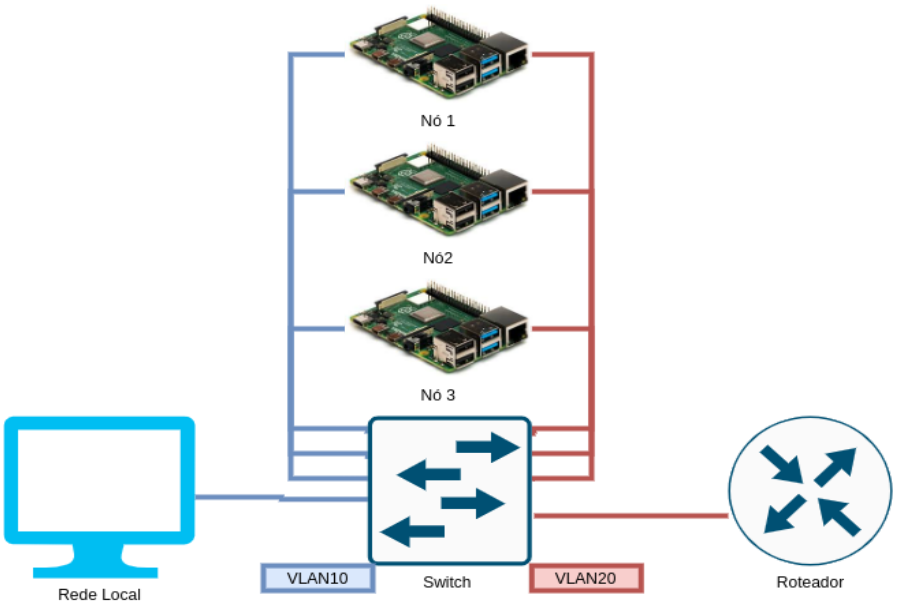
4 APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS

Os testes foram divididos em três fases. A primeira com a montagem do cenário e a operacionalização. A segunda envolve os testes de performance, para demonstração de características referente ao tráfego utilizado. E a última etapa contempla as aplicações de *firewall* no *cluster* e seu funcionamento. Nas segunda e terceira etapas, foram “forçadas” a indisponibilização de uma parte dos nós, tanto físicamente (desligamento de um dos raspberries) quanto logicamente (removendo o cabo de rede ou desligando a porta do switch conectada).

4.1 MONTAGEM DO CENÁRIO

O primeiro passo para se realizar os testes operacionais do trabalho envolveu montar um cenário real de aplicação, para tal foram utilizados três *raspberries*, que são os nós do *firewall*, um computador simulando a rede local, um *switch* gerenciável e um roteador com acesso a *internet*, conforme mostrado na Figura 6.

Figura 6 - Cenário de testes



Fonte: Autoria própria

4.1.1 Hardware utilizado

O roteador (à direita da Figura 6) possui duas interfaces, sendo que uma delas recebe o sinal externo com acesso a *internet* e a outra tem acesso ao cenário.

O *Switch* é o componente onde todos os equipamentos foram conectados para fazer a transmissão de tráfego entre eles.

Os raspberries utilizados possuem duas interfaces de rede, sendo a embutida na própria placa e outra por meio de um adaptador RJ45 USB.

O computador (à esquerda da Figura 6) foi posicionado simulando a rede local, que fica antes do *firewall* e a rede externa.

4.1.2 Segregação de tráfego por VLANs

Na proposta do trabalho, o tráfego da rede local deve ser encaminhado para os nós do *cluster*, e esse encaminhar os dados para a rede externa, filtrando-os quando assim configurado. Para que isso ocorra foram criadas VLANs no Switch utilizado, o tráfego de uma VLAN não é compartilhado por outra. As VLANs 10 e 20 foram aplicadas nas portas físicas do *switch*, a 10 foi configurada nas portas onde estão conectados o computador da rede local e uma interface de cada *raspberry*, e a 20 foi aplicada nas portas onde estão o roteador e a outra interface de cada *raspberry*. Dessa maneira o tráfego da rede local obrigatoriamente passa pelos nós antes de ser encaminhado para o roteador.

4.1.3 Encaminhamento de pacotes no *firewall*

Primariamente foi configurado nos nós do *cluster* uma interface em uma rede com acesso único a rede do roteador e a outra interface na rede local. Onde seria configurado para os nós fazerem NAT (processo de troca dos IPs nos pacotes para

que a rede interna tenha acesso a internet, usando os endereços da rede externa) entre as redes. Dessa maneira cada nó seria um roteador independente. Entretanto, houve um problema lógico, pois os computadores da rede local usam apenas um gateway para direcionar os pacotes para a rede externa, e tendo múltiplos *gateways* um computador comum não conseguiria fazer o acesso correto.

Foi alterada a configuração dos nós para criar uma *bridge* entre as duas interfaces físicas e atribuir a essa *bridge* um IP na rede local, a solução acabou por ser mais transparente do que a ideia anterior

4.1.4 Acesso a rede externa

O roteador possui acesso tanto a rede local quanto a rede externa em interfaces diferentes. Foi configurado o NAT entre as redes e o encaminhamento de pacotes entre as interfaces, fazendo com que os pacotes recebidos sejam enviados entre as redes, mudando o ip de origem para aquele que possui o acesso devido

4.1.5 Balanceamento de carga

Tendo em vista os conceitos de alta disponibilidade citados anteriormente na fundamentação teórica (seção 2.5.2) é necessário que haja o balanceamento de carga entre os nós, para que o tráfego recebido seja dividido entre eles. Sendo assim no *switch* utilizado foi realizada a configuração de um *“Port-channel”*, que agrega portas físicas de um *switch*, a configuração foi aplicada nas portas onde as interfaces dos nós direcionadas a rede externa estavam conectadas. Ainda no *port-channel* podem ser utilizadas as opções de balanceamento de carga com base no endereço mac de origem ou destino, IP de origem ou destino e ainda unir essas opções. Para os testes foi escolhida a opção de mac de origem, o *switch* distribuirá o tráfego com base no endereço físico das placas de rede da rede local

4.1.6 Serviços em *containers*

Para encapsular os softwares dentro dos *containers* do Docker foi necessário programar as imagens. A distribuição usada para os containers foi o Alpine devido ao seu baixo uso de armazenamento (a base ocupa aproximadamente 3MB). Quando foram realizadas as primeiras compilações houveram problemas de compatibilidade, dos três *raspberries* um é do modelo 3B+ e dois são do modelo 1B+, a arquitetura do *chipset* dos dois mais antigos não é compatível com a última versão disponível do Alpine. Para resolver a situação foi alterado na construção do container a versão do sistema base para uma mais antiga (versão 3.12.0).

Os serviços utilizados foram compilados conforme o Anexo D e configurados para funcionar em *cluster* com a aplicação Docker Swarm

4.1.7 Pasta de arquivos compartilhada

Para criar a pasta de configurações e registros sincronizada entre os nós do *cluster* a aplicação GlusterFS foi utilizada, conforme descrito no Apêndice C

4.1.8 Filtro de pacotes

Para se realizar o filtro de pacotes (principal aplicação do *firewall*) foi utilizado o *iptables*, criando um *script* para aplicar as regras em todos os nós simultaneamente, segundo detalhado no APÊNDICE H.

4.1.9 Funcionamento básico

Com o cenário físico e lógico disponibilizado foi possível fazer os testes de tráfego e acesso *web* usando um computador da rede local.

4.2 TESTES DE PERFORMANCE DE REDE

Na segunda etapa da apresentação e discussão dos resultados foram realizados testes com o objetivo de verificar que a inclusão do firewall proposto entre a rede externa e interna traria algum distúrbio na performance de rede da topologia local.

4.2.1 Ping

O teste envolve primeiramente fazer um único *PING* entre a máquina local e um site externo (para teste de conectividade), e verificar se ocorreriam perdas ao se manipular os nós do *cluster*. A princípio, por não ser um ambiente controlado a partir do momento que se usam endereços externos para se fazer o teste, os resultados podem ser voláteis, entretanto para o cerne do trabalho é interessante se utilizar um cenário real, onde se torna importante verificar a estabilidade do tráfego mesmo sem total controle do ambiente. Não houveram problemas de perda de pacotes em nenhum momento, apenas algumas variações de latência no tempo de resposta, que rapidamente foram estabilizadas, conforme as Figuras 7, 8 e 9.

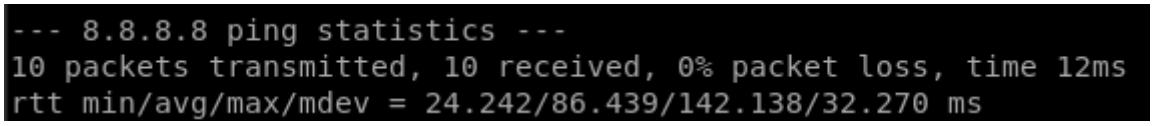
Figura 7 - Um PING sem manipular o *cluster*

```
--- 8.8.8.8 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 21ms
rtt min/avg/max/mdev = 20.495/44.053/167.501/41.970 ms
```

FONTE: Autoria própria

A latência média aferida onde o cenário permaneceu estável (sem interferências no não houve manipulação do *cluster* foi de 44 ms.

