UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
CAMPUS SÃO CARLOS

Marcos Augusto Faglioni Junior

FILTRO STATELESS A 10G DEFINIDO EM BLUESPEC E IMPLEMENTADO EM FPGA

São Carlos
Junho de 2021

MARCOS AUGUSTO FAGLIONI JUNIOR

FILTRO STATELESS A 10G DEFINIDO EM BLUESPEC E IMPLEMENTADO EM FPGA

Trabalho de Conclusão de Curso submetido à Universidade Federal de São Carlos, como requisito necessário para obtenção do grau de Bacharel em Engenharia de Computação

São Carlos, Junho de 2021

UNIVERSIDADE FEDERAL DE SÃO CARLOS

MARCOS AUGUSTO FAGLIONI JUNIOR

Esta Monografia foi julgada adequada para a obtenção do título de Bacharel em Engenharia de Computação, sendo aprovada em sua forma final pela banca examinadora:

Orientador(a): Prof. Dr. Paulo Matias
Universidade Federal de São Carlos
UFSCar

Prof. Dr. Ricardo Menotti
Universidade Federal de São Carlos
UFSCar

Prof. Dr. Lourenço Alves Pereira Junior
Instituto Tecnológico de Aeronáutica
ITA

São Carlos, Junho de 2021

Dedico esse trabalho a meus pais, que me guiaram no mundo para chegar até aqui.
Dedico também aos meus amigos, sempre apoiando e motivando.

Agradecimentos

Os meus mais sinceros e profundos agradecimentos a todos os professores e funcionários da UFSCar, em especial aos docentes e técnicos do departamento de computação, na qual tive maior contato e que me inspiraram e aproximaram na computação e ao Professor Doutor Paulo Matias, e ao seu Orientando Fabiano Losilla por todo conhecimento para que esse trabalho se tornasse realidade. Também agradeço a minha família, motivando, inspirando e presente em todos os anos da minha vida, contribuindo para me tornar quem sou hoje como pessoa. Agradeço também aos meus amigos, que sempre me acompanharam, me apoiaram e estiveram presentes em todos os momentos. Ainda, gostaria de agradecer aos projetos de extensão que tive a oportunidade de contribuir e evoluir, em especial a equipe de robótica Red Dragons UFSCar, que apoiou muito meu crescimento dentro da universidade, tanto ao aplicar meus conhecimentos de computação, como de trabalhar em equipe.

Resumo

Um dos pilares da segurança cibernética é o princípio do isolamento, e uma das principais formas de aplicar esse princípio a redes com endereços globalmente roteáveis é a filtragem de pacotes. Com o aumento da banda consumida pelas aplicações de rede, da quantidade de dispositivos conectados, e com a incidência cada vez maior de ataques de movimentação lateral, torna-se importante construir filtros de alto desempenho customizáveis e de baixo custo. Este trabalho propõe o desenvolvimento de um filtro *stateless* em FPGA. Os filtros *stateless* são programas que dada uma entrada (sequência de bits) retornam uma decisão, com base na correspondências ou não com critérios previamente definidos. A FPGA é um dispositivo que pode ser programado no nível de *hardware* e com isso é possível conseguir um melhor desempenho quando comparado a processadores de propósito geral. Assim, este trabalho começa pelo estudo de alguns protocolos tradicionais de rede, como o IPv4 e IPv6, identificando campos de interesse. Em seguida, é proposto um método para filtrar esses pacotes, inicialmente utilizando a lógica de bloqueio por padrão, assim qualquer pacote fora do padrão do filtro será descartado, permitindo-se explicitamente apenas alguns IPs e MACs de origem e destino. A importância de se trabalhar com filtragem de pacotes é justificada principalmente pela proteção de sistemas computacionais evitando possíveis invasões, visto que os ataques, quando remotos, ocorrem pela rede, assim, quanto mais eficiente o filtro, melhor poderá ser sua detecção e bloqueio de pacotes maliciosos. Assim, essa proposta, mesmo que protótipo inicial de filtro, utiliza dispositivos reconfiguráveis com interfaces mais velozes de conexão e desempenho superior a um custo menor que os dispositivos comerciais existentes atualmente.

Palavras-chave: FPGA, Filtro Stateless, Bluespec, Ethernet, Ethernet 10G, Camada de enlace

Abstract

One of the pillars of cybersecurity is the principle of isolation, and one of the main ways to apply this principle to networks with globally routable addresses is packet filtering. With the increasing bandwidth consumed by network applications, amount of connected devices, and incidence of lateral movement attacks, it's important to build customizable and low-cost high-performance filters. This work proposes the development of a stateless filter in FPGA. Stateless filters are programs that given an input (sequence of bits) return a decision, based on whether or not it matches previously defined criteria. The FPGA is a device that can be programmed at the hardware level and with this it is possible to achieve better performance when compared to general purpose processors. Thus, this work begins with the study of some traditional network protocols, such as IPv4 and IPv6, identifying fields of interest. Next, the work proposes a method to filter these packets, initially using the default blocking logic, so any packet outside the filter pattern will be dropped, explicitly allowing only some source and destination IPs and MACs. The importance of working with packet filtering is justified mainly by the potential of protecting computer systems avoiding possible invasions, since attacks, when remote, occur over the network, so the more efficient the filter, the better it can detect and block malicious packets. Thus, this proposal, even as an initial prototype filter, uses reconfigurable devices with faster connection interfaces and superior performance at a lower cost than commercial devices currently available.

Keywords: FPGA, Stateless filter, Bluespec, Ethernet, Ethernet 10G, Data link layer

Lista de abreviaturas e siglas

FPGA - *Field-Programmable Gate Array*

UFSCar - Universidade Federal de São Carlos

OSI - *Open System Interconnection*

ISO - *International Organization for Standardization*

IMAP - *Internet Message Access Protocol*

HTTP - *Hypertext Transfer Protocol*

BSV - *Bluespec System Verilog*

HDL - *Hardware Description Language*

VHDL - *VHSIC Hardware Description Language*

VHSIC - *Very High Speed Integrated Circuits*

IPV4 - *Internet Protocol Version 4*

IPV6 - *Internet Protocol Version 6*

TCP - *Transmission Control Protocol*

ARP - *Address Resolution Protocol*

TOS - *Type of service*

TTL - *Time to live*

IEEE - *Institute of Electrical and Electronics Engineers*

Gbps - *Gigabits per second*

MAC - *Media Access Control*

ISOC - *Internet Society*

RFC - *Request for Comments*

FIFO - *First in First out*

CAM - *Content-addressable memory*

Lista de ilustrações

Sumário

1 Introdução

Com o avanço das comunicações, exigiu-se que novas formas de determinar e rotear os dados fossem desenvolvidas, a fim de dar suporte ao aumento do tráfego de dados na internet. Na verdade, até hoje existem muitos problemas relacionados com o alto fluxo de dados, requisições e pacotes, e por isso é necessário sempre buscar por pesquisas e novas tecnologias que possam suportar tal fluxo.

Este trabalho propõe a implementação de um filtro *stateless* utilizando linguagens de descrição de *hardware* e implementado em um FPGA. Essas tecnologias, se comparadas com chips de propósito geral possuem um desempenho melhor, com processamento mais rápido e confiável. Além do método de programação, essa placa dispõe de uma interface Ethernet de 10Gb, fornecendo comunicação mais veloz. Portanto será proposto um protótipo de um *firewall* utilizando de *hardware* com custo mais baixos que servidores ou *hardware* de propósito específico mas mantendo uma alta taxa de processamento de bits, entregando assim alto desempenho.

