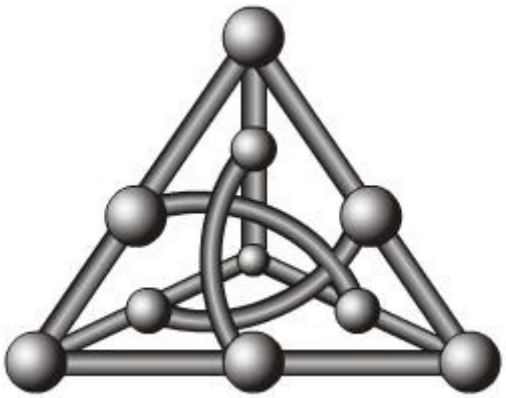
Implementação de Roteamento IPv4 em SDN com P4

Vitor Hugo dos Santos Duarte

Trabalho de Conclusão de Curso
apresentado à Faculdade de Computação da Universidade Federal de Mato Grosso do Sul, como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação.



Orientador: Prof. Ronaldo Alves Ferreira

Campo Grande,
Novembro de 2023

Resumo

A Internet desempenha um papel fundamental na sociedade contemporânea, interligando dispositivos e facilitando a transferência de dados em escala global. Contudo, o constante avanço tecnológico impõe desafios à estrutura e gestão das redes que compõem a Internet. Diante da crescente demanda por velocidade, flexibilidade e eficiência, torna-se imperativo explorar novas abordagens para incorporar funcionalidades inovadoras às redes. Nesse contexto, as Redes Definidas por Software (SDN, do inglês *Software-Defined Networking*) surgem como uma possível solução ao permitir a separação do plano de dados e controle e proporcionar um gerenciamento mais eficaz da rede. Com o aumento do número de protocolos no plano de dados, surge a necessidade de uma programação dinâmica que vá além do modelo estático. Diante disso, a linguagem de programação P4 se destaca, possibilitando a programação independente de protocolo no plano de dados e permitindo lidar com a diversidade crescente de protocolos. No âmbito dessa evolução, os roteadores assumem uma posição central, sendo responsáveis por encaminhar os pacotes para seus destinos corretos. O foco principal deste trabalho é o desenvolvimento de um roteador IP baseado em SDN utilizando a linguagem de programação de plano de dados P4 e que atenda ao padrão estabelecido na RFC 1812. A validação do roteador é realizada por meio de emulação utilizando o Mininet.

Palavras-chave: P4, Roteadores, SDN, Redes de Computadores.

Abstract

The Internet has a fundamental role in contemporary society, connecting devices and facilitating data transfer on a global scale. However, constant technological advancement challenges the structure and management of the networks that make up the Internet. Given the growing demand for speed, flexibility, and efficiency, exploring new approaches to incorporating innovative functionalities into networks is imperative. In this context, Software-Defined Networks (SDN) emerge as a possible solution by allowing the separation of the data and control plane and providing more effective network management. With the increase in the number of protocols in the data plane, the need for dynamic programming goes beyond the static model. Given this, the P4 programming language stands out, enabling protocol-independent programming in the data plane and allowing it to deal with the growing diversity of protocols. Within this evolution, routers are crucial for forwarding packets to their correct destinations. The main focus of this work is the development of an IP router based on SDN using the P4 data-plane programming language that meets the RFC1812 standard. The router validation is performed through emulation using Mininet.

Keywords: P4, Router, SDN, Computer Networks.

Sumário

Capítulo 1

Introdução

A Internet, inicialmente concebida como uma rede de comunicação entre estações de pesquisa do Departamento de Defesa dos Estados Unidos (DoD) na década de 1960 [1], evoluiu ao longo do tempo para se tornar uma infraestrutura global de comunicação. Hoje, é o meio de comunicação mais abrangente, desempenhando um papel crucial na sustentação de tecnologias inovadoras e proporcionando ao mundo contemporâneo maior facilidade na comunicação e conectividade.

De acordo com dados do relatório da Cisco [2], avanços tecnológicos, incluindo a implantação do 5G, Internet das Coisas (IoT), Inteligência Artificial e a expansão da cobertura de redes Wi-Fi, desempenharam um papel fundamental na interconexão de um número crescente de dispositivos e na geração massiva de dados. Em resposta à essa demanda, as novas aplicações necessitam do transporte de grandes volumes de dados pela rede. No núcleo dessas redes, encontram-se os roteadores, que são responsáveis por encaminhar os pacotes para seus destinos. O encaminhamento é feito de maneira a minimizar o processamento e buscar um melhor desempenho de transmissão [3].

Para acompanhar o contínuo avanço da camada de aplicação e lidar com o crescente fluxo de informações e gerenciamento dessas redes, surgiu a abordagem de Redes Definidas por Software ou SDN (do inglês, *Software Defined Network*). Essa abordagem separa os planos de dados e de controle, permitindo a programação do plano de dados. Com a introdução da linguagem P4, o plano de dados transcende a sua natureza estática, tornando-se independente de protocolos e abrindo portas para novas estratégias de processamento. Historicamente, para garantir uma maior eficiência no encaminhamento de pacotes por roteadores, as funções deles por muito tempo foram construídas usando os ASICs (*Application Specific Integrated Circuits*) de forma que tornava seu plano de dados engessado e não permitia atualizações. Para se realizar atualizações, era necessário a construção de um novo hardware, o qual se tornou uma barreira para o avanço das redes [4]. O desenvolvimento de SDN marcou uma transição desse paradigma,permitindo a programação do plano de dados, por meio de linguagens como P4.

O intuito deste trabalho é ir além das implementações convencionais de roteadores, buscando construir um roteador cujo plano de dados é desenvolvido na linguagem P4.

Para isso, seguimos as diretrizes das RFC (*Request for Comments*) para tornar o roteador compatível com o padrão da Internet.

1.1 Objetivo Geral

O objetivo principal deste trabalho consiste na implementação de um roteador IPv4 conforme especificado na RFC 1812 [5] empregando a linguagem de programação P4 para a construção do plano de dados. O roteador resultante deverá incorporar as características essenciais de um roteador convencional, assim como protocolos de roteamento. Para validar a implementação, foram realizados experimentos no emulador de rede Mininet.

1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Implementação do plano de dados, seguindo as diretrizes estabelecidas nas RFCs relativas a IPv4;
- Desenvolvimento do plano de controle, assegurando sua integração com roteadores convencionais;
- Exploração dos protocolos fundamentais utilizados na estrutura da Internet;
- Discussão e uso das funcionalidades da linguagem P4.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta uma revisão de conceitos fundamentais, que são essenciais para compreender o funcionamento e o desenvolvimento do roteador proposto.

2.1 TCP/IP

Para que haja comunicação em diferentes dispositivos conectados a uma rede, é essencial a utilização de protocolos e padrões. Na Internet, o modelo usado é o TCP/IP, que pode ser chamado também de modelo Internet [6]. Este modelo estrutura os protocolos, viabilizando a comunicação entre uma ampla variedade de sistemas de computadores interconectados na rede. O modelo TCP/IP possui quatro camadas: aplicação, transporte, rede e subrede de comunicação, sendo que a subrede de comunicação é dividida em camadas de enlace e física, conforme ilustrado na Figura 2.1. Cabe salientar que as camadas de enlace e física não são objetos de padronização na arquitetura TCP/IP. O modelo TCP assume apenas que a subrede de comunicação é capaz de entregar um quadro entre dois elementos (*e.g.*, roteador and host) de rede.

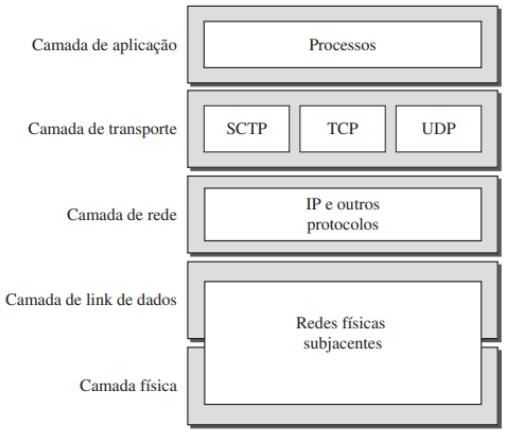


Figura 2.1: Camadas do modelo TCP/IP [1].