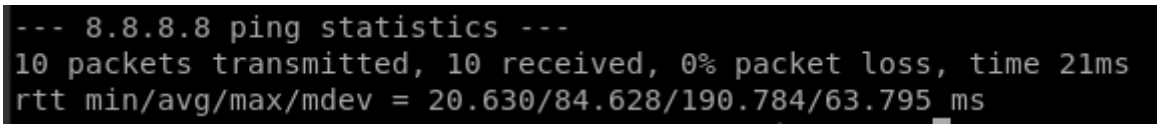
Figura 8 - Um *PING* manipulando a conexão de rede



FONTE: Autoria própria

A latência média aferida onde houve manipulação dos cabos de rede do *cluster* foi de 86 ms

Figura 9 - Um *PING* desligando os nós por comando e pela fonte de energia

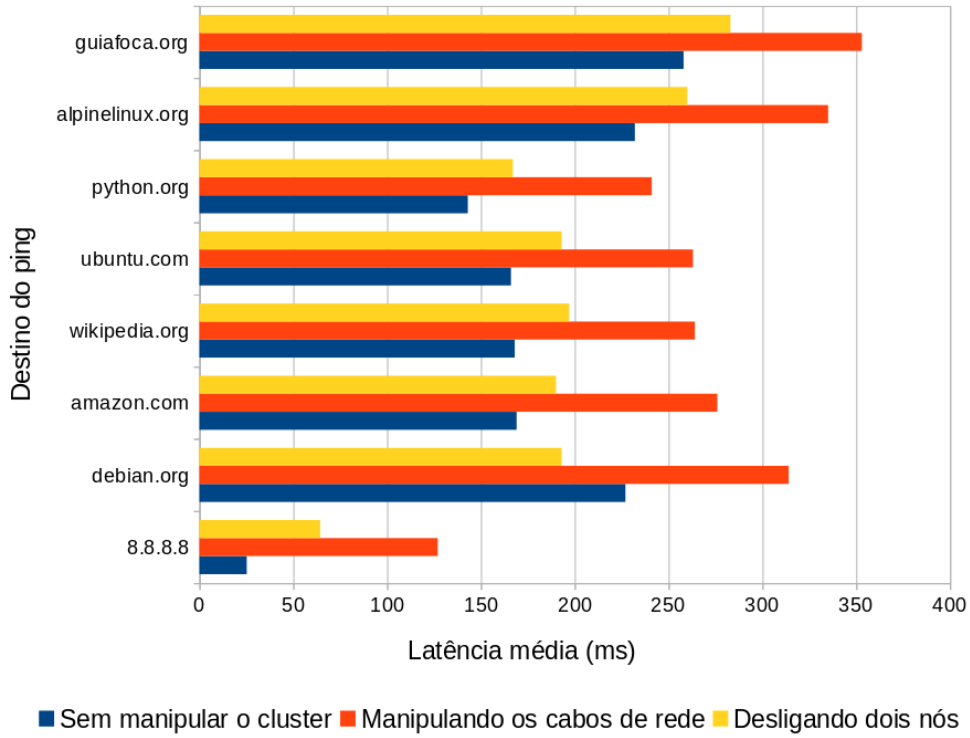


FONTE: Autoria própria

A latência média aferida quando houve o desligamento de dois dos nós do *cluster* foi de 85 ms.

Também foram realizados testes executando simultaneamente vários *PINGs* em diferentes sites. Para se realizar o teste foi utilizado o programa Fping, que permite que a aplicação seja realizada para múltiplos destinos em um único comando. O resultado foi similar ao teste com um único *PING*, os valores medidos constam no gráfico da Figura 10.

Figura 10 - Gráfico relacionando os testes de múltiplos PINGs simultâneos



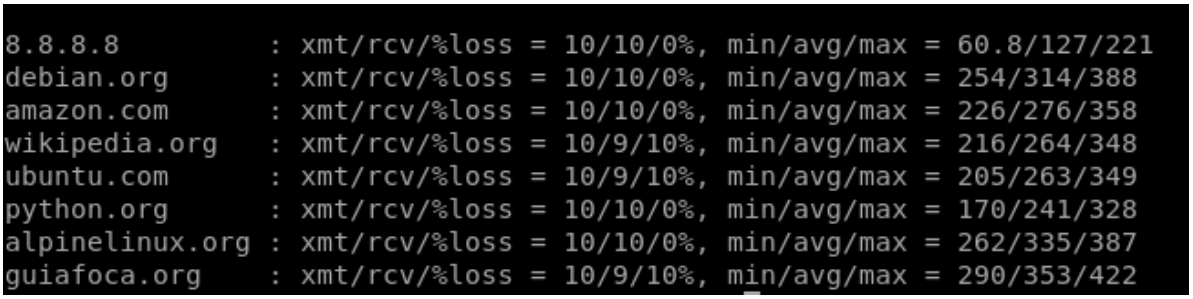
FONTE: Autoria própria

Na Figura 10, o gráfico em barras foi construído com base nos resultados de testes de múltiplos PINGs simultâneos. No eixo vertical consta quais os destinos dos *PINGs* executados, no eixo horizontal consta a latência média entre uma sequência de 10 pacotes enviados. Na cor azul é o resultado da latência executando os *PINGs* com o funcionamento padrão do sistema (sem interferência externa), na cor vermelha consta o resultado ao se remover o cabo de rede de um nó e desligando a porta do *switch* de outro, e em amarelo consta o resultado ao se desligar dois nós, sendo um por comando e outro removendo a fonte de energia abruptamente. É esperado que para cada destino se tenha um tempo de resposta diferente entre eles, por isso é necessário comparar os resultados dos testes entre os mesmos destinos. Avaliando os resultados totais do gráfico é possível observar que a melhor latência ocorre quando o *cluster* não é manipulado, e também que remoção ou desligamento da rede lógica piora o tempo de resposta em comparação com o processo de desligar parte do cluster. Tendo em vista que os testes foram realizados apontando os PINGs para destinos externos (da internet) onde não há controle de

desempenho, mesmo com o aumento de latência medido e um pacote perdido, os resultados se mantiveram estáveis.

No mesmo teste de múltiplos *PINGs* também foi verificado se houveram pacotes perdidos, o que apenas ocorreu durante o segundo teste (manipulação dos cabos de rede), conforme a Figura 11.

Figura 11 - PING para múltiplos destinos com manipulação dos cabos de rede



FONTE: Autoria própria

Conforme visto na Figura 11, houve a perda de dois pacotes, sendo um para o destino “*wikipedia.org*” e outro para o destino “*ubuntu.com*”. Esse resultado é um complemento ao resultado lido no gráfico da Figura 10, tendo em vista que o teste de remoção de cabos teve uma latência menor do que o de desligamento de nós, entretanto houve alguma perda de pacotes no processo.

4.2.2 Largura de banda

Para se realizar os testes de velocidade foram utilizados servidores web para validação da conexão, chamados comercialmente de “*speedtest*”. Foi verificado que conexão estava sempre abaixo de 10 Mbps, sendo que fazendo o mesmo teste a partir do roteador a conexão ficava em torno de 40 Mbps, foi verificado que as interfaces RJ45-USB usadas nos *Raspberries* estavam negociando a apenas 10 Mbps com o *switch*. Após a realização da aquisição de outro conversor com maior

capacidade de negociação, e ao usá-lo em um dos nós e desligar os outros (para que este seja o único dispositivo disponível), a largura de banda aferida passou a ser semelhante a do roteador, com 40 Mbps, conforme verificado no quadro 2:

Quadro 2 - Largura de banda medida com interfaces diferentes

Teste de banda com interface negociando a 10Mbps	Teste de banda com conector negociando a 100Mbps
Ping: 38.056 ms *Download*: 5.41 Mbit/s *Upload:* 2.86 Mbit/s	Ping: 26.616 ms *Download:* 44.17 Mbit/s *Upload:* 7.18 Mbit/s

FONTE: Autoria própria

Tendo em vista que a banda atingiu cerca de 40 Mbps tanto se utilizando o cluster quanto se testando direto no roteador, se conclui que a inclusão do *firewall* não impactou na largura de banda da rede local. Considerando testes de largura de banda limitando a velocidade de 10 Mbps negociada, e ao se remover os nós da rede, enquanto pelo menos um dos nós esteve ligado, as medições foram sempre semelhantes.

4.2.3 Acesso web

Para simular o uso real de uma rede local, tendo múltiplos usuários acessando vários serviços ao mesmo tempo, foi utilizada a aplicação JMeter, que foi desenvolvida para realizar testes de performance em páginas WEB.

