José Augusto Tagliassachi Gavazza

Uma proposta para a coexistência de múltiplas Arquiteturas de Internet do Futuro

Sorocaba, SP

26 de Junho de 2020

José Augusto Tagliassachi Gavazza

Uma proposta para a coexistência de múltiplas Arquiteturas de Internet do Futuro

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC-So) da Universidade Federal de São Carlos como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação. Linha de pesquisa: Engenharia de Software e Sistemas de Computação.

Universidade Federal de São Carlos – UFSCar

Centro de Ciências em Gestão e Tecnologia – CCGT

Programa de Pós-Graduação em Ciência da Computação – PPGCC-So

Orientador: Prof. Dr. Fábio Luciano Verdi

Sorocaba, SP

26 de Junho de 2020

Gavazza, José Augusto Tagliassachi

Uma proposta para a coexistência de múltiplas Arquiteturas de Internet do Futuro / José Augusto Tagliassachi Gavazza -- 2020.
113f.

Dissertação (Mestrado) - Universidade Federal de São Carlos, campus Sorocaba, Sorocaba
Orientador (a): Prof. Dr. Fábio Luciano Verdi
Banca Examinadora: Profa. Dra. Cíntia Borges Margi, Prof. Dr. Rodolfo da Silva Villaça
Bibliografia

1. Arquitetura de Internet do Futuro. 2. Redes Definidas por Software. 3. Linguagem de Processamento de Pacotes.. I. Gavazza, José Augusto Tagliassachi. II. Título.

Ficha catalográfica desenvolvida pela Secretaria Geral de Informática (SIn)

DADOS FORNECIDOS PELO AUTOR

Bibliotecário responsável: Maria Aparecida de Lourdes Mariano - CRB/8 6979

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências em Gestão e Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato José Augusto Tagliassachi Gavazza, realizada em 26/06/2020.

Comissão Julgadora:

Prof. Dr. Fabio Luciano Verdi (UFSCar)

Prof. Dr. Rodolfo da Silva Villaça (UFES)

Profa. Dra. Cintia Borges Margi (USP)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Dedico este trabalho aos meus pais, José Augusto, Maria Elisabeth, à minha esposa Bianca e meu filho Arthur.

Agradecimentos

Agradeço,

a Bianca Regina Proença Gavazza pela compreensão e paciência.

ao Prof. Dr. Fábio Luciano Verdi pelo tempo dedicado durante o período de orientação e revisão técnica.

ao Fellipe Augusto Ugliara e ao Paulo Ditarso Maciel Júnior pelo auxílio com a formatação e sugestões para este trabalho.

ao Jeferson Santiago da Silva pelos esclarecimentos em P4.

a todos os professores do PPGCC da UFSCar - Sorocaba, pelos seus ensinamentos.

aos integrantes do projeto FIXP, Prof. Dr. José Suruagy Monteiro, Prof. Dr. Flávio Silva, Prof. Dr. Fábio Luciano Verdi e Prof. Dr. Antonio Marcos Alberti, pela participação no projeto.

a Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), pelo financiamento do projeto "Alavancando a Internet do Futuro no Brasil através da Coexistência e Interconexão de Múltiplas Arquiteturas" (2015/24518-4)", projeto no qual esta dissertação é inserida.

"Jamais considere seus estudos como uma obrigação, mas como uma oportunidade invejável para aprender a conhecer a influência libertadora da beleza do reino do espírito, para seu próprio prazer pessoal e para proveito da comunidade à qual seu futuro trabalho pertencer."

(Albert Einstein)

Resumo

A atual arquitetura de Internet é a mesma desde a época de sua criação e vêm sofrendo incrementos e modificações em seus protocolos para suprir novas demandas que vem surgindo ao longo do tempo. Como a resolução de todas essas questões não são facilmente sanadas apenas com a adição de novas funcionalidades ou modificações na arquitetura, propostas de desenvolvimento de novas arquiteturas de Internet tiveram início pelo mundo. Com diversas novas arquiteturas de Internet sendo propostas, eleger uma nova arquitetura de Internet para substituir a atual torna-se uma tarefa extremamente complexa. Dessa forma, o surgimento de um ambiente heterogêneo composto por múltiplas arquiteturas de Internet que possibilite a coexistência de novas arquiteturas de Internet do Futuro, através da atual infraestrutura de rede, torna-se uma solução viável. Assim, este trabalho apresenta uma proposta que viabiliza a coexistência de múltiplas arquiteturas de Internet, sobre a infraestrutura da rede atual, de forma a possibilitar a comunicação entre entidades de mesma arquitetura de Internet, possibilitando assim o surgimento de um ambiente heterogêneo composto por múltiplas arquiteturas de Internet. Para isso, esta proposta define a criação e operação conjunta de comutadores programáveis implementados em linguagem P4, de controladores específicos para cada arquitetura de Internet suportada pela proposta e de um Protocolo de Controle, utilizado para padronizar a troca de mensagens de controle, independente da arquitetura, viabilizando assim a troca de tráfego entre entidades de uma mesma Arquitetura de Internet do Futuro, através da infraestrutura de rede existente. Para demonstrar esta proposta, foi implementada uma prova de conceito em ambiente virtualizado, onde foram escolhidas as Arquiteturas de Internet do Futuro ETArch e NovaGenesis para serem inicialmente suportadas pela solução, assim como o protocolo IPv4. Por fim, foram realizadas a validação da prova de conceito e a execução de testes relacionados ao tempo de processamento de pacotes de dados e de controle, onde observou-se a independência do tempo de processamento em relação ao tamanho de um pacote de uma arquitetura assim como outras informações de desempenho.

Palavras-chaves: Arquitetura de Internet do Futuro. Redes Definidas por Software. Linguagem de Processamento de Pacotes.

Abstract

The current Internet architecture has been the same since the time of its creation and has undergone increments and modifications in its protocols to meet new demands that have been arising over time. As the resolution of all these issues is not easily solved with the addition of new functionalities or architectural modifications, proposals for the development of new Internet architectures started around the world. With several new Internet architectures being proposed, choosing a new Internet architecture to replace the current one becomes an extremely complex task. Thus, the emergence of a heterogeneous environment composed of multiple Internet architectures that allows the coexistence of new Future Internet Architectures, through the current network infrastructure, becomes a viable solution. Thereby, this work presents a proposal that enables the coexistence of multiple Internet Architectures on the current network infrastructure, in order to enable communication between entities of the same Internet architecture, enabling the emergence of the heterogeneous environment composed of multiple Internet architectures. To this end, this proposal defines the creation and joint operation of programmable switches implemented in P4 language, specific controllers for each Internet architecture supported by the proposal and a Control Protocol used to standardize the exchange of control messages, regardless of the architecture, thus enabling the exchange of traffic between entities of the same Future Internet Architectures, through the existing network infrastructure. To demonstrate this proposal, a proof of concept was implemented in a virtualized environment, where the Future Internet Architectures ETArch and NovaGenesis were chosen to be initially supported by the solution, as well as the IPv4 protocol. Finally, the proof of concept validation and tests related to the processing time of data and control packets were performed, where the independence of processing time was observed in relation to the size of a package of such an architecture as well as other performance information.

Keywords: Future Internet Architecture. Software Defined Networking. Packet Processing Language.

Lista de ilustrações

Lista de Códigos-Fonte

Lista de tabelas

Sumário

Introdução

Desde sua origem, a Internet tem crescido rapidamente e seu uso se diversificado, passando de uma rede de pesquisa acadêmico-militar, composta por estações fixas de grande porte cujas principais aplicações eram o acesso remoto e a troca de arquivos, para uma plataforma de acesso popular com mais de um bilhão e meio de usuários, formada por diversos dispositivos fixos e móveis, executando aplicações de acesso e compartilhamento de informação, áudio e vídeo (MOREIRA et al., 2009).

A Internet como conhecemos, hoje surgiu da criação dos protocolos TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*), que tinham como objetivo atender a requisitos de interconexão de redes, prover a conectividade fim-a-fim entre estações e garantir a acessibilidade de informações (CLARK et al., 2005). Porém, esses mesmos princípios que inicialmente levaram a simplicidade da implementação dos protocolos e da arquitetura, acabaram por reduzir a flexibilidade da arquitetura quanto a adição de novas correções, dificultando assim modificações mais complexas dos protocolos do núcleo da rede.

Dessa forma, com o crescimento do uso da Internet, limitações da arquitetura começaram a surgir, tais como segurança, mobilidade, qualidade de serviço, endereçamento e escalabilidade. Como resposta a esses novos requisitos da rede, alterações e adições de novas camadas de protocolos vêm sendo realizadas de forma incremental à arquitetura da Internet ao longo dos anos. Apesar dessas modificações, os protocolos do núcleo da arquitetura se mantêm os mesmos de cinco décadas atrás, quando as tecnologias, os recursos, as capacidades computacionais disponibilizadas e os princípios para o desenvolvimento dos projetos desses protocolos eram diferentes dos que possuímos hoje em dia (MONTEIRO, 2015).

Em Dezembro de 1991, os integrantes do *Internet Activities Board* (IAB) e do *Internet Engineering Steering Group* (IESG), duas entidades responsáveis pela manutenção dos protocolos da arquitetura da Internet, reuniram-se para discutir sobre os problemas e limitações da Internet a época e como resultado das discussões, enumeraram cinco principais pontos de necessidade de evolução da arquitetura. O primeiro e mais urgente tópico de evolução envolve os temas *endereçamento e roteamento*, pois estes estão diretamente envolvidos com a capacidade de crescimento da Internet. O segundo tópico sugere a evolução da arquitetura atual para uma nova *arquitetura multi protocolos*, a qual suporte ambos os conjuntos de protocolos TCP/IP e OSI (*Open System Interconnection*). O terceiro item destaca a necessidade de evolução da segurança para uma *arquitetura de segurança*, pois, embora isso tenha sido considerado no projeto da Internet, os problemas

modernos de segurança são muito mais amplos, incluindo requisitos comerciais. Além disso, a experiência ao longo dos anos mostrou que é difícil adicionar segurança a um conjunto de protocolos, a menos que esteja incorporada à arquitetura desde o início. O quarto ponto se refere à evolução de *tráfego e controle de estado*, pois a Internet deve suportar aplicativos de transmissão de voz e vídeo, necessitando assim de novos mecanismos que realizem o controle de tráfego e de estado. Por fim, o quinto ponto apresentado é a criação de *novos aplicativos*, que utilizem as melhorias apresentadas (CLARK et al., 1991).

