UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

DANIEL MAIA CUNHA

Maximizando o Desempenho de Funções de Rede Virtualizadas Utilizando Aceleração de Hardware a Partir de FPGAs

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Alberto Egon Schaeffer-Filho
Co-orientador: Prof. Dr. Gabriel Nazar

Porto Alegre
2019

“If I have seen farther than others,
it is because I stood on the shoulders of giants.”
— Sir Isaac Newton

AGRADECIMENTOS

Aos professores Alberto Egon Schaeffer-Filho e Gabriel Luca Nazar, orientadores deste trabalho, um agradecimento muito especial pela leitura atenta, as sugestões e as críticas.

À minha família por sempre me incentivar através de gestos e palavras a superar todas as dificuldades. Pai e mãe, eu nunca teria conseguido sem o apoio de vocês. Obrigado por tudo, de coração.

À Universidade Federal do Rio Grande do Sul, por me proporcionar todo o aprendizado, experiência e amigos que continuarão por toda a vida.

RESUMO

Virtualização de funções de rede (NFV, *Network Functions Virtualization*) propõe desacoplar funções de rede de plataformas de *hardware* dedicadas e implementá-las em máquinas virtuais. Embora esta tecnologia tenha o potencial de melhorar o escalonamento e provisionamento sob demanda, implementações baseadas em *software* não correspondem, para algumas funções, ao alto desempenho que os dispositivos de *hardware* dedicado oferecem, tipicamente, podendo ser um gargalo para o ambiente de rede com taxas de transferência na ordem de Gigabit/s. Portanto, este trabalho propõe explorar o uso de aceleração de *hardware* no contexto de NFV, com o objetivo de aumentar a taxa de transferência e diminuir a latência das Funções de Rede Virtualizadas (VNFs, *Virtualized Network Function*), assegurando as vantagens previstas por essa tecnologia. Nesse cenário, os Arranjos de Portas Programáveis (FPGAs, *Field Programmable Gate Array*) convêm como aceleradores apropriados, por conta de fornecerem implementações de alto desempenho e serem completamente reprogramáveis.

Palavras-chave: NFV. VNF. FPGA. *middlebox*. aceleração de *hardware*. desempenho.

Maximizing the Performance of Virtualized Network Functions Using Hardware Acceleration from FPGAs

ABSTRACT

Network Function Virtualization (NFV) proposes to decouple network functions from dedicated hardware platforms and to implement them in virtual machines. While this technology improves on-demand scheduling and provisioning, software-based implementations do not match, for some functions, the high performance that dedicated hardware devices typically offer, and can be a bottleneck for the network environment with transfer rates in the order of Gigabit / s. Therefore, this work proposes to explore the use of hardware acceleration in the context of NFV, with the objective of increasing the transfer rate and reducing the latency of Virtualized Network Functions (VNFs), ensuring the advantages provided by this technology. In this scenario, Field Programmable Gate Array (FPGAs) act as appropriate accelerators because they provide high-performance implementations and are fully reprogrammable.

Keywords: NFV, VNF, FPGA, middlebox, hardware acceleration, performance.

LISTA DE ABREVIATURAS E SIGLAS

NFV	Network Functions Virtualization.
VNF	Virtualized Network Function.
VNFC	VNF Component.
ASIC	Application Specific Integrated Circuits.
CPU	Control Process Unit.
FPGA	Field Programmable Gate Array.
IP	Internet Protocol.
IDS	Intrusion detection System.
IPS	Intrusion Prevention System.
AXI	Advanced eXtensible Interface.
NAT	Network Address Translation.
WAN	Wide Area Network .
BRAM	Block RAM.
SRAM	Static RAM.
DMA	Direct Memory Access.
HDL	Hardware Description Language.
DSP	Digital Signal Processor.
VHDL	VHDL Very High Speed Integrated Circuit Hardware Description Language.
EDK	Embedded Development Kit.
GPU	Graphics Processing Unit.
ETSI	European Telecommunications Standards Institute.
CLB	Configurable Logic Block.
LUT	Lookup Table.
EM	Element Management.

API	Application Programming Interface.
JSON	JavaScript Object Notation.

LISTA DE FIGURAS

Figura 2.1 Topologia de rede composta por variadas *middleboxes* 17
Figura 2.2 Visão geral de NFV 18
Figura 2.3 Arquitetura ETSI NFV 19
Figura 2.4 Estrutura CLB 21
Figura 2.5 Arquitetura FPGA 22

Figura 4.1 Infraestrutura 29
Figura 4.2 firewall 31
Figura 4.3 IPS 31
Figura 4.4 Arquitetura de um projeto utilizando NetFPGA 34
Figura 4.5 *Pipeline* de referência da plataforma NetFPGA 36
Figura 4.6 Máquina de estados 38

Figura 5.1 Ambiente configurado para a análise de resultados do *firewall* baseado em FPGA 41
Figura 5.2 Latência média em diferentes configurações de *firewalls* 43
Figura 5.3 Taxa de transferência média em diferentes configurações de *firewalls* 44
Figura 5.4 Tempo necessário para que o conjunto de regras do *firewall* seja atualizado 45

LISTA DE TABELAS

Tabela 4.1 Blocos de processamento de pacote especificados pelo projeto OpenBox ...32
Tabela 4.2 Repositório de VNFCs implementados ...37
Tabela 4.3 Exemplo de um conjunto de regras configurados no *firewall* ...37
Tabela 4.4 Pacote separado em 7 conjuntos de palavras para os cabeçalhos ...38

Tabela 5.1 *Hardware* equipado para execução do *firewall* Iptables ...41
Tabela 5.2 Recursos Utilizados pelo FPGA ...42
Tabela 5.3 Resultado da latência média em ms para diferentes conjuntos de regras ...43
Tabela 5.4 Resultado da taxa de transferência média em Mbps para diferentes conjuntos de regras ...44

SUMÁRIO

1 INTRODUÇÃO 12
1.1 Motivação 13
1.2 Contribuições 14
1.3 Estrutura 15
2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 *Middlebox* 16
2.2 Virtualização de Funções de Rede 17
2.3 Aceleração de *Hardware* 20
3 TRABALHOS RELACIONADOS 24
3.1 Soluções baseadas em GPU 24
3.2 Soluções de DPIs utilizando FPGA 25
3.3 Soluções de *firewalls* utilizando FPGA 26
3.4 Aceleração de *hardware* em ambientes NFV 27
3.5 Discussões 28
4 METODOLOGIA 29
4.1 Infraestrutura 29
4.2 Implementação 32
4.2.1 Arquitetura da NetFPGA 33
4.2.2 Hardware 33
4.2.3 *Pipeline* de referência 35
4.2.4 Conjunto de VNFCs implementado 36
4.2.5 *Firewall* 36
4.2.5.1 Plano de Dados 38
4.2.5.2 Plano de Controle 40
5 RESULTADOS 41
5.1 Recursos Utilizados do FPGA 42
5.2 Latência 42
5.3 Taxa de Transferência 43
5.4 Tempo de Atualização do Conjunto de regras 45
6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS 46
REFERÊNCIAS 47

1 INTRODUÇÃO

NFV é uma tecnologia que consolida em servidores comuns as funções anteriormente executadas por dispositivos de *hardware* específicos usando a tecnologia de virtualização. Antes do aparecimento de NFV, funções de rede eram construídas como uma combinação fechada de *software* e *hardware* de fornecedores, conhecidas como *middleboxes*. NFV, portanto, é um avanço tecnológico a respeito do provisionamento de funções de rede, que permite desacoplar *software* do *hardware* dedicado (Mijumbi et al., 2016).

Este paradigma de virtualização possibilita que as VNFs possam ser instanciadas e removidas em tempo de execução, sendo assim, o ciclo de vida das VNFs pode ser muito mais curto e dinâmico em comparação com dispositivos físicos, uma vez que essas funções podem ser adicionadas sob demanda.

Embora NFV melhore o escalonamento e provisionamento sob demanda, a tecnologia de virtualização apresenta uma limitação quanto a performance, uma vez que o desempenho de uma função virtualizada é reduzido em comparação com Circuitos Integrados de Aplicação Específica (ASIC, *Application Specific Integrated Circuit*) (Watanabe et al., 2017). Portanto, em se tratando de funções de rede que exigem processamento de pacotes em taxas de linha e necessitam de computação intensiva, implementações totalmente baseadas em *software* não oferecem o desempenho necessário. Por conta disso, uma abordagem possível para amenizar este problema é a inserção de aceleradores de *hardware* em um ambiente NFV.

FPGAs consistem em circuitos integrados que podem ser configurados conforme necessário depois de fabricados. Mais especificamente, um FPGA é composto de blocos lógicos primitivos e, a partir do agrupamento e conexão desses blocos, podem-se executar funções mais complexas. Portanto, devido às suas características, a tecnologia de FPGA tem sido amplamente reconhecida como uma das principais plataformas para viabilizar a execução de NFV (Niemiec et al., 2019; Brebner, 2015; Lan et al., 2017; Paolino; Pinneterre; Raho, 2017; GE et al., 2014), ao passo que é possível se beneficiar tanto da aceleração de *hardware* quanto da flexibilidade oferecida pela sua capacidade de reprogramação.

1.1 Motivação

As funções de rede, ou *middleboxes*, desempenham um papel essencial nas atuais redes de datacenters, provedores de Internet e empresas de forma geral. Desempenho, segurança e resiliência são razões comuns para implantar *middleboxes* em redes corporativas. Funções e tipos de *middleboxes* crescem constantemente desde os primeiros dias da Internet. Apesar das preocupações em violar o princípio de ponta a ponta da Internet, os serviços de hoje são inconcebíveis sem o uso de *middleboxes* (Papastergiou et al., 2017).

Atualmente, o tráfego de rede normalmente atravessa uma sequência de *middleboxes* (também conhecida como cadeia de serviços) até chegar no seu destino. As funções das *middleboxes* incluem, entre outras, as seguintes: Balanceador de Carga, Tradução de Endereço de Rede (NAT, *Network Address Translation*), Conversão de Protocolo (por exemplo, IPv4 / v6), Classificação e Marcação de Pacotes, Otimização de Rede de Longa Distância (WAN, *Wide Area Network*), Detecção de Intrusão, firewall, *Proxy* e Rede de Distribuição de Conteúdo (CDN, *Content Delivery Network*).

Por um lado, a proliferação de *smartphones*, *streaming* de vídeo e Redes Virtuais Privadas (VPNs, *Virtual Private Networks*) gera necessidade por novos tipos de funções de rede continuamente. Por outro lado, à medida que mais tráfego atravessa as *middleboxes*, o custo e o desempenho se tornam importantes considerações de projeto.

Os custos de manter um ambiente de rede utilizando equipamentos especializados são enormes, pois há necessidade de comprar e manter vários produtos do mesmo fabricante, que nem sempre se integram facilmente. Além disso, existem os gastos como manutenção, mão-de-obra qualificada, espaço e eletricidade que devem ser levados em conta ao implementar uma tecnologia. Portanto, a NFV foi criada de forma que um *hardware* genérico seja capaz de substituir o *hardware* proprietário. A tecnologia de virtualização permite o desenvolvimento de projetos, implementações e novos modelos de administração de serviços de rede, buscando a substituição de dispositivos de *hardware* dedicados e com alto custo por dispositivos de *software* executados em máquinas virtuais.

No entanto, um dos principais desafios da tecnologia de NFV é garantir que as VNFs instanciadas neste ambiente ofereçam um desempenho comparável com as funções de rede executadas em dispositivos especializados. Diante disso, técnicas de aceleração de *hardware* são exploradas. Devido ao seu desempenho e capacidade de programação, o FPGA emergiu como uma plataforma promissora para a NFV, por conta de desfrutar da alta capacidade de programação de *software* e do alto desempenho do *hardware* dedicado.

1.2 Contribuições

O trabalho apresentado neste documento aborda o projeto e a implementação de uma infraestrutura NFV heterogênea composta por Processadores de Propósito Geral (GPPs, *General Purpose Processors*) e aceleradores de *hardware*, mais especificamente, FPGAs. Portanto, trata-se de uma infraestrutura capaz de acelerar o desempenho de VNFs transferindo o processamento massivo de pacotes para o FPGA.

Uma vez que os recursos do acelerador de *hardware* são caros em comparação aos recursos do GPP, a infraestrutura projetada propõe consumir uma menor quantidade de recursos do acelerador de *hardware* através do compartilhamento de blocos de processamento comum de pacotes. Isto é, explora-se a propriedade de que as funções de rede, tipicamente, utilizam um conjunto muito semelhante de etapas de processamento, sendo assim, é possível que blocos de processamento de funções de rede sejam compartilhados sistematicamente entre mais de uma função de rede simultaneamente, eliminando possíveis redundâncias no acelerador.

Com o objetivo de facilitar a gerência de rede e abstrair o uso de aceleradores de *hardware* para o usuário final, a infraestrutura projetada é capaz de instanciar sistematicamente múltiplas VNFs para uso em diferentes cenários de cargas de trabalho a partir de uma estrutura de gerência. A estrutura de gerência permite que as VNFs possam ser instanciadas, configuradas e desalocadas conforme o interesse do gerente. Desta forma, é transparente à gerência de rede como os aceleradores de *hardware*, GPPs e, de forma geral, os recursos estão alocados na infraestrutura subjacente, uma vez que essas configurações e interconexões são automaticamente geradas.

Usando os dados de desempenho obtidos das implementações e um exemplo realista para estudo de caso, pretende-se apresentar ganhos significativos de desempenho, em relação ao uso de aceleração de *hardware* no contexto de NFV, para diferentes cenários de carga de trabalho por meio de avaliação analítica. Desta forma, como estudo de caso propõe-se a implementação de um *firewall* dinamicamente configurável executado sobre uma infraestrutura heterogênea prototipado em uma placa NetFPGA.

As contribuições deste trabalho incluem:

- Investigar o estado da arte relacionado à aplicação de aceleradores de *hardware* no contexto de NFV.
- Desenvolver uma infraestrutura NFV heterogênea com provisionamento de aceleração de *hardware* de forma dinâmica a partir de FPGAs.