Assim, serão implementadas, em um primeiro momento, as formas de detecção dos protocolos usados atualmente, como os protocolos IPv4 e IPv6, e a partir desta identificação, criar-se-ão listas de bloqueio e permissão para analisar e tratar pacotes de dados que chegam na porta Ethernet. Os filtros propostos neste trabalho têm como objetivo demonstrar a aplicação e usabilidade, assim serão implementado utilizando como base os endereços de IP de destino e origem, endereços MAC e detecção de partes específicas do pacote.

1.1 Problemática

No mercado atual, existem soluções sofisticadas já disponíveis, porém essa tecnologia é geralmente patenteada e de alto custo, tanto para sua implementação como para sua manutenção. Pensando em ambientes com pessoal técnico qualificado, este trabalho pode apresentar uma solução para ambientes como os de universidades públicas, oferecendo um bom desempenho e sendo confiável, por um custo muito abaixo se comparado a compra de equipamentos novos, com *software* proprietários e com manutenção mensal.

1.2 Objetivo

Este trabalho propõe uma maneira de implementar um *firewall* utilizando uma FPGA. Assim é demonstrado o funcionamento de um filtro de rede que ao invés de ser

implementado em placas de propósito específico, foi implementado em uma placa FPGA, que tem a capacidade de atingir desempenho satisfatório, com um custo de *hardware* muito abaixo do mercado e com altas possibilidades de escalabilidade pela sua dinâmica de funcionamento. Ainda, foram implementados alguns protocolos de redes diferentes, para demonstrar sua aplicabilidade e versatilidade na utilização das tecnologias aqui propostas.

1.3 Organização

O restante deste trabalho está organizado da seguinte forma:

- Capítulo 2 - Apresenta os fundamentos teóricos das tecnologias utilizadas;
- Capítulo 3 - Apresenta uma revisão bibliográfica dos trabalhos em estado da arte atual;
- Capítulo 4 - Apresenta a metodologia e as implementações realizadas no trabalho
- Capítulo 5 - Apresenta as conclusões e resultados obtidos

2 Fundamentos Teóricos

Nesta seção serão apresentados os fundamentos necessários para a compreensão de alguns dos termos e tecnologias utilizadas neste trabalho. Será apresentado brevemente o que é e como ocorre a comunicação por meio da internet, a definição de um dispositivo FPGA, a tecnologia disponível no *hardware* de internet 10G, e ainda as linguagens e *software* utilizadas neste trabalho.

2.1 Protocolos

2.1.1 Data Frame

Neste trabalho, definimos como *data frame* um quadro Ethernet que contenha um datagrama IP. O *data frame* contém, portanto, informações de endereço de MAC e IP de origem e destino, a *payload* - que é a sequência de bits que determina a informação, geralmente existe o *checksum* - que é uma forma de validar os dados enviados, o comprimento do campo de dados, entre outros campos que variam conforme o protocolo. Em seguida será detalhado cada um dos protocolos mais usados atualmente segundo [3].

2.1.1.1 IPv4

O protocolo IPv4 é um dos protocolos mais importantes e utilizados atualmente. Este protocolo conta com um endereçamento de 32 bits, sendo possível identificar assim, 2^{32} ou mais de 4 bilhões de dispositivos.

Esse protocolo está definido pela RFC 791 [4], que especifica seu funcionamento e seu datagrama, possibilitando a comunicação de forma padronizada entre dispositivos que possuem tal protocolo. A Figura 1 ilustra a formatação de um datagrama IPv4.

Como mostrado na Figura 1, esse protocolo possui os seguintes campos e respectivas funções:

- Versão - 4 bits: Para este protocolo temos sempre o número 0100 em binário, que representa 4;
- HLEN - 4 bits: Este campo representa a quantidade de palavras de 32 bits que o cabeçalho contém, podendo assumir um número entre 5 e 15;
- Tipo de serviço (TOS) - 8 bits: Este campo é utilizado para informar qual o tipo de serviço;

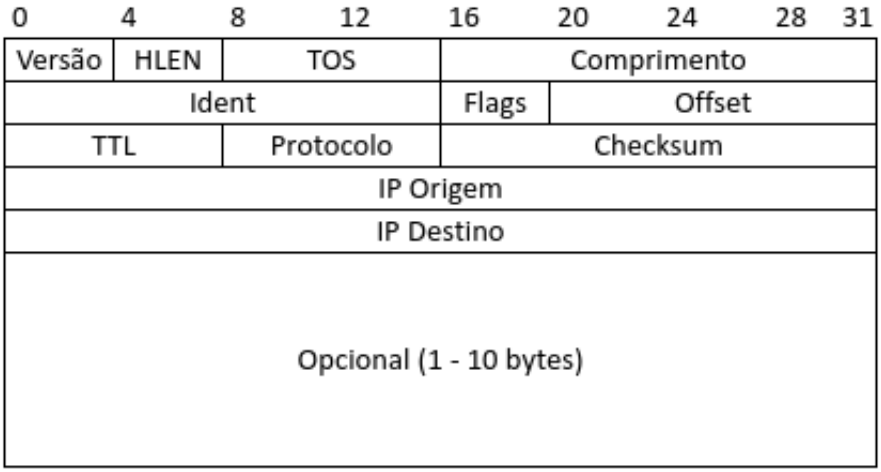

Figura 1 – Datagrama IPv4

- Comprimento Total - 16 bits: Esta informação representa a quantidade de bytes total do pacote. Este campo possui um valor mínimo de 20 bytes (tamanho ocupado pelo cabeçalho) e máximo de 65.535 bytes (limitado por 2^{16} que é máximo estipulado pelo protocolo), no caso da informação a ser enviada ser maior que 2^16 o pacote deve ser fragmentado em mais de um *data frame*;
- Identificação - 16 bits:
- Flags - 3 bits: Possui três *flags* cada uma de um bit. O bit 0 é sempre 0 e representa um bit reservado; o bit 1 representa se é ou não um pacote fragmentado; bit 2 representa se este pacote faz parte de uma fragmentação (no caso de ser um pacote fragmentado, todos os pacotes fragmentados com exceção do último terá o bit 2 setado com 1);
- *Fragment Offset* - 13 bits: Informa o *offset* do fragmento com relação ao início do pacote desfragmentado (suporta máximo de $(2^{13}) - 1) * 8 = 65\,528$ bytes;
- Tempo de vida (TTL) - 8 bits: Tempo de vida estipula por quantos nós o pacote pode passar antes de chegar no seu destino. A cada nó o tempo de vida é decrementado, e ao chegar a 0, o pacote é descartado. Isso evita que o pacote entre em um *loop* eterno entre nós da rede, o que poderia causar colapso por congestionamento;
- Protocolo - 8 bits: Indica o protocolo utilizado;
- *Checksum* do cabeçalho - 16 bits: Contém um valor que é calculado pelo remetente, e que também é calculado pelo destinatário. O cálculo utiliza os bits do cabeçalho e é útil para validar se o conteúdo recebido foi o mesmo que o enviado;
- Endereço de Origem Origem - 32 bits: IP do endereço de origem;

- Endereço de Destino - 32 bits: IP do endereço de destino;
- Opção - até 10 bytes: Representa campos adicionais que o administrador da rede pode utilizar. Este campo é opcional e está presente em pacotes com HLEN maior que 5.