Em decorrência do surgimento das limitações da arquitetura atual da Internet, desde o final dos anos 90 surgiram diversas pesquisas voltadas à especificação e criação de novas arquiteturas, denominadas *Arquiteturas de Internet do Futuro* (*Future Internet Architectures* ou FIAs), que utilizam como base para a proposta de uma nova arquitetura a metodologia *clean-slate* de pesquisa, ou seja, a pesquisa é baseada em novas tecnologias, protocolos e arquiteturas, de forma que não há dependência e/ou restrições dos protocolos nem da arquitetura atual, podendo assim realizar estudos de melhores propostas de solução sem a preocupação dos problemas existentes atualmente (FELDMANN, 2007).

Iniciativas como *NewArch* (CLARK et al., 2004), *Future Internet Design* (FIND) (NSF NETS FIND INITIATIVE, 2020), *Future Internet Architecture* (FIA) (NSF FIA PROJECT, 2020) e *Future Internet Architecture Next Phase* (FIA-NP) (FIA-NEXT PHASE, 2020) mostram que esse assunto tem sido pesquisado desde o início deste século nos Estados Unidos. Na Europa, os programas FP6 (*Framework Program 6*), FP7 (*Framework Program 7*) e H2020 (*Horizon 2020*) financiaram vários projetos de pesquisa focados em novas arquiteturas de rede. Este tópico também estará presente no programa FP9 (*Framework Program 9*) sob a iniciativa Internet de Nova Geração (*Next Generation Internet* ou NGI) (NEXT GENERATION INTERNET, 2020). Na Ásia, podem ser citadas as iniciativas *New Generation Network* (NWGN) do Instituto Nacional de Tecnologia da Informação e Comunicação do Japão, que financiou diversos projetos de pesquisa, dentre eles o projeto *AKARI* (HARAI, 2009) e os projetos chineses *Programa 863*, *Programa 973* e o *12º Plano de Projeto de cinco anos*, criados pelo Ministério de Ciências e Tecnologia da China, que também têm financiado pesquisas voltadas a arquiteturas de Internet do Futuro, como o projeto *SINET* (*Smart Identifier NETwork*) (ZHANG et al., 2016) e ambientes de testes e pesquisas voltadas à evolução da arquitetura de Internet atual.

No Brasil, iniciativas de novas Arquiteturas de Internet do Futuro também vêm sendo propostas, dentre estas, destacam-se dois esforços nacionais de pesquisa, a (*Entity Title Architeture* ou ETArch) (SILVA et al., 2012) e a NovaGenesis (NG) (ALBERTI et al., 2017). Há também a (*Recursive InterNetwork Architecture* ou RINA) (GRASA et al., 2011) (KLOMP, 2016) (MATTA; BOSTON, 2010), que têm sido pesquisada por diversos centros de pesquisa.

Como pode-se observar, o cenário atual com relação a pesquisa das Arquiteturas

de Internet do Futuro mostra a existência de várias novas arquiteturas de rede, com protótipos em diferentes estágios de desenvolvimento, em diversos locais do mundo, cada qual com propostas, objetivos e paradigmas de comunicação específicos.

Duas vertentes de solução são comumente apresentadas. A primeira propõe a substituição da Internet atual por uma nova arquitetura, composta por novos protocolos que não apresentem as limitações atuais, permitindo assim a sua evolução. Porém, a escolha de uma dessas novas arquiteturas como a substituta para a arquitetura atual de rede é uma tarefa complexa. A segunda sugere a continuidade de utilização da arquitetura atual, seguindo com as correções incrementais na arquitetura e em seus protocolos, permitindo assim atender as limitações atuais. De qualquer forma, ambas as soluções descritas não preveem a coexistência das novas propostas de Arquiteturas de Internet do Futuro atualmente existentes ou em desenvolvimento. Assim, ilhas de Arquiteturas de Internet do Futuro continuam existindo.

Desta forma, o surgimento de um ambiente que possibilite a coexistência de diferentes arquiteturas de Internet simultaneamente, sob uma mesma infraestrutura de rede, desponta como uma solução plausível contra a questão da escolha de uma nova arquitetura para ser a substituta da arquitetura atual.

Na infraestrutura de rede atual, um Ponto de Troca de Tráfego de Internet (*Internet Exchange Point* ou IXP) é o local propício para a aplicação dessa proposta, pois, IXPs têm se mostrado receptivos à aplicação de novas tecnologias, que permitem a expansão das suas operações, como Redes Definida por Software (SDN), Virtualização de Funções de Rede (NFV) e mais recentemente, a Programação da Rede, que acabam trazendo benefícios como flexibilização na aplicação de políticas e a centralização da lógica de controle, que simplificam a gestão e manutenção do IXP. Outro motivo da escolha de IXPs, é que estes já possuem toda a estrutura de rede necessária para a realização de troca de pacotes entre as redes de diferentes Provedores de Serviço de Internet conectados ao IXP.

Assim, esta dissertação de mestrado tem como objetivo, apresentar uma proposta a ser implantada em um Ponto de Troca de Tráfego de Internet, para que este passe a processar cabeçalhos de Arquiteturas de Internet do Futuro, possibilitando assim o surgimento de um novo ambiente heterogêneo, propício à coexistência de múltiplas arquiteturas de Internet sobre a infraestrutura de rede atual, tornando possível assim a comunicação de entidades de mesma Arquitetura de Internet do Futuro hoje isoladas.

Para isso, a proposta apresenta a especificação de três camadas, dos processos existentes em cada camada e de um *Protocolo de Controle*. A primeira camada é a *Camada de Controle*, composta pelos servidores que executam os controladores do Plano de Controle de cada arquitetura suportada pela proposta. A segunda é a *Camada de Abstração*, composta por um servidor, que realiza o roteamento de pacotes entre os controladores e

comutadores programáveis multi arquitetura, possibilitando a utilização de uma única interface de rede do comutador programável multi arquitetura para a realização da troca de pacotes entre o comutador e os controladores. A terceira camada é a *Camada de Comutação*, composta por um servidor que atua como comutador programável multi arquitetura que realiza o processamento do cabeçalho do protocolo IP e das Arquiteturas de Internet do Futuro, ETArch e NovaGenesis. Por fim, é definido o *Protocolo de Controle*, que tem por objetivo padronizar a forma de envio e recebimento das operações de controle entre os componentes da proposta.

Com a aplicação desta proposta em um Ponto de Troca de Tráfego de Internet, este acaba por expandir sua funcionalidade, passando a processar, além dos pacotes da arquitetura atual, os pacotes de Arquiteturas de Internet do Futuro, podendo então o IXP passar a ser denominado como um Ponto de Troca de Tráfego de Internet do Futuro (*Future Internet Exchange Point* ou FIXP), como apresentado inicialmente em (MONTEIRO, 2015).

Como forma de incentivar o apoio de pesquisas do Brasil nas iniciativas desta área, auxiliando a alterar o cenário típico onde soluções de inovação surgem inicialmente do exterior, foram selecionadas as Arquiteturas de Internet do Futuro ETArch (SILVA et al., 2012) e NovaGenesis (ALBERTI et al., 2017) para serem as duas primeiras arquiteturas suportadas pela proposta. Estas arquiteturas de Internet do Futuro foram escolhidas por serem desenvolvidas por parceiros envolvidos no projeto *Alavancando a Internet do Futuro no Brasil através da Coexistência e Interconexão de Múltiplas Arquiteturas* (MONTEIRO, 2015), financiado pela *Fundação de Amparo à Pesquisa do Estado de São Paulo* (FAPESP) (FAPESP, 2020), projeto no qual este trabalho se inspira.

Para o desenvolvimento deste trabalho foram realizadas pesquisas sobre as estruturas dos cabeçalhos das Arquiteturas de Internet do Futuro *ETArch*, *NovaGenesis* e do protocolo IP, bem como sobre implementação em linguagem P4, para a definição de um comutador programável multi arquitetura com suporte aos cabeçalhos de novas arquiteturas de Internet. Durante os estudos da estrutura do cabeçalho da NovaGenesis, foi identificada que a arquitetura apresenta uma questão relacionada à fragmentação de pacotes, sendo definida uma solução para a fragmentação de pacotes da NovaGenesis no comutador P4. Para apresentar o funcionamento da proposta, foi criada uma prova de conceito com suporte a ETArch e IP. Inicialmente pretendia-se também utilizar a NovaGenesis durante a validação da prova de conceito e dos testes, entretanto, o seu controlador encontra-se em desenvolvimento. Assim, a NovaGenesis teve seu cabeçalho e o tratamento da fragmentação de pacotes implementados na prova de conceito, sendo validada separadamente. Por fim, foram executados testes para a análise do tempo de processamento de pacotes de dados, de controle e do tempo de conclusão do fluxo de pacotes.

Esta dissertação está organizada da seguinte forma: o Capítulo 1 é dividido em duas subseções que apresentam os conceitos básicos e trabalhos relacionados. No Capítulo 2, é apresentada a definição de Arquiteturas de Internet do Futuro e o que é a metodologia *clean-slate*. Também são apresentadas neste capítulo as Arquiteturas de Internet do Futuro abordadas inicialmente neste trabalho, ETArch e NovaGenesis, descrevendo como cada uma funciona e os formatos das estruturas dos seus cabeçalhos. O Capítulo 3 apresenta, de forma detalhada, a descrição do funcionamento de cada um dos elementos que compõem a proposta, demonstrando também o processo de encaminhamento de pacotes, desde o envio de um pacote de um *Computador A* até o recebimento deste pacote pelo *Computador B*. O Capítulo 4 é separado em duas subseções, onde a primeira descreve de forma mais detalhada sobre cada elemento da proposta, enquanto a segunda subseção apresenta a descrição do ambiente e da realização dos testes de prova de conceito, de processamento de pacotes e dos resultados obtidos. Por fim, é apresentada a conclusão deste trabalho, onde são observados os principais pontos desenvolvidos neste trabalho e onde são propostos trabalhos futuros relacionados ao mesmo.