Na primeira tentativa de configuração foi montado um plano de teste no jMeter onde seriam simulados 50 usuários acessando três páginas web a cada 5 segundos, entretanto os próprios *sites* possuem medidas de segurança para evitar tantas requisições de um mesmo local, onde depois de alguns minutos todas as conexões passaram a ser recusadas. Diversas configurações foram realizadas, as

únicas que não acabaram por resultar nos *sites* recusando acesso tinham um espaçamento de aproximadamente 30 segundos entre cada requisição, e com menos usuários. Como o intuito é ter um teste rápido para simular acessos pesados que explorem ao máximo o *cluster*, o uso de *sites* externos não se adequou à proposta.

Para realizar o teste e ainda ter escalabilidade de serviço *web*, foi instalado um servidor web acima do *cluster*, e nele instalado uma página html, e justamente por ser um servidor próprio não foi configurado nenhum recurso de segurança (pois tal situação é limitada nos sites externos utilizados inicialmente) foi possível simular o estresse de múltiplas requisições HTTP.

Para o teste foi configurado no JMeter um grupo de 150 usuários, executando 100 acessos cada para o *site* instalado acima do *cluster*, a execução durou 24 segundos. Sem manipular nenhuma parte do *cluster* não houveram erros de requisição. Na Figura 12 consta o resultado.

Figura 12 - Requisições HTTP sem manipulações do cluster

Label	# Samples	Average	Median	90% Line	95% Line
HTTP Request	25000	39	33	70	87
TOTAL	25000	39	33	70	87

99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
139	3	1146	0.00%	1015.8/sec	3641.77	114.08
139	3	1146	0.00%	1015.8/sec	3641.77	114.08

Fonte: Autoria própria, utilizando os *software* JMETER

Para o segundo teste da sequência foi iniciado o processo de remoção de cabos, enquanto o JMeter era executado os cabos de rede foram removidos um a um e retornados aos nós. O resultado consta na Figura 13.

Figura 13 - Requisições HTTP removendo os cabos de rede do *cluster*

Label	# Samples	Average	Median	90% Line	95% Line
HTTP Request	25000	277	113	273	1033
TOTAL	25000	277	113	273	1033

99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
3294	3	31623	0.08%	498.5/sec	1786.62	55.94
3294	3	31623	0.08%	498.5/sec	1786.62	55.94

Fonte: Autoria própria, utilizando os software JMETER

Como consta na Figura 13 a porcentagem de erro foi de apenas 0,08%, representando as perdas de requisições HTTP durante o processo, algo que para o uso real não seria notável ao usuário final.

Para o último teste dois nós foram desligados, um deles por comando e outro removendo abruptamente a fonte de energia, os resultados constam na Figura 14.

Figura 14 - Resultado de requisições HTTP desligando nós do *cluster*

Label	# Samples	Average	Median	90% Line	95% Line
HTTP Request	25000	318	110	214	484
TOTAL	25000	318	110	214	484

99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
7204	3	31625	0.18%	488.9/sec	1751.89	54.81
7204	3	31625	0.18%	488.9/sec	1751.89	54.81

Fonte: Autoria própria, utilizando os *software* JMETER

Conforme a imagem 14 a taxa de erro foi de 0,18%, ainda assim é uma taxa baixa para ser percebida pelo usuário final. Esse tipo de serviço acessado pelo usuário é classificado como *best effort* (melhor esforço), que por definição considera a concorrência de fluxos trabalhando para que os pacotes sejam entregues por completo (KUROSE; ROSS, 2006), não impactando na percepção do usuário.

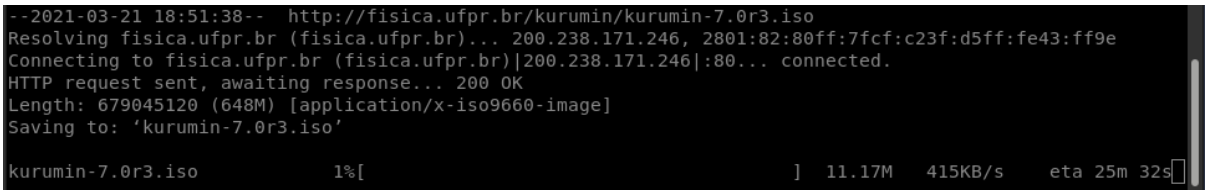
4.2.4 *Download* de um arquivo

O teste consiste em realizar o *download* de um arquivo grande e remover nós do *cluster* durante o processo, ao final do processo verificar que não houveram perdas e o arquivo foi recebido de maneira íntegra.

Foi feito o *download* do arquivo de instalação do sistema Linux Kurumin do servidor da UFPR. O arquivo possui cerca de 700 MB e o servidor possui a informação de *hash* md5, essa *hash* é um valor gerado com base no arquivo, independente do computador onde a *hash* seja gerada, ela terá o mesmo valor, assim será possível aferir que o arquivo estará íntegro.

O *download* foi realizado conforme a Figura 15, durante o processo um dos nós foi desligado e ligado da tomada, um foi reiniciado por linha comando, após esses retornarem foi retirado o cabo de rede de um e desligado na porta do *switch* por comando em outro.

Figura 15 - *Download* do arquivo de instalação do Linux Kurumin



Fonte: Autoria própria

Após as manipulações e o download finalizado, foi utilizado o programa md5sum e comparada a *hash* gerada com a que consta no *site* da UFPR, concluindo que ambas são idênticas, conforme a Figura 16 abaixo.

Figura 16 - *Hash* md5

```
mma@PENSADOR-PROFUNDO:~/tcc_logs$ md5sum kurumin-7.0r3.iso
44e6ea85b57dc0757ebf537cbdd4daea  kurumin-7.0r3.iso
mma@PENSADOR-PROFUNDO:~/tcc_logs$ cat kurumin-7.0r3.md5sum.txt
44e6ea85b57dc0757ebf537cbdd4daea  kurumin-7.0r3.iso
mma@PENSADOR-PROFUNDO:~/tcc_logs$
```

FONTE: Autoria própria

Em sequência na Figura 16, na segunda linha é o resultado da hash md5 ao se aplicar o comando md5sum no arquivo que foi realizado o *download*, e na quarta linha o conteúdo disponibilizado em texto pelo servidor da UFPR. Os resultados obtidos são os mesmos, mostrando que não houveram perdas no *download* durante as perdas de conexão com alguns nós do *cluster*.

4.2.5 *Streaming* de áudio e vídeo

O último teste de performance envolve verificar se os serviços de *streaming "online"* se mantêm estáveis durante a manipulação dos nós.

Usando a plataforma de *software* livre Jitsi, foi realizada uma chamada e feitas as manipulações das conexões. Durante o processo de transmissão de áudio e vídeo houveram muitos travamentos e em alguns momentos uma demora de recepção de quatro segundos para recepção do som principalmente, independente das remoções de cabo de rede e desligamento de nós. A plataforma Jitsi possui uma notificação de qualidade de conexão, conforme a Figura 17.

Figura 17 - Qualidade da conexão "não ótima"

Fonte: Autoria própria, utilizando a plataforma Jitsi

Como verificado na Figura 17, a própria plataforma acusou a qualidade da conexão, apontando que não está conforme o esperado para uma transmissão fluida. Entretanto é importante ressaltar que, como informado no capítulo 5.2.2, as interfaces de rede USB usadas possuíam a taxa de negociação de 10 Mbps (que atingiu aproximadamente 3 Mbps no teste de largura de banda), não possibilitando uma conexão mais rápida. Foi realizada a conexão de áudio e vídeo usando apenas um nó com a interface USB-RJ45 que negociou a 100 Mbps, e com ela a transmissão foi satisfatória, mas por ser apenas uma peça não foi possível simular a quebra de conexão e desligamento de nós com uma transmissão satisfatória. Sendo assim é possível supor que tendo mais interfaces suportando a taxa de 100 Mbps a transmissão melhoraria, mas esse teste fica pendente pela falta de material.

4.3 APLICAÇÕES DE *FIREWALL*

Nesta etapa da discussão dos resultados foram realizados os testes específicos da aplicação das ferramentas de firewall, visando verificar se elas funcionam como esperado na topologia em cluster conforme apresentado.

4.3.1 Criação de regras no Iptables

O primeiro teste foi um bloqueio simples de pacotes do protocolo ICMP (usado no PING), a regra foi aplicada conforme o *script* mostrado no Apêndice H. Para validar que todos os nós possuíam a regra, foi verificado um a um com o comando “iptables -L” que a regra de descarte constava. Foi realizado o PING no computador da rede local para a rede externa, os pacotes de fato foram filtrados. Também foram removidos os cabos de rede de um nó e desligado outro, os pacotes continuam sendo filtrados.