- Economizar recursos do acelerador de *hardware* a partir da reutilização de blocos de processamento de pacotes entre múltiplas VNFs.
- Apresentar uma estrutura de gerência para que o operador de rede possa configurar dinamicamente a infraestrutura projetada.
- Realizar experimentos para avaliar eventuais ganhos significativos de performance em comparação a plataformas NFV tradicionais a partir da infraestrutura desenvolvida.

1.3 Estrutura

Este trabalho é organizado da seguinte forma: o Capítulo 2 define os conceitos técnicos importantes para entendimento do trabalho e da solução proposta. O Capítulo 3 faz uma revisão do estado da arte de aceleradores de *hardware* no contexto de NFV. Neste capítulo também são apresentados as oportunidades de contribuição para o estado da arte, baseado nos artigos revisados. O Capítulo 4 apresenta a plataforma proposta e discute a arquitetura e implementação de referência em detalhe. O Capítulo 5 apresenta os resultados experimentais para avaliar as opções de implementação. O Capítulo 6 apresenta conclusões e discute possibilidades de trabalho futuro.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo fornece informações gerais sobre os tópicos que serão abordados nos capítulos seguintes. Inicia-se com uma visão geral do conceito de *middleboxes* e os desafios técnicos de migrar-se de *hardware* especializado para *software*. Na sequência, aborda-se detalhadamente os conceitos técnicos de Virtualização de Funções de Rede. Por fim, argumenta-se a respeito de Aceleração de *Hardware* e sua aplicação no contexto de ambientes de redes virtualizados.

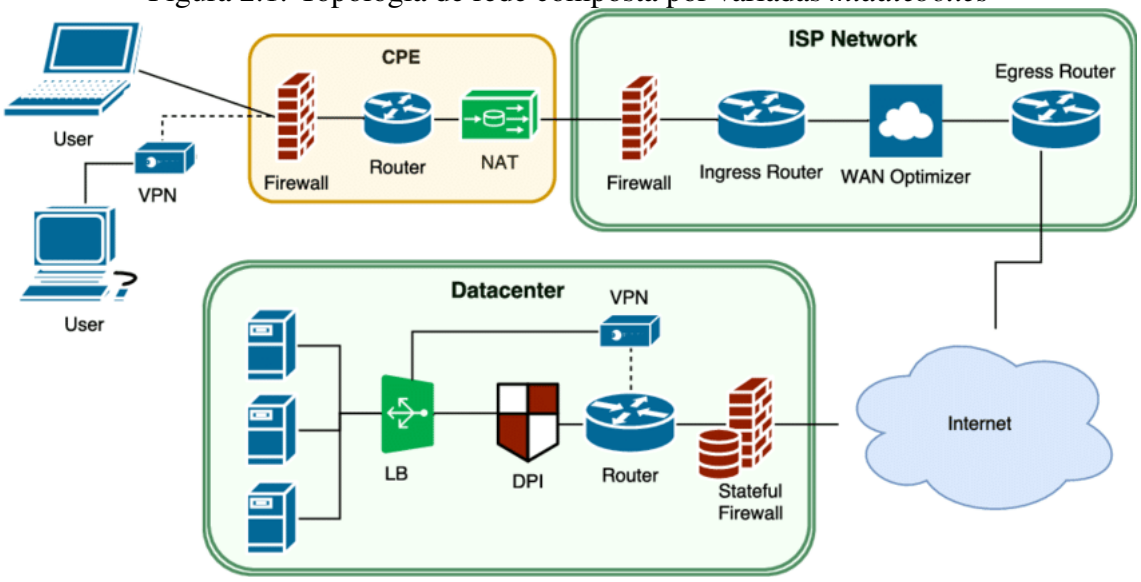
2.1 *Middlebox*

No início da Internet, os elementos de rede operavam de maneira isolada e suas funções eram limitadas ao simples encaminhamento de pacotes IP. Isso era compatível com um dos objetivos fundamentais de alcançar conectividade contínua, mesmo com a perda de elementos de rede. A popularidade da Internet, no entanto, trouxe novos requisitos para operadores e usuários de rede. Por exemplo, a necessidade de segurança aprimorada levou muitos operadores de rede a implantar *firewalls* e Inspeções Profunda de Pacotes (DPIs, *Deep Packet Inspections*), permitindo que se tenha um controle mais refinado sobre quais pacotes são permitidos em suas redes, além de atenuar possíveis ataques. Essas funções são implementadas tipicamente em elementos de rede mais especializados, chamados *middleboxes* (Chang et al., 2017). A Figura 2.1 apresenta uma topologia de rede realista composta por diversas *middleboxes*.

"Middleboxes" são definidas na RFC 3234 como "qualquer dispositivo intermediário que execute funções diferentes da função padrão normal de um roteador IP no caminho do datagrama entre um hospedeiro de origem e um hospedeiro de destino"(CARPENTER; BRIM, 2002). Além de melhorar a segurança, as *middleboxes* podem ser usadas para melhorar o desempenho (por exemplo, Balanceadores de carga), fornecer prestação de dados e informações de monitoramento (por exemplo, monitor de tráfego), compatibilizar protocolos diferentes (por exemplo, tradutor de protocolo IPv4 para IPv6) e solucionar as limitações existentes (por exemplo, NAT, que permite manter o redimensionamento da Internet diante do esgotamento dos endereços IPv4) (Stiemerling; Quittek, 2002).

Comparado aos dispositivos de encaminhamento, como *switches* e roteadores, as *middleboxes* são complexas. Elas operam em fluxos de pacotes em várias camadas da pilha de rede, da camada de rede para a camada de aplicação, além disso, tipicamente

Figura 2.1: Topologia de rede composta por variadas *middleboxes*



Fonte: (Chang et al., 2017)

o fazem na taxa de linha (Huang; Cuadrado; Uhlig, 2017). Entretanto, as *middleboxes* interferem na transmissão de pacotes ponta a ponta, na funcionalidade de aplicativos no usuário final e na restrição ou impedimento de que os aplicativos nos hospedeiros finais funcionem corretamente (Huang; Cuadrado; Uhlig, 2017).

2.2 Virtualização de Funções de Rede

Até os últimos anos, as *middleboxes* eram implantadas principalmente usando *hardware* criado para esse fim. Isso, no entanto, tem várias deficiências:

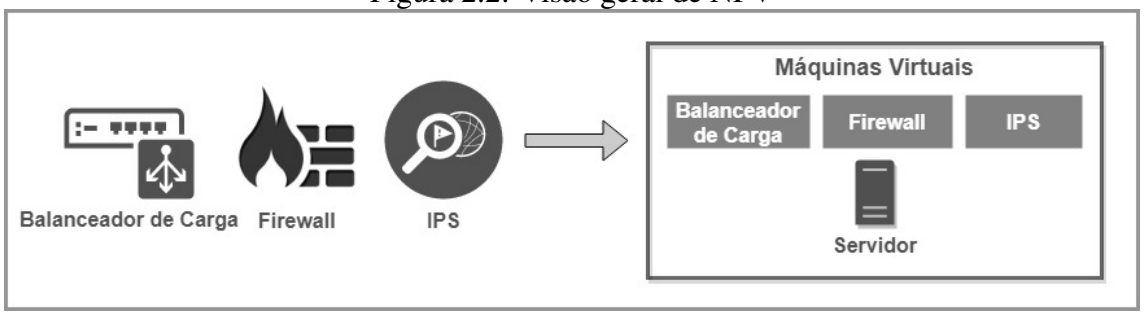
- Rigidez: como a funcionalidade é implementada diretamente no hardware, a mudança é muito difícil - geralmente impossível.
- Difícil de gerenciar: as caixas intermediárias de vários fornecedores têm suas próprias interfaces de gerenciamento que não fornecem uma compatibilidade.
- Desenvolvimento lento: o *hardware* é mais lento e mais difícil de desenvolver do que o *software*.
- Custo: tipicamente, as *middleboxes* apresentam um custo elevado. Além disso, as *middleboxes* subutilizadas não oferecem oportunidade para reutilização.

NFV visa resolver os problemas acima, migrando a funcionalidade das *middleboxes*, de caixas dedicadas para *software* em execução em servidores comuns.

Proposto em 2012 pelo Instituto Europeu de Normas de Telecomunicação (ETSI,

European Telecommunications Standards Institute) (NFV White Paper, 2012), NFV surgiu como uma tecnologia de rede da indústria de telecomunicações para fornecer agilidade e flexibilidade na implantação de serviços de rede e para reduzir as despesas de capital (CAPEX, *Capital Expenditure*) e as despesas operacionais (OPEX, *Operational Expenditure*) (Tipantuña; Yanchapaxi, 2017).

Figura 2.2: Visão geral de NFV



Fonte: o autor

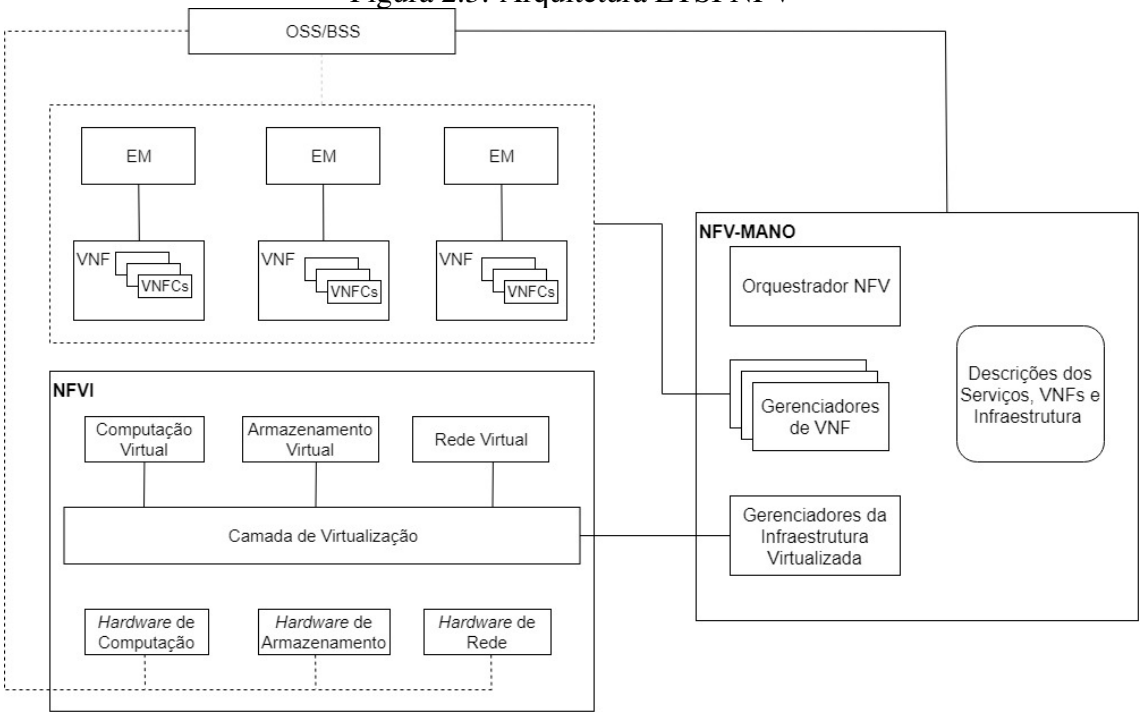
A ideia principal de NFV é desacoplar da rede física do equipamento as funções que são executadas nele. Para esse fim, concentra-se em virtualizar funções de rede que de outra forma seriam executadas em *hardware* proprietário, tais como: balanceadores de carga, *firewalls* e sistemas de detecção de intrusões. Sendo assim, migram-se estes serviços, anteriormente executados em *hardware*, para dispositivos baseados em *software* em execução em máquinas virtuais (VM, *Virtual Machine*) (Figura 2.2). Isso significa que uma função de rede - como um *firewall* - pode ser despachada como uma instância de *software* simples. Isso permite a consolidação de muitos equipamentos de rede (funções) em servidores de alto volume, que podem estar localizados em *data centers*, nós da rede e nas instalações do usuário final. Portanto, o modelo NFV adiciona flexibilidade e melhora a manutenibilidade, permitindo que os provedores de serviços iniciem, melhorem e otimizem incrementalmente os serviços usando atualizações de *software*, ao invés de substituição de *hardware* (Alwakeel; Alnaim; Fernandez, 2019).

O ETSI, entidade responsável pelo Grupo de Especificação da Indústria para NFV (ETSI ISG NFV), propôs uma arquitetura para NFV (ETSI GS NFV 002, 2013), empenhando-se em apresentar uma estrutura para o operador de rede instanciar e gerenciar blocos funcionais. Essa arquitetura consiste em três componentes fundamentais: Funções Virtualizadas de Rede (VNFs), Arquitetura de Gerenciamento de Virtualização e Orquestração (NFV MANO) e Infraestrutura de NFV (NFVI, NFV *Infrastructure*).

As VNFs são implementações de blocos funcionais de rede, gerenciadas pelo Gerenciador de Elemento (EM, *Element Management*), que podem ser fragmentadas em

diversos componentes de mais baixo-nível, os chamados componentes de VNF (VNFCs, VNF *Components*). A NFVI é a totalidade de componentes de *hardware* e *software* que criam o ambiente onde as VNFs são implantadas. O NFV MANO é a coleção de todos os blocos funcionais, repositórios de dados usados por esses blocos e pontos de referência e interfaces através dos quais esses blocos funcionais trocam informações com o propósito de gerenciar e orquestrar a NFVI e as VNFs. Além disso, o NFV MANO inclui a especificação de interfaces que permitem a comunicação com outros componentes, incluindo Sistemas de Suporte a Operações (OSS, *Operations Support Systems*) e Sistemas de Suporte de Negócios (BSS, *Business Support Systems*). A Figura 2.3 ilustra os módulos que compõem essa arquitetura.