2.1.1.2 IPv6

Um dos problemas detectados no protocolo IPv4 é com relação à quantidade de endereços, ou seja, de dispositivos que podem ser utilizados, que é exatamente de $2^{32} = 4\,294\,967\,296$ endereços. Assim, com o crescimento exponencial de dispositivos conectados, esse número tornou-se pequeno, e desenvolveu-se portanto o protocolo IPv6, que dispõe de 2^{128} endereços, ou cerca de $3{,}4 \times 10^{38}$.

O protocolo IPv6 está definido na norma RFC 2460 [5], que especifica o seu funcionamento, afim de se obter padronização na comunicação entre os dispositivos. A Figura 2 exibe o datagrama e discute quais são os campos e suas finalidades.

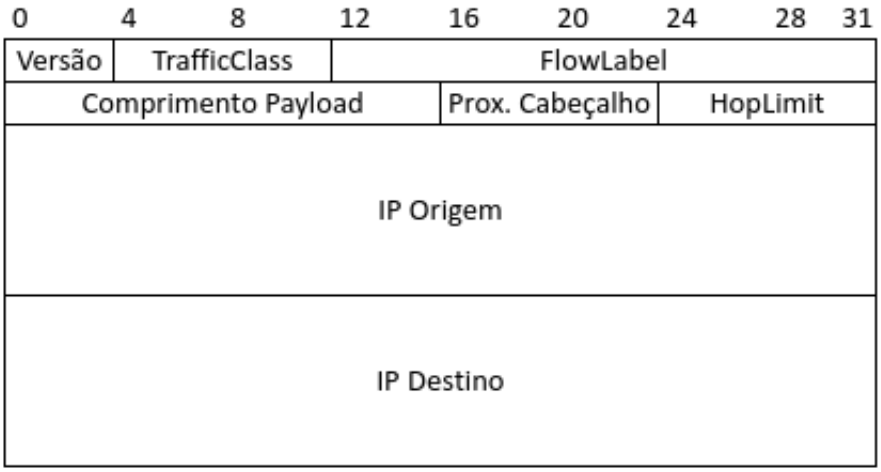

Figura 2 – Datagrama IPv6

Assim como no protocolo IPv4, o protocolo IPv6 possui versão (que neste caso é sempre 0110 em binário ou 6), *Traffic class* similar ao tipo de serviço (TOS), *Hop Limit* equivalente ao tempo de vida, além dos seguintes campos:

- Comprimento da *Payload* - 16 bits: Contém apenas o comprimento da *payload* do pacote;
- *Next header* - 8 bits: Se houver outro encapsulamento na informação presente na *payload*, esse campo indica a presença e qual o tipo do protocolo existente, geralmente protocolos de camadas superiores como TCP e UDP;
- Endereço de Origem Origem - 128 bits: IP do endereço de origem;

- Endereço de Destino - 128 bits: IP do endereço de destino.

2.2 Firewall

Um *firewall* [6] é um método de adicionar mais segurança a uma rede de internet, este é implementado entre a rede interna e a rede externa, também denominada rede confiável e não confiável. A definição de rede confiável é aquela que está dentro da organização, assim, interliga os nós, conexões de servidores, *data centers*, sistemas *host*, roteadores, túneis de criptografia, entre outros, enquanto que a rede não confiável é a que trafega todos os dados de internet para fora da organização. Com isso um *firewall* é responsável por analisar e impedir que uma ameaça externa penetre na rede confiável de uma organização.

Segundo Hunt [1] a operação de um *firewall* é definida pela sua política. A política define os serviços e os tipos de acessos permitidos entre os domínios confiáveis e não confiáveis. Assim a definição de uma política de segurança exige uma explicação clara de perímetro de segurança, uma vez que diferentes arquiteturas de um *firewall* definem diferentes níveis de garantia contra ataque.

Com o perímetro de segurança definido, pode-se determinar a zona de risco, que cobre todas as interfaces, redes e servidores conectados diretamente à internet. Portanto pode-se definir que o objetivo da política de um *firewall* é minimizar a zona de risco, reduzindo ao máximo a possibilidade de um ataque proveniente da rede externa. Assim o *firewall* torna-se a ponte entre a zona de risco e a rede confiável.

Com a prevalência cada vez maior de ataques de movimentação lateral, aconselha-se que as redes internas sejam divididas em múltiplos segmentos, de forma que um segmento não confie no outro. Com isso, torna-se necessária a existência de *firewall* entre todos os segmentos da rede que possam se comunicar entre si.

2.2.1 Filtro *Stateless*

Um firewall do tipo *stateless* [3] é aquele que avalia apenas o pacote recebido, não avaliando as conexões envolvidas. Assim, com este tipo de filtro, é possível avaliar os endereços IP e MAC de origem e destino, e algumas informações presentes no cabeçalho do pacote, mas se limitando a essas informações.

Em contraponto aos filtros *stateless*, existem os filtros *stateful*. Considerado uma evolução do *stateless*, o filtro *stateful* permite a análise da conexão. Por manter e conseguir armazenar em tabelas, dados da conexão, essa forma de filtragem permite a aplicação de um conjunto menor de regras resultando em um desempenho maior e conseguem avaliar um conjunto maior de validação, visto que possuem a validação da conexão além do

pacote.

Neste trabalho será utilizado o filtro *stateless*. Apesar deste filtro possuir desempenho menor, ele é suficiente para demonstrar a aplicação desejada e sua implementação é menos complexa que um filtro *stateful* em vista da implementação de tabelas para a utilização do estado da conexão.

2.2.2 Arquitetura de um *firewall*

Existem muitas maneiras de classificar um *firewall*, mas uma muito comum é levando em consideração as divisões do modelo de 7 camadas OSI [1], como ilustrado na Figura 3.

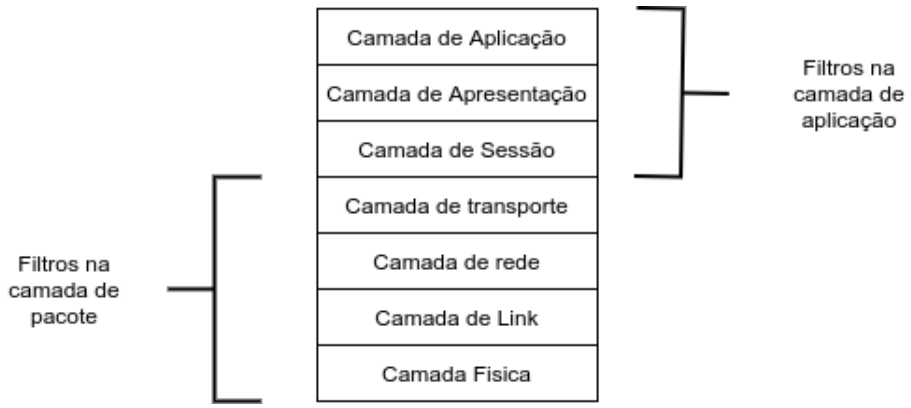

Figura 3 – Arquitetura de *firewall* com relação ao modelo OSI (adaptado [1])

Nessa classificação, um *firewall* pode existir em dois níveis diferentes:

- Nível de pacote - Os *firewalls* aplicados nesse nível operam na camada de rede (IP) e na camada de transporte (TCP). São comumente chamados de roteadores de triagem ou de filtro de pacotes e bloqueiam a transmissão de determinadas classes de tráfego
- Nível de aplicação - Usualmente são aplicados nas camadas de sessão, apresentação e aplicação. Geralmente são implementados em *hosts* dedicados e oferecem serviços de retransmissão.