Uma regra de segurança comum é impedir que computadores externos consigam acessar a gerência de equipamentos na rede interna, então foi criada outra regra que impede o acesso via SSH e Telnet pelo *firewall*. Para validar foi iniciada a tentativa de um acesso SSH do roteador para uma máquina da rede local, e ele de fato foi bloqueado. Também foram removidos os cabos de rede de um nó e desligado outro, os acessos continuaram sendo bloqueados. Para averiguar se o bloqueio realmente veio do *cluster* a regra foi removida e o acesso foi possível.

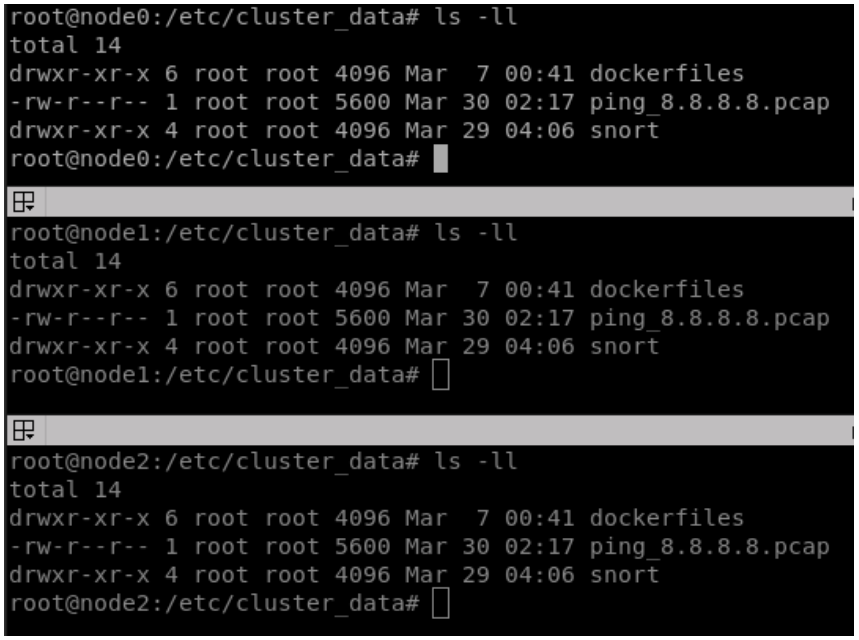
4.3.2 Captura de pacotes no *Cluster*

Os testes consistem em realizar capturas de pacotes usando filtros de regras específicas e verificar principalmente se ao realizar o processo entre múltiplos equipamentos sincronizados alguma informação de tráfego seria perdida durante o desligamento de parte dos nós.

Para fazer o teste foi criada uma regra no tcpdump para capturar os pacotes ICMP da rede, as capturas foram direcionadas para um arquivo dentro da pasta compartilhada entre os nós do *cluster*. Foi iniciado um PING configurado para enviar 100 pacotes de uma máquina da rede local para a rede externa. Após realizar o processo, em um nós foi removido o cabo de rede fisicamente e em outro desligada a porta do *switch* por comando, em seguida feito o desligamento forçado de um dos nós e a reinicialização por comando em outro.

Após todas as manipulações o serviço criado para capturar os pacotes foi finalizado (quando o comando é aplicado em um dos nós, todos finalizam o serviço simultaneamente), a primeira verificação foi o tamanho do pacote, mostrado na Figura 18.

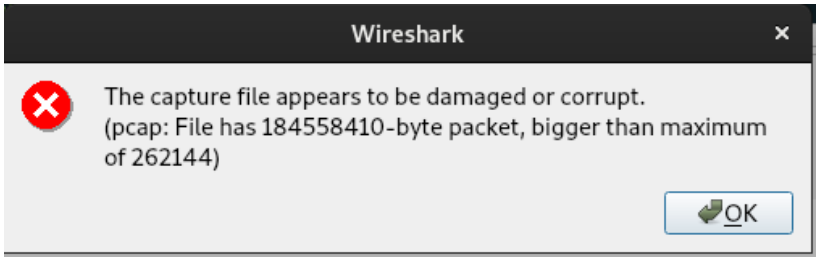
Figura 18 - Verificação da captura de pacotes em todos os nós do *cluster*



Fonte: Autoria própria

Conforme a Figura 18 é possível verificar que o arquivo “ping_8.8.8.8.pcap” possui o mesmo tamanho nos três nós do *cluster*. Em seguida o arquivo foi removido de um dos nós para uma máquina na rede local. Primeiro foi verificado usando o próprio leitor do tcpdump (em linha de comando) que os pacotes estavam presentes no arquivo. Após verificar o arquivo a partir do próprio *cluster*, o arquivo de captura foi exportado para um computador da máquina local para se analisar o log usando interface gráfica com o *software* Wireshark. Ao verificar o arquivo o programa apresentou um erro conforme a Figura 19 abaixo.

Figura 19 - Erro ao abrir o arquivo de captura no Wireshark

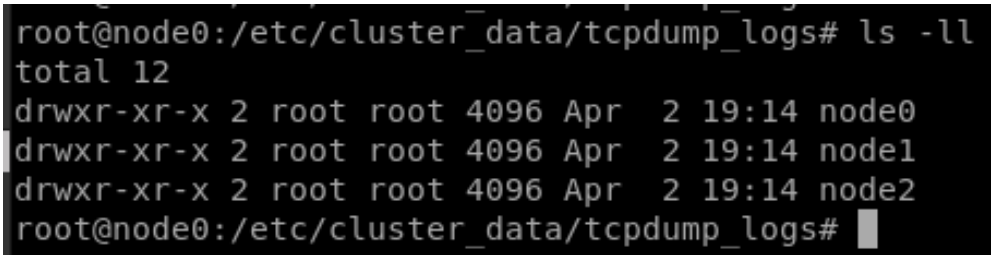


Fonte: Autoria própria

Após a realização de mais testes foi verificado que, ao se criar um arquivo pelo tcpdump tendo várias origens, diversas partes são corrompidas gerando pacotes mal formados.

Tendo em vista o erro encontrado anteriormente, procurando uma alternativa para que os arquivos não sejam corrompidos, a maneira de criar e processar o tcpdump foi alterada. Foram criadas pastas de logs diferentes para cada nó na pasta compartilhada do *cluster*, conforme a Figura 20.

Figura 20 - Pastas de logs do tcpdump separadas por nó

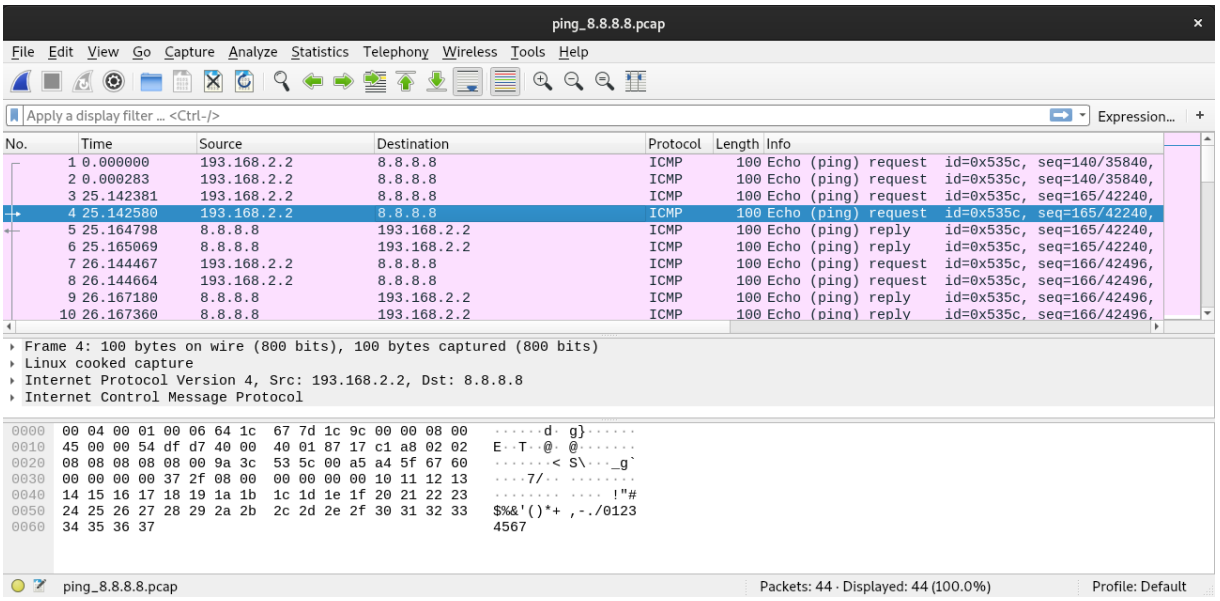


Fonte: Autoria própria

Na nova lógica, cada imagem do *Docker* que roda o tcpdump cria os registros dentro de uma pasta separada, sendo assim para uma captura se terão três arquivos diferentes, organizadas na pasta compartilhada por pastas, que não serão corrompidos e ainda atestaram uma qualidade melhor de análise, pois mostrará exatamente que tráfego passou por cada nó do *cluster*. Ao repetir os testes feitos anteriormente, nenhum arquivo foi corrompido, e mesmo no caso de desligar o nó,

esse arquivo de *log* continua acessível pelos outros *raspberries* porque a pasta estaria compartilhada entre eles. Na Figura 21 é possível ver um dos arquivos abertos externamente pelo wireshark sem apresentar erros.