Figura 2.3: Arquitetura ETSI NFV



Fonte: o autor

A arquitetura NFV apresenta vantagens no gerenciamento de rede, uma vez que permite que as funções de rede sejam desenvolvidas com uma preocupação ínfima com relação ao *hardware* onde serão executadas. Em contrapartida, a migração do *hardware* para o *software* não apresenta apenas benefícios. O *hardware* criado para propósito geral geralmente oferece desempenho de taxa de linha difícil de alcançar em *software*. De fato, com a rápida adoção de padrões Ethernet de alta velocidade, atingir taxas de linha é uma tarefa árdua (Taniguchi et al., 2015). Por conta disso, quando a NFVI não provê nenhum acelerador e as VNFs são puramente baseadas em *software*, as funções de rede provisionadas por esse ambiente tendem a ter uma performance aquém em relação a fun-

ções executadas em *hardware* dedicado, tipicamente, prejudicando a adoção da tecnologia de NFV. Por conta disso, investiga-se o uso de técnicas de aceleração de *hardware* com objetivo de melhorar o aspecto performático nas plataformas de NFV.

2.3 Aceleração de *Hardware*

Uma rotina pode ser processada puramente em *software* executado em uma CPU genérica, puramente em *hardware* dedicado ou em alguma combinação de ambos. Sendo assim, aceleração de *hardware* consiste em transferir a execução de uma tarefa, ou parte dela, da CPU para um *hardware* mais apropriado, a fim de se obter mais eficiência. O conceito de aceleração de *hardware*, tipicamente, está atrelado a uma menor latência, maior vazão e um consumo de energia reduzido.

Embora a arquitetura padrão NFV baseada em servidores comuns pode atender às metas de desempenho de muitos casos de uso de virtualização, por exemplo, no caso de funções de rede que não apresentam um gargalo para rede (NAT, *proxies*) (Bonafiglia et al., 2015; Chia-Nan Kao et al., 2015; Chou et al., 2018). No entanto, alguns casos de uso são mais desafiadores e exigem processamento intensivo em redes ou computação intensiva, por exemplo, no caso de funções de rede que requerem uma alta taxa de transferência e uma baixa latência (firewalls, DPIs) (Naik; Shaw; Vutukuru, 2016). Sendo assim, faz-se necessário, tipicamente, o provisionamento de aceleração de *hardware* pela NFVI, com o objetivo de permitir o processamento de VNFs na taxa de linha (por exemplo, 10Gbps). Para este fim, o acelerador deve ser adaptável a diferentes funções e fornecer um alto desempenho.

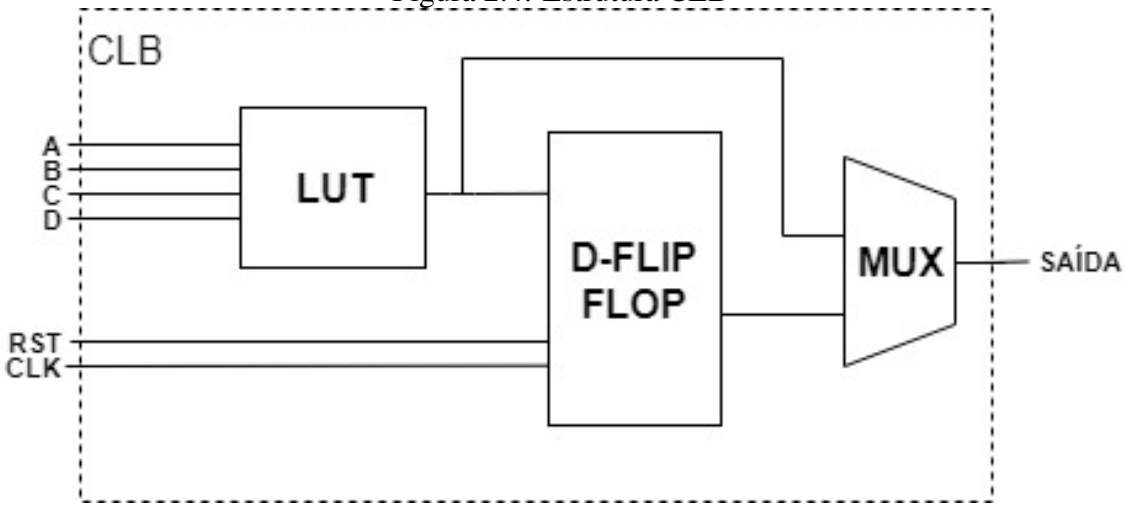
As Unidades de Processamento Gráfico (GPUs, *Graphics Processing Unit*) foram utilizadas com sucesso para acelerar funções não relacionadas a processamento gráfico, frequentemente do domínio de aprendizado de máquina. Devido ao grande número de núcleos e a alta largura de banda da memória, as GPUs também podem ser úteis no contexto de NFV para aceleração de VNFs que podem ser mapeadas com eficiência para seu modelo de computação.

Além das GPUs, os FPGAs são uma opção altamente viável para aceleração de VNFs, por conta de representarem um equilíbrio entre uma implementação ASIC e uma implementação em *software*, oferecendo tanto a flexibilidade e a rápida implementação das soluções em *softwares* quanto o rendimento mais próximo dos dispositivos de *hardware* especializados. Embora nenhuma solução aceleradora seja provavelmente ideal para

todos os casos, este trabalho concentra-se em soluções baseadas em FPGA, principalmente devido ao uso anteriormente bem-sucedido no domínio de rede e sua grande flexibilidade, o que permite a implementação de uma variedade de diferentes funções.

FPGAs possuem uma matriz de blocos lógicos configuráveis e conectados via interconexões programáveis que permitem programar FPGAs para aplicações ou funcionalidades específicas. Um FPGA incorpora uma boa quantidade de tabelas de busca (LUTs, *lookup tables*). Cada LUT pode ser configurada para desempenhar uma função lógica específica (AND, OR, NOT, etc), e são tipicamente agrupadas em Blocos Lógicos Configuráveis (CLBs, *Configurable Logic Blocks*) juntamente com *flip-flops* e mutiplexadores, como é exibido na Figura 2.4. Além disso, FPGAs geralmente incluem blocos de Processamento de Sinal Digital (DSP, *Digital Signal Processor*), Blocos de Memória de Acesso Aleatório Dinâmico (BRAMs, *Block Random Access Memories*), Memórias Estáticas de Acesso Randômico (SRAMs, *Static Random Access Memories*), Memórias Dinâmicas de Acesso Randômico (DRAMs, *Dynamic Random Access Memory*) e, por fim, blocos de I/O. A distribuição e interconexão dos componentes pela área do FPGA são exibidas na Figura 2.5.

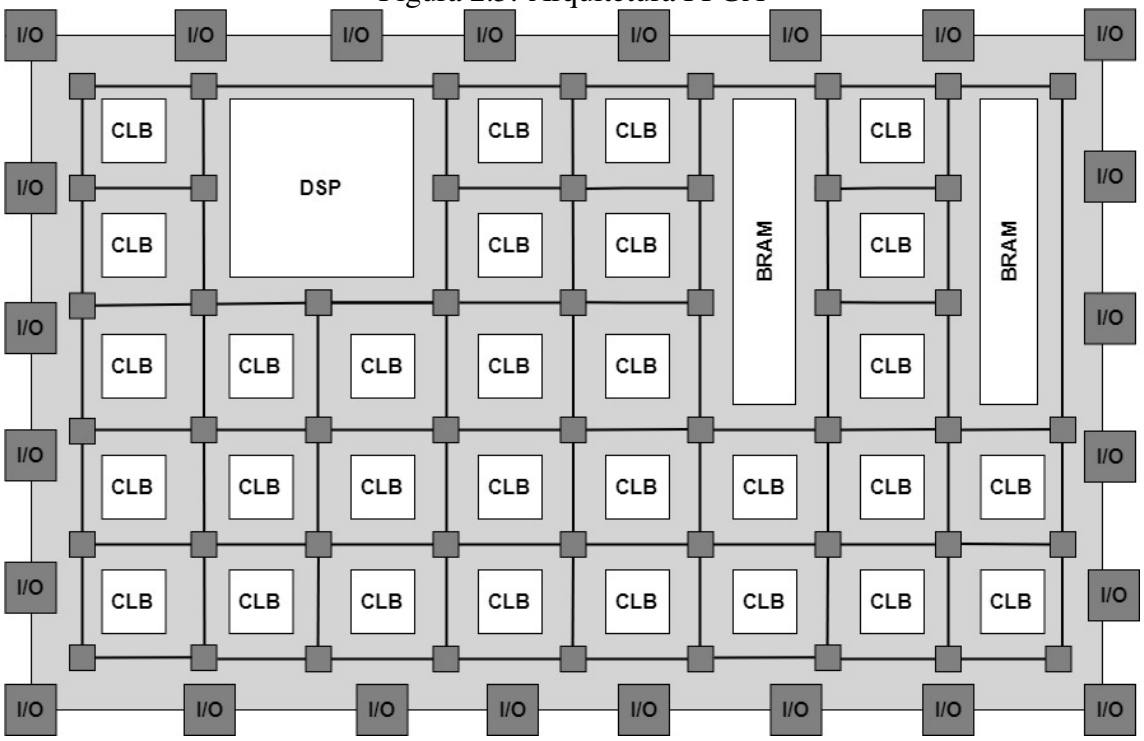
Figura 2.4: Estrutura CLB



Fonte: o autor

Os FPGAs são ideais para execução de sistemas paralelos em que várias tarefas devem ser executadas simultaneamente (Dorta et al., 2009). Por conta disso, os ganhos de performance alcançados pelos FPGAs estão altamente atrelados ao algoritmo a ser executado. Por ser possível construir várias unidades de execução paralela em um FPGA, algoritmos que manipulam um grande volume de dados podem ser paralelizados através da distribuição dos dados pelas unidades de execução paralelas e obter maiores ordens de taxa de transferência do que pode ser alcançado mesmo com uma CPU *multi-core*.

Figura 2.5: Arquitetura FPGA



Fonte: o autor

Os benefícios em eficiência energética também podem ser alcançados com os FPGAs, quando comparados à execução de *software* em um GPP ou até mesmo em uma GPU (Yih et al., 2018), embora os ganhos exatos possam variar bastante, dependendo do *hardware* específico implementado. Em contrapartida, esses ganhos são tipicamente menores do que os obtidos com um ASIC.

No contexto de NFV, a principal vantagem dos FPGAs como aceleradores de *hardware* descendem do fato de serem inerentemente reconfiguráveis. Bugs em sua lógica de aplicação podem ser corrigidos, recursos adicionados ou plataformas reconfiguradas para implementar funcionalidades diferentes. Os FPGAs podem ser reconfigurados estaticamente e dinamicamente (Lie; Feng-yan, 2009). Por um lado, a reconfiguração estática significa que a execução do FPGA é interrompida e o FPGA é configurado com um novo arquivo de configuração (*bitstream*). Por outro lado, a reconfiguração dinâmica (parcial) refere-se à opção de configurar parte do FPGA, enquanto o resto do FPGA está operando sem qualquer interrupção. A reconfiguração dinâmica, tipicamente, é a mais utilizada no âmbito de NFV, por conta de sua flexibilidade (Mishra; Chen; Zervas, 2016).

Apesar das vantagens apresentadas pelo uso de FPGAs como aceleradores de *hardware* em um ambiente de NFV, existem empecilhos a respeito de sua implantação (Chiotakis; Pinneterre; Paolino, 2019). Os CLBs em uma placa FPGA são limitados, portanto, haveria um alto custo para implementação de todas as VNFs em um FPGA,

decorrente de serem exigentes em recursos. Ademais, quando opta-se pelo uso de reconfiguração estática o FPGA precisa ser reprogramado quando a lógica da VNF muda, o que pode levar um longo tempo para sintetização do código, inviabilizando o uso de FPGA para implementar toda a cadeia de serviços de NFV. No entanto, uma vez que se escolhe pelo uso de reconfiguração dinâmica, é necessário uma plataforma mais complexa e capaz de gerenciar a coexistência de múltiplas funções em um mesmo FPGA.

3 TRABALHOS RELACIONADOS

Um grande desafio de NFV é alcançar o desempenho conhecido de dispositivos de *hardware* dedicados, portanto, uma série de trabalhos exploram técnicas para viabilizar a execução de Funções de Rede Virtualizadas. Essas técnicas, tipicamente, são baseadas no uso de aceleradores de *hardware* para aumentar o rendimento e reduzir o atraso das Funções de Rede Virtualizadas.

Houve muitos esforços recentes para permitir a integração de aceleradores de *hardware* no contexto de NFV, sendo assim, este capítulo apresenta os principais trabalhos relacionados à utilização de técnicas para melhorar o desempenho de funções de rede, a partir de plataformas capazes de gerenciar a integração de VNFs com aceleradores de *hardware*.

3.1 Soluções baseadas em GPU

As GPUs são eficientes no processamento de funções de rede, porque oferecem paralelismo suficiente para o grande número de pacotes recebidos. Como as CPUs e as GPUs têm características arquiteturais diferentes, elas geralmente funcionam em *pipeline* para executar tarefas específicas de alta eficiência (HAN et al., 2010). As CPUs geralmente são responsáveis por executar operações de I/O, lote e análise de pacotes. As tarefas de computação e uso intensivo de memória são transferidas para GPUs para aceleração, como operações criptográficas (JANG et al., 2011), DPIs (JAMSHED et al., 2012) e *matching* de expressões regulares (KALIA et al., 2015).

Um sistema GPU-NFV é proposto pelos autores em (YI; DUAN; WU, 2017). Este trabalho explora o poder de processamento paralelo oferecido pelas GPUs com objetivo de aumentar o rendimento do sistema NFV. GPU-NFV é um sistema heterogêneo, onde as CPUs se encarregam de realizar tarefas de controle da infraestrutura e as GPUs são encarregadas de realizar o processamento massivo dos pacotes. Por fim, este sistema atinge uma alta taxa de processamento de pacotes, explorando ao máximo os recursos de processamento paralelo das GPUs, em contrapartida, não apresenta estratégias de reconfigurabilidade das funções executadas pelas GPUs.

Os autores em (ZHANG et al., 2018) propuseram o G-NET, um sistema NFV com um esquema de virtualização de GPU que suporta o compartilhamento espacial de GPU, um agendador de GPU baseado em cadeia de serviços e um esquema para garantir o iso-

lamento de dados na GPU. Também foi desenvolvida uma abstração para criar funções eficientes de rede no G-NET, o que reduz significativamente os esforços de desenvolvimento.