1. Camada de aplicação — Representa o nível mais elevado do modelo TCP/IP, fornecendo interfaces para serviços de alto nível utilizados por programas que acessam a rede. Essa camada abriga diversos protocolos, incluindo, mas não se limitando a HTTP, SMTP, FTP, DNS e TELNET.

2. Camada de transporte — É responsável pela gestão do controle de fluxo de dados e controle de erro entre a origem e o destino da comunicação. Nessa camada, destacam-se os protocolos TCP e UDP. O TCP assegura a entrega de pacotes com garantia de integridade dos dados, voltada a conexão. Em TCP, há necessidade de estabelecimento de uma conexão entre a origem e destino antes do início da transferência de dados. Ele é adequado para aplicações que demandam confiabilidade, como transferência de arquivos. Por outro lado, o UDP não visa garantir a integridade dos dados transmitidos e é não orientado a conexão. Ele tem menor sobrecarga de largura de banda e introduz menos latência quando comparado ao TCP. O UDP é muito utilizado em aplicações de streaming pela baixa sobrecarga e latência introduzida.

3. Camada de rede — É responsável em entregar um pacote da origem ao destino final, realizando o encaminhamento de pacotes entre roteadores e escolhendo o melhor caminho por onde o pacote deve ser transferido. O protocolo de rede utilizado na Internet é o protocolo IP. O formato de um datagrama IP é ilustado na Figura 2.2.

4. *Enlace* e física — São as camadas de nível mais baixo em uma rede de computadores e desempenha um papel crucial ao proporcionar a abstração do hardware e configurar os mecanismos de transmissão no meio físico. Além disso, assume a responsabilidade de representar os níveis lógicos de comunicação no meio de transmissão e efetuar a própria transmissão. Nesta instância, os pacotes de dados são encapsulados em quadros (*frames*) para a subsequente transmissão. Nesse contexto, o exemplo mais comum de protocolo é o Ethernet, o qual desempenha um papel central na comunicação eficiente nesse estrato da rede.

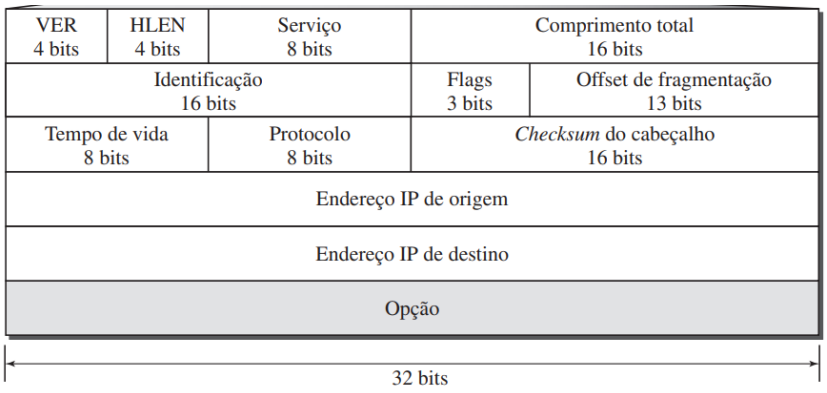


Figura 2.2: Formato do datagrama IP [1].

2.2 Roteadores

A Internet é composta por redes interligadas e os dispositivos que fazem a conexão entre essas redes são os roteadores [6]. Operando na camada 3 do modelo TCP/IP, esses dispositivos são responsáveis por encaminhar pacotes na rede de computadores. Um roteador conecta diversas redes distintas, sendo que a identificação de uma rede é realizada por meio de uma máscara de rede. Para determinar qual interface o roteador deve usar para enviar um pacote, é empregado um algoritmo de roteamento.

A definição da rota que um pacote deve percorrer é estabelecida com base em tabelas de encaminhamento mantidas pelo roteador. Essas tabelas possuem entradas para cada rede diferente, as quais podem ser inseridas manualmente por um administrador de rede ou atualizadas periodicamente por um algoritmo, como no caso do protocolo RIP [7]. Quando um pacote chega a uma das interfaces do roteador, este analisa o IP de destino do protocolo IP e, utilizando a tabela de encaminhamento, determina a porta de saída correspondente. A Figura 2.3 apresenta um exemplo da disposição de roteadores em uma rede, indicando suas respectivas saídas para cada entrada.

A RFC 1812 [5], intitulada "*Requirements for IP Version 4 Routers*", especifica os requisitos para roteadores que suportam o Protocolo de Internet versão 4 (IPv4). Seu resumo do algoritmo para a transmissão unicast descreve os passos que um roteador deve seguir ao encaminhar um pacote IP para seu destino. As etapas são:

1. Receber o pacote da camada de enlace.
2. Validar o cabeçalho IP.
3. Realizar processamento no cabeçalho IP, como exemplo verificação de *checksum*, versão, tamanho, etc.
4. Examinar o endereço de destino do cabeçalho IP, e verificar se o pacote tem como destino o próprio roteador.
5. Determinar próximo salto, para determinar qual interface de saída do pacote.
6. Verificar se o destino é válido, e se há uma rota válida
7. Decrementa o TTL (*Time to Live*) para evitar *loop* na rede
8. Realizar fragmentação do pacote caso necessário, se o pacote for muito grande para ser transmitido em uma única transmissão, ele deve ser fragmentado para se adequar aos limites da rede.
9. Determinar o endereço da camada de enlace, consulta a tabela ARP para determinar o endereço MAC do próximo salto do qual o pacote deve ser encaminhado.
10. Encapsular o datagrama IP em quadro da camada de enlace e enfileira na fila de saída determinada pela etapa 5.

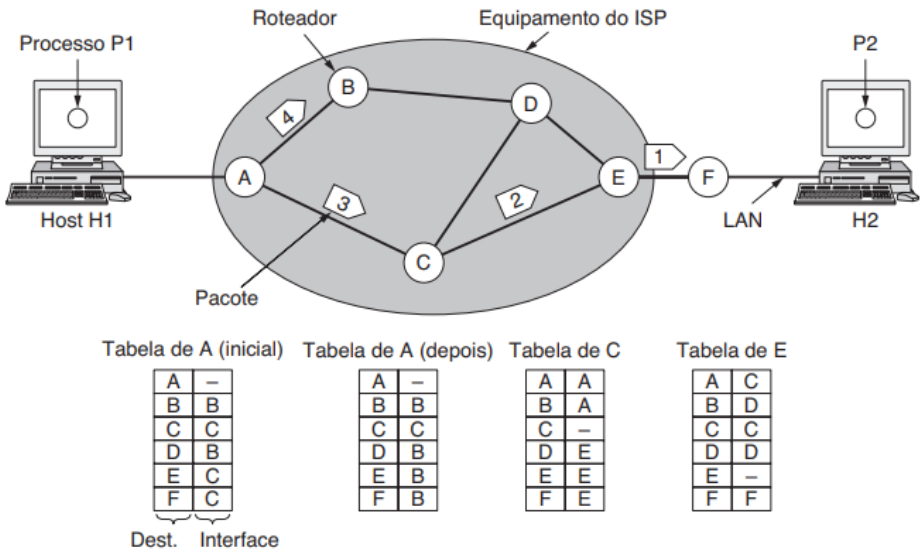


Figura 2.3: Roteamento usando tabela de encaminhamento [8].