Figura 21 - Captura pelo tcpdump verificada pelo Wireshark



Fonte: Autoria própria

Foi realizado também um teste de stress na captura de pacotes para avaliar a performance do *cluster*. Para avaliar foi criado um serviço no tcpdump para o *cluster* capturar todos os pacotes trafegados e enviar para um arquivo na pasta compartilhada. Para gerar a carga foram iniciados serviços de *streaming* na rede local e eles executaram durante aproximadamente 8 horas.

O *cluster* conseguiu processar esses pacotes durante todo o período especificado e o resultado foi satisfatório, entretanto foi identificado um provável problema em relação a armazenamento, o arquivo gerado pelo teste possuía o tamanho próximo de 3 GB, e a memória de cada nó tem apenas 4GB, caso o teste tivesse durado mais tempo provavelmente teria ocupado todo o armazenamento disponível.

Para resolver essa situação existem duas possibilidades, a primeira seria a utilização de cartões de memória maiores em cada *raspberry*. A segunda envolve a criação de um repositório externo ao *cluster* para armazenamento dos registros, entretanto nesse caso a banda utilizada para a rede local seria compartilhada para esse envio dos *logs*. Para a execução deste trabalho de conclusão de curso foram utilizados dispositivos de armazenamento de 4GB, considerando a adoção como prova de conceito.

4.3.3 Funcionamento de aplicação IDS

Conforme o Apêndice E descreve, foi criado com o *Docker* um serviço no *cluster* executando o *snort* (IDS escolhido), a aplicação foi configurada para gerar *logs* com qualquer pacote ICMP trafegando nas interfaces de rede do *cluster*. O *snort* cria um arquivo chamado *"alert"* para imprimir os dados que foram identificados pelo seu filtro. Após iniciar o serviço entre os nós do *cluster* e um PING entre um computador da máquina local e um IP da rede externa o arquivo *"alert"* gerado foi lido e verificado que as mensagens ICMP estão sendo identificadas pelo serviço do snort, conforme a Figura 22 abaixo.

Figura 22 - Leitura do arquivo *"alert"* gerado pelo snort

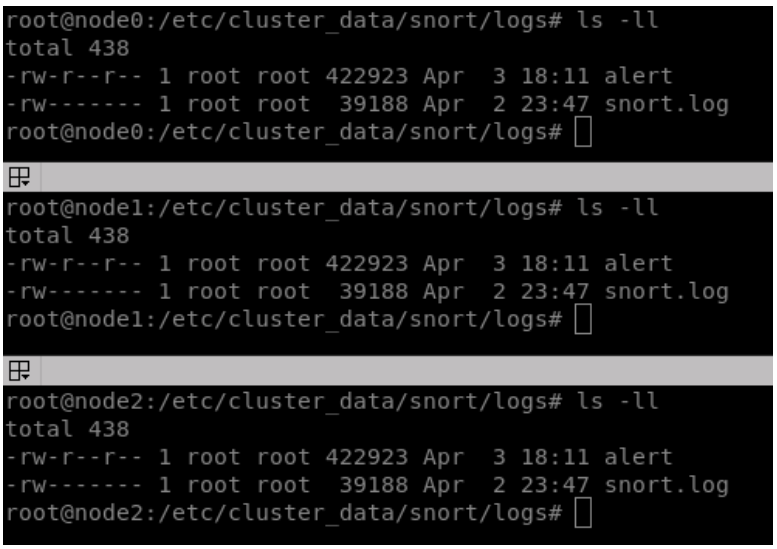
```
root@node1:/etc/cluster_data/snort/logs# tail  alert
04/03-17:08:00.995468  [**] [1:1000001:0] ICMP Packet found [**] [Priority: 0] {ICMP} 8.8.8.8 -> 193.168.2.2
04/03-17:08:00.995468  [**] [1:10001:0] ICMP Echo Reply [**] [Priority: 0] {ICMP} 8.8.8.8 -> 193.168.2.2
04/03-17:08:01.969333  [**] [1:1000001:0] ICMP Packet found [**] [Priority: 0] {ICMP} 193.168.2.2 -> 8.8.8.8
04/03-17:08:01.969333  [**] [1:10000:0] ICMP Echo Request [**] [Priority: 0] {ICMP} 193.168.2.2 -> 8.8.8.8
04/03-17:08:01.989104  [**] [1:1000001:0] ICMP Packet found [**] [Priority: 0] {ICMP} 8.8.8.8 -> 193.168.2.2
04/03-17:08:01.989104  [**] [1:10001:0] ICMP Echo Reply [**] [Priority: 0] {ICMP} 8.8.8.8 -> 193.168.2.2
04/03-17:08:02.970943  [**] [1:1000001:0] ICMP Packet found [**] [Priority: 0] {ICMP} 193.168.2.2 -> 8.8.8.8
04/03-17:08:02.970943  [**] [1:10000:0] ICMP Echo Request [**] [Priority: 0] {ICMP} 193.168.2.2 -> 8.8.8.8
04/03-17:08:02.995553  [**] [1:1000001:0] ICMP Packet found [**] [Priority: 0] {ICMP} 8.8.8.8 -> 193.168.2.2
04/03-17:08:02.995553  [**] [1:10001:0] ICMP Echo Reply [**] [Priority: 0] {ICMP} 8.8.8.8 -> 193.168.2.2
root@node1:/etc/cluster_data/snort/logs#
```

Fonte: Autoria própria

Fazendo os testes de performance, durante o funcionamento do serviço e do envio de pacotes ICMP foi novamente retirado um cabo de rede de um dos nós, desligado outro por comando no *switch*, retornados eles a posição original, e em sequência desligado um *raspberry* da fonte de energia e reiniciado outro por comando.

Após todos os nós retornarem foi verificado na pasta compartilhada que o arquivo de registro *“alert”* ainda pode ser lido de maneira correta, e possui o mesmo tamanho em todos os nós, como visto na Figura 23.

Figura 23 - Verificação do arquivo “alert” nos múltiplos nós do *cluster*



Fonte: Autoria própria

4.4 RESULTADOS OBTIDOS

O Quadro 3 apresenta os resultados obtidos ao longo dos testes realizados, os testes contemplam tanto o funcionamento sem interferências quanto manipulando o cluster para simular estados de falha em parte dos nós, além de sempre validar se os registros criados foram mantidos com persistência ao usar a escalabilidade de armazenamento do GlusterFS, conforme descrito anteriormente neste capítulo.

Quadro 3 - Resumo dos testes realizados

Teste	Resultado	Observações
Ping da rede local para rede externa	Aprovado	Houveram resultados satisfatórios com baixa diferença de latência durante a interferência nos nós, tanto no teste do ping para um único destino quanto para múltiplos
Largura de banda aferida	Aprovado / Incompleto	Os testes foram impactados devido à negociação física das interfaces RJ45-USB. Tendo apenas uma interface que comporta a banda disponibilizada no *uplink* foi possível verificar que a velocidade alcançada abaixo do cluster foi a mesma alcançada pelo roteador acima dele. Para realizar os testes com interferência externa nos nós (se limitando a 10Mbps) não houveram perdas significativas durante os testes
Escalabilidade de acessos web	Aprovado	O cluster suportou uma grande quantidade de acessos a página web (150 usuários fazendo 100 requisições HTTP cada um) sem perdas quando o cluster operava sem interferência externa. E houveram perdas menores que 0,2% nos casos com interferência nos nós
Download de um arquivo	Aprovado	O escopo deste teste envolvia o download de um arquivo grande. O processo foi realizado e mesmo com manipulando os nós para interferir no seu funcionamento a integridade do arquivo no final foi mantida
Streaming de áudio e vídeo	Aprovado/ Incompleto	O teste foi afetado devido a limitação física das interfaces RJ45-USB. Quando se utilizou uma interface que negocia a uma taxa maior (100Mbps) o resultado foi satisfatório, mas quando limitado a 10 Mbps a qualidade aferida em uma chamada de áudio e vídeo foi baixa
Criação de regras do Iptables	Aprovado	Todas as regras criadas no cluster foram efetivas independente da interferência aplicada no sistema
Captura de pacotes no cluster	Aprovado	O sistema criado com *containers* do Docker foi efetivo para realizar as capturas dos pacotes trafegados (usando o tcpdump) mesmo com interferências no *cluster*
Funcionamento da aplicação IDS	Aprovado	O funcionamento das regras criadas pelo Snort não foram afetadas durante as

		interferências nos nós do *cluster*.