3.2 Soluções de DPIs utilizando FPGA

A inspeção profunda de pacotes envolve a pesquisa do cabeçalho e da carga útil de um pacote em relação a milhares de regras para detectar possíveis ataques. Com o aumento no uso da Internet e o crescente número de ataques existentes, a aceleração do *hardware* se tornou essencial para evitar que o DPI se tornasse um gargalo na rede, se usado em um roteador de borda ou de núcleo.

Os proponentes de trabalhos nesta área argumentam que a detecção de intrusões é uma aplicação perfeita da computação reconfigurável, pois é computacionalmente intensiva, orientada à taxa de transferência, e os conjuntos de regras mudam frequentemente. Como os FPGAs são inerentemente reconfiguráveis, a maioria dos trabalhos anteriores nessa área concentra-se em maneiras eficientes de mapear uma regra estabelecida para um circuito especializado que implementa a pesquisa. A configuração (o circuito implementado no FPGA) é projetada para tirar proveito da natureza de um determinado conjunto de regras e qualquer alteração no conjunto de regras exigirá a geração de um novo circuito, tipicamente, necessitando uma ressintetização do FPGA.

Uma arquitetura de DPI com uma estrutura de memória hierárquica e mecanismos de *matching* paralelo baseados no FPGA centralizado na memória é proposto em (Lin et al., 2017; Tharaka et al., 2017). O protótipo de DPI implementado é capaz de fornecer taxa de transferência de *matching* de texto de até 60 Gbps a partir do acelerador de *hardware*. Explorando a capacidade de processamento paralelo de alta velocidade do FPGA, vários mecanismos de *matching* são implementados neste protótipo.

Dois algoritmos são apresentados em (Tuck et al., 2004) com base na abordagem Aho-Corasick (Tran et al., 2013) para *matching* de strings. Eles são projetados com a aceleração do *hardware* e reduzem o consumo de memória através do uso de bitmaps e compactação de caminho.

Por fim, uma solução tipicamente utilizada para implementação de DPI são as memórias de conteúdo endereçáveis (CAM, *Content Addressable Memories*), que são sistemas de memória que pesquisam uma sequência de dados e retornam seu endereço. As consultas podem ser feitas em paralelo e geralmente são muito mais rápidas do que pes-

quisar em uma RAM tradicional com *software*. Uma alternativa popular, Ternary CAM (TCAM), adiciona o bit curinga à pesquisa. Ambas as abordagens permitem a pesquisa rápida de padrões maliciosos, conforme implementado em (Deng et al., 2009; Woo-Sug Jung; Kwon, 2010). No entanto, CAMs e TCAMs são notoriamente ineficientes em energia e memória, o que pode ser visto a partir da análise feita na literatura (Le; Prasanna, 2010; KUMAR et al., 2007).

3.3 Soluções de *firewalls* utilizando FPGA

Um *firewall* de filtragem de pacotes controla o campo de cabeçalho em cada pacote de dados da rede com base em sua configuração e permite ou nega os dados que passam pela rede. Esta seção aborda questões de alguns esforços recentes de pesquisa, com foco nas arquiteturas das soluções. Os objetivos comuns da pesquisa são tornar o *design* de *hardware* o mais compacto e eficiente possível e ter uma solução de *firewall*, que seja simples de oferecer suporte e atualização.

Um esquema típico para implementação e prototipagem é ter um FPGA com um mecanismo *firewall* como coprocessador em solução incorporada. Assim, uma tarefa de gerenciamento está em uma parte do *software* do sistema e a atualização de regras do *firewall* envolve apenas a reinicialização de memória no *software* sem nenhuma alteração de *hardware*. Um exemplo é um *firewall* baseado em *hardware* e um mecanismo de limitação de taxa projetado pelos autores em (Sang-Kil Park; Jin-Tae Oh; Jong-Soo Jang, 2006), onde o filtro é implementado no FPGA e o gerenciamento de regras é operado por *software* na CPU incorporada. Outro exemplo é um sistema embarcado baseado em processador com sistema operacional em tempo real de (Ajami; Dinh, 2011), projetado para atingir alteração de configuração altamente personalizada e dinâmica no *firewall*.

A reconfiguração parcial permite modificações lógicas, mantendo o sistema operacional usando dois mecanismos paralelos e alterando a configuração do FPGA. Os trabalhos em (EZZATI et al., 2010; YIN et al., 2011) exploram um conceito interativo de *firewall* reconfigurável. Portanto, a partir reconfiguração parcial, nestes trabalhos, a alocação dinâmica de rede virtual depende do recurso de reconfiguração dinâmica no FPGA, sendo assim, os autores utilizaram a interface JTAG para carregar dinamicamente o fluxo de bits parcial no dispositivo FPGA toda vez que a política de regras do *firewall* é modificada.

Outra abordagem é alterar a configuração do FPGA para cada atualização no con-

junto de regras do *firewall*, otimizando ainda mais a implementação para um conjunto regras específicas. Apesar de exigir reconfiguração na atualização do conjunto de regras, os trabalhos (Yang et al., 2014; Hager et al., 2014) podem alcançar um bom desempenho, no entanto o *firewall* está inoperável durante o tempo de ressintetização - podendo ser um longo período no contexto de rede. Além disso, essas estratégias dependem muito dos recursos do conjunto de regras e, em alguns casos, otimizações podem não ser possíveis, tornando o desempenho variável, dependendo do conjunto de regras específicas.

A TCAM realiza pesquisa simultânea nos endereços armazenados em um único ciclo de relógio, desta forma, uma pesquisa em um conjunto de regras endereçadas a partir desta memória para verificação em um conjunto de regras de um *firewall* pode ser feita de forma otimizada. Em (Ajami; Dinh, 2011) os autores apresentam um *firewall* completo implementado em um FPGA, contando com CAMs limitados a 16 posições. Os autores em (DONG et al., 2015) propuseram quatro algoritmos heurísticos simples para endereçar regras de um *firewall* otimizando o número entradas necessárias na memória para cada conjunto de regra.

3.4 Aceleração de *hardware* em ambientes NFV

Em um ambiente NFV, é essencial acelerar o desempenho de determinadas VNFs em execução por meio de aceleradores de *hardware* externos. Em (KACHRIS; SIRAKOULIS; SOUDRIS, 2014), uma estrutura foi proposta para utilizar FPGAs para implementar funções de rede completas em *hardware*. Essa estrutura permite criar VNFs baseadas apenas em FPGA e não permite que a VNF em execução em uma CPU descarregue tarefas selecionadas para aceleradores de FPGA, isto é, a VNF deve estar executando no FPGA em sua totalidade. O compartilhamento da malha FPGA entre várias VNFs é realizado pela tecnologia de reconfiguração parcial.

Os autores em (Li et al., 2018) desenvolveram a *Dynamic Hardware Library* (DHL), uma biblioteca para abstrair aceleradores baseados em FPGA para VNFs, que podem ser acessados usando um conjunto de APIs. Este trabalho concentrou-se em diminuir a quantidade de esforço para acessar o acelerador a partir de VNFs. Byma et al. (2014) propôs uma estrutura que agrega regiões parcialmente reconfiguráveis em vários FPGAs para oferecer um único recurso de FPGA a um inquilino da nuvem que pode programar sua região alocada. Essa estrutura é útil para provedores de serviços em nuvem que desejam oferecer FPGAs como outros recursos de computação de alto desempenho.

O trabalho descrito em (GE et al., 2014) apresenta uma estrutura baseada em OpenStack para aceleração de *hardware* no contexto de NFV. Sendo assim, um nó do hipervisor fornece recursos FPGA para VNFs em execução nele. A estrutura descrita fornece provisionamento elástico e automatizado para aceleração de *hardware* para funções de rede. As VNFs têm acesso a um conjunto predefinido de comportamento acelerado (por exemplo, Tradução de Endereço de Rede (NAT, *Network address translation*), DPI, *firewall*) e se comunicam por meio de uma interface independente de *hardware* com o hipervisor para configurar o acelerador. Os resultados da avaliação mostram uma melhoria na produtividade em torno de uma ordem de magnitude em comparação com a VNF baseada em processador de propósito geral. No entanto, o provisionamento de aceleração de *hardware* não é livremente programável pela infraestrutura de rede.

3.5 Discussões

Diferentemente dos trabalhos mencionados, este trabalho propõe dividir cada instância de VNF em um conjunto de Componentes de VNF (VNFCs, VNF *components*), isto é, componentes internos que compreendem subconjuntos de uma funcionalidade completa de VNF. Desta forma, é possível economizar recursos do acelerador de *hardware*, uma vez que VNFs que tenham etapas de processamento em comum, podem compartilhar o mesmo módulo de processamento no acelerador. Além disso, este trabalho apresenta uma interface de gerência para o usuário final que não está presente em nenhum trabalho mencionado, sendo possível que o gerente de rede configure a infraestrutura para instanciar ou desalocar VNFs sistematicamente, abstraindo a disposição dos aceleradores de *hardware* no contexto da tarefa de gerência.

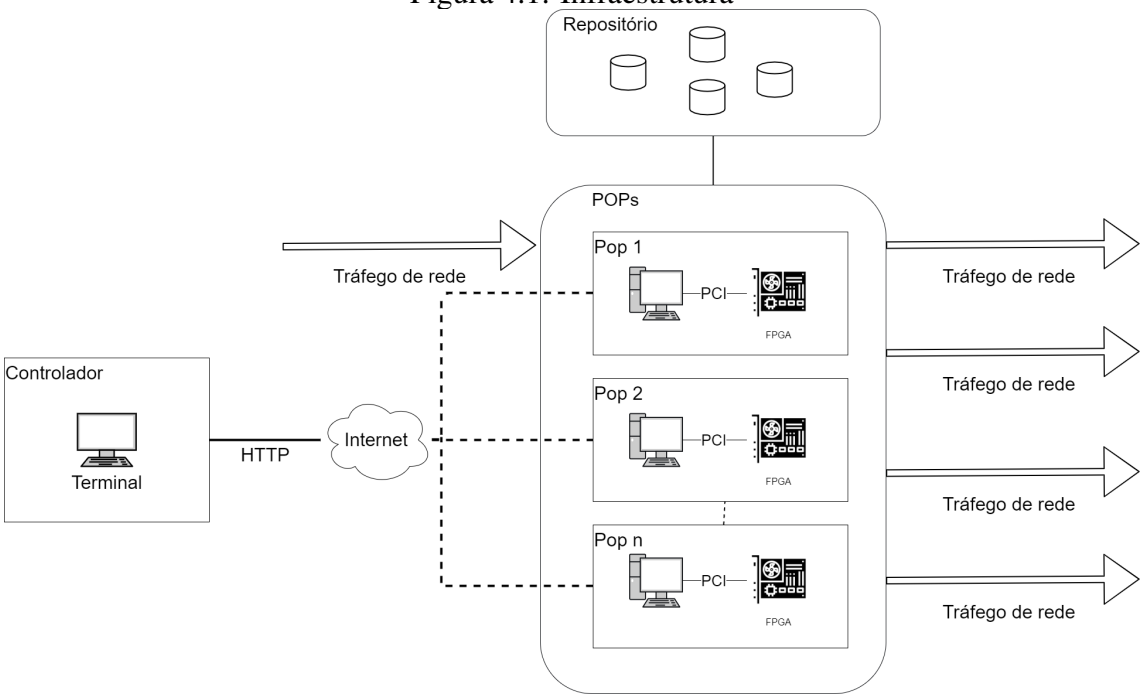
4 METODOLOGIA

Este capítulo apresenta os parâmetros de referência para execução e implementação de um ambiente NFV de alto desempenho com suporte à aceleração de *hardware* para processamento massivo de pacotes.

4.1 Infraestrutura

Foi projetada uma Infraestrutura NFV com suporte à aceleração de *hardware* para execução de funções de rede. Na Figura 4.1 é exibida a visão geral desta infraestrutura. A ideia principal é que o processamento massivo de pacotes seja efetuado pelos FPGAs para que seja possível manter uma alta taxa de transferência e uma baixa latência do ponto de vista das funções de rede instanciadas neste ambiente.

Figura 4.1: Infraestrutura



Fonte: o autor

A infraestrutura é composta por *Pontos de Presença* (PoPs, *Points of Presence*) que são integrados por um computador hospedeiro e um FPGA. Uma função de rede pode ser configurada nesse ambiente através da alocação e disposição de diversos PoPs. O tráfego de rede atravessa os PoPs, que são responsáveis, de fato, por fazerem o processamento de pacotes. A CPU presente no PoP está diretamente conectada com um FPGA através de uma interface PCIe. No FPGA é feito o contato direto com o tráfego de rede através das

interfaces Ethernet. A CPU tem o papel de consultar o *Repositório* para obter a base de dados responsável por reconfigurar o FPGA. Estes dados serão o conjunto de regras que determinam a funcionalidade da função de rede. Por fim, o *Controlador* é uma interface para o operador de rede poder gerenciar as VNFs que devem ser instanciadas por este ambiente em cima do tráfego corrente.

Na infraestrutura projetada, as VNFs instanciadas são particionadas em um conjunto de VNFCs. Esses componentes são alocados dinamicamente para compor sistematicamente uma VNF de interesse. Portanto, uma das principais características desta infraestrutura é que as VNFCs podem estar geograficamente divididas entre os PoPs. Desta forma, o tráfego pode atravessar diversos desses pontos para que, por fim, seja encaminhado para o próximo nodo da rede.

Essa fragmentação de VNFs implica que recursos possam ser distribuídos entre os PoPs, diminuindo a redundância de dados entre eles. Por exemplo, no caso de um *firewall* instanciado nesse ambiente, em um PoP pode estar presente um módulo de processamento que intercepta o pacote da interface de rede e, tipicamente, em outro módulo um bloco que filtra os pacotes com base nas regras do *firewall*. Por fim, em outro PoP pode estar a lógica de transmissão do pacote para o próximo nodo da rede.