2.3 Redes Definidas por Software

O conceito de redes definidas por software (do inglês, *Software Defined Networking* – SDN) possibilita a separação física dos dispositivos de rede entre o plano de dados e o plano de controle, sendo configurados por meio de um controlador central. Essa abordagem visa atingir uma maior automação na rede [9]. O plano de dados é responsável sobre o encaminhamento do tráfego de pacotes em direção ao seu destino, orientado pela tabela de encaminhamento fornecida pelo plano de controle. Este, por sua vez, gerencia o roteamento e reside no servidor controlador, o qual detém uma visão abrangente da topologia, estabelecendo comunicação com os dispositivos de rede, conforme ilustrado na Figura 2.4

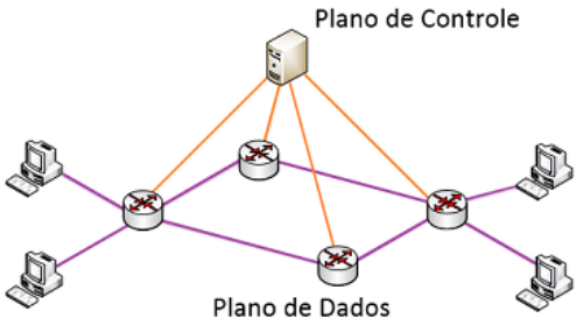


Figura 2.4: Visão de uma rede SDN [10].

A comunicação entre os dispositivos de rede e o controlador tem como principal protocolo o *OpenFlow* [11] permitindo assim o controlador realizar operações como

inserção, atualização e exclusão na tabela de encaminhamento dos comutadores. O *OpenFlow* possui alguns campos fixos para tabela de encaminhamento. Essa abordagem contribuiu para o gerenciamento centralizado da rede como um todo. No entanto, com o surgimento de novas aplicações, os campos do protocolo de comunicação tendem a aumentar, demandando atualizações. A Tabela 2.1 ilustra o aumento do número de campos com passar dos anos.

Tabela 2.1: Campos reconhecidos pelo OpenFlow [12]

Version	**Date**	**Header Fields**
OF 1.0	Dec 2009	12 fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 fields (MPLS, inter-table metadata)
OF 1.2	Dec 2011	36 fields (ARP, ICMP, IPv6, etc.)
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields

Com o avanço da tecnologia no plano de dados, surgiu a linguagem P4 como uma estratégia para superar a rigidez dos campos de cabeçalho no *OpenFlow*, que anteriormente não permitia a adição flexível de novos campos. Antes, a introdução de um novo campo de cabeçalho exigia a criação de uma nova versão do protocolo, resultando em maior complexidade e aumento do tamanho da tabela de encaminhamento [12]. A adoção da linguagem P4 representa um passo significativo para aprimorar a flexibilidade e a escalabilidade na definição de protocolos em redes, permitindo inovações sem a necessidade de revisões do protocolo.

2.4 A Linguagem P4

P4 (do inglês, *Programming Protocol-independent Packet Processors*) é uma linguagem para programação de plano de dados de alto nível, que tem como objetivo de flexibilizar a análise dos pacotes no plano de dados, como protocolo independente de programação [12]. A linguagem surgiu com objetivo de superar as limitações do *OpenFlow*, além disso superar as limitações do processamento de pacote através de ASICs, o qual tornava a inserção de novas funcionalidade uma barreira na evolução das redes [11]. Diante disso o P4 busca três objetivos principais:

1. **Independência de Protocolo:** O dispositivo de rede deve ser capaz de analisar qualquer tipo de protocolo, possibilitando o processamento dinâmico de pacotes. Essa abordagem proporciona maior flexibilidade para gerenciar e adicionar os cabeçalhos dos pacotes e simplifica a introdução de novas funcionalidades na rede, sem a necessidade de alterações complexas no protocolo.

2. **Independência de Arquitetura:** O compilador gera o programa de execução independente do hardware subjacente, o que facilita o desenvolvimento em diferentes plataformas.

3. **Reconfiguração em Campo:** Os programadores podem alterar o programa P4 e redefinir o processamento de pacotes nos comutadores.

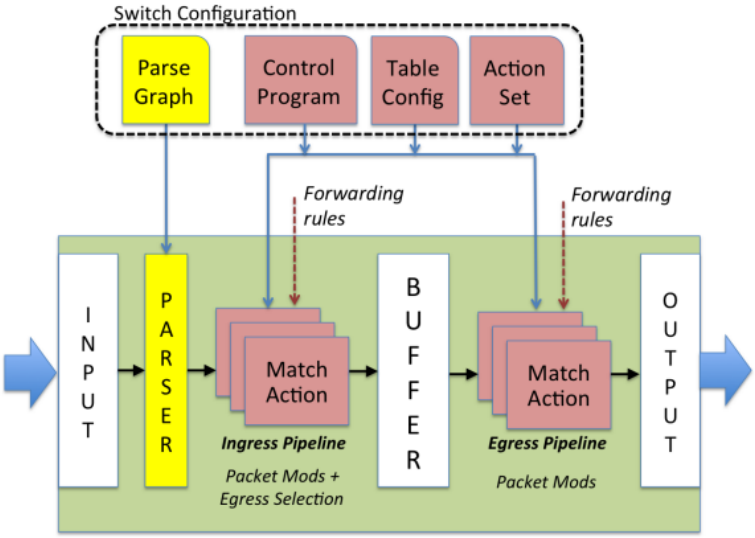


Figura 2.5: Modelo do pipeline de encaminhamento [12].

Baseada na arquitetura *Protocol-Independent Switch Architecture* (PISA), a linguagem P4 adota uma abordagem baseada em pipelines, onde cada segmento é um bloco programável. A arquitetura inclui declarações de cabeçalhos, analisadores implementados por meio de máquinas de estados, e tabelas de correspondências [11]. Os cabeçalhos desempenham um papel crucial, definindo não apenas o formato, mas também o tamanho de cada campo nos pacotes de dados. O analisador, utilizando máquinas de estados, realiza a leitura e interpretação desses cabeçalhos durante o processamento dos pacotes. As tabelas de correspondências são elementos-chave na arquitetura P4, armazenando informações usadas para realizar correspondências durante o processamento. Quando ocorre uma correspondência, a tabela pode acionar a execução de ações específicas, adicionando uma camada de programabilidade ao processamento de pacotes [13].

O modelo de pipeline, exemplificado na Figura 2.5, representa visualmente o fluxo de pacotes pelos diferentes estágios. Cada estágio é composto por blocos programáveis que desempenham funções específicas, como análise de cabeçalhos, correspondência em tabelas e execução de ações baseadas nessas correspondências. Essa abordagem modular e programável confere uma notável flexibilidade à linguagem P4 na definição e implementação de protocolos de rede. Essa flexibilidade faz da P4 uma escolha poderosa para programação de dispositivos de rede, incluindo switches e roteadores.

Capítulo 3

Trabalhos relacionados

Este capítulo apresenta uma breve revisão bibliográfica contendo alguns trabalhos que pertencem ao escopo deste projeto. A revisão visa situar o projeto no contexto mais amplo da pesquisa acadêmica.

3.1 Click

A arquitetura de software Click [14] proporciona flexibilidade no processamento de pacotes, sendo composta por um conjunto de elementos desenvolvidos em C++. Esses elementos podem ser integrados para construir um roteador de maneira modular, mediante a configuração do roteador pela interconexão desses elementos. Embora o Click demonstre eficiência no processamento de pacotes com o uso de CPU, observam-se algumas limitações. Especificamente, a arquitetura não foi concebida para criar tabelas de correspondência de forma dinâmica, o que pode representar uma restrição em ambientes que exigem adaptações frequentes. Adicionalmente, é relevante mencionar que o Click impõe uma limitação ao processamento em uma única *thread*, resultando em uma restrição ao processamento simultâneo. Essa característica pode ser uma consideração importante em cenários nos quais o processamento em paralelo é essencial para otimizar o desempenho do sistema.

Um exemplo de trabalho que realiza análise de programa em Click é a ferramenta Clara [15]. Seu objetivo principal consiste em avaliar o desempenho de estratégias de *offloading* e tem como propósito automatizar e propor insights de desempenho para essas funções implementadas em Click. Para isso, a ferramenta transforma um código de função de rede Click em nível de programação para representação intermediária usando LLVM. Essas estratégias fazem uso de técnicas de aprendizado de máquina como, por exemplo, *Long-short Term Memory* (LSTM) para prever o desempenho entre plataformas através do número de instruções e acesso à memória.