FONTE - Autoria própria

Os testes realizados possuíam o objetivo de operacionalizar o *firewall* proposto, bem como demonstrar sua performance e o devido funcionamento das ferramentas escolhidas para compor o sistema. O Quadro 3 acima resume os testes realizados após a montagem do cenário, para os casos de performance. Houveram, por exemplo, problemas referentes a uma peça adquirida (interface RJ-45) mas que podem ser solucionadas com a troca desse componente. Referente a compatibilidade com as ferramentas de segurança em operação o sistema foi funcional, também não houveram falhas quanto a robustez do *firewall* em manter os serviços em operação mesmo com perda parcial dos componentes do cluster.

Mostrando assim que na maior parte dos casos as escolhas realizadas para o sistema foram efetivas, e nos casos de falha existem alternativas para a correção dos problemas.

5 CONCLUSÃO

O estudo desenvolvido demonstrou que existem no mercado diversas tecnologias de baixo custo que podem ser utilizadas para implementar um sistema de alta disponibilidade focado em segurança de redes.

A união da interface de linha de comando com as configurações de *cluster* dos *softwares* Docker e GlusterFS permitiram uma integração otimizada entre os componentes em *cluster*, mesmo tendo arquiteturas diferentes, foi possível operar sem grandes problemas de compatibilidade, esses poucos que foram contornados.

A proposta considerava pequenas redes locais como objetivo final de aplicação, nesses casos existe uma entrada de rede externa onde é feito o roteamento para a rede local, e este ponto é onde o *cluster* disponibilizado será aplicado.

Tendo em vista que a proposta inicial sugere ter um sistema transparente a rede local que fizesse a função de analisador e filtro de pacotes, é possível afirmar que os resultados foram satisfatórios ao criar um *cluster* de baixo custo que operou com os testes propostos. Durante a prova de conceito realizada foram utilizados três nós com *raspberries*, mas em uma rede real isso poderia ser ampliado para uma quantidade maior de pontos, tendo assim uma capacidade escalável de atender redes locais com mais usuários.

Um ponto importante é que o *cluster* foi projetado para operar de maneira passiva referente ao tráfego recebido, sendo assim ele depende de um equipamento externo que faça o balanceamento de distribuição de pacotes. No caso do presente trabalho foi utilizado um *switch* de camada 2 com a *feature* de *link-aggregation* que já possui balanceamento nativo.

Todos os testes realizados tinham o objetivo de validar o ambiente em seu estado “normal” e também interferindo em seu funcionamento para simular problemas de operação como perda de conexão em alguns dos nós e o desligamento deles, como por falha de alimentação ou mesmo a queima de um dos

equipamentos. Os testes além de validar que os serviços continuavam funcionando nas quebras, também mediu o quanto de impacto que o sistema teria em operação. Esses testes foram baseados em serviços comumente utilizados por usuários de uma rede local, como acesso web e streaming de áudio e vídeo. Alguns dos testes também consideraram a escalabilidade de acessos, simulando uma utilização provável de múltiplos usuários abaixo do *firewall* disponibilizada. Esses testes em sua maioria tiveram sucesso e apresentaram uma boa performance, as únicas ressalvas são quanto aos adaptadores a interfaces de rede, essas que podem ser trocadas por outras mais robustas em uma futura instalação em um cenário real.

Para o funcionamento de segurança de redes houveram testes focados em filtragem de pacotes (utilizando o Netfilter e Iptables), um sistema de IDS (utilizando o snort) e o processo de captura de pacotes para análise de tráfego (utilizando o tcpdump). E para os três casos o funcionamento foi satisfatório ao ser aplicado em cluster, todos os nós operaram em conjunto para realizar as aplicações testadas. Esses testes garantiram que as aplicações de segurança são compatíveis com a proposta do trabalho.

Um dos principais pontos abordados na análise do problema é a falta de modularidade nos sistemas de *firewall* existentes. Com a proposta desenvolvida foi criada uma plataforma onde diferentes aplicações podem ser encapsuladas para trabalhar em cluster, tendo um funcionamento personalizado a cada topologia. Possuindo um sistema genérico a ser implementado de acordo com cada necessidade.

Com isso se conclui que, com base nos objetivos e testes realizados, o *firewall* em *cluster* utilizando *softwares open-source* atenderia de forma satisfatória as necessidades de segurança de uma rede local para pequenas instituições.

O trabalho foi desenvolvido com o foco na aplicação de um *firewall* de borda em pequenas instituições. Entretanto devido a sua modularidade aferida pelo uso de *containers* e ferramentas *open-source*, e também do uso de hardware de baixo custo. É possível prever que sua utilização pode ser facilmente adaptada para cenários com foco em IoT, cidades inteligentes, monitoramento de tráfego e demais

sistemas de redes que utilizam hardwares com baixo uso de energia elétrica e precisam de um alto desempenho com pouco custo.

5.1 SUGESTÕES PARA TRABALHOS FUTUROS

Para trabalhos futuros que venham a seguir na mesma linha de pesquisa, pode ser interessante estudar outros *hardwares*, que sejam diferentes de computadores de placa única, como utilizar computadores “normais” que sejam antigos para aplicações comuns mas que possam ser reutilizados num *cluster* de segurança, a arquitetura desenvolvida permitiria isso.

Também se sugere que busquem por alternativas ao Docker como ferramenta para encapsular os *softwares* de segurança, é possível continuar na linha de *containers* com ferramentas como o Podman, Rocket e LXD. Além de buscar alternativas a orquestração dos *containers* com aplicações como o Kubernetes, que pode trazer mais complexidade ao funcionamento atual do *cluster*.

Também é possível eliminar o uso dos *containers* do projeto, se utilizando *scripts* para orquestrar as ferramentas de segurança, onde assim elas possam operar de maneira nativa nos equipamentos, trazendo assim vantagens e desvantagens a serem estudadas.

Outro ponto seria estudar as outras topologias (além de um firewall de borda) onde que o trabalho se aplicaria.

E por fim também é possível, com base no baixo armazenamento disponível nos *raspberries*, estudar alternativas de repositório externo ao *cluster* para guardar registros coletados durante a aplicação dos serviços de segurança.

REFERÊNCIAS

about | **Alpine Linux**. Disponível em: <https://www.alpinelinux.org/about>. Acesso em: 19 de Jul. 2020.

Banana Pi. Disponível em: <http://www.banana-pi.org/>. Acesso em: 17 de Nov. 2020.

BARBOSA, T. S.; REIS, F. A. O uso de Ferramentas Open Source para Aplicações de Segurança em Redes Corporativas: Um estudo baseado em Firewalls. **Saber Digital**, v. 5, n. 01, p. 72-90, 2018.

BeagleBoard.org - about, **BeagleBoard**. Disponível em: <https://beagleboard.org/about>. Acesso em: 17 de Nov. 2020.

CAMPOS, A. **O que é uma distribuição Linux**. BR-Linux. Florianópolis,março de 2006. Disponível em <http://br-linux.org/linux/faq-distribuicao>. Consultado em 12/06/2021.

CAPURRO, R.; HJORLAND, B. The concept of information as we use in everyday. **Perspectivas em ciência da informação**, v. 12, n. 1, p. 148-207, 2007.

DA SILVA, G. M. **Guia Foca GNU/Linux**. 1999.

Diferença entre empilhar, cascatear e fazer um cluster com Switch Cisco | **DIteC do Brasil**. 2014 Disponível em: <http://www.dltec.com.br/blog/cisco/diferenca-entre-empilhar-cascatear-e-fazer-um-cluster-com-switch-cisco/>. Acesso em: 15 de Out. 2020.

EMILSSON, R. **Container performance benchmark between Docker, LXD, Podman & Buildah**. 2020.

FERREIRA, F.; SANTOS, N.; ANTUNES, M. **Clusters de alta disponibilidade – abordagem OpenSource**. Relatório desenvolvido na disciplina de Projecto I, 2005.

FILHO, J. E. M.. **Análise de Tráfego em Redes TCP/IP: Utilize tcpdump na análise de tráfegos em qualquer sistema operacional**. Novatec Editora, 2013.

FILHO, J. E. M. **Descobrindo o Linux-3ª Edição: Entenda o sistema operacional GNU/Linux**. Novatec Editora, 2012.

FORSHAW, J. **Attacking network protocols: a hacker's guide to capture, analysis, and exploitation**. No Starch Press, 2018.

FREUND, G. P. **Redes de Computadores I**. Palhoça, UnisulVirtual, 2009.

GASQUE, K. C. G. D. Te**oria fundamentada: nova perspectiva à pesquisa exploratória**. 2007.

Gluster | Storage for your Cloud. Disponível em: <https://www.gluster.org/>. Acesso em: 19 de Jul. 2020.

GUPTA, A.; SHARMA, L. S. A categorical survey of state-of-the-art intrusion detection system-Snort. **International Journal of Information and Computer Security**, v. 13, n. 3-4, p. 337-356, 2020.