1 Conceitos Básicos e Trabalhos Relacionados

Neste capítulo, são apresentados e descritos os conceitos básicos utilizados neste trabalho, as motivações para suas pesquisas, suas características e seus benefícios. Também são apresentados trabalhos relacionados e seus conceitos, sendo realizados comentários sobre as similaridades ou diferenças destes com o trabalho apresentado.

1.1 Conceitos Básicos

Nesta seção, são descritos os conceitos básicos utilizados neste trabalho como *Redes Definidas por Software*, a linguagem de *Programação de Processador de Pacotes Independente de Protocolo* conhecida como P4, *Pontos de troca de Tráfego da Internet* (IXP) e *Pontos de Troca de Tráfego de Internet Definido por Software*, mostrando suas características, vantagens e benefícios.

1.1.1 Redes Definidas por Software

As redes de computadores são compostas por diferentes dispositivos de rede como roteadores, comutadores, vários tipos de *middle boxes*, que são dispositivos que manipulam os pacotes trafegados para fins que não o de encaminhamento, como *firewalls*, balanceadores de carga, tradutores de endereço de rede, entre outros. Dessa forma, os operadores de rede são responsáveis por configurar políticas de alto nível em cada um desses dispositivos individualmente, a fim de responder a uma ampla variedade de eventos da rede, como desvios de tráfego, intrusões, detecção de vírus e de riscos à segurança ou detecção de ataques. Comumente, essa tarefa de configuração é realizada através da utilização de ferramentas limitadas, tornando o gerenciamento de rede bastante complexo. Assim, a idéia de utilizar 'redes programáveis' foi proposta como uma forma de facilitar a manutenção, o gerenciamento e a evolução das redes de computadores (KREUTZ et al., 2014).

As pesquisas no campo de Redes Definidas por Software são relativamente novas, mas têm evoluído rapidamente. Em 1995, o *Open Signaling Working Group* (OPENSIG) (OPEN ALLIANCE SIG, 2020) apresentou uma série de palestras propondo a separação entre o hardware de comunicação e o software de controle de redes ATM, redes móveis e da Internet, fornecendo acesso ao hardware da rede por meio de interfaces de rede programáveis.

Também em meados dos anos 1990, a iniciativa *Active Networking* propôs a idéia de uma infraestrutura de rede programável para serviços personalizados, através de duas abordagens principais. A primeira considerava comutadores programáveis pelo usuário, com transferência de dados *in-band* e gerenciamento através de canais *out-of-band* e a segunda propunha cápsulas, que eram fragmentos de programa que podiam ser transportados nas mensagens de um usuário e seriam então interpretados e executados por roteadores.

Outra iniciativa surgida no meio dos anos 1990 é a *Devolved Control of ATM Networks* (DCAN) (DEVOLVED CONTROL OF ATM NETWORKS, 2020), sugerindo que as funções de controle e gerenciamento de dispositivos, no caso comutadores ATM, fossem dissociadas dos próprios dispositivos e delegadas a entidades externas dedicadas a esse objetivo. Neste ponto temos uma característica importante pois, esta virá a se tornar um dos conceito básico por trás das futuras Redes Definidas por Software.

Em 2004, o projeto 4D (4D PROJECT, 2020) propõem a separação entre a lógica de roteamento e os protocolos de controle entre os elementos da rede, dando ao plano de *decisão* uma visão geral da rede, através um plano de *divulgação* e *descoberta*, para realizar o controle de um plano de *dados* responsável pelo encaminhamento de tráfego.

Em 2006, o *IETF Network Configuration Working Group* propôs o NETCONF (RFC 4741-NETCONF, 2006) como um protocolo de gerenciamento para configuração de dispositivos de rede, permitindo que os dispositivos de rede disponibilizassem uma interface por meio da qual a configuração seria realizada.

Também em 2006, o projeto *Ethane* (ETHANE ARCHITECTURE, 2020) propunha o uso de um controlador centralizado para gerenciar políticas e segurança em uma rede, empregando para isso, dois componentes: um controlador para decidir se um pacote deve ser encaminhado e um comutador *Ethane*, que consiste em uma tabela de fluxo e um canal seguro para o controlador, definido assim o que se tornaria a base de uma Rede Definida por Software.

Fundada em 11 de Março de 2011, a *Open Networking Foundation* (ONF) (OPEN NETWORKING FOUNDATION, 2020) é um consórcio sem fins lucrativos formado pelas companias: *Deutsche Telekom*, Google, Microsoft, Verizon e Yahoo, responsável por difundir a idéia de *Redes Definidas por Software* (*Software Defined Networking* ou SDN) e dedicado ao desenvolvimento e a padronização do protocolo *OpenFlow*, sendo proposto por esta mesma entidade a definição atualmente aceita do que é uma Rede Definida por Software.

Uma Rede Definida por Software se refere a um paradigma emergente em redes de computadores que apresenta duas características principais. A primeira é a separação da lógica de controle dos dispositivos de rede responsáveis pelo encaminhamento do

tráfego e a segunda é a utilização de uma lógica de controle programável.

Dessa forma, no Plano de Dados, os dispositivos de rede se tornam simples dispositivos de encaminhamento de pacotes, enquanto que no Plano de Controle, a responsabilidade da lógica de controle da rede é delegada a uma entidade externa programável chamada *controlador* ou *sistema operacional de rede*.

A Figura 1 mostra as representações de uma rede tradicional, onde a lógica de controle é distribuída, pois é implementada por cada dispositivo da rede, e de uma *Rede Definida por Software*, onde a lógica de controle é desacoplada dos dispositivo de rede, sendo centralizada por um dispositivo controlador, não pertencente ao Plano de Dados.

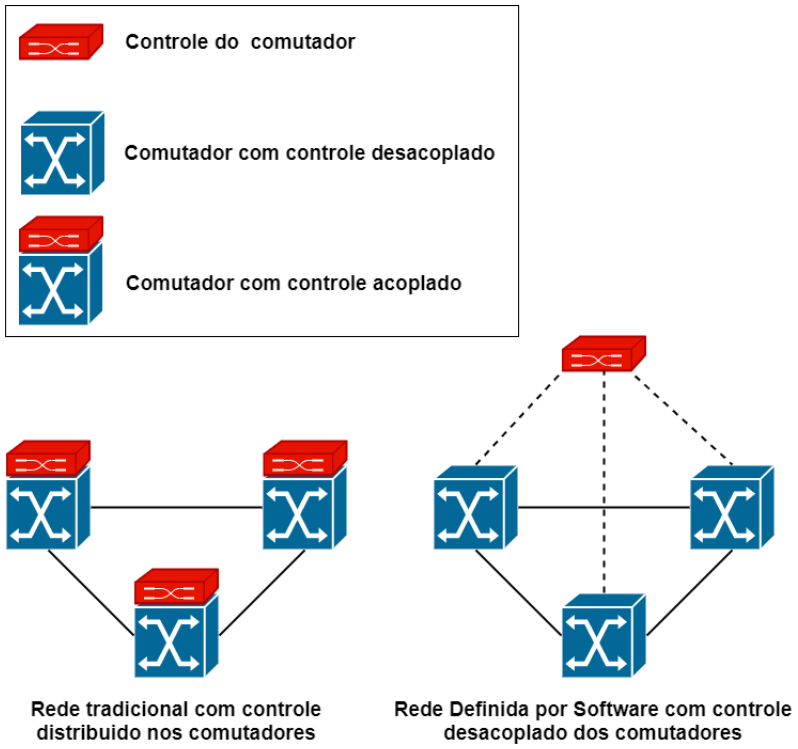


Figura 1 – Estrutura de uma rede tradicional e de uma Rede Definida por Software (baseado em (NUNES et al., 2014)).

Redes Definidas por Software podem ser aplicadas em uma larga variedade de ambientes de rede. Ao desassociar o Plano de Controle do Plano de Dados, as redes programáveis permitem um controle personalizado, gerando assim uma oportunidade para minimizar ou mesmo remover os *middle boxes*, além de simplificar a configuração dos dispositivos e das políticas de controle de tráfego e de segurança, permitindo o desenvolvimento e implantação de novos serviços e protocolos de rede. Com a programação da lógica de controle da rede, uma Rede Definida por Software abstrai a complexidade do gerenciamento individual dos dispositivos, possibilitando a aplicação de políticas e configurações de forma centralizada, simplificando assim a gestão da rede por parte dos operadores, reduzindo custos de gerenciamento, aumentando a segurança e confiabilidade da rede.

1.1.2 Linguagem de Programação de Processador de Pacotes independente de Protocolo

Com a difusão das Redes Definidas por Software (SDN), diversos benefícios passaram a ser propostos às operadoras e provedores de conectividade em redes, como a centralização da gestão de rede através da programação do controle dos dispositivos de rede por um controlador. Pode-se assim observar que, até então, as vantagens oferecidas são referentes ao Plano de Controle. Porém, o processamento de pacotes ainda era realizado via hardware por componentes ASIC (*Application Specific Integrated Circuits*) dos dispositivos de rede, o que engessava a alteração e inserção de novas funcionalidades nestes dispositivos (CAMERA; ZANETTI, 2019).

Com a evolução das redes, estas começaram a migrar funcionalidades do *hardware* para o *software*, onde soluções antes baseadas em *hardware* passaram a ser desenvolvidas inteiramente via *software*, surgindo assim novas tecnologias de redes baseadas em *software* como a Virtualização de Servidores (*Server Virtualization*), a Computação em Nuvem (*Cloud Computing*), as *Redes Definidas por Software* e mais recentemente a Virtualização de Funções de Rede (*Network Function Virtualization* ou NFV), ficando este novo conceito conhecido como *softwarização de redes* (SOUSA; ROTHENBERG, 2017).

Mudanças na forma de como realizar o processamento dos pacotes foram necessárias e a programabilidade de rede acabou se direcionando para o Plano de Dados, não sendo suficiente dizer o que fazer com os fluxos de dados, mas também sendo necessário descrever como fazê-lo. Nesse cenário, surgem propostas de realizar a programação do Plano de Dados e entre elas surge a linguagem de programação de processadores de pacotes independente de protocolo (FEFERMAN et al., 2018).