Neste trabalho será utilizado o nível de pacote, uma vez que o filtro proposto atuará principalmente liberando pacotes que possuam determinadas classes de tráfego. Apesar de filtros no nível de aplicação terem o potencial de mitigar uma maior variedade de ataques, sua implementação em *hardware* é um demasiadamente complexa para o escopo de um Trabalho de Graduação, além de esbarrar em dificuldades tais como a alta prevalência de tráfego cifrado no nível de aplicação.

2.2.2.1 Filtro de Pacote

Conforme os pacotes passam pelo roteador, são filtrados de acordo com um conjunto de regras [1]. Para a criação de regras, geralmente são utilizadas regras baseadas nos endereços IPs de origem e destino e no número da porta de origem e de destino.

A filtragem, então, pode ser utilizada para bloquear conexões de *hosts* específicos, bem como bloquear requisições de portas específicas. Com esse modelo de bloqueio, e objetivando-se essas camadas, pretende-se propor um filtro de perímetro. Criando assim uma rede segura, separada do chamado zona desconhecida (internet geral) e aplicando este *firewall* nas interfaces de comunicação, fisicamente entre a chegada de internet e os servidores e computadores locais.

2.2.3 Camadas de atuação

Por atuar no nível de pacote, o filtro aqui proposto será implementado nas camadas de enlace e rede, que é onde ocorre a identificação de alguns aspectos dos protocolos a serem analisados. Essas camadas fazem parte de um conjunto maior ([7]), definido pelas normas ISO, afim de padronizar os protocolos de comunicação entre os sistemas de rede. Esse conjunto chamado de *OSI* foi criado em 1971 e formalizado em 1983, embora não defina um protocolo ou serviços exatos, ele determina o escopo de cada uma das 7 camadas. A seguir cada uma das camadas que permitem a filtragem por pacote será conceituada.

A primeira camada que trata os dados é a camada de enlace ou de dados, que é responsável pela montagem dos bits recebidos na camada anterior em *data frames* e correção de pequenos erros que podem ter acontecido. Em alguns protocolos de camada de enlace também existe a repensabilidade de alertar o emissor se a informação chegou corretamente e administrar o controle de acesso a canais compartilhados.

Com o *data frame* pronto, a camada de rede lida com o endereçamento de larga escala (possivelmente global) do pacote. Roteadores usam essas informações de endereçamento para direcionar o pacote de dados do emissor para o receptor. Utilizando-se de uma tabela de valores, que relaciona faixas de endereços de destino com interfaces físicas, os roteadores fazem o encaminhamento dos pacotes até que este chegue no receptor.

Por se tratarem de camadas que lidam diretamente com o *data frame*, torna-se possível capturar e analisar sequências de bits de interesse, e com isso manipular os pacotes para liberar somente os que obtiverem permissão para passar.

2.3 FPGA

Um FPGA é um circuito integrado projetado para ser configurado pelo seu usuário final, diferindo assim de chips que são construídos com um único propósito. Assim sendo,

um FPGA é constituída basicamente por blocos lógicos, interfaces de entrada e saída e matrizes de interconexões.

Os blocos lógicos são organizados na forma de uma matriz bidirecional. Todos os blocos possuem a mesma construção e são formados por arranjos de flip-flops (elementos armazenadores de memória) e tabelas de busca (*lookup tables*) reconfiguráveis. Entre estas, existem matrizes de conexão, que interligam os blocos lógicos entre si e as interfaces de entrada e saída.

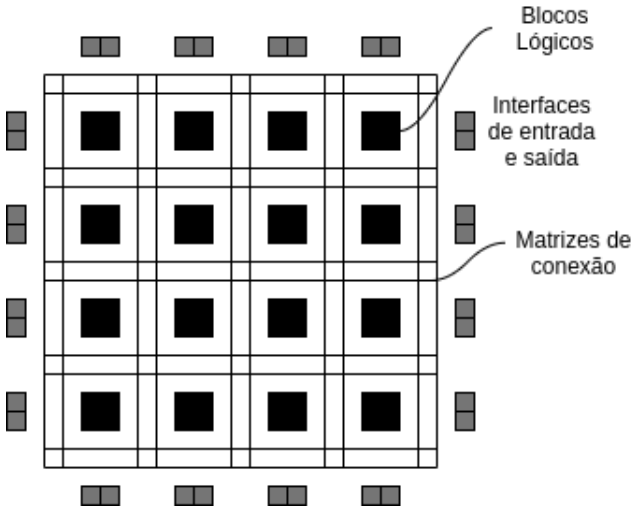

Figura 4 – Diagrama esquemático básico de uma FPGA (adaptado [2])

A programação da FPGA ocorre no nível de *hardware* - um dos motivos que consegue atingir desempenho próximo a *hardware* dedicados - onde a matriz de conexão permite a ligação entre os vários blocos lógicos da estrutura como pode ser observado na Figura 5

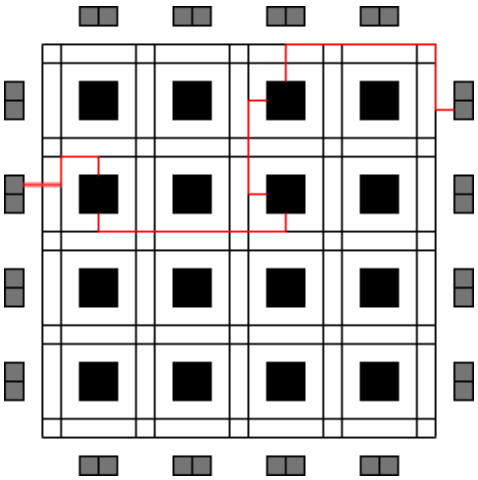

Figura 5 – Estrutura matricial entre uma entrada e saída (adaptado [2])

Assim, a programação modifica fisicamente as ligações dos blocos lógicos e das matrizes de conexão. Este trabalho será modelado com a linguagem *Bluespec*, que será

descrita na Seção 2.3.1, utilizando também o visualizador de simulações GTKWave [8]. Para a implementação será utilizado um dispositivo Xilinx Kintex 7.

2.3.1 Bluespec

Bluespec SytemVerilog (BSV) é uma linguagem para projeto de *hardware* [9]. Bluespec é um compilador, simulador e possui ferramentas para a geração de códigos em Verilog padrão, oferecendo assim compatibilidade com as ferramentas e dispositivos já disponíveis no mercado. Esse recurso estava sendo desenvolvido a quase 20 anos pela empresa Bluespec Inc, que recentemente liberou sua licença para utilização em código aberto.

O BSV amplia o conjunto de projeto de *SystemVerilog*, incluindo tipos de *SystemVerilog*, módulos, instanciação de módulos, interfaces, instanciação de interfaces, parametrização e elaboração estática. Com esses recursos, o BSV possui a grande vantagem da utilização pois facilita a implementação de módulos em *hardware*, de uma maneira mais moderna e simplificada se comparada a linguagens padrões de descrição de *hardware* como a linguagen Verilog, exportando como resultado de sua compilação um arquivo na linguagem Verilog.