HENNESSY, J. L.; PATTERSON, D. A. **Organização e Projeto de Computadores: a interface hardware/software**. Elsevier Brasil, 2014.

HERTZOG, R.; MAS, R. **The Debian Administrator's Handbook: Debian Buster From Discovery To Mastery**. Freexian, 2019.

HUAMAN, P.; JOSEPH. **Metodología integral para evaluar el rendimiento de firewalls**. 2018

KAPLAN, B., DUCHON, D. **Combining qualitative and quantitative methods in information systems research: a case study. MISQuarterly**, v. 12, n. 4, p. 571-586, Dec. 1988.

KASTNER, W.; CSEBITS, C.; MAYER, M. Linux in factory automation? Internet controlling of fieldbus systems!. In: **1999 7th IEEE International Conference on Emerging Technologies and Factory Automation. Proceedings ETFA'99 (Cat. No. 99TH8467)**. IEEE, 1999. p. 27-31.

KLETTENBERG, J. **Segurança da informação: Um estudo sobre o uso da engenharia social para obter informações sigilosas de usuários de Instituições Bancárias**. 2016.

KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet**. São Paulo: Person, 2006.

LAUDON, K. C.; LAUDON, J. P. **Sistemas de informação gerenciais**. 5ª Edição. 2004.

LE-TRUNG, Q. A Container-Based Edge Computing System for Smart Healthcare Applications. In: **Industrial Networks and Intelligent Systems: 7th EAI International Conference, INISCOM 2021, Hanoi, Vietnam, April 22–23, 2021, Proceedings**. Springer Nature, 2021. p. 324.

MAVRIDIS, I.; KARATZA, H. Orchestrated sandboxed containers, unikernels, and virtual machines for isolation‐enhanced multitenant workloads and serverless computing in cloud. **Concurrency and Computation: Practice and Experience**, p. e6365.

netfilter/iptables project homepage - The netfilter.org project, **Netfilter**. Disponível em: <https://www.netfilter.org/>. Acesso em: 01 de ago. 2019.

POCINO, J. L. **Analysis and practice of micro-services deployments technologies based on containers**. 2021. Trabalho de Conclusão de Curso. Universitat Politècnica de Catalunya.

Raspberry Pi Documentation, **Raspberry**. Disponível em: <https://www.raspberrypi.org/documentation/>. Acesso em: 30 de Jul. 2020.

RUSLING, D. A. **The linux kernel**. 1999.

Ryuk Ransomware Forces Prosegur Security Firm to Shut Down Network | **Bleeping Computer**. 2019 Disponível em: <https://www.bleepingcomputer.com/news/security/ryuk-ransomware-forces-prosegur-security-firm-to-shut-down-network/>. Acesso em: 23 de Ago. 2020.

SEMIDÃO, R. A. M. **Dados, informação e conhecimento enquanto elementos de compreensão do universo conceitual da ciência da**

informação: contribuições teóricas. 2014.

SILVA, W. F. **Aprendendo Docker**. São Paulo, Novatec, 2016.

Snort - Network Intrusion Detection & Prevention System, **Snort**. Disponível em: <https://www.snort.org/>. Acesso em: 04 de ago. 2019.

SOPPELSA, F.; KAEWKASI, C. **Native Docker Clustering with Swarm**. Packt Publishing Ltd, 2016.

Teach, Learn, and Make with Raspberry Pi – Raspberry Pi, **Raspberry**. Disponível em: <https://www.raspberrypi.org/downloads/raspberry-pi-os/>. Acesso em: 19 de Jul. 2020.

YAMANOOR, N. S.; YAMANOOR, S. High quality, low cost education with the Raspberry Pi. In: **2017 IEEE Global Humanitarian Technology Conference (GHTC)**. IEEE, 2017. p. 1-5.

YU, T et al. PSI: Precise Security Instrumentation for Enterprise Networks. In: **NDSS**. 2017.

APÊNDICE A - CONFIGURAÇÕES INICIAIS DOS NÓS

Utilizando o sistema Debian (ou o Raspberry PI OS) é necessário atualizar os repositórios e instalar as aplicações com o gerenciador de pacotes apt, usando o seguinte *script* e possuindo permissão de root é possível instalar todos os pacotes de necessários que já são pré-compilados:

```
apt-get update && apt-get install -y curl apt-transport-HTTPS iproute2
net-tools iptables-persistent wget gnupg
```

A empresa responsável pelo Docker disponibiliza um *script* para a instalação, usando o pacote curl é possível baixar e executar ele na mesma linha de comando:

```
curl -fsSL HTTPS://get.docker.com | bash
```

Para instalar a aplicação do gluster é necessário instalar as chaves de permissão e cadastrar o repositório da aplicação no arquivo de fontes do sistema antes de fazer a instalação de fato

Para adicionar as chaves:

```
wget -O - HTTPS://download.gluster.org/pub/gluster/glusterfs/7/rsa.pub |
apt-key add -
```

Para adicionar o repositório é necessário observar qual o sistema está utilizando, caso seja um computador de arquitetura amd64 (como um desktop ou notebook comum) se deve adicionar a seguinte linha no arquivo de fontes do sistema:

```
echo deb [arch=amd64]
HTTPS://download.gluster.org/pub/gluster/glusterfs/7/LATEST/Debian/buster/amd
64/apt buster main > /etc/apt/sources.list.d/gluster.list
```

Caso o sistema um computador de arquitetura arm (como o raspberry PI e outros computadores de placa única) se usa a seguinte linha:

```
echo deb [arch=arm64]
HTTPS://download.gluster.org/pub/gluster/glusterfs/7/LATEST/Debian/buster/amd
64/apt buster main > /etc/apt/sources.list.d/gluster.list
```

Tendo as chaves instaladas e o repositório adicionado nas fontes do sistema o próximo passo é atualizar as fontes e instalar a aplicação, é possível fazer os dois simultaneamente com a seguinte linha de comando:

```
apt-get update && apt-get install glusterfs-server -y
```

Então para iniciar a aplicação é utilizado o comando:

```
systemctl start glusterd && systemctl enable glusterd
```

Para realizar as configurações de acesso entre os nós do *cluster* é necessário que seja permitido o acesso remoto como usuário de permissão máxima (root) as devidas alterações nos arquivos de funcionamento de ssh, com o seguinte comando é possível fazer isso sem usar editores de texto:

```
echo "PermitRootLogin yes" >> /etc/ssh/sshd_config && systemctl restart ssh
```

Para que a comunicação entre nós também ocorra de maneira segura e facilitada é importante que as chaves de acesso públicas sejam trocadas entre os nós, evitando que se tenha que digitar senhas a cada novo acesso.

Para gerar uma chave de acesso pública:

```
ssh-keygen -t rsa
```

Após gerar ela é necessário copiar para todos os nós, o comando abaixo deve ser replicado para todos os nós do *cluster*:

```
ssh-copy-id root@${NODE_IP}
```

Outro ponto importante é cadastrar os nomes dos nós entre eles, criando a associação IP por NOME em seus arquivos de comunicação, para fazer isso é necessário editar o arquivo /etc/hosts com o seguinte padrão:

```
IP    NOME_DO_HOST
192.168.150.10    node0
192.168.150.11    node1
192.168.150.12    node2
```

APÊNDICE B - CONFIGURAÇÕES DO CENÁRIO COM DOCKER

Para tratar os nós como *cluster* no Docker é necessário iniciar uma sessão do Docker Swarm, escolhendo um dos nós se executa nele o seguinte comando:

```
docker swarm init --advertise-addr $(hostname -I | cut -d " " -f 1)
```

Ao se executar a linha acima irá retornar o token usado para se adicionar mais nós no *cluster*, que também pode ser localizado com o comando:

```
Docker swarm join-token worker
```

Basta executar a saída do comando acima nos demais nós para formar o *cluster* via Docker, com o seguinte comando é possível verificar o status corrente de todos os nós ligados em *cluster*:

```
docker node status
```

Fazendo todas as configurações apresentadas no APÊNDICE B já é possível a criação de “docker services”, que são os responsáveis por executar as aplicações em *cluster*.