Em resumo, embora a arquitetura Click ofereça flexibilidade no processamento de pacotes, ela possui algumas limitações, especialmente no que concerne à dinamicidade na criação de tabelas e à capacidade de processamento paralelo, aspectos que devem

ser considerados em ambientes que demandam maior adaptabilidade e eficiência operacional.

3.2 SDN e P4

O artigo que propõe o *switch OpenFlow* [16] apresenta uma proposta para enfrentar os desafios associados à realização de experimentos em ambientes de rede real. Neste trabalho destaca a dificuldade enfrentada pela comunidade científica ao tentar testar novos experimentos em larga escala e propõe o *OpenFlow* como uma solução viável. O trabalho explora a necessidade de ambientes de teste mais acessíveis e práticos, especialmente em contextos universitários, para permitir a experimentação com novos protocolos de rede. Além disso, descreve os componentes essenciais dos *switches OpenFlow*, como a tabela de fluxo, o canal seguro de comunicação com o controlador e o protocolo *OpenFlow*, apresentando dois tipos de *switches OpenFlow*: dedicados e com *OpenFlow* habilitado. Nesse contexto, o artigo discute a flexibilidade do *OpenFlow* para suportar uma variedade de experimentos, desde testes de novos protocolos de roteamento até redes não IP, mobilidade, VLANs e processamento de pacotes. Destaca-se a importância do *OpenFlow* como uma plataforma experimental que permite aos pesquisadores criar ambientes de teste semelhantes a redes reais, facilitando a inovação na área de Redes de Computadores. Este trabalho serve como uma referência valiosa para compreender as possibilidades e benefícios do *OpenFlow* na criação de ambientes de teste dinâmicos e práticos, especialmente em contextos acadêmicos e de pesquisa em redes de computadores.

Dentro desse contexto, a linguagem P4 que surgiu como alternativa para sair dos padrões do *OpenFlow* e tem sido amplamente adotado em diversos trabalhos para a programação do plano de dados. Um exemplo notável é a implementação do método *Discrete Wavelet Transform* [17], que aborda a análise de fluxo de dados para a detecção de periodicidade em um SmartNic Netronome. Nesse contexto, o método se destaca por decompor um sinal específico em diferentes componentes de frequência, analisando cada componente com uma resolução que corresponde à sua escala. A utilização da linguagem P4 revelou-se um mecanismo eficaz para explorar o plano de dados. Este trabalho destaca sua eficácia na manipulação de dados complexos e na extração de informações relevantes. O foco na análise de periodicidade em um SmartNic específico adiciona um aspecto prático e aplicado ao uso dessa abordagem inovadora.

Capítulo 4

Implementação

Neste capítulo, é fornecida uma visão sobre a implementação do roteador proposto. A linguagem P4, embora robusta, apresenta algumas limitações, como a ausência de estruturas como *loops*, recursão, desvios condicionais, ponteiros, gerência de tempo, uso de *threads* e estruturas de dados como vetores e matrizes. Essas limitações demandaram a implementação de certas funcionalidades como, por exemplo, a atualização das tabelas de correspondência no plano de controle, além das funcionalidade padrões deste plano como a gerência e supervisão de como os dados trafegam na rede.

Como resultado, a implementação do roteador foi dividida em duas partes: o plano de dados, implementado utilizando a linguagem P4, e o plano de controle, desenvolvido no controlador utilizando a linguagem Python e sua API para comunicação. Todo o código desenvolvido neste projeto está disponível no GitHub [18].

4.1 Plano de Dados

O modelo arquitetural escolhido da linguagem P4 para a implementação do plano de dados, que melhor se adequa ao algoritmo de processamento exigido pela RFC 1812, é o modelo comportamental conhecido como "v1model". Essa escolha é respaldada pela sua divisão em um pipeline de seis estágios, composto por blocos programáveis: *parser*, verificação de *checksum*, *ingress*, *egress*, atualização de *checksum* e *deparser* com mostra a Figura 4.1. Essa estrutura apresenta uma notável semelhança com o algoritmo descrito na RFC, tornando-o a opção mais apropriada para atender aos requisitos específicos.

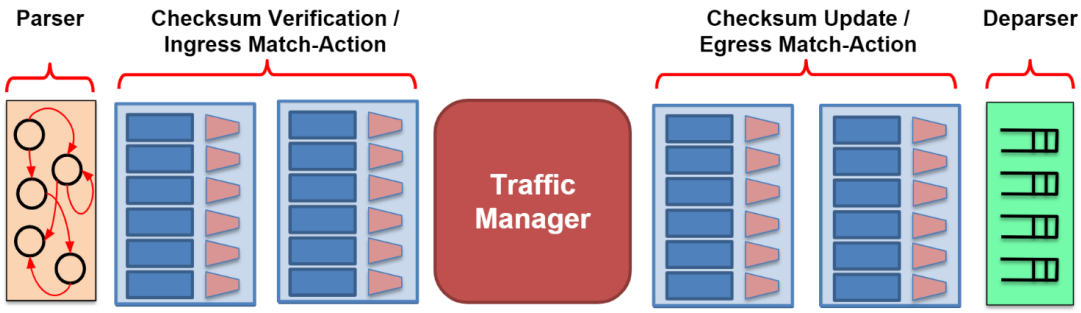


Figura 4.1: Arquitetura v1 model [19]

4.1.1 Cabeçalhos

Nos cabeçalhos são definidos os campos que podem ser processados pela linguagem P4, que são extraídos pelo *parser*. Nesse espaço devem ser definidas todas as estruturas que especifica o nome e a largura de cada campo. Para o roteador proposto, foram definidos cabeçalhos como Ethernet, IPv4, ICMP, ARP e estruturas para metadados. Além desses, duas estruturas para controle de comunicação para pacotes trocados entre o plano de dados e controle foram especificadas: *packet_in* e *packet_out*. Quando o roteador envia ou recebe um pacote do controlador, um novo cabeçalho é adicionado ao pacote: o *packet_in* quando o pacote vai para o controlador e *packet_out* quando sai do controlador. A Figura 4.2 apresenta o formato dos pacotes.

```
@controller_header("packet_in")
header packet_in_header_t {
    bit<8>   opcode;
    bit<32> operand0;
    bit<48> operand1;
    bit<48> operand2;
}
```

(a) Packet out.

```
@controller_header("packet_out")
header packet_out_header_t {
    bit<8>   opcode;
    bit<32> operand0;
    bit<32> operand1;
    bit<32> operand2;
}
```

(b) Packet in

Figura 4.2: Cabeçalhos de controle.

4.1.2 *Parser*

Após a entrada de um pacote no roteador P4 no buffer de entrada, o primeiro processamento ocorre no *parser*. É nesse ponto que os cabeçalhos do pacote são lidos. Esse estágio, definido por blocos que utilizam máquina de estado finita, guia o pacote ao longo de um caminho específico, permitindo a extração dos cabeçalhos.

No diagrama da Figura 4.3, são representados os possíveis trajetos que um pacote pode seguir no roteador implementado, com base nos cabeçalhos definidos. O primeiro passo no diagrama é verificar se o pacote foi recebido pelo plano de controle e

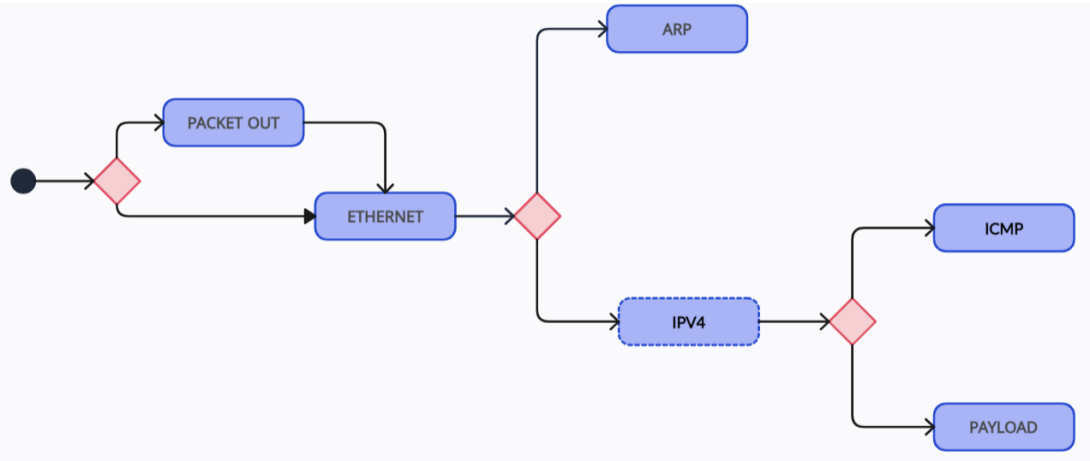


Figura 4.3: Modelo lógico do parser

extrair o *packet_out* se isso for confirmado. Caso contrário, o pacote Ethernet é extraído e segue o modelo TCP/IP até a camada de rede. Em cada estado, o próximo curso do pacote é determinado; caso o próximo estado não seja identificado, o pacote pode ser marcado como rejeitado e descartado ou enviado por padrão para o próximo estado. Quando o pacote alcança os estados finais, eles são aceitos e encaminhados para o próximo bloco de processamento.