A linguagem de programação de processadores de pacotes independente de protocolo, denominada P4 (*Programming Protocol-Independent Packet Processors*), apresentada na Conferência SIGCOMM de Julho de 2014 (BOSSHART et al., 2014), é uma linguagem de domínio específico e de alto nível, voltada para a implementação de aplicações que realizam tratamento e encaminhamento de pacotes, trabalhando em conjunto com protocolos de controle de redes definidas por software, permitindo assim a criação de soluções customizadas de dispositivos de rede, como comutadores, roteadores, *firewalls* e balanceadores de carga.

A linguagem P4 foi desenvolvida com três objetivos em mente. O primeiro é permitir a *reconfigurabilidade em tempo de execução*, possibilitando aos controladores modificar, em tempo de execução, a forma como os dispositivos de rede analisam e processam os pacotes, sem interromper a execução dos dispositivos de rede em produção. O segundo objetivo é a *independência de protocolo*, que desvincula os elementos de rede de um protocolo específico. Esta independência é resultado da possibilidade de

definição dos cabeçalhos, da existência de um analisador e de ações de encaminhamento programáveis que conseguem extrair e trabalhar com qualquer cabeçalho definido. Finalmente, o terceiro objetivo é a *independência de dispositivo alvo*, permitindo que programas criados em P4 sejam independentes das especificidades de um único *hardware* alvo. Esta característica é conseguida pois um programa P4, uma vez implementado, pode ser compilado para diferentes dispositivos alvo, como computadores, máquinas virtuais, *hardware* programável, entre outros (BOSSHART et al., 2014).

A linguagem P4 é baseada no *Modelo de Encaminhamento PISA* (*Protocol Independent Switch Architecture Abstract Forwarding Model*), através do qual, dispositivos de rede realizam o encaminhamento de pacotes através de um analisador programável que permite trabalhar com novas definições de cabeçalhos, seguido de uma sequência de verificações de regra de encaminhamento e aplicação de ações de encaminhamento (*match+action*), definidas também pelo programador. Dessa forma, o modelo de encaminhamento abstrato PISA permite à linguagem P4 generalizar como os pacotes são processados por diferentes dispositivos de encaminhamento. A Figura 2 apresenta um diagrama com os componentes integrantes do modelo de encaminhamento abstrato PISA.

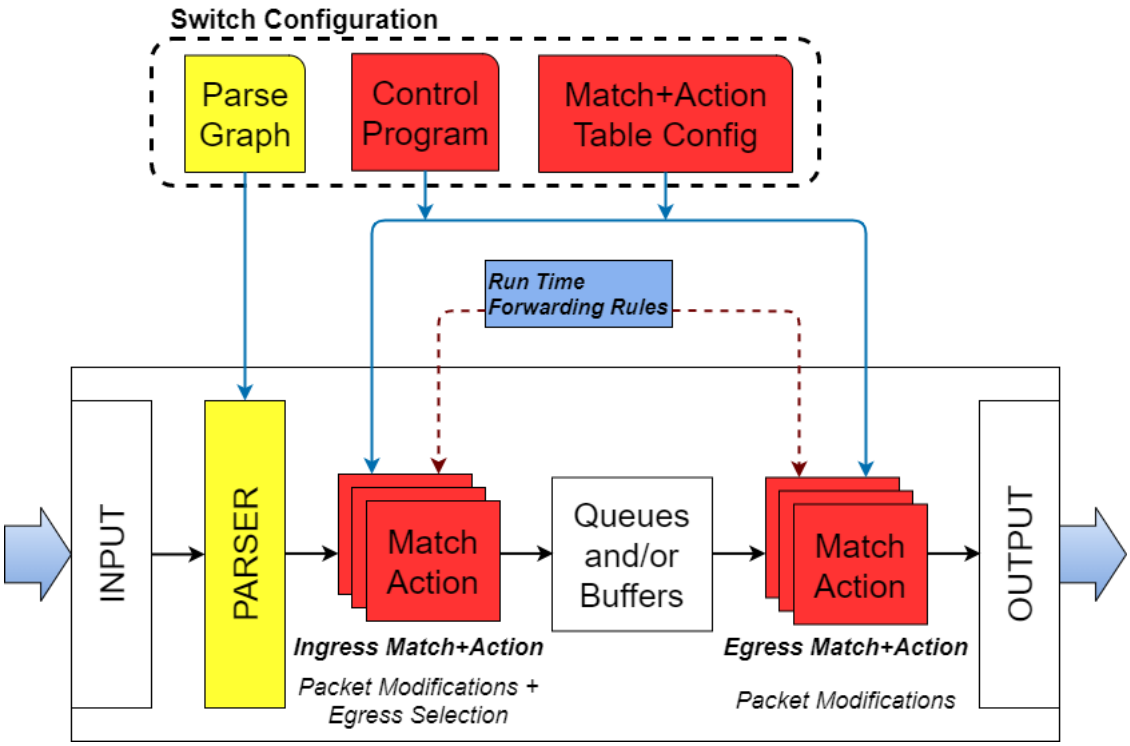


Figura 2 – Diagrama do Modelo de Encaminhamento Abstrato PISA do P4 (BOSSHART et al., 2014).

O modelo de encaminhamento abstrato PISA utiliza dois tipos de operações para a realização do seu controle: as operações de *configuração* e as operações de *população*. As operações de *configuração* programam o *analisador*, definem a ordem de utilização dos estágios de *correspondência e ações* de encaminhamento e especifica os campos do

cabeçalho processados em cada estágio, determinando quais protocolos são suportados e como os dispositivos programados irão processar os pacotes. As operações de *população* adicionam e removem entradas nas *tabelas* de regras de encaminhamento, especificadas durante a configuração, permitindo assim a reconfiguração das tabelas de regras de encaminhamento sem a interrupção no processo de encaminhamento de pacotes.

Um programa em P4 é implementado através da definição e cinco conceitos principais da linguagem: a definição de *cabeçalhos* (*headers*) válidos, a criação de um *analisador* (*parser*) de pacotes, a especificação de *tabelas* (*tables*) de regras de encaminhamento, a definição das *ações* (*actions*) de encaminhamento e a definição do *programa de controle* (*control program*), que especifica a ordem de utilização das tabelas e ações que são aplicadas no encaminhamento de um pacote.

O primeiro conceito da linguagem, a especificação dos formatos dos *cabeçalhos*, é normalmente o primeiro passo para a implementação de uma solução em P4. Um cabeçalho é definido por uma estrutura declarada com o comando *header*, similar a uma estrutura em C, composta por uma sequência de um ou mais campos, formados por um tipo (*varbit*, *bit*, *int*), o tamanho do campo, definido entre os caracteres '<' e '>' e um nome, como apresentado na Figura 3. Se um campo é definido com o tipo (*bool*), basta especificar o nome do campo, pois o tamanho é definido como sendo 1 *bit*. Um campo também pode ser formado por uma estrutura secundária do tipo *struct*, contendo um ou mais campos, sendo seguido apenas por um nome. É importante observar que a linguagem P4 não suporta a definição de estruturas de cabeçalhos que possuam campos que definam outras estruturas do tipo cabeçalho. Por fim, toda estrutura de cabeçalho define automaticamente um campo booleano oculto denominado *validity*, inicializado com *false*, sendo este campo utilizado para verificar se o formato de um cabeçalho recebido pelo analisador é válido ou não com o formato do cabeçalho definido em P4.

```
header Ethernet {
    bit<48>   dstAddr;
    bit<48>   srcAddr;
    bit<16>   etherType;
}
```

(a) Cabeçalho formado por tipos básicos.

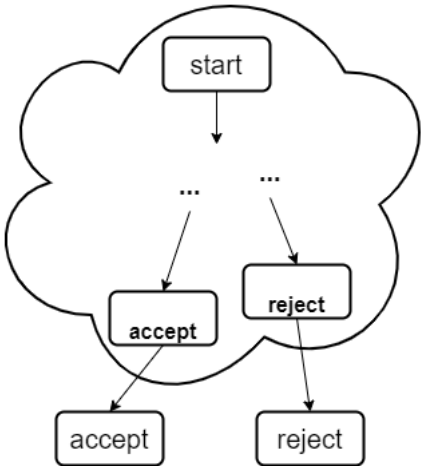
```
struct ethernet_fields {
    bit<48>   dstAddr;
    bit<48>   srcAddr;
    bit<16>   etherType;
}

header Ethernet {
    ethernet_fields   eth;
}
```

(b) Cabeçalho formado por estrutura.

Figura 3 – Exemplos de formatos de cabeçalhos definidos em P4.

O próximo conceito, a implementação do *analisador* de pacotes, têm por objetivo realizar a validação do cabeçalho dos pacotes recebidos através da definição de uma má-

quina de estados com um estado inicial e dois possíveis estados finais. O estado inicial chamado de *start* é obrigatório, devendo sempre estar presente e os dois estados finais são o *accept*, que representa um cabeçalho válido e o *reject*, que indica um cabeçalho inválido. Os estados de aceitação e rejeição são distintos dos estados implementados pelo programador e estão logicamente fora do analisador. O usuário pode implementar quantos estados necessitar, sendo os estados de aceitação e rejeição invocados pelo programador dentro dos estados criados. A validação de um pacote é iniciada com a extração de parte do cabeçalho no estado inicial e continua até que um estado final de aceitação ou rejeição seja explicitamente alcançado. Em caso do fluxo de validação do cabeçalho alcançar um estado que declara a chamada do estado de aceitação *accept*, os cabeçalhos extraídos são encaminhados para análise de *correspondência e ação* (*match+action*) das regras de encaminhamento. A Figura 4 ilustra a estrutura geral de uma máquina de estado do analisador.



(a) Representação da máquina de estados do Analisador.

```
parser start {
   transition parse_ethernet;
}

state parse_ethernet {
   extract(ethernet);

   transition accept;
}
```

(b) Exemplo de codificação do Analisador.

Figura 4 – Representação e exemplo de código da máquina de estados do Analisador.