APÊNDICE C - SINCRONIZAÇÃO DE ARQUIVOS COM GLUSTER

O primeiro passo é criar o *Cluster* pela aplicação, é necessário aplicar o próximo comando em um dos nós do *cluster*, mas repetir o comando com a quantidade de nós presentes no *cluster*

```
gluster peer probe $NODE_NAME
```

Após aplicar os comandos é possível verificar o status dos nós do *cluster* com o seguinte comando.

```
gluster pool list
```

Após todos os nós estarem configurados é necessário criar e iniciar um volume para sincronizar entre os dispositivos, usando o seguinte comando para associar os nós a esse volume, no exemplo o nome do volume é ${VOLUME_NAME}, a quantidade de nós é 3, seus nomes são node0, node1 e node2 e por fim o diretório do volume está representado por ${VOLUME_DIRECTORY}:

```
gluster volume create ${VOLUME_NAME} transport tcp replica 3
node0:${VOLUME_DIRECTORY} node1:${VOLUME_DIRECTORY} node2:${VOLUME_DIRECTORY}
force
```

Então é necessário iniciar o volume com o seguinte comando:

```
gluster volume start ${VOLUME_NAME}
```

Em todos os nós será necessário montar o disco em uma pasta que será mapeada pelo sistema, primeiro então é preciso criar essa pasta com o comando abaixo:

```
mkdir -p /etc/${VOLUME_NAME}
```

Para não apenas montar o disco mas também fazer com que ela seja persistente, sendo montada automaticamente em todas as inicializações dos nós é necessário adicionar as informações no arquivo de discos do sistema. Usando os seguintes comandos:

```
echo 'localhost:/${VOLUME_NAME} /etc/${VOLUME_NAME} glusterfs
defaults,_netdev 0 0' >> /etc/fstab
```

```
mkdir /etc/systemd/system/srv.mount.d/
```

```
echo "[Unit]" >> /etc/systemd/system/srv.mount.d/override.conf
```

```
echo "After=glusterfs-server.service" >>
/etc/systemd/system/srv.mount.d/override.conf
```

```
echo "Wants=glusterfs-server.service" >>
/etc/systemd/system/srv.mount.d/override.conf
```

Será necessário reiniciar cada nó duas vezes nesse momento para entrar em operação corretamente:

Primeira reinicialização:

```
systemctl reboot
```

Após o boot completo, a segunda reinicialização:

```
systemctl reboot
```

APÊNDICE D - CONSTRUÇÃO DE IMAGENS DOCKER COM DOCKERFILES

Para se encapsular as aplicações a serem utilizadas nos *containers* foram utilizados Dockerfiles, o sistema base escolhido para todos foi o Alpine pelo seu baixo consumo de recursos físicos.

Abaixo um modelo de Dockerfile usado para a aplicação do Snort:

```
FROM alpine:latest

MAINTAINER @mateusmantoan

LABEL Description="snort IDS"

RUN apk add -u snort && rm -f /var/cache/apk/*
```

No código acima existem apenas quatro parâmetros, o primeiro FROM define o sistema base a ser utilizado, no caso o Alpine. O parâmetro MAINTAINER é a declaração do responsável pelo código. O LABEL Description é a definição do que é o container de maneira simples. Por fim o RUN diz que sequência de comandos deve ser aplicada para configurar o container, no exemplo é usada a sistema de gerência de pacotes do Alpine, o apk, para atualizar seus repositórios e instalar o snort, e em seguida apaga os temporários usados para as instalações, diminuindo o tamanho do container em si.

Para criar as imagens das aplicações snort e tcpdump é usado o mesmo padrão do código acima.

APÊNDICE E - CONFIGURAÇÕES DA APLICAÇÃO SNORT

O snort é a aplicação de IDS a ser utilizada, já tendo a imagem Docker montado e funcionando, é preciso possuir os arquivos de configuração desejados para que a aplicação funcione. Tendo essa relação pronta de arquivos, é necessário adicioná-las numa pasta única de configurações, que nos exemplos será usada como ${RULES_DIRECTORY}.

Utilizar a seguinte linha de comando para iniciar a aplicação em um único nó pelo Docker:

```
docker run -itd --rm --name ${CONTAINERNAME} --net=host --volume
${RULES_DIRECTORY}:/etc/snort/ ${DOCKER_IMAGE} snort -v -c
${RULES_DIRECTORY}/snort.conf -l ${SNORT_LOGS_FILE} -A fast
```

No comando acima o nome do container é definida pelo ${CONTAINERNAME}, o parâmetro --net é definido como host para que o container compartilha da máquina do nó as informações de rede, tanto lógica como IP e portas abertas como física, como endereço mac e demais informações da placa de rede, o parâmetro --volume monta uma pasta da máquina hospedeira dentro do container, o ${DOCKER_IMAGE} é o nome do Docker image criado para a aplicação, e todos os comandos depois disso são os parâmetros normais da aplicação utilizada.

Para o caso de trabalhar em *cluster*, é necessário criar um Docker service, para tal é usado o seguinte comando:

```
docker service create --name ${NAME_OF_SERVICE} --mode global --network host
--mount type=bind,source=${SNORT_RULES_DIRECTORY},target=/etc/snort/
${DOCKER_IMAGE} snort -v -c ${SNORT_CONF_FILE} -l /var/log/ -A fast
```

${NAME_OF_SERVICE} é o nome dado ao serviço criado, o parâmetro --mode, definido como global, diz que todos os nós do *cluster* deverão ter uma cópia do serviço operando, o parâmetro --mount é o usado para montar o disco do host dentro do container. Ao se utilizar a pasta de operação (o ${SNORT_RULES_DIRECTORY} no exemplo acima dentro da pasta compartilhada pelo GlusterFs, todas as alterações realizadas e registros realizados serão automaticamente replicado por todos os nós do *cluster*.

APÊNDICE F - CONFIGURAÇÕES DA APLICAÇÃO TCPDUMP

O tcpdump é a aplicação de análise de pacotes a ser utilizada, já possuindo a imagem do Docker montado e funcionando. É importante escolher uma pasta para salvar as capturas realizadas

Utilizar a seguinte linha de comando para iniciar a aplicação em um único nó pelo Docker:

```
docker run -itd --restart always --net host --name ${CONTAINERNAME} --volume
${HOST_LOGS_DIRECTORY}:${CONTAINER_LOGS_DIRECTORY} ${DOCKER_IMAGE}
${TCPDUMPCMD} -w ${CONTAINER_LOGS_DIRECTORY}/${TCPDUMPRULENAME}.pcap
```

No comando acima o nome do container é definida pelo ${CONTAINERNAME}, o parâmetro --restart com o argumento always é usado para que, caso o container seja interrompido, ele reinicie automaticamente o parâmetro --net é definido como host para que o container compartilha da máquina do nó as informações de rede, tanto lógica como IP e portas abertas como física, como endereço mac e demais informações da placa de rede, o parâmetro --volume monta uma pasta da máquina hospedeira dentro do container, para salvar as capturas geradas, o ${DOCKER_IMAGE} é o nome do Docker image criado para a aplicação, e a variável ${TCPDUMPCMD} deve ser a linha de comando utilizada para a aplicação, em geral filtrando a interface a analisar e os tipos de pacotes desejados. O parâmetro -w é do próprio tcpdump e é usado para registrar as capturas realizadas em um arquivo, aqui é importante colocar na mesma pasta que foi montada no container para poder ter acesso fácil aos logs.

Para o caso de trabalhar em *cluster*, é necessário criar um Docker service, para tal é usado o seguinte comando:

```
docker service create --name ${NAME_OF_SERVICE} --mode global --network host
--mount
type=bind,source=${HOST_LOGS_DIRECTORY},target=${CONTAINER_LOGS_DIRECTORY}
${DOCKER_IMAGE} ${TCPDUMPCMD} -w
${CONTAINER_LOGS_DIRECTORY}/${TCPDUMPRULENAME}.pcap
```

${NAME_OF_SERVICE} é o nome dado ao serviço criado, o parâmetro --mode, definido como global, diz que todos os nós do *cluster* deverão ter uma cópia do serviço operando, o parâmetro --mount é o usado para montar o disco do host dentro do container. Ao se utilizar a pasta de operação dentro da pasta compartilhada pelo GlusterFs, todas as alterações realizadas e registros realizados serão automaticamente replicado por todos os nós do *cluster*. Os registros realizados de vários nós dentro do mesmo arquivo criaram um registro único, tratando todos os tráfegos recebidos como apenas um.

APÊNDICE G - CONFIGURAÇÕES DA APLICAÇÃO IPTABLES

O iptables é configurado por meio de regras de direção de tráfego e tratativa de pacotes, como, por exemplo, no comando abaixo:

```
iptables -A FORWARD -p icmp -j DROP
```

Com o comando acima qualquer pacote do tipo ICMP que passe pelo nó será descartado.

O comando iptables-save faz com que as configurações aplicadas sejam persistentes após a reinicialização do dispositivo:

```
iptables-save
```

Para que as regras sejam aplicadas em todos os nós simultaneamente um simples *script* pode ser utilizado com base nas configurações já utilizadas. Supondo que no arquivo de hosts (descrito no APÊNDICE A) tenha sido devidamente preenchido o *script* abaixo funcionará para aplicar as regras no *cluster* corretamente:

```
#!/bin/bash

echo Digite a regra a ser criada

read REGRA

for server in $(cat /etc/hosts | grep node | cut -f 1)

do

      ssh root@$server "$REGRA"

      ssh root@$server iptables-save

      done
```

É necessário copiar o *script* para um arquivo normal, adicionar nele a permissão de execução com o comando chmod:

```
chmod +x scriptfile
```

Tendo o *script* pronto basta executar ele e em seguida digitar a regra desejada, que ela será replicada para todos os nós do *cluster*