4.1.3 Verificação de *Checksum*

Nesse estágio é verificado o *checksum* do cabeçalho IP do pacote recebido e comparado com o campo de *checksum* do mesmo. Para isso, é usada a função "*verify_checksum*" disponibilizada pela arquitetura v1model. Caso tenha erro na verificação o campo *checksum_error* do metadado do pacote *standard_metadata* é definido com o valor diferente de zero, senão ele é definido como zero. Essa etapa assegura a integridade dos dados do cabeçalho IP, ajudando a identificar eventuais erros na transmissão.

4.1.4 *Ingress*

Nesse estágio é realizada a maior parte do processamento do pacote recebido, com base nos cabeçalhos extraídos pelo *parser*. Além disso, nesse bloco é onde são estabelecidas as tabelas de correspondência (*matches*) usadas. Detalhes da implementação podem ser verificados em [18].

Em termos gerais, o processamento segue as seguintes etapas:

1. Identificação do protocolo IP: Determina se o cabeçalho IPv4 foi extraído no parser.

2. Descarte do pacote IP em caso de erro de *checksum*: Se for identificado um erro no *checksum*, o pacote é descartado.

3. Verificação se o pacote é destinado ao roteador: Determina se o destino do pacote é o próprio roteador.

4. Determinação da interface e endereço IP do próximo salto para o pacote: Define a interface de saída e o endereço IP do próximo salto para onde o pacote deve ser encaminhado utilizando a tabela de encaminhamento.

5. Decréscimo do Time To Live (TTL) do pacote: Reduz o valor do TTL no cabeçalho IP e descarta o pacote caso o valor seja zero.

6. Resposta em caso de pacote ICMP direcionado ao roteador: Se o pacote for um ICMP destinado ao roteador, define a resposta apropriada.

7. Criação de pacote de ICMP: caso seja necessário o plano de dados cria pacotes ICMP para comunicar problemas com a transmissão de dados.

8. Determinação do endereço da camada de enlace do pacote: Define o endereço MAC de destino do pacote na camada de enlace.

9. Decisão sobre o encaminhamento do pacote para o controlador: Define se o pacote deve ser encaminhado para processamento no controlador.

Essas etapas são fundamentais para um correto processamento do pacote, onde são tomadas diversas decisões com base nos cabeçalhos e no conteúdo do pacote. Este bloco é crucial para encaminhar os pacotes de forma adequada na rede. Uma parte essencial desse procedimento é a utilização das tabelas de correspondência, desempenhando papéis distintos na configuração e no encaminhamento dos pacotes.

- Tabela de Configuração de Interface:
  Essa tabela é a primeira a ser consultada e é responsável por associar os endereços IP e MAC de cada interface do roteador. Utiliza o número de identificação da interface como chave de entrada, exigindo correspondência exata para configuração dos metadados do pacote.

Tabela 4.1: Tabela de configuração das interfaces do roteador

Interface	IP	MAC
1	11.0.0.10	08:00:00:00:01:00
2	22.0.0.10	08:00:00:00:02:00

Se a entrada da tabela corresponder ao pacote, uma função é chamada para configurar os seus metadados com os dados da interface de entrada.

- Tabela de Encaminhamento

  A segunda tabela de correspondência é a tabela de encaminhamento, composta por entradas que utilizam o endereço IP em conjunto com a máscara de rede no modo de correspondência *Longest Prefix Match* (LPM). Seu retorno inclui a interface de saída do pacote, o endereço IP do próximo salto e a métrica, representando o custo para que o pacote alcance o destino por meio dessa rota. Essa métrica é atualizada pelo protocolo RIP, implementado no plano de controle. Quando há uma correspondência nesta tabela, duas funções podem ser acionadas: *My_router*, quando o pacote é destinado ao próprio roteador, e *ipv4_forward*, quando o pacote deve ser enviado por uma das interfaces disponíveis. Abaixo segue um exemplo dessa tabela de encaminhamento:

Tabela 4.2: Tabela de encaminhamento

IP	Porta de saída	IP do próximo salto	Métrica
11.0.2.0/8	1	11.0.2.10	2
11.0.1.0/8	2	11.0.1.10	4

- Tabela de endereço de camada de enlace

  Por fim, A última tabela desempenha um papel crucial ao determinar o endereço de camada de enlace para o próximo salto do pacote. Neste caso, a entrada na tabela é o endereço IP do próximo salto, e a correspondência é do tipo exata. Como resultado, são fornecidos o endereço MAC do próximo salto e o endereço MAC da interface de saída do roteador. Para ilustrar, um exemplo dessa tabela pode ser visualizado na Tabela 4.3

Tabela 4.3: Tabela de ARP

Interface	MAC de destino	MAC de saída
1	08:00:00:00:05:00	08:00:00:00:01:00
2	08:00:00:00:09:00	08:00:00:00:02:00

4.1.5 Atualização de *Checksum*

Após realizar todo o processamento nos cabeçalhos, que incluiu análise, modificação e configuração, é essencial proceder à atualização dos campos de *checksum* do pacote. O roteador desenvolvido realiza essa etapa para os protocolos IP e ICMP, seja quando estes são lidos no parser ou inseridos no pacote ao longo do processamento. A integridade dos dados é fundamental e, portanto, a atualização dos *checksums* é uma medida imprescindível para assegurar que quaisquer alterações realizadas nos cabeçalhos não comprometam a validade e a segurança dos dados transmitidos.

4.1.6 *Deparser*

No final do processamento do plano de dados, o pacote precisa ser reconstruído na ordem do modelo TCP/IP com as alterações realizadas, pois os cabeçalhos foram extraídos no parser e agora precisam ser reinseridos. Isso é feito no *deparser*, seguindo a ordem definida para os cabeçalhos. Posteriormente, o pacote é encaminhado para a porta correspondente.

4.2 Plano de Controle

O plano de controle é executado em um único controlador conectado a todos os roteadores da rede, sendo que cada roteador é tratado como um processo distinto dentro desse controlador. A comunicação entre os diferentes planos é realizada por meio de uma API. Especificamente, no contexto deste trabalho, a comunicação entre os planos foi efetuada utilizando o P4Runtime com o gRPC.

As definições das tabelas de correspondência usadas são realizadas no P4, entretanto, o cadastramento deve ser realizado pelo controlador, sendo responsável pela gestão individual de cada entrada.

Dentre as funcionalidades desse plano temos:

1. Receber e processar pacotes do plano de dados.
2. Limpar as entradas na tabela ARP após a expiração de seu tempo de vida.
3. Armazenar pacotes que estão aguardando resposta ARP.
4. Reconfigurar a tabela ARP com base nas respostas ARP recebida.
5. Enviar periódicamente pacotes RIP de atualização para todos os roteadores vizinhos a cada 30 segundos.
6. Atualizar a tabela de roteamento usando respostas recebidas do RIP.
7. Enviar o pacote para o plano de dados.

Para implementar essas funcionalidades, foram usadas *threads* de usuário. Onde cada pacote recebido gera uma nova thread para seu processamento. A decisão sobre como processar o pacote no plano de controle é identificada pelo campo '*opcode*' no cabeçalho do pacote de entrada. Para aguardar o tempo de expiração das entradas do RIP e limpeza da tabela ARP, uma *thread* é utilizada para cada entrada.