Uma implementação tradicional de arquivos Bluespec inclui a importação de bibliotecas, de interfaces, declarações de registradores e os métodos do módulo. A criação de um arquivo pode ser entendida como um módulo do sistema, assim, a linguagem permite que se importe um módulo no outro para a utilização de ferramentas disponíveis no módulo terceiro. As interfaces devem ser entendidas como as conexões disponíveis em cada um dos módulos, permitindo assim entrada e saída de informações. As declarações permitem a alocação de registradores ou outros elementos de estado para armazenar as informações utilizadas, as regras realizam os cálculos e processamentos que o módulo executa e os métodos permitem ao mundo externo acessar informações ou controlar o módulo. Assim é a estrutura básica de um arquivo, e esses módulos podem ser compilados para então gerar o arquivo correspondente em Verilog para cada módulo. Ainda executar os arquivos gerados em um simulador, gerando um arquivo de extensão .vcd que pode ser visualizado usando ferramentas como o GTKWave [8].

2.3.2 *Ethernet* 10G

A comunicação 10Gbps representa um grande avanço na velocidade de envio e recepção dos dados. Com uma velocidade mil vezes mais rápida que a *Ethernet* original, a 10 Gbps foi pensada para que servidores, *data centers* e centrais pudessem se conectar a roteadores, *switches* e servidores, além de permitir conexões de alta velocidade a longas distâncias, podendo chegar a 40 km [7].

A fim de padronizar a *Ethernet*, o IEEE definiu alguns critérios e normas para o padrão 10 gigabits. Ainda, definiram-se diretrizes para a compatibilidade com as outras versões existentes, podendo ser alternada automaticamente entre nós que possuam velocidade mais alta ou mais baixa, dependendo da ligação física disponível.

A implementação final deste trabalho será realizada em um dispositivo que permite a conexão com essa velocidade de comunicação, servindo assim como exemplo potencial de um equipamento com capacidade de alto processamento, que acompanha demandas atuais de velocidade de conexão.

3 Trabalhos Relacionados

Uma breve análise sobre outros estudos indica que a busca por soluções de firewall de baixo custo tem-se amplificado. No estudo de Ajami e Dinh [10], avalia-se a implementação de um *firewall* em FPGA, assim como proposto neste trabalho, comparando-o com *iptables* do Linux, exibindo resultados promissores de desempenho e velocidade. Ainda, é interessante ressaltar que nesta comparação, os resultados de performance foram independentes do tamanho do pacote ou da quantidade de regras utilizadas, demonstrando assim grande vantagem em se trabalhar com esse tipo de implementação.

Em Keni e Mande [11], também realiza-se a comparação entre uma implementação em *hardware* e uma em *software*. Esse trabalho propõe um filtro para pacotes de 512 bits cuja aprovação ou rejeição é validada pelo endereço IP. Nesse método, é utilizada uma lista de permissão, na qual todos os endereços IPs que permitem a liberação do pacote seguem para o processamento e os pacotes que não atendem esse critério são descartados. Nesse método também foi observado que o tempo gasto no processamento foi mais rápido se comparado ao *software*.

Em Cho e Mangione-Smith [12], trabalha-se com a implementação de *firewall* em FPGA. Além de concluir o que o sistema é mais rápido, o trabalho apresenta um comparativo entre uma placa *Xilinx* e um processador *Pentium III* de 1GHz. Nesse comparativo, o ganho de desempenho salta de uma saída de 50 Mbps para mais de 3 Gbps de tráfego. Outro ponto interessante é a comparação entre diversos modelos de FPGA mostrando resultados possíveis entre 0.4 Mbps e 6.4 Gbps.

Em Liu e Gouda [13], realizam-se implementações sem utilizar o FPGA, mas há observações interessantes sobre o tema. Uma dos grandes diferenciais do trabalho é que os autores utilizam da linguagem Java e além da implementação, eles propõe um *software* de validação de *firewalls* automatizado que, a partir de dois métodos diferentes, avalia tanto as regras do sistemas como seu desempenho. Assim, não será seguida a ideia de implementação, mas a ideia de automatização de testes de desempenho e regras pode ser aproveitada em trabalhos futuros.

Em Lin et al. [14], propõe-se algo muito similar ao presente trabalho, com a filtragem de pacotes através dos dados de IP e MAC em uma FPGA. Os dados obtidos indicam que o o sistema proposto possui uma taxa de saída de dados em 950 Mbps e um atraso com relação à entrada de 61 μs. Segundo os autores, os resultados foram testados por três diferentes métodos, que comprovaram os valores de taxa de saída de dados e o atraso entre a entrada e saída.

No trabalho de Wang et al. [15] é proposto um filtro implementado em Bluespec.

Neste trabalho é desenvolvido um filtro mais complexo chamado P4 e utilizado o compilador do Bluespec para gerar os códigos em Verilog para FPGA. A proposta deste trabalho é interessante mas muito aprofundada e não portável, assim será tratado uma proposta também em Bluespec, que seja mais simples, com o objetivo de entender os processos e mecanismos de filtragem, em conjunto com a elaboração de um filtro mais portável e adaptável, conforme a necessidade.

4 Desenvolvimento do trabalho

Neste capítulo, aborda-se a metodologia e implementação do *firewall* proposto neste trabalho.

4.1 Metodologia

Para a execução deste trabalho, foi utilizada a lógica de *firewall* de lista de permissão. Assim, foi implementada uma lista dos possíveis IPs e portas permitidos para aceitar o pacote.

A implementação foi realizada em BSV (Seção 2.3.1). Foram desenvolvidos alguns módulos principais, detalhados na Seção 4.2, para receber o pacote, em seguida detectar a qual protocolo de comunicação este pertence, e por fim, utilizar o módulo de filtro para realizar a filtragem dos dados.

Durante o desenvolvimento do trabalho, optou-se por realizar toda a implementação, desde a detecção dos bits da interface de rede até a filtragem do pacote já montado. Assim, a utilização de bibliotecas disponíveis no Bluespec restringiu-se à aquelas que tratam de dados como FIFO e algumas matemáticas, mas nenhuma outra biblioteca responsável pela captação de dados ou tratamento de ethernet em si. Assim, todos os métodos foram desenvolvidos pelo autor em conjunto com o orientador e seu doutorando.

O módulo de detecção de pacotes tem por objetivo de encontrar onde começa e onde termina cada quadro que contenha um pacote IP, além de verificar à qual versão (4 ou 6) do protocolo IP esse pacote pertence.

O módulo de filtro, por sua vez, analisa os endereços IPs e MAC que devem ou não ser liberados. Uma vez que o cabeçalho está analisado e os dados disponíveis para verificação, este módulo consulta todos os endereços de IPs e portas de origem e destino que permitem a liberação do pacote. Havendo compatibilidade entre pacote e a lista de permissão, a informação é então liberada e direcionada para a interface de saída da placa.

O padrão utilizado neste trabalho será de lista de permissão. Entretanto, a implementação ou a utilização de um modelo como lista de bloqueio ou interfaces que utilizam outros métodos como comparação de endereços MAC, ou alguma *flag* disponível no protocolo também podem ser utilizadas com implementações parecidas e tal tarefa também pode ser estendida em trabalhos futuros.