O conceito seguinte é a definição da *tabela* de regras de encaminhamento e a descrição, pelo programador, de como os campos do cabeçalho ou de metadados devem se corresponder nos estágio de correspondência e ações (*match+action*), indicando assim quais ações serão executadas decorrentes de uma correspondência positiva. O programador implementa a definição de uma tabela para a análise da correspondência (*match*) através da instrução *table*, seguido do nome para a estrutura da tabela. O atributo *key* define quais campos do cabeçalho ou de metadados serão utilizados durante a verificação de correspondência. Os campos chave da tabela pode ser declarados como do tipo *exact*, onde o valor do campo chave da tabela deve corresponder exatamente ao valor do campo do cabeçalho ou metadado, do tipo *lpm*, que utiliza o algoritmo *longest prefix match* para verificar se existe correspondência entre o valor do cabeçalho ou metadado em um intervalo definido na tabela ou do tipo *ternary*, onde o valor do

campo do cabeçalho ou metadado é comparado com uma ou mais máscaras de *bits* definidas na tabela. O programador também define através do atributo *actions*, uma lista com os nomes das ações que podem ser aplicadas a um pacote por essa tabela caso ocorra uma correspondência. O atributo *size*, é um atributo opcional que define a quantidade máxima de regras de encaminhamento que a tabela suportará, sendo o valor máximo deste atributo, dependente do dispositivo alvo. Por fim o atributo *default_action* também é um atributo opcional e o valor a ele atribuído é o nome de uma das ações declaradas na lista de ações *actions*, esta ação será definida como a ação padrão a ser utilizada pela tabela no caso de pacotes que não apresentem correspondência com nenhuma regra existente nesta tabela. A Figura 5 mostra um exemplo de definição de tabela para o protocolo IPv4.

```
table ipv4 {

    key = {
        hdr.ipv4.dstAddr: exact;
    }

    actions = {
        forward;
        drop;
    }

    size = 1023;

    default_action = drop();
}
```

Figura 5 – Exemplo de definição de uma tabela em P4.

Outro conceito de P4 é a definição de *ações* (*actions*) para cada tabela. As ações são como funções em C, declaradas inicialmente com a instrução *action*, seguida do nome da ação e de parênteses. Em caso da ação possuir parâmetros, estes são definidos com um tipo e nome do parâmetro entre os parênteses e separados por vírgula. Caso uma ação utilize parâmetros, os valores dos parâmetros serão fornecidos em tempo de execução a partir da regra correspondente existente na tabela de correspondências. Ao implementar uma função de ação, o programador pode atribuir e modificar valores de campos do cabeçalho e de metadados. A Figura 6 apresenta um exemplo de definição de uma ação.

O último conceito de P4 é a definição do *programa de controle* (*control program*), que especifica a sequência da execução de uma tabela para a seguinte, caso necessário. Um programa desenvolvido em $P4_{16}$ implementa o fluxo de uso das tabelas de correspondência e as ações dentro da seção *apply*, através da utilização de comandos

```
action forward(egressSpec_t port) {
    standard_metadata.egress_spec = port;
}
```

Figura 6 – Exemplo de definição de uma ação em P4.

condicionais, que direcionam a utilização das tabelas. A Figura 7 mostra uma representação do fluxo de controle desejado.

```
apply {

    if(hdr.ipv4.isValid()) {
        forward.apply();
    } else {
        drop();
    }
}
```

Figura 7 – Exemplo de definição de um programa de controle em P4.

Após a análise, a tabela de verificação de origem verifica a consistência entre o pacote recebido e a porta de entrada. As tabelas posteriores no fluxo podem corresponder a esses metadados para evitar a remarcação do pacote. A tabela de comutação local é então executada. Se esta tabela "falhar", isso indica que o pacote não está destinado a um *host* conectado localmente e a ação definida como ação padrão é então executada.

Assim, a linguagem P4 foi criada com o objetivo de possibilitar a programação de aplicações de dispositivos do Plano de Dados, como comutadores, roteadores, balanceadores de carga, entre outros, afim de torná-los mais flexíveis, permitindo ao desenvolvedor definir os cabeçalhos dos protocolos utilizados e decidir como o Plano de Dados realiza o processamento dos pacotes sem a preocupação de detalhes específicos sobre o protocolo ou sobre o dispositivo alvo que executará a solução.

1.1.3 Ponto de Troca de Tráfego de Internet

A Internet que temos hoje evoluiu do *backbone* da *National Science Foundation* (NATIONAL SCIENCE FOUNDATION, 2020) americana que, através do financiamento de um esforço para conectar pesquisadores a diversos centros americanos de super-computação com o propósito de fornecer uma forma rápida e conveniente para compartilhamento de conhecimento, a *National Science Foundation Network* (NSFNET). Em 1988, o *Federal Networking Council* permitiu que o *Corporation for National Research Initiatives* (CNRI) implementasse um *gateway* para uma entidade comercial, a MCI Mail, com isso, outros provedores comerciais de e-mail obtiveram permissão para possuir conexões semelhantes, levando à interconexão de muitos serviços de e-mail até então

desconectados da Internet. Em 1991, a NSF permitiu à *Advanced Network and Services* (ANS), uma empresa sem fins lucrativos, transportar tráfego comercial através do backbone da ANSNet, que utilizava a mesma infraestrutura do backbone da NSFNET. O aumento no tráfego comercial na NSFNET levou a um grupo de fornecedores de backbone comercial (PSINet, UUNET, e CERFne) a estabelecer uma rede privada denominada *Comercial Internet eXchange* (CIX), que servia como ponto de troca de tráfego entre as entidades comerciais integrantes da NSFNET (WHEELER; O'KELLY, 1999).

Com a transição do serviço de *backbone* da NSFNET para o setor privado, por volta de 1994/1995, foram criadas quatro instalações localizadas em Washington DC, Nova Jersey, Chicago e São Francisco, denominadas *Network Access Points* (NAPs), com o objetivo de atuarem como pontos de conexão parar provedores comerciais e para garantir que a rede permanecesse conectada. Os NAPs eram instalações responsáveis pela troca de rede onde os Provedores de Serviço de Internet comerciais poderiam se conectar para realizar troca de tráfego entre si. Desde a sua criação, por volta de 1995, estes quatro NAPs foram substituídos por diversos Pontos de Troca de Tráfego de Internet modernos (90 no início de 2020) (CHATZIS; SMARAGDAKIS; FELDMANN, 2013).

Um Ponto de Troca de Tráfego de Internet é um local físico, podendo estar localizado em uma única instalação, em múltiplos locais como uma mesma cidade ou região ou mesmo distribuído em escala global, com infraestrutura de rede, de camada 2, com propósito de suportar e facilitar a troca de tráfego entre múltiplas redes de diferentes Provedores de Serviço de Internet. Cada cliente de um IXP é denominado *membro* ou *participante*. Normalmente um IXP opera como uma entidade comercial com fins lucrativos, cobrando uma mensalidade pelo acesso e realização de troca de tráfego e possivelmente cobrando uma taxa anual pela filiação ao IXP, continuando a cumprir o papel original dos NAPs de fornecer uma infraestrutura física e um ponto de encontro neutro onde os ISPs ou outras redes podem se comunicar, ou seja, trocar seu tráfego localmente (AGER et al., 2012).

A Figura 8 apresenta a arquitetura de um IXP típico com os comutadores do IXP e os roteadores de borda de cada membro.

Para que membros do IXP realizem a troca de tráfego, é necessário que estes se pareiem, ou seja, que estabeleçam uma sessão BGP entre seus respectivos roteadores de borda e apliquem políticas para a definição e exportação de rotas. Cada rota BGP é composta por diversas informações como o endereço IP de destino, o caminha a ser utilizado pelo tráfego de um pacote na rede membro e o endereço IP do roteador vizinho que receberá o tráfego. Assim que uma rota é definida, o roteador de borda cria uma entrada na tabela de encaminhamento que mapeia o endereço IP de destino para a porta de saída conectada ao IXP e para o endereço MAC de destino do roteador de borda vizinho. Dessa forma, o comutador IXP não necessita de nenhuma informação

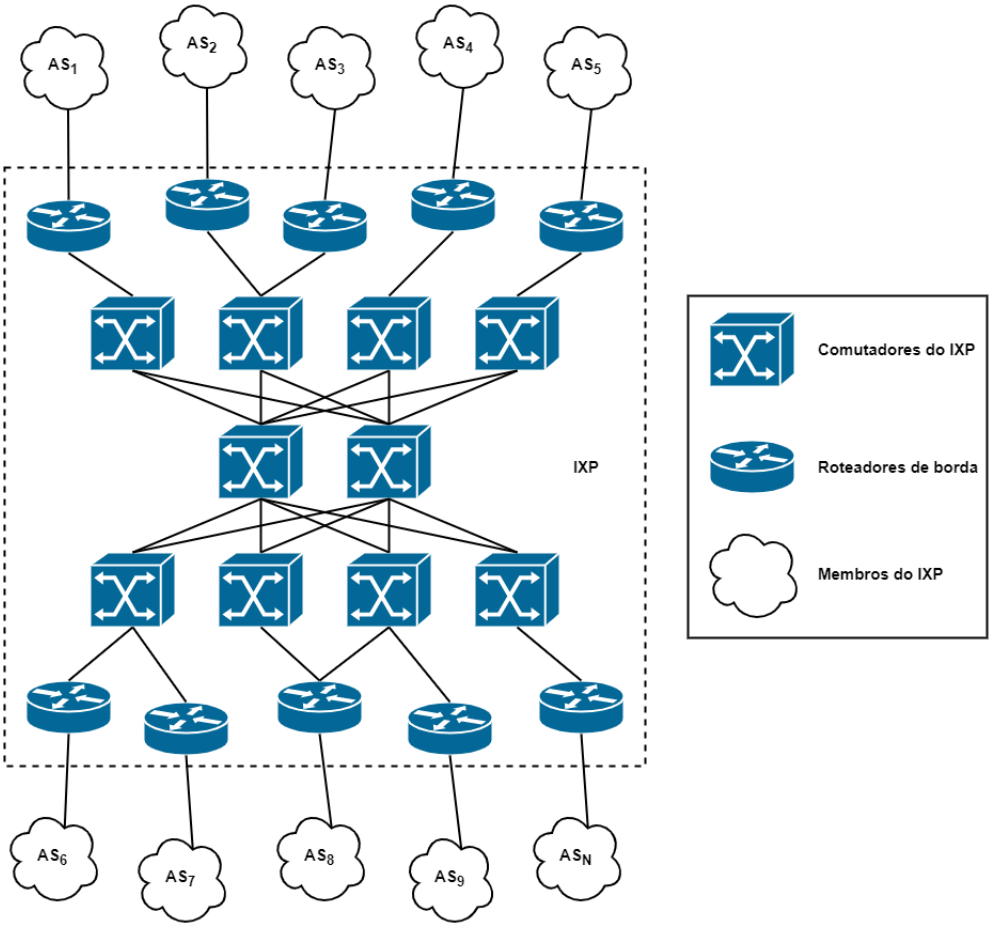


Figura 8 – Representação da arquitetura de um IXP tradicional.

sobre o endereço IP e apenas encaminha o pacote baseado no endereço MAC de destino.