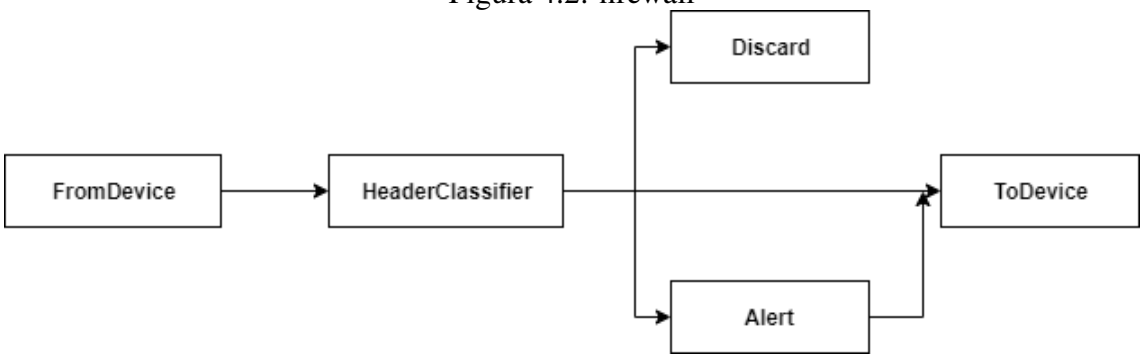
Os VNFCs exploram a propriedade de que as funções de rede utilizam um conjunto muito semelhante de etapas de processamento. Por exemplo, a maioria das funções de rede faz algum tipo de classificação baseada em cabeçalho. Em seguida, algumas delas realizariam alguma modificação de pacote (por exemplo, tradutores, balanceadores de carga). Outros, como Sistemas de Prevenção de Intrusões (IPSs, *Intrusion Prevention Systens*) e Sistemas de Prevenção de Vazamento de Dados (DLPs, *Network Data Loss Prevention*), classificariam ainda mais os pacotes com base no conteúdo da carga útil (um processo geralmente chamado de DPI). Algumas funções de rede usariam o gerenciamento de filas ativo antes de transmitir pacotes. Outros (como *firewalls* e IPSs) descartariam alguns pacotes ou acionariam um alerta ao administrador do sistema. Desta forma, cada bloco de processamento representa uma única unidade lógica encapsulada a ser executada em pacotes, como classificação de campo de cabeçalho ou modificação de campo de cabeçalho. Cada bloco possui uma única porta de entrada e zero ou mais portas de saída. Ao manipular um pacote, um bloco pode enviá-lo para uma ou mais de suas portas de saída. Cada porta de saída está conectada a uma porta de entrada de outro bloco.

O projeto OpenBox (BREMLER-BARR; HARCHOL; HAY, 2016) define um

conjunto de blocos de processamento que, tipicamente, são utilizados por funções de redes. A Tabela 4.1 exibe os blocos mais utilizados neste contexto. Com o objetivo de projetar um repositório com uma coleção de VNFCs, este trabalho usou como inspiração as subdivisões apresentadas pelo projeto OpenBox.

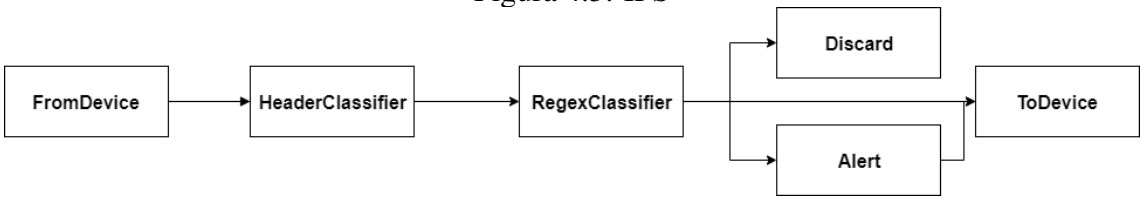
Nas Figuras 4.2 e 4.3 são ilustrados os principais componentes de um *firewall* e de um IPS compostos por blocos OpenBox. Pode-se notar que as duas funções têm blocos de processamento de pacotes semelhantes, diferindo apenas pela presença de um classificador regex no IPS. A Infraestrutura implementada visa explorar essa similaridade entre as funções de rede, eliminando possíveis redundâncias, uma vez que esses blocos de processamento podem ser reaproveitados e compartilhados entre uma ou mais funções de rede que estejam sendo instanciadas em determinado instante.

Figura 4.2: firewall



Fonte: o autor

Figura 4.3: IPS



Fonte: o autor

Tabela 4.1: Blocos de processamento de pacote especificados pelo projeto OpenBox

Função	Descrição	Dispositivo
FromDevice	Lê pacotes de um dispositivo de rede	FPGA
ToDevice	Envia pacotes para um dispositivo de rede	FPGA
Discard	Descarta pacotes e registra número de pacotes descartados	CPU
HeaderClassifier	Classifica pacotes através de uma lista de map (string,value), onde este map corresponde a um nome de campo e o valor esperado neste campo. Esta lista será percorrida em ordem e o pacote é enviado para a saída correspondente ao primeiro padrão encontrado	FPGA
RegexClassifier	Classifica pacote através de uma lista de expressões regex. Esta lista será percorrida em ordem e o pacote é enviado para a saída correspondente ao primeiro padrão encontrado	FPGA
Alert	Envia uma mensagem de alert ao controlador	CPU
Ipv4AddressTranslator	Tradutor de endereços em uma rede baseada em IPv4	CPU

4.2 Implementação

Nesta seção será apresentado o sistema final implementado, demonstrando os detalhes de implementação e as tecnologias utilizadas. Será apresentada a plataforma NetFPGA, que foi escolhida como principal *framework* para o desenvolvimento da infraestrutura baseada em FPGA.

O sistema final implementado é composto por um PoP e um controlador. Para simplificação, foi eliminada a presença do elemento Repositório, sendo o controlador o responsável por carregar, de fato, a base de dados para configuração de determinada função de rede.

Finalmente, será descrito o experimento final, tratando-se da implementação de um *firewall* dinamicamente configurável. Desta forma, é possível que o gerente de rede, configure *on the fly* o conjunto de regras aplicáveis aos pacotes que transpassem o *firewall*

instanciado.

4.2.1 Arquitetura da NetFPGA

O projeto NetFPGA é um esforço para desenvolver *hardware* e *software* de código aberto para prototipagem rápida de dispositivos de redes de computadores. O projeto teve como alvo pesquisadores acadêmicos, usuários da indústria e estudantes (Gibb et al., 2008). O NetFPGA utiliza uma abordagem baseada em FPGA para criar protótipos de dispositivos de rede. Isso permite que os usuários desenvolvam projetos capazes de processar pacotes na taxa de linha, algo que não é possível em abordagens baseadas em *software*.

A NetFPGA é particularmente útil como uma alternativa de *hardware* flexível e de baixo custo para avaliação de protótipos de pesquisa. NetFPGAs já foram utilizadas em vários projetos de pesquisa (Sun; Kim, 2011; Chou et al., 2011; Zilberman et al., 2014; Dorosh; Debita; Schauer, 2017). Comparada com a abordagem de implementar soluções em *software*, a NetFPGA garante processamento e encaminhamento na taxa de transmissão das interfaces (sem sacrificar banda), baixa latência e não onera a CPU do computador.

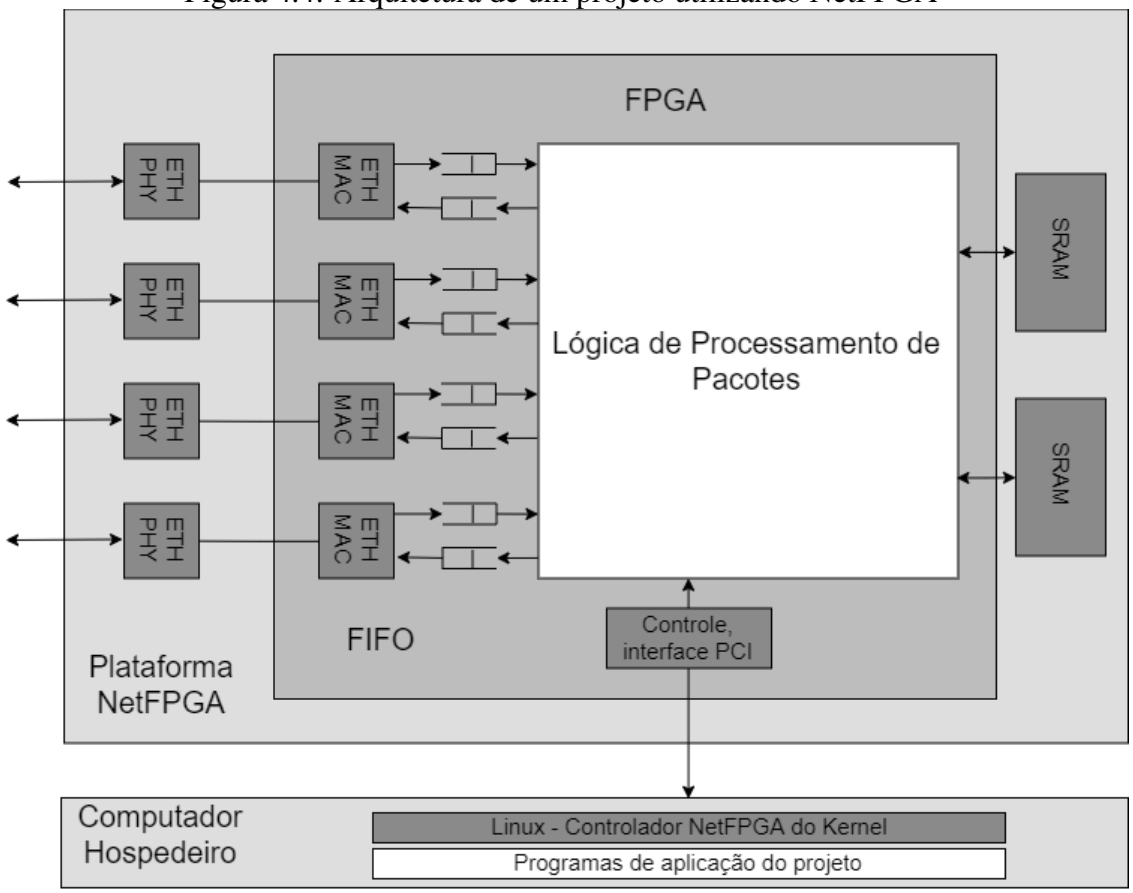
4.2.2 Hardware

Este trabalho será baseado no modelo NetFPGA-1G-CML, que é um dispositivo de *hardware* combinado com uma base de código-fonte aberto que, em conjunto, permite a rápida criação de protótipos de dispositivos de rede. O *hardware* em si é uma placa PCI Express com portas Gigabit Ethernet 4 x 1 e um FPGA Xilinx Kintex-7 XC7K325T-1FFG676. Com base no fluxo de projeto do *software* Xilinx Embedded Development Kit (Patel, 2006) e na especificação da interface AXI4 da ARM, a base de código-fonte aberto da plataforma inclui projetos de referência, uma biblioteca IP e ferramentas para ajudar o usuário a realizar um novo projeto rapidamente.

O FPGA é capaz de processar pacotes recebidos pelas portas Ethernet em plena velocidade (1 Gbps em modo de operação full-duplex). A NetFPGA possui também um controlador PCI que permite programação e comunicação com o FPGA. Uma SRAM Cypress CY7C2263KV18 QDRII + Quad Data de 4,5 MB é fornecida para aplicativos que

exigem memória de alta velocidade e baixa latência. A SRAM pode realizar uma operação de leitura ou escrita por ciclo de relógio e retorna dados lidos em três ciclos de relógio. A SRAM é ideal para aplicações que fazem acessos frequentes a pequenas quantidades de memória, como encaminhamento de pacotes e implementação de contadores SNMP. O NetFPGA-1G-CML inclui um SDRAM DDR3 de 512 MB da Micron MT41K512M8, que emprega um barramento de dados de 800 MHz, capaz de operar a uma taxa de dados de 1600 mbps. A SDRAM funciona de forma assíncrona e requer atualização (*refreshing*) contínua dos dados. Por estes fatores, o controlador da SDRAM é significativamente mais complexo que o controlador da SRAM. Apesar da maior latência, a DRAM tem vazão alta. A DRAM é ideal para aplicações que precisam de uma quantidade maior de memória, como armazenamento temporário (*buffering*) de pacotes. A SDRAM também permite uma hierarquização da memória da NetFPGA. A Figura 4.4 mostra uma visão geral da placa da NetFPGA.