Ao compilar os arquivos BSV, são gerados os arquivos em Verilog, que podem ser carregados na placa através do *software* proprietário. Além de carregar o código na placa,

pode-se gerar um outro arquivo, a partir do Verilog que simula o estado dos registradores e conexões da FPGA durante o tempo e com o auxílio de um código que simula o envio de dados, é possível então verificar o estado dos registradores antes de testar o código na placa. Para isso será utilizado o GTKWave.

4.2 Implementação

De forma a guiar e auxiliar as implementações, foi construído um esquema como o da Figura 6. Essa figura descreve um diagrama de estados simplificado, iniciando pela recepção do pacote pela interface de rede, que em seguida é armazenado e então processado a fim de detectar qual protocolo está sendo utilizado para a comunicação. No caso de não ser um IP, o pacote é descartado, senão, segue para o filtro e, sendo aprovado, é enviado para a interface de saída para que possa ser entregue ao destinatário.

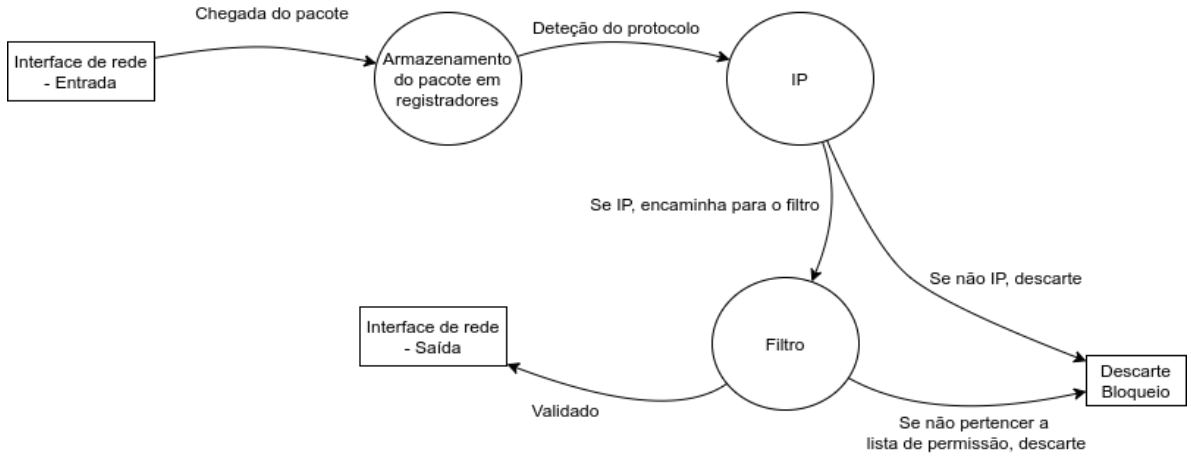

Figura 6 – Modelo esquemático da máquina de estados utilizado como base para a implementação

4.2.1 Simulação dos pacotes

Para que fosse possível simular no computador, foi criada uma rotina específica para enviar à interface de rede alguns pacotes aleatórios. A estrutura do arquivo contém a criação de instâncias que indica aos módulos se o pacote é valido (*valid*), se é a última parte do pacote (*last*), um contador (cnt) para auxiliar na determinação de qual sequência de bits deve ser enviada (considerando envio de sequências na ordem correta), e por fim um registrador de 64 bits que contém a sequência de bits.

Em seguida, a depender do contador, determina-se qual sequência é enviada. Além de atribuir o valor da sequência ao registrador de dados, os registrados *last* e *valid* também são gravados. Uma vez definidas as regras para controlar os registradores, uma função efetivamente envia os dados, utilizando uma das interfaces de saída do módulo.

4.2.2 IP

No módulo IP ocorre a detecção dos campos de dados pertencentes ao protocolo IP, de versão 4 ou 6. Assim, neste módulo é criado um registrador para cada uma das opções (Seção 2.1.1) de versão do protocolo. Isso inclui o próprio IP de destino e origem e as portas, que será utilizado na etapa de filtragem para ambas as versões do protocolo.

Um método é executado a partir de um chamada da interface para receber os dados e verifica a compatibilidade de um protocolo do tipo IP. Assim que é detectado, uma *flag* é acionada, e então o módulo recebe os dados que estão chegando na interface, e os analisa, atribuindo a cada registrador o seu valor, relacionado a cada um dos campos.

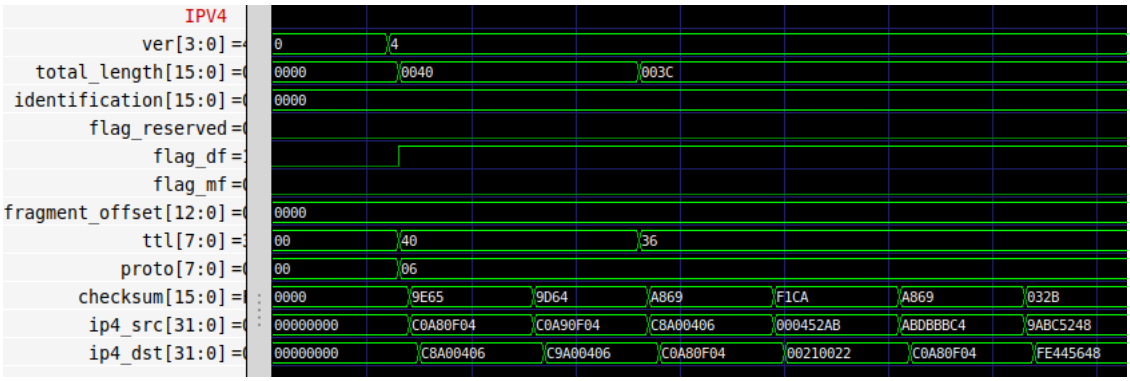

Figura 7 – Saída da simulação para os dados de IPv4

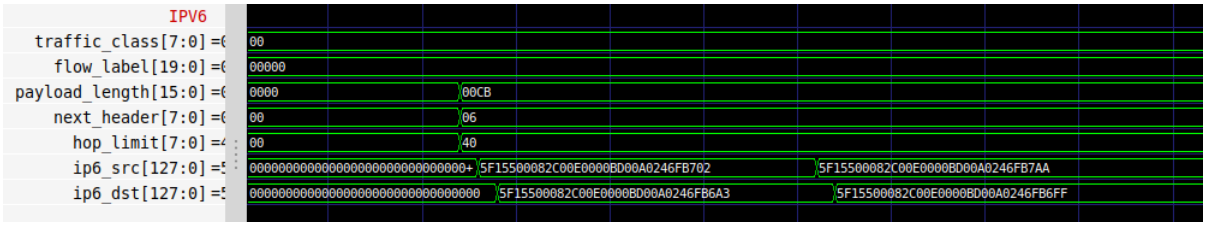

Figura 8 – Saída da simulação para os dados de IPv6

Observado as Figuras 7 e 8, é possível notar os pacotes montados na simulação, e assim temos os seguintes endereços IP de destino e origem obtidos na simulação:

- IPv4
  - fonte: C0A80f04 e destino: C8A00406
  - fonte: C0A90f04 e destino: C9A00406
  - fonte: C8A00406 e destino: C0A80f04
  - fonte: 000452AB e destino: 00210022
  - fonte: ABDBBBC4 e destino: FE445648

- IPv6
  - fonte: 5F15500082C00E0000BD00A0246FB702
    e destino: 5F15500082C00E0000BD00A0246FB6A3
  - fonte: 5F15500082C00E0000BD00A0246FB7AA
    e destino: 5F15500082C00E0000BD00A0246FB6FF

Portanto, para os testes de filtragem serão utilizados esses pacotes.