Apesar da realização de sessões BGP diretamente entre pares de membros, os IXPs frequente utilizam um Servidor de Rota (*Route Server* ou RS) que atua na realização de sessões BGP onde cada membro estabelece uma sessão BGP ao roteador ao invés de realizar uma sessão BGP para cada outro roteador de borda dos membros aos quais se deseja realizar o pareamento de forma que o servidor de rota aplica as políticas de definição e exportação de rotas baseado nas políticas fornecidas por cada membro a cada outro roteador membro. Os membros do IXP podem utilizar o servidor de rotas para a realização da maioria dos pareamentos, podendo realizar pareamentos através de sessões BGP diretamente a outros membros em casos especiais, simplificando desta forma a quantidade de sessões realizadas por cada roteador membro. O servidor de rotas envia cada rota BGP para cada endereço IP de destino alvo da política de exportação do membro que anunciou a rota. Assim, o servidor de rota opera apenas como uma entidade do Plano de Controle, sem participação ativa no encaminhamento de pacotes.

A Figura 9 apresenta, a esquerda, a representação de um IXP que realiza os pareamentos entre seus membros sem a utilização de servidor de rota, enquanto a imagem da direita representa um IXP que utiliza um servidor de rota para a reali-

zação dos pareamentos.

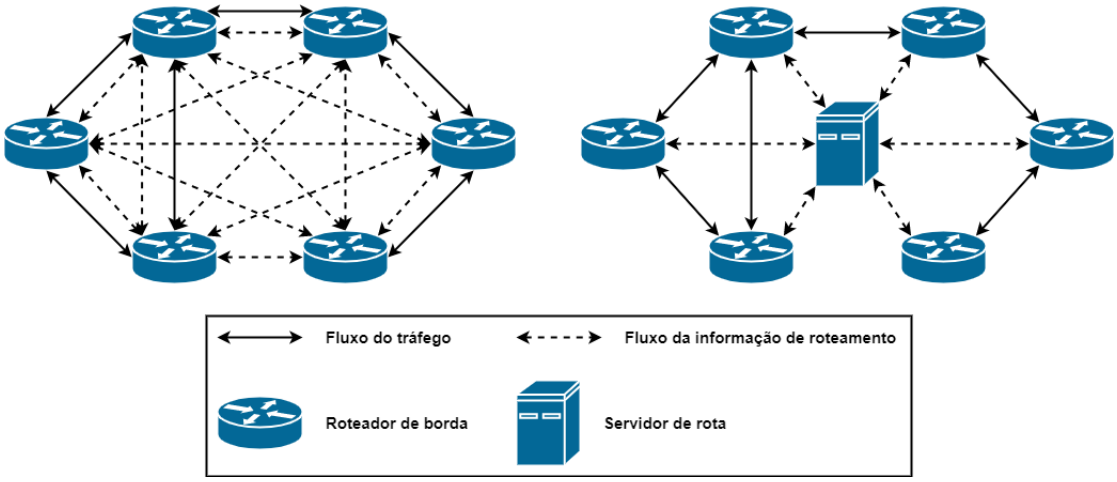


Figura 9 – Pareamento sem servidor de rota e pareamento com servidor de rota (CHATZIS; SMARAGDAKIS; FELDMANN, 2013).

1.1.4 Ponto de Troca de Tráfego de Internet Definido por Software

A princípio, um IXP se assemelha a uma SDN, com comutadores de camada 2 no Plano de Dados e o servidor de rotas como controlador. Dessa forma, o servidor de rotas se apresenta como um ponto estratégico para a aplicação de melhorias para as políticas de roteamento BGP e lógicas de decisão. Porém, o servidor de rotas apresenta limitações significativas para a arquitetura do IXP pois este não é capaz de determinar o estado de encaminhamento nos comutadores, nem de utilizar os campos de endereço IP de origem, nem o número das portas TCP/UDP, nem utilizar os bits ToS (*Type of Service*), limitando assim a aplicação de políticas mais flexíveis e robustas. As possibilidades de uso do servidor de rotas podem ir além da definição de rotas padrão, do provisionamento automático ou da gestão de sessões BGP, necessitando de suporte específico para tornar mais flexível a aplicação das políticas para troca de tráfego com o BGP, o que não poderia ser possível sem o uso de SDN. Dessa forma, os IXPs tradicionais não suportam uma forma mais flexível de aplicação de políticas do Plano de Dados, tornando altamente complexas tarefas como a aplicação de regras de segurança, a resolução de problemas, a prevenção de ataques, a implementação de contratos comerciais, o balanceamento de tráfego e o controle de congestionamento (GUPTA et al., 2014).

A utilização de *Redes Definidas por Software* podem eliminar essas limitações do roteamento entre domínios pois, os comutadores SDN podem encaminhar pacotes baseados em diferentes campos de cabeçalho de um pacote, permitindo assim a definição de políticas de encaminhamento mais flexíveis do que apenas pelo roteamento do endereço IP de destino. O controlador da SDN consegue exercer controle direto, instalando novas regras de processamento de pacotes nos dispositivos do Plano de

Dados. Contudo, apenas possuir comutadores compatíveis com SDN não soluciona a questão do roteamento entre domínios. Outra questão é a dificuldade na implantação incremental de novos projetos em uma Internet global com uma enorme base instalada de roteadores baseados em BGP.

Em Abril de 2020, o site *Packet Clearing House* (PCH) (PACKET CLEARING HOUSE, 2020) registra a existência de 556 IXPs em todo o mundo, com 28 deles localizados no Brasil. Com o aumento do tráfego da Internet, os IXPs tem aumentado a carga de tráfego, necessitando de melhores mecanismos de gestão e controle para oferecer. Com a implementação de melhorias em apenas um IXP, os benefícios seriam oferecidos a dezenas ou centenas de Provedores de Serviço de Internet membros deste IXP. Estas características tornam os Pontos de Troca de Tráfego de Internet locais propícios para a implantação de novas soluções de inovação, permitindo que os IXPs diferenciem seus serviços dos demais concorrentes de mercado. Assim, a implantação de SDN em um IXP permite que o Plano de Controle, hoje em BGP, evolua independentemente dos dispositivos de encaminhamento, trazendo controle e lógica de software para o roteamento entre domínios, então um *Ponto de Troca de Tráfego de Internet Definido por Software* pode ser uma forma de oferecer a solução para esses problemas.

Um *Ponto de Troca de Tráfego de Internet Definido por Software* (*Software Defined Internet eXchange* ou SDX) consiste de um IXP com aplicação de SDN em sua infraestrutura de rede. Assim, o IXP passa a incorporar as vantagens oferecidas pela SDN como a flexibilização do protocolo de roteamento BGP e o desenvolvimento do Plano de Controle do BGP de forma independente dos dispositivos de rede, trazendo assim o controle e a lógica via *software*, para o roteamento do IXP, reduzir a complexidade da configuração e gestão do IXP.

Para simplificar as regras nos comutadores, o SDX necessita suportar o encaminhamento padrão de pacotes entre os roteadores membros com base nas políticas de roteamento BGP, como nos IXPs tradicionais. Para permitir a definição de políticas mais sofisticadas no Plano de Dados, o controlador SDX aplica composição sequencial e/ou paralela para combinar as políticas das redes membro, em uma única política para os comutadores do SDX, modificando sintaticamente as políticas dos participantes para garantir que suas políticas não entrem em conflito. Além disso, o controlador SDX pode agregar informações auxiliares como registros de infra-estrutura de chave pública de recursos (RPKI), informações do servidor de roteamento, entre outras, que afetam as decisões de roteamento (GUPTA et al., 2015).

O controlador SDX fornece a cada membro do SDX, a abstração de um comutador dedicado que pode ser programado usando políticas de correspondência e ação para realizar o controle dos fluxos de troca de tráfego, podendo definir políticas para o tráfego de entrada e/ou saída, garantindo assim que o tráfego não seja encaminhado

para um membro vizinho que não tenha anunciado uma rota BGP para o endereço IP de destino do pacote. Cada membro executa um aplicativo de controle no controlador central para que seu roteador de borda troque mensagens BGP com o servidor de rota do SDX (GUPTA et al., 2016).

O SDX pode ter integrado em sua estrutura um servidor de rotas, de forma que os membros interajam com este servidor da mesma maneira como um servidor de rota convencional. O servidor de rotas SDX recebe as rotas anunciadas por cada roteador de borda e seleciona a melhor rota para cada endereço de cada membro. Diferente dos servidores de rota tradicionais, onde cada participante define e usa uma rota por endereço, o servidor de rota do SDX permite que cada membro encaminhe o tráfego para todas as rotas possíveis para um mesmo endereço. A Figura 10 mostra a representação da arquitetura de um SDX, onde os roteadores de borda dos membros do SDX se conectam ao controlador SDX e ao servidor de rotas.

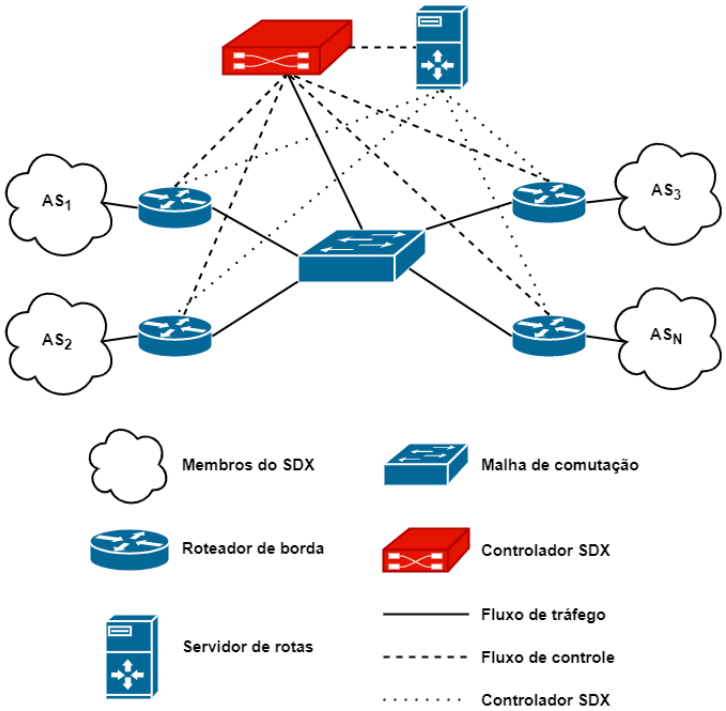


Figura 10 – Exemplo de arquitetura de um SDX.