Capítulo 5

Ambiente de Execução

Para validar a implementação do roteador proposto, foi utilizada uma máquina virtual com o sistema operacional Ubuntu 20.04 com todas as dependências necessárias para a emulação [20]. Para a execução da imagem, utilizou-se o VirtualBox na versão 7.0. O mininet [21], emulador que permite executar simulação de rede com switches, links e hosts usando virtualização do *kernel* do Linux, foi usado para criar uma topologia virtual de rede para avaliar a implementação desenvolvida. Alguns roteadores foram instanciados com os programas P4 compilados e foram executados juntos com outros roteadores ligados.

A topologia de teste usada neste trabalho é ilustrada na Figura 5.1. Nela, foi instanciado um controlador, dois roteadores que executam P4 (R1 e R2), um roteador usando o software de roteamento BIRD (R3) e 6 hosts (H1 a H6). Esses dispositivos estão interconectados, formando uma rede que interliga oito redes distintas.

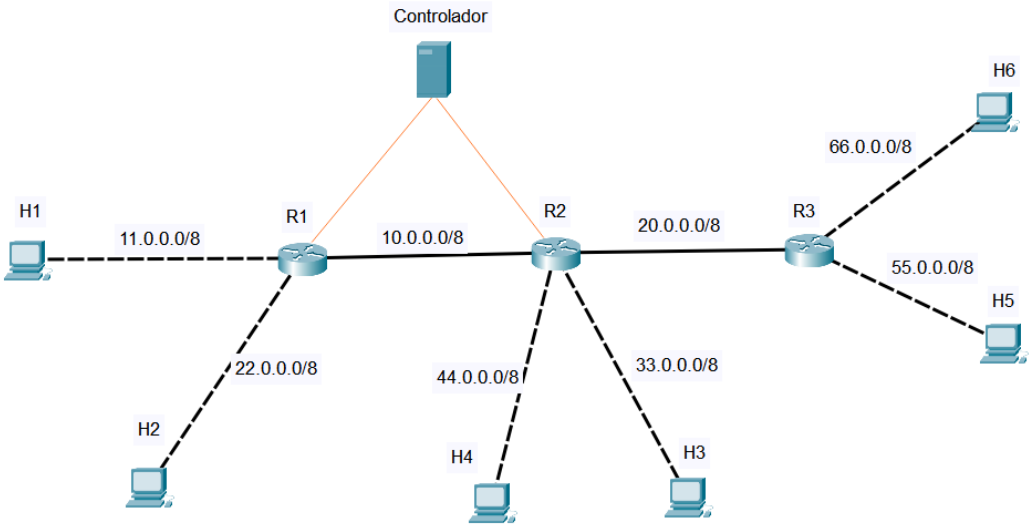


Figura 5.1: Topologia de teste.

5.1 Avaliação Experimental

Para a avaliação, foram realizados testes de funcionalidade, durante os quais foram realizadas capturas de tela para a documentação. Inicialmente, um script foi executado para inicializar a topologia descrita no arquivo *topologia.json*, o qual é utilizado pelo Mininet para instanciar a rede através do *kernel* do Linux. Posteriormente, o programa P4 foi compilado pelo compilador p4c, gerando um arquivo executável que é carregado nos roteadores presentes na rede. Logo em seguida, foram iniciados os canais de comunicação entre o plano de dados e o plano de controle dos roteadores.

- **Teste de RIP**

  Após a configuração do ambiente de validação, o primeiro teste a ser realizado é verificar se os roteadores estão compartilhando suas tabelas de roteamento usando o protocolo RIP. Para isso, foi verificada a tabela de roteamento do roteador R3, que serviu como teste. Foram registrados três momentos da tabela de roteamento usando o Bird, com intervalos entre os anúncios de rede RIP: o primeiro logo após a inicialização dos roteadores, que inicialmente só conhecia rotas para H5 e H6; o segundo, após receber o primeiro pacote RIP do roteador R2; e o terceiro, após receber novamente o pacote RIP.

  Nesse teste, é possível perceber que inicialmente o roteador não conhece rotas para H3 e H4, e após o primeiro pacote RIP chegar, ele cadastra essas redes, como é mostrado na segunda consulta. Por último, após o roteador R2 aprender as rotas de R1, ele as compartilha com o roteador R3, permitindo assim o este conhecer rotas para as redes H1 e H2.

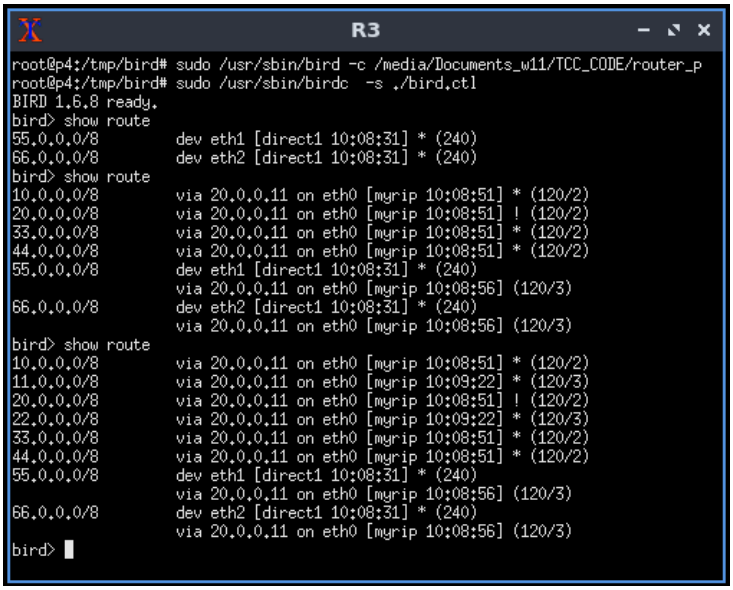


Figura 5.2: Teste de RIP.

- **Teste de ICMP - *Time Exceeded***

Em seguida, foi conduzido um teste específico para verificar se o roteador responde adequadamente quando o campo *Time To Live* (TTL) do pacote IP atinge zero. Para realizar este teste, foi desenvolvido um script que envia um pacote do host H2 para o H4 com o valor TTL configurado como 1. No entanto, ao atingir o primeiro roteador (R1), este descarta o pacote e envia uma resposta ICMP do tipo *Time Exceeded*, conforme evidenciado na Figura 5.3.

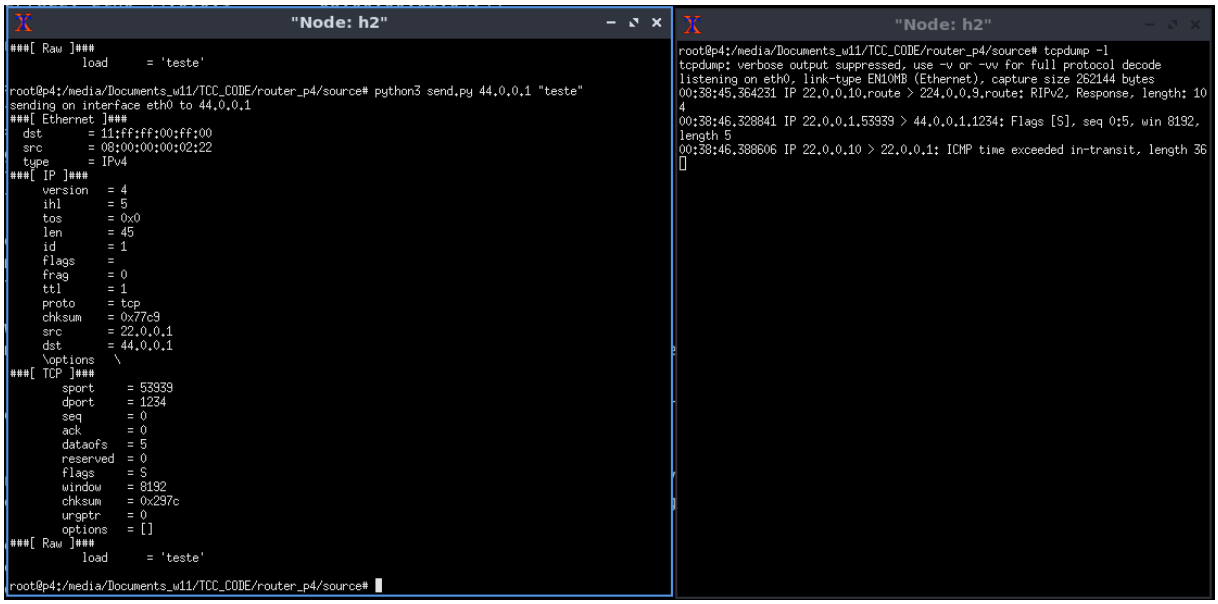


Figura 5.3: Teste de ICMP - time exceeded.