Figura 4.4: Arquitetura de um projeto utilizando NetFPGA

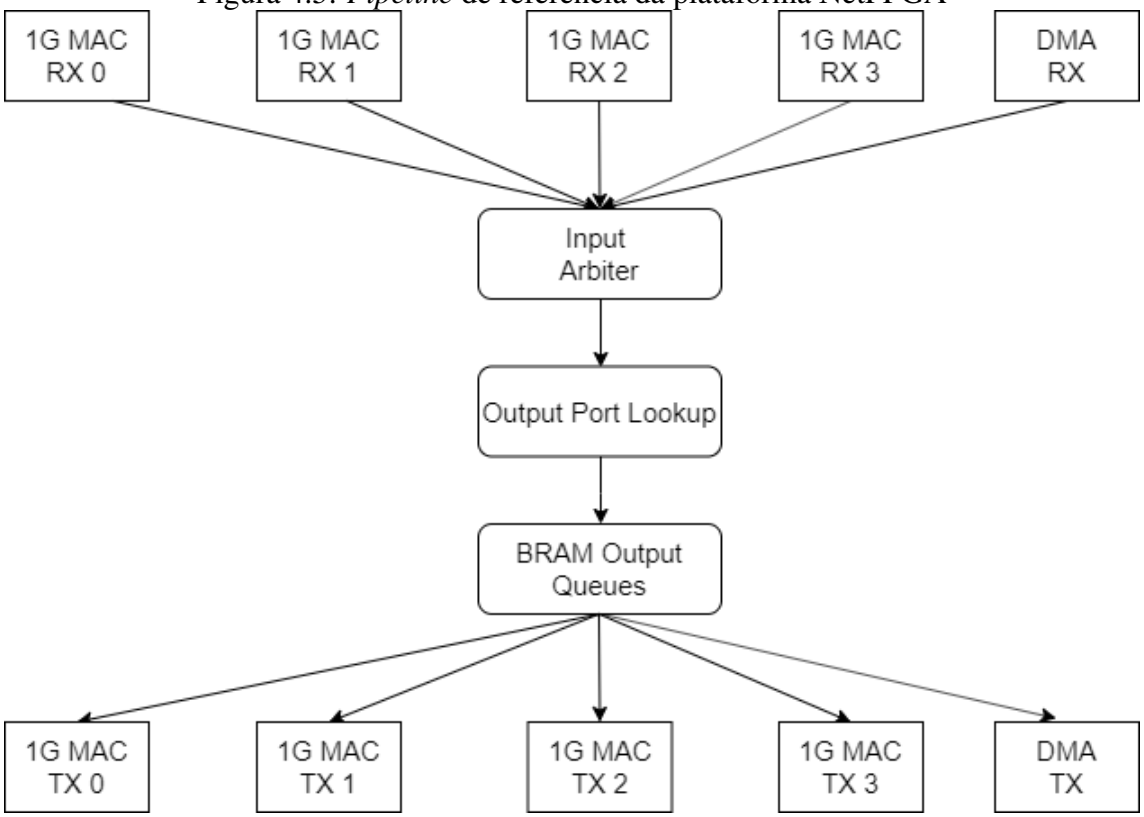


Fonte: o autor

4.2.3 *Pipeline* de referência

Um *pipeline* de referência NetFPGA, ilustrado na Figura 4.5, consiste em filas associadas às portas físicas, isto é, quatro portas físicas para quatro filas de entrada de *frames* recebidos e 4 filas de saída para *frames* transmitidos. Essas portas podem ser detectadas pelo sistema operacional através de um *driver*. Um *frame* de entrada é colocado em uma fila adequada (por exemplo, Rx 0) e enviado ao módulo *Input Arbiter* que escolhe uma das filas de entrada, pega o pacote e o envia ao longo do *pipeline*. Devido a esse fato, não é crucial por qual porta / fila o pacote está chegando, porque todos os pacotes são enviados para um único *pipeline*. Em seguida, os pacotes são processados em módulos ao longo do *pipeline*. Logo, o módulo *Output Port Lookup* decide para qual fila de saída o pacote deve ser enviado. Por fim, assim que um pacote chega ao módulo BRAM *Output Queues*, ele já possui um destino marcado (fornecido em um canal lateral), de acordo com este destino, ele é inserido em uma fila de saída dedicada. Existem cinco filas de saída: uma por cada porta 1G e uma no bloco DMA. Observe que um pacote pode ser descartado se sua fila de saída estiver cheia ou quase cheia. Quando um pacote atinge o topo de sua fila de saída, ele é enviado para a porta de saída correspondente, sendo um módulo MAC ou o módulo DMA. Um novo projeto no NetFPGA pode ser iniciado do zero, o que significa preparar todo o código para o processamento de pacotes, ou pode-se preparar apenas um módulo e inseri-lo no *pipeline* de referência.

Figura 4.5: *Pipeline* de referência da plataforma NetFPGA



Fonte: o autor

4.2.4 Conjunto de VNFCs implementado

A fim de se ter uma base de dados para composição sistemática de múltiplas VNFs, foi implementado um conjunto de VNFCs. Essa coleção de VNFCs foi descrita para ser executada no acelerador de *hardware*, mais especificamente nos FPGAs, a partir da linguagem de descrição de *hardware* com o objetivo de provisionar uma alta taxa de transferência à infraestrutura e, além disso, economizar recursos dos aceleradores, uma vez que muitos desses VNFCs podem ser, tipicamente, compartilhados entre diversas VNFs instanciadas no sistema. A Tabela 4.2 apresenta e descreve a coleção de VNFCs implementada.

4.2.5 *Firewall*

Os *firewalls* são usados para examinar o tráfego de rede e aplicar políticas com base nas instruções contidas no conjunto de regras configurado no *firewall*. O *firewall* projetado neste trabalho se baseia em uma política de regras de 5-tupla (IP de Origem,

Tabela 4.2: Repositório de VNFCs implementados

VNFC	Descrição
Receive Packet	Lê pacote da fila de entrada
Forward Packet	Encaminha pacote para fila de saída, onde posteriormente será redirecionado para o próximo nodo da rede
Decrement TTL	Decrementa tempo de vida do pacote
Update Checksum	Atualiza checksum do pacote com base no decremento
Header Classifier	Classifica o pacote com base em um campo do cabeçalho e uma regra referente a este campo. Além disso, retorna uma flag para indicar se houve correspondência do campo com a regra recebida.

Porta de Origem, IP de Destino, Porta de Destino, Protocolo) para classificar o tráfego. Portanto, para cada pacote analisado, deve-se verificar o conjunto de regras configurado para determinar se o pacote deve ser descartado ou encaminhado para o próximo nodo da rede. A Tabela 4.3 ilustra um conjunto de regras que pode ser configurado no *firewall* projetado.

Tabela 4.3: Exemplo de um conjunto de regras configurados no *firewall*

Regra	IP Origem	IP Destino	Porta Origem	Porta Destino	Protocolo
r1	10.0.0.1	*	*	80	TCP
r2	10.0.0.2	192.168.0.5	6000	*	*
r3	192.168.0.25	192.168.0.32	5666	80	UDP
r4	*	10.0.0.33	*	*	ICMP

É importante ressaltar que em *firewalls* comerciais existe uma coluna adicional, chamada Ação. Essa coluna é responsável por ditar a operação que deve ser efetuada, uma vez que haja uma correspondência com os demais campos da 5-tupla de uma determinada regra (por exemplo, *discard* para descartar um pacote). Para simplificação, essa coluna foi eliminada na política de regras do *firewall* implementado. Sendo assim, uma vez que haja a correspondência com qualquer regra do conjunto total de regras configurado, o pacote será descartado.

O *firewall* foi projetado para ser executado sobre a infraestrutura heterogênea implementada. Além disso, a sua instância é composta pelo conjunto de VNFCs do repositório criado. Para descrever o funcionamento do *firewall*, é interessante analisar o plano de dados e o plano de controle. Por um lado, o plano de dados do *firewall* está inteiramente implementado em FPGA e seu código escrito em Verilog, visando maximizar a taxa de transferência do tráfego de rede corrente. Este módulo está descrito na Subseção 4.2.5.1. Por outro lado o plano de controle, responsável por configurar o conjunto de regras a serem aplicadas pelo *firewall* em cima do tráfego, é executado no computador hospedeiro e

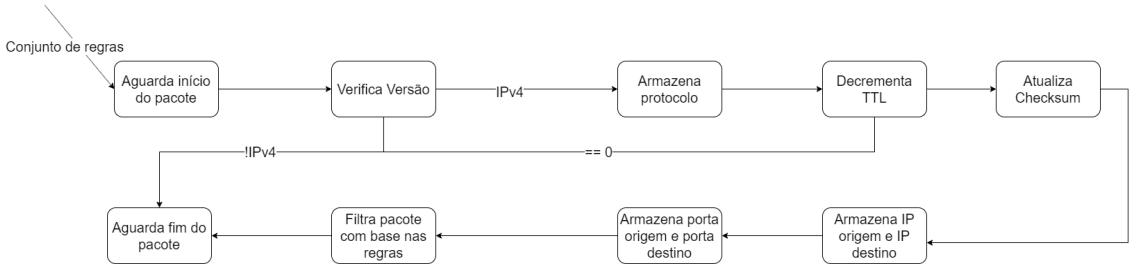
está escrito em linguagem C. Este módulo está descrito na Subseção 4.2.5.2.

4.2.5.1 Plano de Dados

O plano de dados do *firewall* segue a lógica da máquina de estados da Figura 4.6, onde o conjunto de regras, isto é, a política do *firewall*, está armazenada na memória SRAM do FPGA. Portanto, a ideia básica do plano de dados é processar um conjunto de regras e analisar o cabeçalho do pacote, através da máquina de estados, para verificar se ele precisa ser descartado antes de transmiti-lo ao próximo módulo.

A cada ciclo de relógio é executado um bloco de processamento da máquina de estados apresentada e uma palavra do pacote - isto é, 16 bytes - é encaminhada para o bloco subsequente. A Tabela 4.4 mostra quais dados são transferidos no barramento de encaminhamento a cada ciclo de relógio. É importante evidenciar que para receber e armazenar o cabeçalho do pacote é necessário uma fila de saída auxiliar, onde colocam-se as palavras de dados que já foram recebidas e processadas até decidir se pacote deve ser enviado para o módulo seguinte ou ser descartado.

Figura 4.6: Máquina de estados



Fonte: o autor

Tabela 4.4: Pacote separado em 7 conjuntos de palavras para os cabeçalhos

Palavra	64:48	47:32	31:16	15:0
1	Eth source addr			Eth dest addr
2	Eth dest addr		EtherType	IPVer, HL, ToS
3	packet size	IP ID	flags, frag	TTL, proto
4	checksum	source IP		destination IP
5	destination IP	source port	dest port	seq. no.
6	seq. no.	acknowledgement		flags
7	adv. window	checksum	urgent ptr.	payload
...	payload			

A seguir, será descrito o processamento realizado pelo *firewall* em cada um dos

estados descritos na Figura 4.6.

- Primeiro estado: esperando o início de pacotes. O primeiro estado da máquina de estados aguarda o início do recebimento de um pacote. Se não há dado a processar ou se o próximo módulo não está pronto para receber, continua-se neste estado sem avançar as filas de entrada e saída. Se há dado a processar e o próximo módulo está pronto para recebê-lo, a máquina de estados avança a fila.
- Segundo estado: verificando a versão. O segundo estado do *firewall* processa a segunda palavra do pacote e verifica se o pacote é um pacote IPv4. Para isso é verificado se o tipo do cabeçalho Ethernet (EtherType) é 0x0800, que indica um pacote IP, e se a versão do protocolo IP é 4. Em caso positivo, prossegue-se para o próximo estado para verificar o protocolo do pacote. Caso o pacote não seja um pacote IPv4 ele deverá ser encaminhado pela rede sem ser filtrado, neste caso, avança-se para o nono e último estado, onde será aguardado o término de recepção do pacote.
- Terceiro estado: verificando o protocolo. O terceiro estado do *firewall* armazena os dados da terceira palavra do pacote temporariamente até que o campo protocolo seja verificado no oitavo estado da máquina de estados.
- Quarto estado: descrementando TTL. Neste estado apenas o tempo de vida do pacote é decrementado. Caso o tempo de vida esteja com o valor zero após o decremento, este pacote é descartado, desta forma, avança-se para o nono e último estado, onde será aguardado o término de recepção do pacote. Caso contrário, avança-se para o quinto e estado.
- Quinto estado: atualizando checksum. Uma vez que o tempo de vida do pacote é decrementado, é necessário que o campo *checksum* seja atualizado. Sendo assim, é necessário subtrair neste campo, o valor subtraído ao TTL.
- Sexto estado: verificando IP origem e destino. O sexto estado do *firewall* armazena os dados da quarta palavra do pacote temporariamente até que os campos IP origem e destino sejam verificados no oitavo estado da máquina de estados.
- Sétimo estado: verificando porta origem e destino. O sétimo estado do *firewall* armazena os dados da quinta palavra do pacote temporariamente até que os campos porta origem e destino sejam verificados no oitavo estado da máquina de estados.
- Oitavo estado: verificando porta origem e destino. O oitavo estado é responsável por processar os dados de interesse, do pacote, temporariamente armazenados.

Caso os campos da 5-tupla correspondam a alguma das regras configuradas, o pacote é instantaneamente marcado para descarte e encaminhado para o próximo estado. Caso contrário, o pacote é apenas encaminhado para o próximo estado.

- Nono estado: Aguardando o fim do pacote. O nono e último estado é responsável por aguardar o recebimento total do pacote e, logo em seguida, verificar se o pacote deve ser descartado ou não. Caso o pacote não seja descartado ele é encaminhado ao próximo estágio do *pipeline*.

4.2.5.2 Plano de Controle

O computador hospedeiro executa um servidor REST, implementado em PHP através do *framework* Lumen, responsável por escutar chamadas de API. Existem duas chamadas possíveis. A primeira é responsável por ligar ou desligar o funcionamento do *firewall*. A segunda chamada tem como objetivo configurar o conjunto de regras aplicadas pelo firewall. Desta forma é necessário transmitir um JSON que defina a política de regras do *firewall*. Uma vez que um comando de configuração é recebido, é feita uma atualização em um arquivo JSON que armazena o conjunto de regras atual.

Uma segunda aplicação é responsável por monitorar o diretório onde o arquivo de configurações está inserido, sendo assim, toda vez que há uma atualização neste local esta aplicação trata de coletar os dados desse arquivo, fazendo o parser e reconfigurando as regras no FPGA através da interface PCIe. Essa reconfiguração é feita através da escrita das regras do *firewall* na memória SRAM do FPGA.

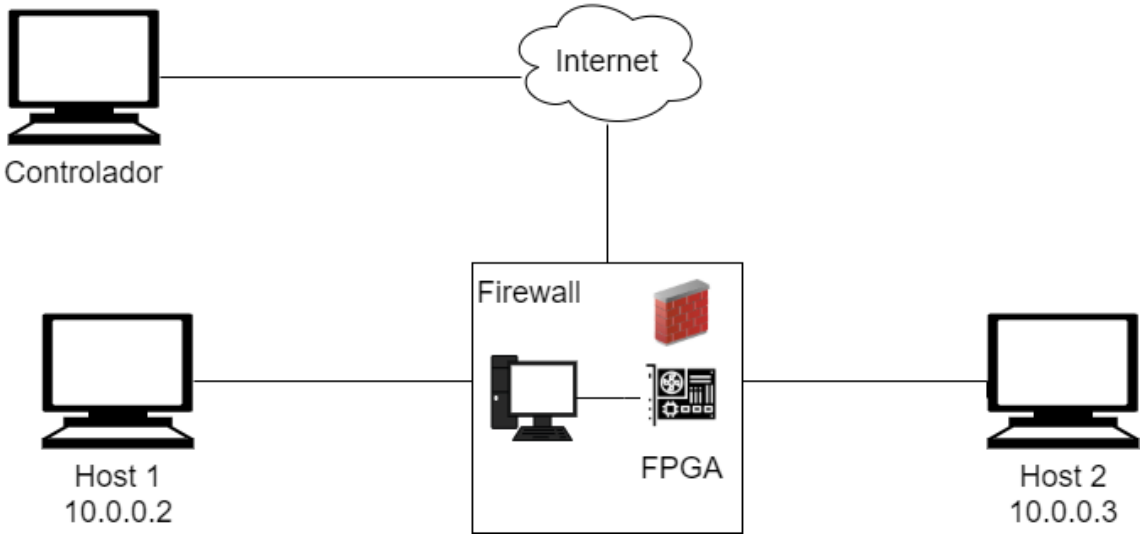
A comunicação entre o computador hospedeiro e o FPGA se dá a partir de escritas diretas na memória, isto é, através da interface DMA, que pode ser diretamente acessada do sistema operacional. Os dados escritos nessa interface são diretamente alocados na memória SRAM do FPGA a partir de uma definição de endereço e um valor correspondente. A plataforma NetFPGA é responsável por essa abstração, portanto, é importante ressaltar que esses dados enviados à interface DMA são repassados ao FPGA através da interface PCI, na qual o FPGA está diretamente conectado.