1.2 Trabalhos Relacionados

Nesta seção, são descritos trabalhos que apresentam propostas relacionadas à coexistência entre diferentes arquiteturas de Internet, sendo apresentados com mais detalhes dois dos trabalhos mais recentes, o *NDN em Redes Locais* e o *framework Fusão de Internet do Futuro*, apresentando suas características, vantagens, benefícios e o que cada trabalho relacionado difere ou se assemelha a este trabalho.

Em pequisa realizada em 2011 sobre Internet do Futuro, são apontados como principais tópicos estudados os seguintes temas: computação em nuvem, segurança, ambientes de testes experimentais, paradigmas orientados a conteúdo ou a dados, mobilidade e computação ubíqua (PAN; PAUL; JAIN, 2011). Porém, em 2012, uma pesquisa propondo a definição de um *gateway* que realiza a conversão de requisições e respostas HTTP em mensagens de *interesse*, utilizadas por *Content Centric Networking*, foi apresentada em (WANG et al., 2012).

Em 2014, são propostos dois *frameworks*, o *Versatile ICN* (REN et al., 2014), que habilita a interoperação entre diferentes redes ICN, através de tunelamento ou tradução e o *Internames*, que define um conjunto de serviços de resolução de nomes que permite o mapeamento dos nomes entre redes IP e ICN.

1.2.1 NDN em Redes Locais

Named Data Networing (NDN) é uma arquitetura de Internet centrada em dados (*Data-Centric Network*), baseada na recuperação de dados e não na entrega de pacotes. Diferente do IP, que carrega o endereço de destino nos pacotes, os pacotes NDN carregam nomes de conteúdos, que são usados por roteadores e *hosts* NDN para validar as requisições dos usuários, chamadas de *interesse*, com os atuais objetos de conteúdo, chamados*dados*.

Os benefícios trazidos pela NDN incluem a segurança centrada em dados, distribuição de conteúdo escalável, mobilidade, redes resilientes, entre outros. Outras soluções existentes implantam NDN através de tunelamento IP, o que normalmente requer configuração manual extra nos destinos. Dessa forma, para melhor proveito das comunicações centradas em dados, sem a sobrecarga de camadas extra de protocolo, é proposta a implantação de NDN diretamente sobre Ethernet, permitindo assim a coexistência de tráfego de NDN e IP.

O NDN possui dois tipos de pacotes: o de *interesse* e o de *dados*. Um *host* NDN que solicita dados é denominado *consumidor* e o servidor de conteúdo é chamado de *produtor*. Existem três estruturas de dados no fluxo de encaminhamento NDN: o *Content Store* (CS) que funciona como um *cache* de dados, a *Pending Interest Table* (PIT), que registra as mensagens de *interesse* que já foram encaminhadas, mas ainda não receberam a mensagem de *dados* e a *Forward Information Base* (FIB), que armazena os próximos saltos para cada prefixo de nome.

Quando um *consumidor* deseja recuperar um dado, ele envia para a rede um pacote de *interesse* que carrega o nome do dado. Um nó NDN, como um roteador, um comutador ou *host*, que recebe o *interesse*, consulta inicialmente seu CS para verificar se possui os dados solicitados. Caso o dado já exista, este é enviado para a interface de

entrada por onde o *interesse* foi recebido, senão, ele verifica se existe alguma mensagem de *interesse* pendente com o mesmo nome em seu PIT. Em caso afirmativo, o nó registra uma nova entrada do *interesse* no seu PIT.

Desta forma, mensagens de *interesse* de diferentes *consumidores* que solicitam os mesmos dados são agregados. Caso a pesquisa PIT também falhar, o nó verifica seu FIB para determinar o próximo passo para o encaminhamento da mensagem de *interesse* e registra uma nova entrada no PIT. Quando o*dado* é retornado, o nó encaminha o *dado* para todos os vizinhos que solicitaram esse dado e armazena o*dado* em cache, no CS local. Então, um pacote de *dado* sempre retorna pelo caminho inverso do pacote de *interesse*.

Assim, é proposto um *Comutador de Pilha Dupla* (*Dual-Stack Switch* ou D-Switch), a ser implementado em qualquer linguagem de programação geral ou de processamento de pacotes, que é capaz de realizar o encaminhamento baseado em nomes para tráfego NDN e o encaminhamento baseado em endereços para o tráfego de IP, utilizando o campo *Ethertype* do Ethernet para identificar o tipo do pacote, definindo assim a forma de como é realizado o encaminhamento. A Figura 11 apresenta a arquitetura de um *Comutador de Pilha Dupla*.

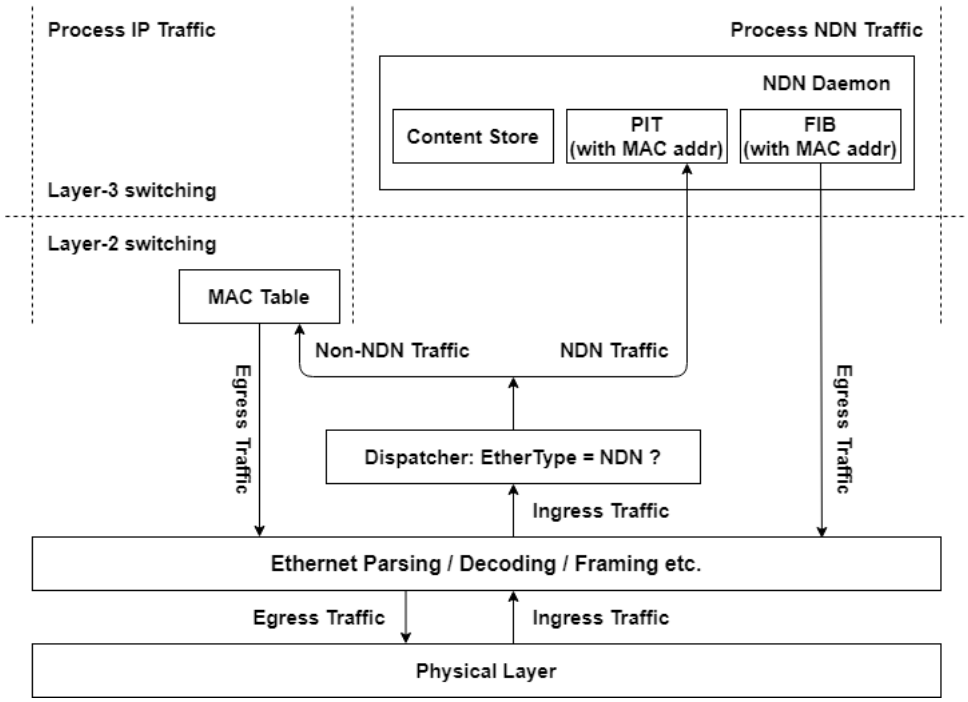


Figura 11 – Arquitetura do *Comutador de Pilha Dupla* (WU et al., 2017)

Um *Comutador de Pilha Dupla*, é um nó NDN que é inserido em uma Rede Local para realizar o encaminhamento de tráfego NDN com base nos nomes de conteúdo. O *dispatcher* e um processo que identifica o tráfego NDN verificando o campo *Ethertype* do cabeçalho de um pacote. Quando um pacote IP é recebido, o *Comutador de Pilha Dupla* se comporta como um comutador Ethernet comum, encaminhando o pacote basedo no seu

endereço MAC. Porém, quando um pacote NDN é recebido, o *Comutador de Pilha Dupla* encaminha o pacote com base no nome do conteúdo transportado no cabeçalho NDN.

O *Comutador de Pilha Dupla* também mantém o FIB, que associa prefixos de nome a interfaces de saída. Quando um pacote de *interesse* é encaminhado, o *Comutador de Pilha Dupla* busca o nome do conteúdo, contra o FIB baseado em nomes, envia o pacote pela interface registrada na entrada FIB correspondente e preenche o campo de endereço MAC de destino com o endereço MAC armazenado.

Comutadores de Pilha Dupla constroem seu FIB baseado em nome por autoaprendizado (*self-learning*). Quando um *interesse* precisa ser encaminhado e não há correspondência no FIB, o *interesse* é enviado para todas as interfaces de saída, exceto a interface de rede por onde o *interesse* chegou. Quando um pacote de *dados* é recebido, seu prefixo de nome, interface de rede e MAC de origem são registrados em uma nova entrada no FIB, de forma que pacotes de *interesses* futuros sob o mesmo prefixo de nome sejam encaminhados sem inundar os vizinhos. Um *consumidor* não precisa conhecer o MAC de destino do *produtor*, podendo simplesmente enviar pacotes de *interesse* para o grupo *ALL-NDN*, e os *Comutadores de Pilha Dupla* realizarão o encaminhamento dos pacotes de *interesse*, com base no FIB preenchido por autoaprendizagem, para *produtores* de conteúdo sem inundar segmentos da Rede Local que não têm o conteúdo.

Por fim, são observadas várias semelhanças do trabalho *NDN em Redes Locais* com a proposta apresentada, a definição de um comutador programável para a realização do encaminhamento de pacotes de diferentes arquiteturas de Internet e a utilização do campo *Ethertype* do Ethernet para a identificação do tipo de pacote, nos comutadores das duas propostas.

Outra semelhança importante, é que ambas as soluções possibilitam a coexistência de novas arquiteturas de Internet com a arquitetura atual, mas a comunicação é realizada apenas entre entidades uma mesma arquitetura, não havendo interoperabilidade entre as arquiteturas. Assim como a proposta desta dissertação não realiza a tradução ou conversão de pacotes entre arquiteturas, o mesmo ocorre no *NDN em Redes Locais*, onde um *hosts* NDN apenas se comunicam com outros *hosts* NDN.