4.2.3 Filtro

O módulo do filtro é responsável por efetivamente realizar o bloqueio do pacote. Assim, conforme os pacotes de dados chegam na interface, são armazenados em uma estrutura de dados no formato de fila, aguardando sua validação para liberação ou descarte do pacote. Como esse pacote já contém todo o *data frame* analisado, então os dados de análise (IP de origem e destino) estão disponíveis, o módulo de filtro realiza a comparação destes dados com os valores padrão para liberação disponíveis no sistema, e em caso de permissão, o pacote é liberado para a interface de saída, ou então é descartado.

Para a implementação deste trabalho para amplos valores de IP e MAC pode ser recomendado a utilização de memórias associadas (CAM) para obtenção de um desempenho superior na consulta da lista de permissão. Neste trabalho não utilizou desta implementação pois a proposta não verificava um número suficientemente alto de endereços, que justificasse a implementação deste método, ainda que para uma implementação em ambiente real, este método deve ser considerado por apresentar alto desempenho em consultas.

A filtragem por análise do *data frame* permite não somente a utilização dos endereços de origem e destino, mas sim de todos os valores disponíveis. Portanto pode-se analisar pelo valor de alguma das *flags*, ou somente um trecho do endereço, como pacotes que são direcionados para IP com finais específicos, ou ainda limitar pelo comprimento do pacote, dependendo do filtro que fizer sentido para a aplicação.

4.3 Resultados

Como está sendo utilizada uma lógica de lista de permissão, todos os endereços que pertencem a lista de permissão devem ser entregues, e todos os demais, descartados. Portanto, analisando inicialmente para os protocolos IPv4 e para o teste inicial, foram bloqueados todos os pacotes com IPv4 menor que hC0A80F04, e com a simulação é possível notar que somente as informações dos pacotes permitidos foram entregues na interface tx_data, que é a interface responsável por levar o dado para a saída.

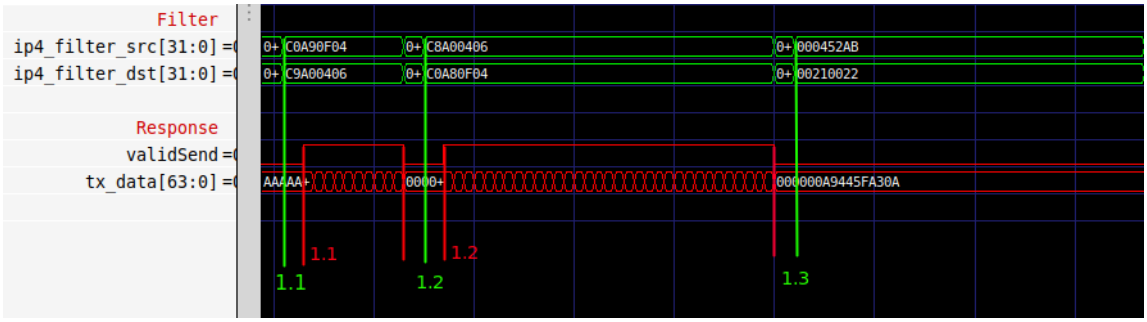

Figura 9 – Saída da simulação para o primeiro exemplo de filtro IPv4 (a)

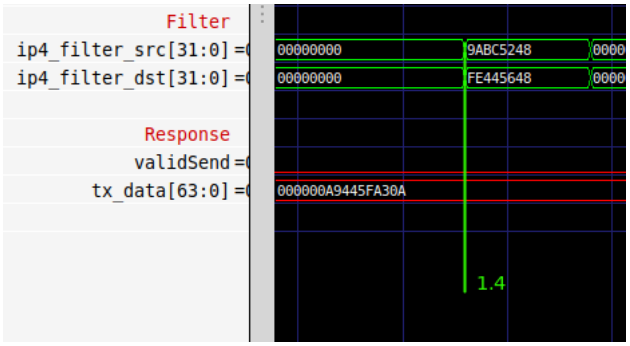

Figura 10 – Saída da simulação para o primeiro exemplo de filtro IPv4 (b)

Observando as Figuras 9 e 10 e focando nos traçados verdes (1.1 a 1.4 em verde), é possível verificar os pacotes chegando no módulo de filtro (indicado pelas interfaces "ip4_fiter_src" e "ip4_fiter_dst" para IPv4 fonte e destino respectivamente), e é possível observar também sobre os traçados vermelhos, as informações sendo repassadas para a conexão de saída (visível pela interface "ValidSend") somente para os pacotes cujo IP de origem é hC0A90F04 e hC8A00406 (números 1.1 e 1.2 em vermelho), pois são maiores que hC0A80F04, enquanto que para os demais pacotes o sinal foi bloqueado.

Como prova de conceito, pode-se filtrar pacotes apenas abaixo desta faixa de IPs. As Figuras 11 e 12 demonstram o efeito dessa mudança no filtro.

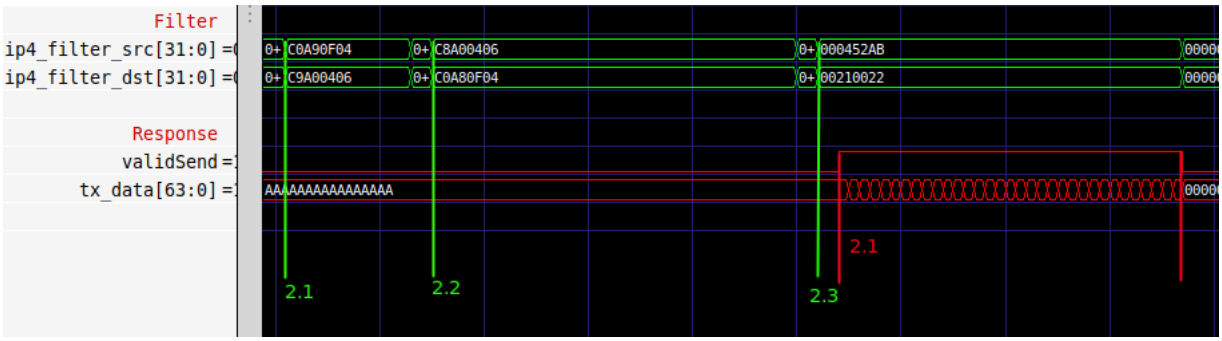

Figura 11 – Saída da simulação para o segundo exemplo de filtro IPv4 (a)

Neste caso, os sinais em verde 2.1 e 2.2 não possuem mais permissão para serem

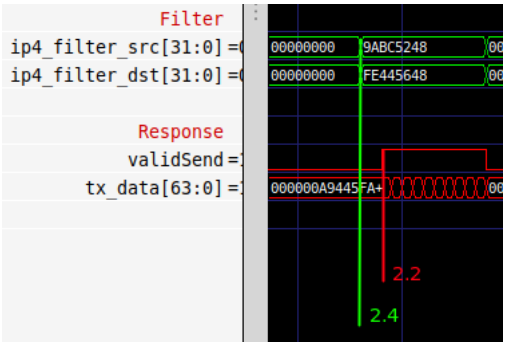

Figura 12 – Saída da simulação para o segundo exemplo de filtro IPv4 (b)

entregues, enquanto que os sinais 2.3 e 2.4 ativam a interface "ValidSend" liberando assim o envio do pacote, indicados por 2.1 e 2.2 vermelho.