- **Teste de Traceroute**

  O emprego do programa traceroute desempenha um papel crucial ao revelar a trajetória que um pacote percorre até alcançar seu destino. No âmbito deste teste específico, procedeu-se ao rastreamento das rotas que os pacotes seguiriam do host H2 para o H3 e do H2 para o H4. Cada salto ao longo do percurso foi analisado, envolvendo o envio para cada um de 5 pacotes de teste. Os resultados podem ser examinados na Figura 5.4, proporcionando uma visão abrangente da eficiência e latência envolvidas em cada salto.

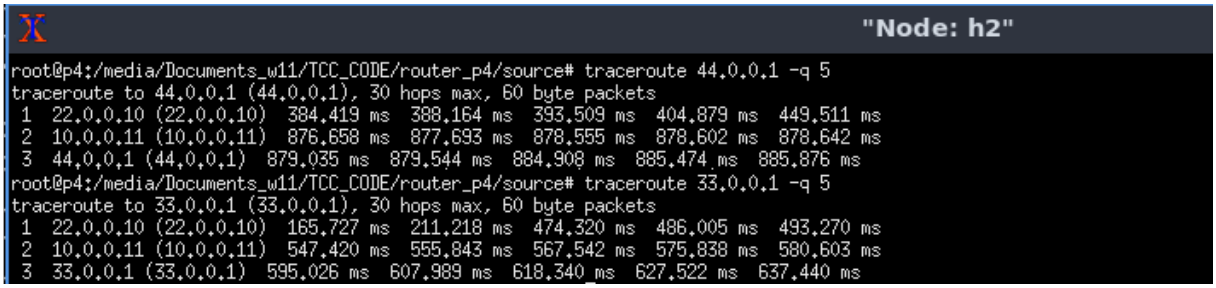


Figura 5.4: Teste de Traceroute.

- **Teste de ICMP – Envio de Pacote TCP**

  O quarto teste foi executado para verificar a eficácia do envio de um pacote TCP de um host para outro. Neste caso, enviou-se um pacote do host H4 para o H2. Na Figura 5.5, são apresentados os valores de cada campo do pacote para o emissor da mensagem e para o receptor. Destaca-se que o pacote foi transmitido com sucesso e o campo TTL decrementado corretamente.

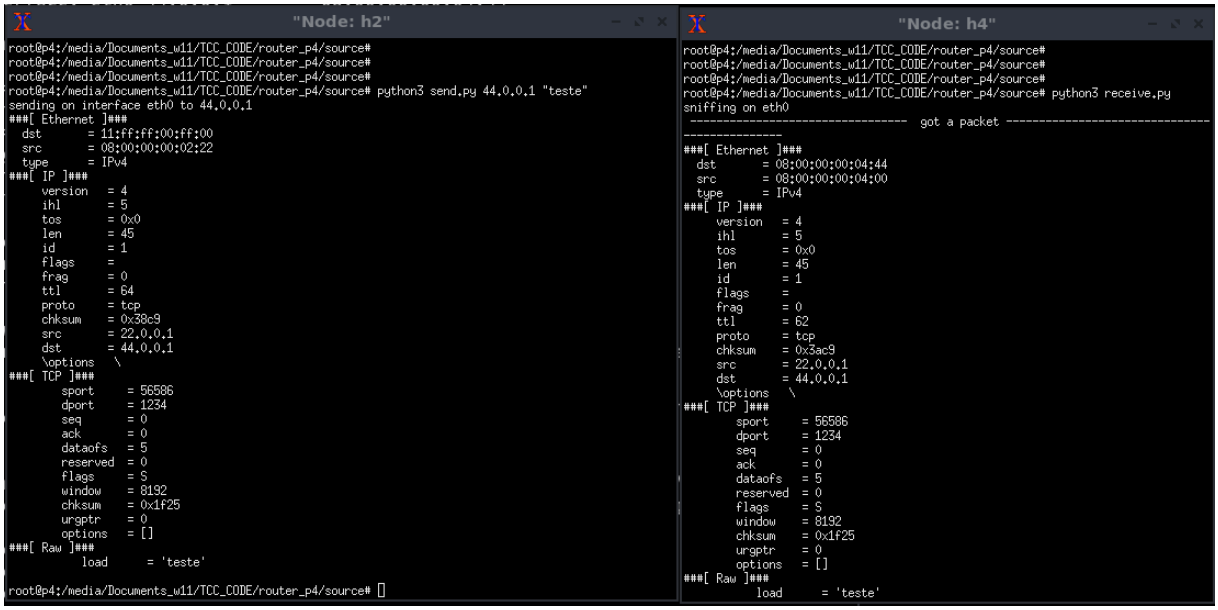


Figura 5.5: Teste de ICMP – envio de pacote TCP.

- **Teste de ICMP - Rede de Destino Inalcançável**

  O teste tem como objetivo verificar a resposta do roteador quando a rede de destino de um pacote é inalcançável. Para realizar este teste, configurou-se uma situação em que o host H2 envia um pacote TCP para um destino da rede 55.0.0.1 a qual o roteador não conhece nenhuma rota. Ao enviar o pacote, o roteador, ao perceber que a rede de destino é inalcançável, retorna um pacote ICMP indicando que a rede de destino é inalcançável como é mostrado na Figura 5.6. Que ocorre quando o roteador não possui uma rota válida para a rede de destino especificada no pacote.

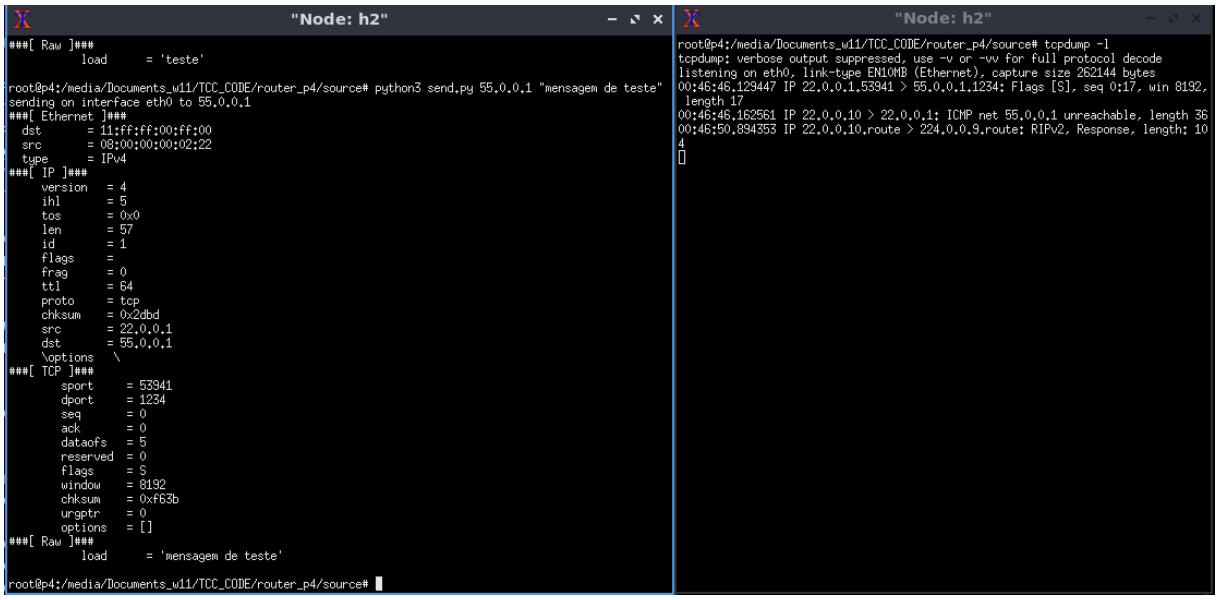


Figura 5.6: Teste de ICMP - Rede de Destino Inalcançável.