Por fim, o gerente de rede, conhecedor desta API previamente definida pode operar sobre a infraestrutura de forma livre através de chamadas HTTP. Do ponto de vista deste usuário, o arranjo dos recursos da infraestrutura é totalmente transparente.

5 RESULTADOS

Neste capítulo são relatados e discutidos os resultados obtidos a partir da infraestrutura projetada. Para isso, foi avaliada a eficácia da implementação através de experimentos com conjuntos de regras de classificação de pacotes baseados na política de 5-tupla. Para avaliar o desempenho do sistema de *firewall* baseado em NetFPGA, a rede mostrada na Figura 5.1 foi configurada.

Figura 5.1: Ambiente configurado para a análise de resultados do *firewall* baseado em FPGA



Fonte: o autor

Com o objetivo de comparar o desempenho do *firewall* baseado em FPGA com uma solução baseada totalmente em *software*, foi utilizado o pacote de *firewall* em execução no Linux, o *Iptables*. Sendo assim, configurou-se, em todos os experimentos, o *software Iptables* com o mesmo conjunto de regras configurado no *firewall* baseado em FPGA. O computador onde este experimento foi executado é equipado com o *hardware* especificado na Tabela 5.1.

Os experimentos apresentados neste capítulo compararam a latência e a taxa de transferência de ambos os *firewalls* com seis tamanhos diferentes de conjunto de regras. Foi um experimento quantitativo que teve como objetivo apresentar a diferença estatística em duas soluções de *firewalls* diferentes. Além disso, é importante ressaltar que cada um dos experimentos para análise de desempenho foi executado 30 vezes para cada configu-

Tabela 5.1: *Hardware* equipado para execução do *firewall* Iptables

CPU	Intel Core i5-4670 @ 3,40 GHz
RAM	8 GB 1600 MHz DDR3
Placa de rede	Intel 1000 82574

ração.

5.1 Recursos Utilizados do FPGA

Com o objetivo de analisar os recursos utilizados pelo FPGA, variou-se a quantidade máxima de regras suportadas pelo *firewall* e o FPGA foi configurado sistematicamente para cada uma dessas variações. Sendo assim, a Tabela 5.2 exibe os resultados após os processos de posicionamento e de roteamento do FPGA para cada uma das configurações mencionadas.

Tabela 5.2: Recursos Utilizados pelo FPGA

Número de Regras	Clock máximo (MHz)	CLB slices utilizados
50	160.2	6545 (12,8 %)
100	159.2	7641 (14,9 %)
200	159.1	7876 (15,1 %)
300	158.3	8182 (15,4 %)
400	155.1	8253 (16,1 %)
500	153.5	8537 (16,7 %)

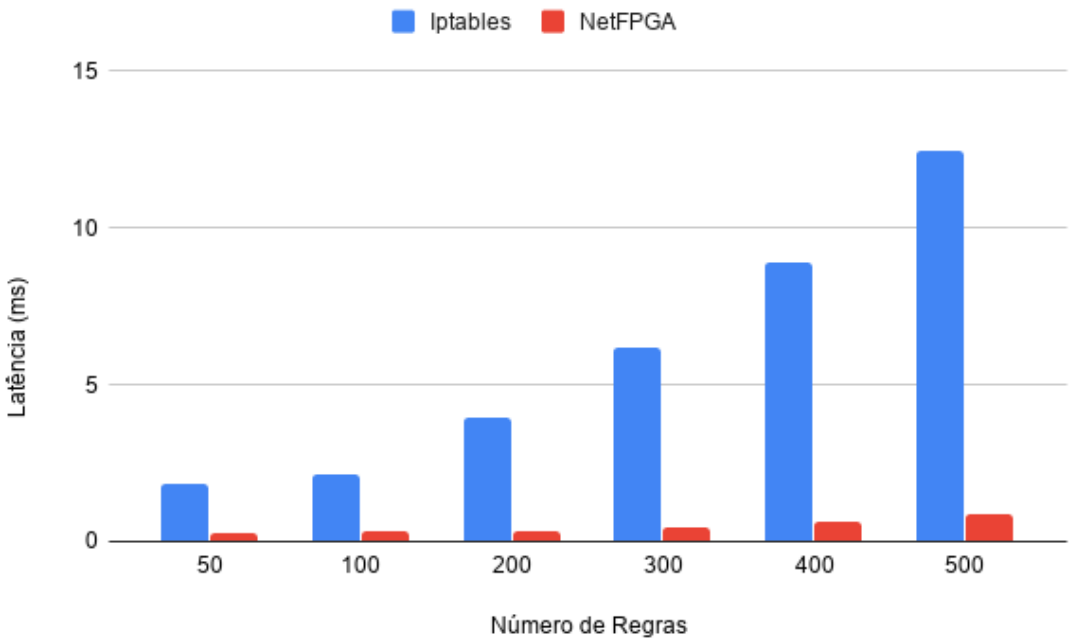
É possível notar que o projeto atingiu uma frequência máxima de *clock* de 160.2 MHz para o caso em que havia suporte para no máximo 50 regras, enquanto no caso mais crítico, isto é 500 regras suportadas, foi possível obter a frequência máxima de *clock* de 153.5 MHz. O número de *slices* utilizados pelo chip após a sintetização é relativamente pequeno, 16,7 % para o cenário de 500 regras configuradas.

5.2 Latência

A latência é uma característica imediatamente perceptível das comunicações em rede, independentemente da aplicação, e um fator significativo no fornecimento de uma experiência positiva ao usuário. Por conta disso, este experimento foi realizado.

Os resultados da latência foram coletados utilizando a ferramenta *ping* e representam o tempo de ida e volta dos pacotes ICMP, ou seja, foram enviadas solicitações de eco do ICMP de um hospedeiro para outro e foi registrando o tempo decorrido até o momento que a resposta de eco do ICMP é retornada.

Figura 5.2: Latência média em diferentes configurações de *firewalls*



Fonte: o autor

Tabela 5.3: Resultado da latência média em ms para diferentes conjuntos de regras

	50	100	200	300	400	500
Iptables	1,86	2,16	3,98	6,17	8,88	12,49
NetFPGA	0,26	0,33	0,35	0,47	0,66	0,89

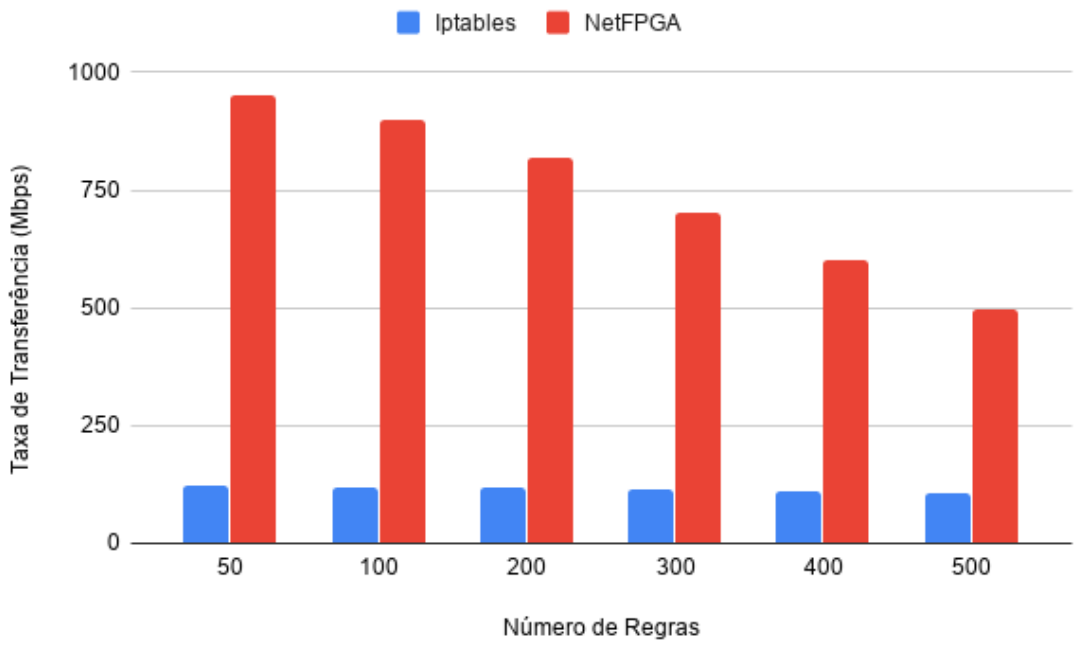
O gráfico da Figura 5.2 e a Tabela 5.3 exibem os resultados do experimento. Nota-se que à medida que mais regras são configuradas a latência aumenta em ambos *firewalls*, no entanto no *firewall Iptables* esse crescimento é significativamente mais acentuado. Por fim, é possível observar que a latência é consideravelmente menor em todos os cenários, para a solução em FPGA.

5.3 Taxa de Transferência

A taxa de transferência é um indicador importante do desempenho e da qualidade de uma conexão de rede, é responsável por indicar a taxa de entrega bem-sucedida de pacotes de um ponto na rede para outro. A taxa de transferência da rede é afetada por vários fatores. Isso inclui atributos como poder de processamento de *hardware*, incluindo cabos e roteadores.

Para executar a transferência de dados bidirecional entre os computadores conectados ao *firewall* e assim avaliar a taxa de transferência média, foi utilizada a ferramenta Nping. O Nping permite que sejam gerados pacotes de rede com uma ampla variedade de protocolos, permitindo que se configure praticamente qualquer campo do cabeçalho do protocolo.

Figura 5.3: Taxa de transferência média em diferentes configurações de *firewalls*



Fonte: o autor

Tabela 5.4: Resultado da taxa de transferência média em Mbps para diferentes conjuntos de regras

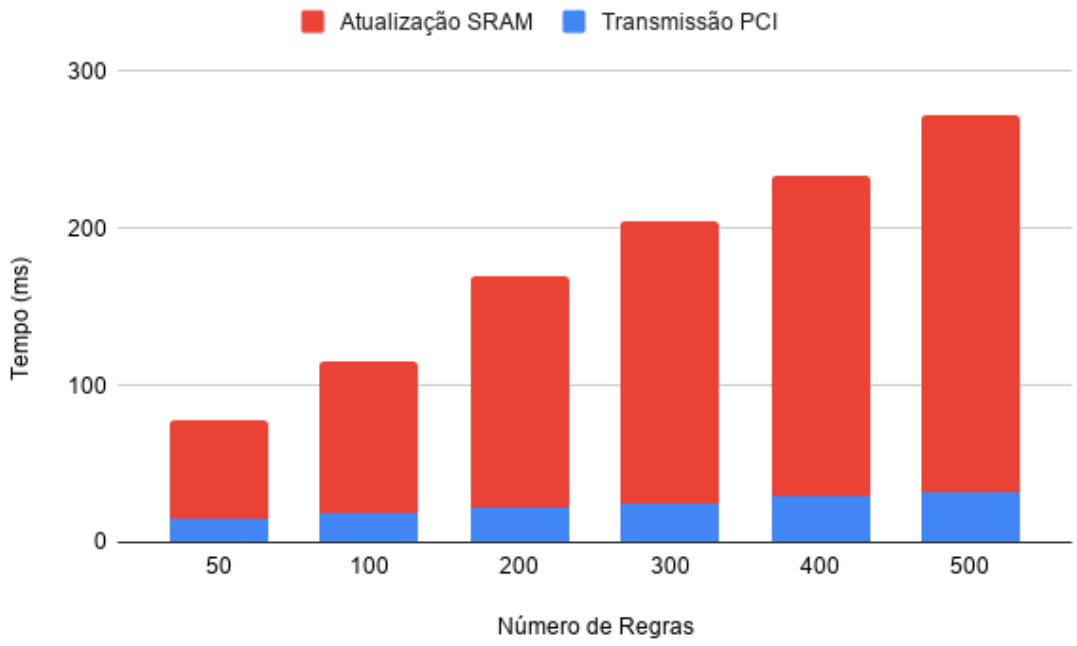
	50	100	200	300	400	500
Iptables	123,3	119,6	118,2	115,9	110,4	102,7
NetFPGA	951,1	898,6	820,1	701,4	601,8	498,23

O gráfico com as respectivas taxas de transferência é ilustrado na Figura 5.3, e na Tabela 5.4 pode ser verificado o valor de cada medida. Novamente é possível observar que o *firewall* baseado em FPGA tem um desempenho consideravelmente melhor, desta vez aproximando-se da taxa de linha (1 Gbps) no melhor caso (50 regras). À medida que conjuntos de regras maiores são configurados, a taxa de transferência se distancia da máxima e decai gradualmente.

5.4 Tempo de Atualização do Conjunto de regras

Neste experimento foi medido o tempo necessário para atualizar o conjunto de regras que especificam endereços IPv4 de origem e destino, o protocolo de transporte, e as portas de origem e destino. A partir do gráfico da Figura 5.4 nota-se que o tempo de atualização do conjunto de regras é diretamente proporcional à quantidade de regras transferidas. É importante ressaltar que o tempo de atualização é dividido entre a transferência dos dados do computador hospedeiro (via PCIe) e o endereçamento desses dados na memória SRAM por parte do FPGA e representa o período de inoperabilidade da função de rede corrente.