Para o IPv6 pode-se utilizar também a mesma politica de filtros. Para uma validação inicial, permitiu-se a passagem de pacotes que estejam comunicando com um IP finalizado com final hA3.

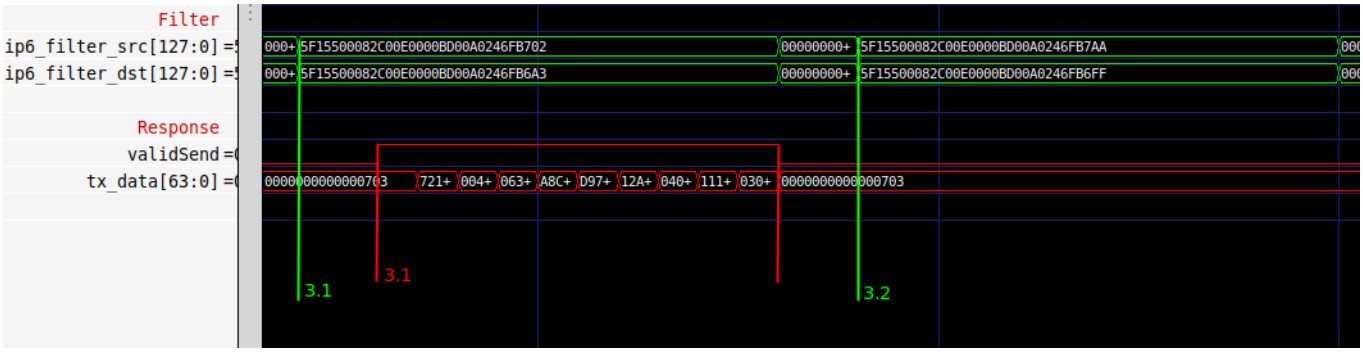

Figura 13 – Saída da simulação para o exemplo de filtro IPv6

Na Figura 13 é possível visualizar dois pacotes IPv6, sendo que um deles foi liberado e o outro não. O pacote 3.1 (verde) possui o endereçamento para um IP que finaliza com hA3, portanto dentro dos valores permitidos, enquanto que o segundo pacote não possui um destinatário com esse valor, e consequentemente foi descartado.

Todas as implementações realizadas neste trabalho podem ser acessadas pelo GITHUB no link https://github.com/MarcosFagli/TCC-Filtro_Stateless. Neste link é disponibilizado um README com as informações de compilação e simulação para obter os resultados aqui apresentados.

5 Conclusão

Filtros são importantes aliados na busca de soluções para prevenir ciberataques, uma vez que eles contribuem para o isolamento e segmentação das redes. Este trabalho mostra o funcionamento de um filtro simples que pode ser implementado no *firewall* e que detecta e impede a entrada de pacotes indesejados, utilizando diferentes meios de validação.

Durante a investigação do estado da arte, foi possível comprovar a implementação de mecanismos de segurança em *hardware* é uma tendência, pois ela pode ser muito eficiente, visto que as FPGAs possuem um ótimo desempenho, e isso pode significar um custo menor a depender da implementação e utilização.

Neste trabalho, os resultados obtidos podem ser medidos pelos pulsos de *clock* da simulação. O tempo entre um dado ser processado, considerando o tempo entre chegar na interface e iniciar a saída deste dado, é de 7 ciclos de *clock*. Considerando agora o tempo entre o primeiro fragmento a chegar e a saída completa do pacote da interface, mede-se um total de 17 ciclos, este dependente diretamente do tamanho do pacote, e como os pacotes analisados continham tamanhos similares, é possível defini-lo como o tempo médio para os pacotes enviados. Assim, considerando-se um *clock* de 450 MHz [16] é possível prever um tempo de processamento do pacote em torno de 1.7 μs. Como não foi possível concluir em tempo hábil a implementação na placa, os dados de projeção em tempo são teóricos, mas para trabalhos futuros é possível gerar o teste e validar as informações de tempo de execução para pacotes reais.

Referências

1 HUNT, R. Internet/intranet firewall securitypolicy, architecture and transaction services. *Computer Communications*, v. 21, n. 13, p. 1107–1123, 1998. ISSN 0140-3664. Disponível em: <https://www.sciencedirect.com/science/article/pii/S014036649800173X>. Citado 4 vezes nas páginas 15, 22, 23 e 24.

2 7 Series FPGAs Configuration User Guide. [S.l.]. Citado 2 vezes nas páginas 15 e 25.

3 PETERSON, L. L.; DAVIE, B. S. *Computer Networks - A Systems Approach*. 5th. ed. [S.l.]: Pearson, 2011. I. Citado 2 vezes nas páginas 19 e 22.

4 POSTEL, J. *Internet Protocol*. RFC Editor, 1981. RFC 791. (Request for Comments, 791). Disponível em: <https://rfc-editor.org/rfc/rfc791.txt>. Citado na página 19.

5 HINDEN, B.; DEERING, D. S. E. *Internet Protocol, Version 6 (IPv6) Specification*. RFC Editor, 1998. RFC 2460. (Request for Comments, 2460). Disponível em: <https://rfc-editor.org/rfc/rfc2460.txt>. Citado na página 21.

6 BELLOVIN, S.; CHESWICK, W. Network firewalls. *IEEE Communications Magazine*, v. 32, n. 9, p. 50–57, 1994. Citado na página 22.

7 TANENBAUM, A. S. *Computer Networks*. 5th. ed. [S.l.]: Pearson, 2011. I. Citado 2 vezes nas páginas 24 e 26.

8 GTKWAVE. 1998–2020. <http://gtkwave.sourceforge.net/>. Citado na página 26.

9 BLUESPEC System Verilog Reference Guide. [S.l.]. Citado na página 26.

10 AJAMI, R.; DINH, A. Embedded network firewall on fpga. In: *2011 Eighth International Conference on Information Technology: New Generations*. [S.l.: s.n.], 2011. p. 1041–1043. Citado na página 29.

11 KENI, S. M.; MANDE, S. Design and implementation of hardware firewall using fpga. In: *2018 3rd International Conference for Convergence in Technology (I2CT)*. [S.l.: s.n.], 2018. p. 1–4. Citado na página 29.

12 CHO, Y.; MANGIONE-SMITH, W. Deep packet filter with dedicated logic and read only memories. In: *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. [S.l.: s.n.], 2004. p. 125–134. Citado na página 29.

13 LIU, A. X.; GOUDA, M. G. Diverse firewall design. *IEEE Transactions on Parallel and Distributed Systems*, v. 19, n. 9, p. 1237–1251, 2008. Citado na página 29.

14 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. Citado na página 29.

15 WANG, H. et al. P4fpga: A rapid prototyping framework for p4. In: *Proceedings of the Symposium on SDN Research*. New York, NY, USA: Association for Computing Machinery, 2017. (SOSR '17), p. 122135. ISBN 9781450349475. Disponível em: <https://doi.org/10.1145/3050220.3050234>. Citado na página 29.

16 XILINX. *7 Series Product Tables and Product Selection Guide*. 2014–2021. <https://www.xilinx.com/support/documentation/selection-guides/7-series-product-selection-guide.pdf>. Citado na página 37.