5.2 Discussão

A seção experimental deste trabalho buscou validar a implementação do roteador proposto por meio de uma série de testes. A partir destes, foi possível observar que o roteador obteve êxito em todos os testes realizados, apresentando o comportamento esperado. A topologia de teste, conforme ilustrado na Figura 5.1, representou um cenário realista com um controlador e três roteadores interconectando oito redes distintas. Esse ambiente proporcionou uma base para avaliação, simulando condições que o roteador poderia encontrar em um ambiente de rede operacional.

O primeiro teste realizado utilizando RIP para compartilhamento das tabelas de roteamento mostrou eficácia no uso do protocolo para o compartilhamento dinâmico de informações de roteamento entre os dispositivos da rede. A dinâmica observada nesse processo é essencial para manter as tabelas de roteamento atualizadas e com conectividade em resposta a variações na topologia da rede. O teste específico de ICMP - *Time Exceeded* demonstrou que o roteador respondeu adequadamente ao descartar pacotes com TTL atingido, enviando uma resposta ICMP do tipo *Time Exceeded*. Isso confirma a correta implementação do controle de TTL, crucial para a integridade da rede e não permitindo a ocorrência de *loops* de pacotes na rede. O teste de traceroute ofereceu uma análise detalhada de cada salto ao longo do percurso entre hosts, fornecendo informações sobre a eficiência e a latência em cada etapa. Essa avaliação é essencial para compreender o desempenho do roteador em condições práticas de roteamento.

O teste de envio de pacote TCP demonstrou não apenas a eficácia do roteador na transmissão bem-sucedida de pacotes, mas também a capacidade de decrementar corretamente o campo TTL. Isso valida a robustez do roteador ao manipular diferentes tipos de pacotes. O teste de "Rede de Destino Inalcançável"evidenciou a resposta ade-

quada do roteador quando confrontado com uma situação em que a rede de destino não era conhecida. O retorno de um pacote ICMP indicando "Rede de Destino Inalcançável"confirmou a capacidade do roteador de lidar com está situação.

Esses testes proporcionaram uma compreensão abrangente do desempenho do roteador implementado em diferentes cenários, abordando desde a manipulação de pacotes ICMP até o envio bem-sucedido de pacotes TCP. Os resultados obtidos contribuem para validar a eficácia e robustez da implementação proposta, enquanto os testes específicos revelam detalhes importantes sobre o comportamento do roteador em situações diversas.

Capítulo 6

Conclusão

A estratégia de programação utilizando SDN, com a implementação do plano de dados por meio de P4, demonstrou ser uma abordagem eficaz para simplificar a implementação do plano de dados. Nesse contexto, o trabalho obteve êxito ao desenvolver um roteador com implementação em SDN que adere às normas estabelecidas na RFC. Embora algumas funcionalidades detalhadas na RFC não tenham sido completamente implementadas, as essenciais foram incorporadas e testadas com sucesso.

Os resultados provenientes dos testes de funcionalidade atestam a validação do roteador proposto, destacando-se pelo sucesso nos testes realizados. Como perspectiva para trabalhos futuros, há espaço para aprimoramentos e expansões. Pode-se considerar a implementação de funcionalidades que não foram abordadas no escopo atual, visando uma conformidade mais abrangente com a RFC e uma maior diversidade de casos de uso. Além disso, a realização de testes de desempenho comparativos com roteadores tradicionais pode fornecer insights valiosos sobre a eficiência relativa do roteador P4 em diferentes cenários.

Outra linha de exploração consiste em sair dos limites dos emuladores, como mencionado anteriormente, adotando placas de redes programáveis. Essa abordagem poderia estender as capacidades do roteador para ambientes práticos, capitalizando a flexibilidade oferecida pelo P4. Tal expansão não apenas consolidaria a robustez do roteador em contextos do mundo real, mas também abriria portas para inovações e aplicações em redes de próxima geração.

Em síntese, este trabalho não apenas valida a abordagem de programação P4 para o plano de dados do roteador, mas também estabelece uma base para futuras explorações, aprimoramentos e implementações mais amplas em cenários práticos de redes.

Referências Bibliográficas

[1] Behrouz A. Forouzan. *Comunicação de Dados e Redes de Computadores*. AMGH Editora, 4 edition, 2010. 1, 2.1, 2.2

[2] Cisco. Cisco Annual Internet Report (2018–2023) White Paper, 2020. 1

[3] James F Kurose, Keith W Ross, and Wagner Luiz Zucchi. *Redes de Computadores e a Internet: Uma Abordagem Top-Down*. Pearson Addison Wesley, 6 edition, 2013. 1

[4] Allan Christian M Silva, Luiz Claudio G Maia, and Humberto F Villela. Redes Definidas por Software – SDN – Um Estudo Sobre as Vantagens e suas Características. *Computação & Sociedade*, 1(1), 2019. 1

[5] Fred Baker. Requirements for IP Version 4 Routers. RFC 1812, June 1995. 1.1, 2.2

[6] Marcial Porto Fernández. *Redes de Computadores*. EdUECE, 2 edition, 2015. 2.1, 2.2

[7] Gary S. Malkin. RIP Version 2 - Carrying Additional Information. RFC 1723, November 1994. 2.2

[8] A. S. Tanenbaum and D. Wetherall. Redes de Computadores. 5ª edição. *Rio de Janeiro: Editora Campus*, 5:77, 2011. 2.3

[9] Luis Fernando Uria Garcia, Rodolfo S Villaça, Moisés RN Ribeiro, Regis Francisco Teles Martins, Fábio Luciano Verdi, and Cesar Marcondes. Introdução à Linguagem P4-Teoria e Prática. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)-Minicursos*, 2018. 2.3

[10] Antonio Lobato, Ulisses Figueiredo, and Leandro Alves. Redes Definida por Software, 2013. 2.4

[11] Daniel Lazkani Feferman, Juan Sebastian Mejia, N Saraiva, and Christian Esteve Rothenberg. Uma Nova Revolucao em Redes: Programacao do Plano de Dados com P4. *Escola Regional de Informátiva do Piauí (ERIPI), Teresina, Brazil*, 2018. 2.3, 2.4, 2.4

[12] P. Bosshart, D. Daly, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014. 2.1, 2.3, 2.4, 2.5

[13] Pedro Eduardo Camera and Alisson Borges Zanetti. Introdução à Linguagem de Programação P4, o Futuro das Redes. *Sociedade Brasileira de Computação*, 2019. 2.4

[14] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000. 3.1

[15] Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. Automated SmartNIC Offloading Insights for Network Functions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 772–787, New York, NY, USA, 2021. Association for Computing Machinery. 3.1

[16] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, mar 2008. 3.2

[17] Briggette R. Huaytalla, Arthur S. Jacobs, Marcus V. B. Silva, Fabrício B. Carvalho, Ronaldo A. Ferreira, Walter Willinger, and Lisandro Z. Granville. DWT in P4: Periodicity Detection in the Data Plane. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 6343–6348, 2022. 3.2

[18] V. Duarte. Repositório do Projeto. `https://github.com/vitor-ufms/router_p4.git`, 2023. 4, 4.1.4

[19] Noa Zilberman. P4 Tutorial: Welcome. `https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf`, 2018. 4.1

[20] Andy Fingerhub. Máquina Virtual. `https://github.com/jafingerhut/p4-guide/blob/master/bin/README-install-troubleshooting.md`, 2023. 5

[21] Mininet. Mininet Overview. `https://mininet.org`, 2022. 5