Figura 5.4: Tempo necessário para que o conjunto de regras do *firewall* seja atualizado



Fonte: o autor

Embora o tempo de reconfiguração do *firewall* através da escrita de regras na memória seja significativo (na ordem de milissegundos), é muito menor em relação ao processo de programar e ressintetizar o FPGA a partir de um novo arquivo de configuração sempre que haja uma modificação na política de regras, uma vez que este processo pode levar de minutos a horas para ser completado.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

No contexto de NFV, é importante utilizar um sistema eficiente e que tenha pouco ou nenhum efeito sobre o desempenho geral da rede. Portanto, este trabalho propôs uma infraestrutura NFV heterogênea, escalável e de alto rendimento, baseada em FPGA para processamento de pacotes.

A infraestrutura NFV proposta por esse trabalho pode ser usada para construção de soluções de alto desempenho e custo otimizado de sistemas. Foram projetados módulos reconfiguráveis e repositório de VNFCs para permitir reduzir a engenharia e esforços durante implementações de novas funções de rede. Além disso, foi apresentado uma estrutura de gerência para o usuário final para que as funções de redes possam ser instanciadas, configuradas e desalocadas, sistematicamente, conforme o interesse do gerente de rede.

Através deste estudo e experiência de pesquisa foi possível avaliar os benefícios da aceleração de *hardware* no contexto de NFV. A partir da implementação de um *firewall* heterogêneo baseado em FPGA e dinamicamente configurável, verificou-se os ganhos de desempenho relacionados à aplicação de FPGAs em ambientes de rede para processamento massivo de pacote. Esta solução foi comparada com soluções tradicionais baseadas em *software* com o objetivo de apresentar as diferenças de desempenho entre as duas abordagens.

Por fim, é importante que o tempo de inoperabilidade das VNFs instanciadas seja minimizado, desta forma, para trabalhos futuros, é possível explorar técnicas de otimização de busca e endereçamento de dados na memória do FPGA, com o objetivo de que o tempo de reconfiguração seja o menor possível. Para facilitar a implementação de novas funções de rede é ideal que haja uma coleção maior de VNFCs no repositório construído e que o estudo apresentado neste trabalho seja estendido a diversas funções de rede.

REFERÊNCIAS

Ajami, R.; Dinh, A. Design a hardware network firewall on fpga. In: **2011 24th Canadian Conference on Electrical and Computer Engineering(CCECE)**. [S.l.: s.n.], 2011. p. 000674–000678.

Alwakeel, A. M.; Alnaim, A. K.; Fernandez, E. B. Toward a reference architecture for nfv. In: **2019 2nd International Conference on Computer Applications Information Security (ICCAIS)**. [S.l.: s.n.], 2019. p. 1–6.

Bonafiglia, R. et al. Assessing the performance of virtualization technologies for nfv: A preliminary benchmarking. In: **2015 Fourth European Workshop on Software Defined Networks**. [S.l.: s.n.], 2015. p. 67–72. ISSN 2379-0369.

Brebner, G. Programmable hardware for software defined networks. In: **2015 European Conference on Optical Communication (ECOC)**. [S.l.: s.n.], 2015. p. 1–3.

BREMLER-BARR, A.; HARCHOL, Y.; HAY, D. Openbox: A software-defined framework for developing, deploying, and managing network functions. In: **Proceedings of the 2016 ACM SIGCOMM Conference**. New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 511–524. ISBN 978-1-4503-4193-6. Available from Internet: <http://doi.acm.org/10.1145/2934872.2934875>.

CARPENTER, B.; BRIM, S. **Middleboxes: Taxonomy and Issues**. IETF, 2002. RFC 3234 (Informational). (Request for Comments, 3234). Available from Internet: <http://www.ietf.org/rfc/rfc3234.txt>.

Chang, Y. et al. Composing middlebox and traffic engineering policies in sdns. In: **2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)**. [S.l.: s.n.], 2017. p. 396–401.

Chia-Nan Kao et al. Fast proxyless stream-based anti-virus for network function virtualization. In: **Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)**. [S.l.: s.n.], 2015. p. 1–5.

Chiotakis, S.; Pinneterre, S.; Paolino, M. vfpgamanager: A hardware-software framework for optimal fpga resources exploitation in network function virtualization. In: **2019 European Conference on Networks and Communications (EuCNC)**. [S.l.: s.n.], 2019. p. 47–51.

Chou, L. et al. Design of sfc management system based on sdn and nfv. In: **2018 International Conference on Information and Communication Technology Convergence (ICTC)**. [S.l.: s.n.], 2018. p. 391–395.

Chou, L. et al. The implementation of multilayer virtual network management system on netfpga. In: **2011 IEEE 17th International Conference on Parallel and Distributed Systems**. [S.l.: s.n.], 2011. p. 988–991.

Deng, X. et al. A sequence encoding scheme for multi-match packet classification. In: **2009 International Conference on Networks Security, Wireless Communications and Trusted Computing**. [S.l.: s.n.], 2009. v. 1, p. 641–644. ISSN null.

DONG, Q. et al. Packet classifiers in ternary cams can be smaller. In: **Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems**. New York, NY, USA: ACM, 2015. (SIGMETRICS '06/Performance '06), p. 311–322. ISBN 1-59593-319-0. Available from Internet: <http://doi.acm.org/10.1145/1140277.1140313>.

Dorosh, S.; Debita, G.; Schauer, P. Network hardware analyzer based on netfpga 1g. In: **2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)**. [S.l.: s.n.], 2017. p. 150–154.

Dorta, T. et al. Overview of fpga-based multiprocessor systems. In: **2009 International Conference on Reconfigurable Computing and FPGAs**. [S.l.: s.n.], 2009. p. 273–278.

ETSI GS NFV 002. **Network Functions Virtualization (NFV); Architectural Framework v1.1.1**. [S.l.], 2013. Available from Internet: <http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf>.

EZZATI, S. et al. Intelligent firewall on reconfigurable hardware. **European Journal of Scientific Research**, v. 47, p. 509–516, 12 2010.

GE, X. et al. Openanfv: Accelerating network function virtualization with a consolidated framework in openstack. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 4, p. 353–354, aug. 2014. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/2740070.2631426>.

Gibb, G. et al. Netfpga—an open platform for teaching how to build gigabit-rate network switches and routers. **IEEE Transactions on Education**, v. 51, n. 3, p. 364–369, Aug 2008.

Hager, S. et al. Mpfc: Massively parallel firewall circuits. In: **39th Annual IEEE Conference on Local Computer Networks**. [S.l.: s.n.], 2014. p. 305–313. ISSN 0742-1303.

HAN, S. et al. Packetshader: a gpu-accelerated software router. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 41, n. 4, p. –, aug. 2010. ISSN 0146-4833. Available from Internet: <http://dl.acm.org/citation.cfm?id=2043164.1851207>.

Huang, S.; Cuadrado, F.; Uhlig, S. Middleboxes in the internet: A http perspective. In: **2017 Network Traffic Measurement and Analysis Conference (TMA)**. [S.l.: s.n.], 2017. p. 1–9.

JAMSHED, M. A. et al. Kargus: A highly-scalable software-based intrusion detection system. In: **Proceedings of the 2012 ACM Conference on Computer and Communications Security**. New York, NY, USA: ACM, 2012. (CCS '12), p. 317–328. ISBN 978-1-4503-1651-4. Available from Internet: <http://doi.acm.org/10.1145/2382196.2382232>.

JANG, K. et al. Sslshader: Cheap ssl acceleration with commodity processors. In: **Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2011. (NSDI'11), p. 1–14. Available from Internet: <http://dl.acm.org/citation.cfm?id=1972457.1972459>.

KACHRIS, C.; SIRAKOULIS, G. C.; SOUDRIS, D. Network function virtualization based on fpgas: A framework for all-programmable network devices. **CoRR**, abs/1406.0309, 2014. Available from Internet: <http://arxiv.org/abs/1406.0309>.

KALIA, A. et al. Raising the bar for using gpus in software packet processing. In: **Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2015. (NSDI'15), p. 409–423. ISBN 978-1-931971-218. Available from Internet: <http://dl.acm.org/citation.cfm?id=2789770.2789799>.

KUMAR, S. et al. Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia. In: **Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems**. New York, NY, USA: ACM, 2007. (ANCS '07), p. 155–164. ISBN 978-1-59593-945-6. Available from Internet: <http://doi.acm.org/10.1145/1323548.1323574>.

Lan, T. et al. Fpga-based packets processing acceleration platform for vnf. In: **2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)**. [S.l.: s.n.], 2017. p. 314–317. ISSN 2327-0594.

Le, H.; Prasanna, V. K. A memory-efficient and modular approach for string matching on fpgas. In: **2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines**. [S.l.: s.n.], 2010. p. 193–200. ISSN null.

Li, X. et al. Dhl: Enabling flexible software network functions with fpga acceleration. In: **2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)**. [S.l.: s.n.], 2018. p. 1–11.

Lie, W.; Feng-yan, W. Dynamic partial reconfiguration in fpgas. In: **2009 Third International Symposium on Intelligent Information Technology Application**. [S.l.: s.n.], 2009. v. 2, p. 445–448.

Lin, S. et al. A design of the ethernet firewall based on fpga. In: **2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)**. [S.l.: s.n.], 2017. p. 1–5.

Mijumbi, R. et al. Network function virtualization: State-of-the-art and research challenges. **IEEE Communications Surveys Tutorials**, v. 18, n. 1, p. 236–262, Firstquarter 2016. ISSN 1553-877X.

Mishra, V.; Chen, Q.; Zervas, G. Reon: A protocol for reliable software-defined fpga partial reconfiguration over network. In: **2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)**. [S.l.: s.n.], 2016. p. 1–7.

Naik, P.; Shaw, D. K.; Vutukuru, M. Nfvperf: Online performance monitoring and bottleneck detection for nfv. In: **2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. [S.l.: s.n.], 2016. p. 154–160.

NFV White Paper. Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1. 2012.

Niemiec, G. S. et al. A survey on fpga support for the feasible execution of virtualized network functions. **IEEE Communications Surveys Tutorials**, p. 1–1, 2019. ISSN 2373-745X.

Paolino, M.; Pinneterre, S.; Raho, D. Fpga virtualization with accelerators overcommitment for network function virtualization. In: **2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)**. [S.l.: s.n.], 2017. p. 1–6.

Papastergiou, G. et al. De-ossifying the internet transport layer: A survey and future perspectives. **IEEE Communications Surveys Tutorials**, v. 19, n. 1, p. 619–639, Firstquarter 2017. ISSN 2373-745X.

Patel, P. Embedded systems design using fpga. In: **19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design (VLSID'06)**. [S.l.: s.n.], 2006. p. 1 pp.–.

Sang-Kil Park; Jin-Tae Oh; Jong-Soo Jang. High-speed attack mitigation engine by packet filtering and rate-limiting using fpga. In: **2006 8th International Conference Advanced Communication Technology**. [S.l.: s.n.], 2006. v. 1, p. 6 pp.–685.

Stiemerling, M.; Quittek, J. Middlebox configuration protocol design. In: **IEEE Workshop on IP Operations and Management**. [S.l.: s.n.], 2002. p. 222–226.

Sun, Y.; Kim, M. S. Dfa-based regular expression matching on compressed traffic. In: **2011 IEEE International Conference on Communications (ICC)**. [S.l.: s.n.], 2011. p. 1–5.

Taniguchi, A. et al. Impact of management data placement in nfv service coordinated across multiple datacenters and wans. In: **2015 11th International Conference on Network and Service Management (CNSM)**. [S.l.: s.n.], 2015. p. 406–409.

Tharaka, P. M. K. et al. Runtime rule-reconfigurable high throughput nips on fpga. In: **2017 International Conference on Field Programmable Technology (ICFPT)**. [S.l.: s.n.], 2017. p. 251–254.

Tipantuña, C.; Yanchapaxi, P. Network functions virtualization: An overview and open-source projects. In: **2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)**. [S.l.: s.n.], 2017. p. 1–6.

Tran, N. et al. High throughput parallel implementation of aho-corasick algorithm on a gpu. In: **2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum**. [S.l.: s.n.], 2013. p. 1807–1816. ISSN null.

Tuck, N. et al. Deterministic memory-efficient string matching algorithms for intrusion detection. In: **IEEE INFOCOM 2004**. [S.l.: s.n.], 2004. v. 4, p. 2628–2639 vol.4. ISSN 0743-166X.

Watanabe, Y. et al. Accelerating nfv application using cpu-fpga tightly coupled architecture. In: **2017 International Conference on Field Programmable Technology (ICFPT)**. [S.l.: s.n.], 2017. p. 136–143.

Woo-Sug Jung; Kwon, T. An independently partial pattern matching for content inspection at multi gigabit networks. In: **2010 The 12th International Conference on Advanced Communication Technology (ICACT)**. [S.l.: s.n.], 2010. v. 2, p. 1574–1579. ISSN 1738-9445.

Yang, B. et al. Practical multituple packet classification using dynamic discrete bit selection. **IEEE Transactions on Computers**, v. 63, n. 2, p. 424–434, Feb 2014. ISSN 2326-3814.

YI, X.; DUAN, J.; WU, C. Gpunfv: A gpu-accelerated nfv system. In: **Proceedings of the First Asia-Pacific Workshop on Networking**. New York, NY, USA: ACM, 2017. (APNet'17), p. 85–91. ISBN 978-1-4503-5244-4. Available from Internet: <http://doi.acm.org/10.1145/3106989.3106990>.

Yih, M. et al. Fpga versus gpu for speed-limit-sign recognition. In: **2018 21st International Conference on Intelligent Transportation Systems (ITSC)**. [S.l.: s.n.], 2018. p. 843–850. ISSN 2153-0009.

YIN, D. et al. Customizing virtual networks with partial fpga reconfiguration. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 41, n. 1, p. 125–132, jan. 2011. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/1925861.1925882>.

ZHANG, K. et al. G-net: Effective gpu sharing in nfv systems. In: USENIX ASSOCIATION. **15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)**. [S.l.], 2018. p. 187–200.

Zilberman, N. et al. Netfpga sume: Toward 100 gbps as research commodity. **IEEE Micro**, v. 34, n. 5, p. 32–41, Sep. 2014.