Um ponto que faltou ser mencionado pelo trabalho *NDN em Redes Locais*, é sobre a descrição de como os cabeçalhos NDN de *interesse* e de *dado* foram adaptados no *Comutador de Pilha Dupla* e como foram encapsulados dentro do Ethernet, pois a adaptação dos campos TLV (Type-Length-Value) dos cabeçalhos NDN pode ser realmente trabalhosa. Também não há nenhum cometário sobre a utilização de controladores para a realização do controle das regras de encaminhamento no *Comutador de Pilha Dupla*.

1.2.2 Fusão de Internet do Futuro

Em uma rede heterogênea, onde múltiplas arquiteturas de rede coexistem sob uma mesma infraestrutura de rede, o projeto de cada arquitetura de rede é baseado em diferentes orientações de projeto, protocolos, paradigmas de comunicação e formas de endereçamento. Estas diferenças dificultam a interoperabilidade e por consequência, a comunicação entre entidades em arquiteturas de rede diferentes, o que motivou a criação de uma entidade de interoperabilidade que serve de intermediário para a comunicação entre as entidades destino.

Para uma Internet do Futuro heterogênea ser funcional, são necessários mecanismos que possibilitem a interoperabilidade entre diferentes arquiteturas de Internet do Futuro, permitindo assim acesso aos recursos independente da arquitetura de rede das entidades envolvidas. Com esta finalidade, é apresentada a definição de um conjunto de mecanismos para a realização do mapeamento de recursos, tornando-os acessíveis entre diferentes arquiteturas. Desta forma, é proposto um *framework* flexível chamado Fusão de Internet do Futuro (*Future Internet Fusion* ou FIFu), que visa possibilitar a coexistência de Arquiteturas de Internet do Futuro, proporcionando ao mesmo tempo, compatibilidade com a Internet atual, permitindo assim o desenvolvimento e evolução das arquiteturas de Internet, de forma independente (GUIMARãES et al., 2019).

Inspirada no conceito de SDN, a *FIFu* separa as operações de conversão de mensagens do processo de controle e configuração. Assim o *framework* proposto consiste de uma estrutura em duas camadas, onde as entidades da camada de adaptação interconectam diferentes ilhas de redes de Internet do Futuro sob o controle de entidades logicamente centralizadas que compõem a camada de inteligência.

Para atingir esse objetivo, são definidas entidades de interoperação, denominadas Pontos de Troca de Tráfego de Internet do Futuro (*Future Internet eXchange Points* ou FIXPs), que compõem a camada de adaptação, convertendo mensagens entre diferentes arquiteturas de rede e Controladores de Internet do Futuro (*Future Internet Controllers* ou FICs), que atuam na camada de inteligência, sendo responsáveis por configurar, gerenciar e dar suporte à operação dos FIXPs, além de armazenar, gerenciar e manter os identificadores de recursos de cada arquitetura de rede.

No FIFu, os identificadores de recursos são representados na forma de URIs (*Uniform Resource Identifiers*), que fornecem toda a informação necessária sobre como acessar os recursos identificados. Para que recursos implantados em uma determinada arquitetura de rede estejam disponíveis em uma arquitetura diferente, eles devem ser acessíveis usando endereços compatíveis com os protocolos suportados por cada arquitetura de rede, constituindo uma representação virtual do recurso real. O recurso virtual pode ser visto como um mapeamento de um determinado recurso nas arqui-

teturas de rede, fornecendo assim uma abstração do recurso real em cada arquitetura. A Figura 12 apresenta uma representação do *framework* FIFu com suas duas camadas, seus componentes e os seus relacionamentos.

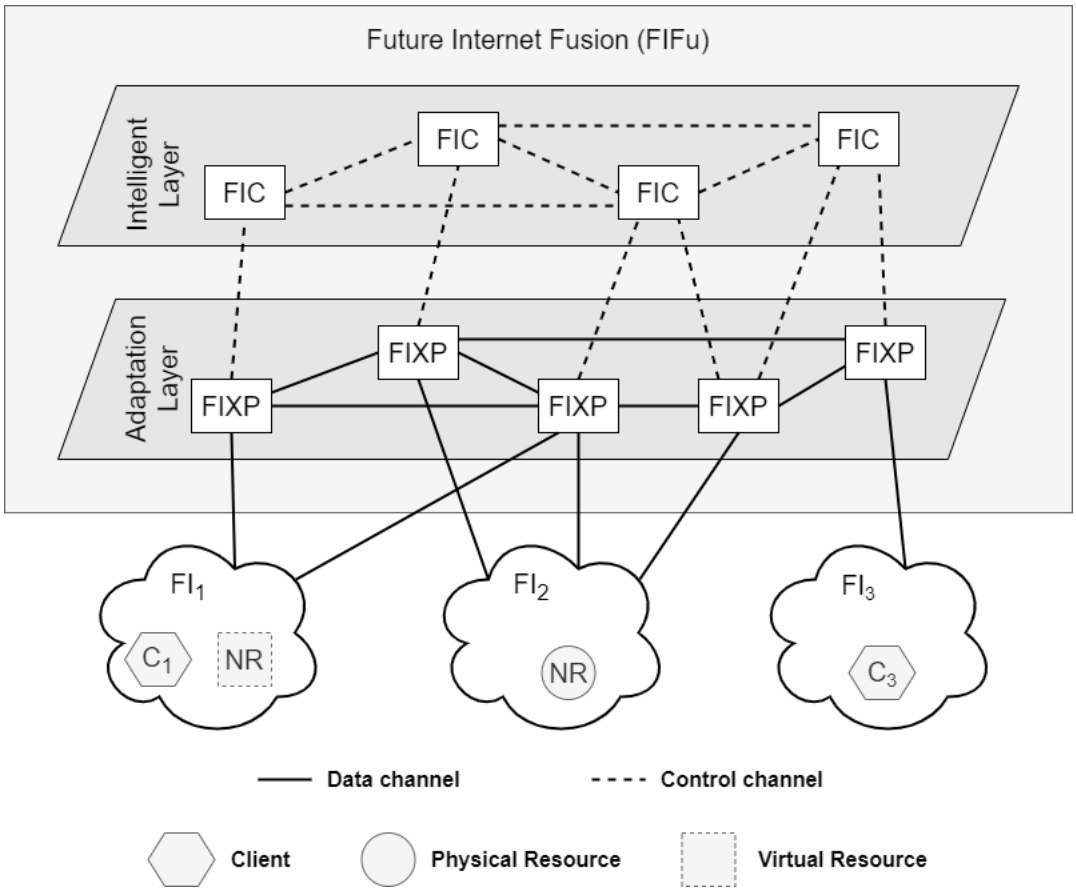


Figura 12 – Visão geral do modelo do framework FIFu (GUIMARãES et al., 2019).

Os FIXPs são entidades intermediárias na comunicação entre diferentes arquiteturas de rede, responsáveis por fornecer a interoperabilidade entre duas arquiteturas de rede diferentes. Atuando como *gateways*, emulam o destino de comunicação da arquitetura de rede original na arquitetura estrangeira e vice-versa, se comportando simultaneamente como destino e origem das mensagens.

Como parte do processo da camada de adaptação, as entidades FIXP convertem as mensagens recebidas, tornando-as compatíveis com a arquitetura de rede destino. Para reconhecer os diferentes protocolos de cada arquitetura de rede suportada, os FIXPs são configurados com informações relacionadas ao formato do pacote de cada protocolo, incluindo o significado semântico de cada campo. Estas informações são atualizadas sem a necessidade de interrupção no serviço ou substituição de equipamento, sendo esse processo realizado pelas entidades da camada de inteligência que, para cada protocolo suportado, inicia uma instância que permitirá ao FIXP agir como ponto de origem e de destino em relação a um determinado protocolo.

A camada de inteligência, composta pelos FICs, controla e suporta a operação

FIXP, armazenando uma lista dos FIXPs implantados pela Internet e mantendo um repositório de mapeamentos entre identificadores de recursos em diferentes arquiteturas. Esta abordagem torna mais flexível a adição de novas arquiteturas de rede e como estas são interconectadas, não sendo necessária alteração manual da configuração de cada FIXP.

Os FICs são responsáveis por controlar, auxiliar e sincronizar a operação da camada de adaptação, realizando o monitoramento de recursos e a configuração dos destinos do protocolo da arquitetura suportada por cada FIXP. Um FIC também é capaz de identificar os recursos de FIXPs subjacentes. Essas informações podem ser obtidas por meio de solicitações realizadas a cada FIXP ou através do recebimento de eventos configurados no FIXP. Os FICs também atuam como um sistema de mapeamento distribuído hierárquico, gerenciando os identificadores de recursos em cada arquitetura de rede, armazenando e gerenciando o mapeamento entre URIs em diferentes arquiteturas e fornecendo serviço recuperação destas informações quando solicitado.

A organização hierárquica lógica dos FICs é dividida em 3 níveis principais. O nível mais alto é o nível *Raiz*, sendo consultado para obter os FICs existentes no nível seguinte. O segundo nível é o nível de *Arquitetura de Rede*, que é responsável pela criação e manutenção de URIs externas pertencentes ao escopo da sua arquitetura de rede e pela distribuição destes URIs aos FICs do próximo nível abaixo. O terceiro nível é o nível de *URI*, responsável por manter os mapeamentos entre URIs em diferentes arquiteturas de rede. Diferentes estratégias de armazenamento podem ser usadas, como mecanismos de resolução hierárquica ou Tabelas de Hash Distribuídas (*Distributed Hash Tables* ou DHT).

Embora os mapeamentos possam ser registrados pelo provedor de recursos, os URIs na arquitetura de rede externa também podem ser gerados pelo FIC e mapeados para seu correspondente na rede original sempre que necessário. Após o recebimento de solicitações de resolução para uma determinada URI, se um mapeamento não existir, o FIC do nível de Arquitetura de Rede responsável pela arquitetura externa gera automaticamente a URI para ser usado na arquitetura externa.

O URI gerado é então entregue ao solicitante e armazenado nos FICs apropriados no nível de URI, podendo os FIXPs ou outros FICs resolver os URIs de arquiteturas de rede externa em sua forma original e vice-versa. Esta resolução pode ser recursiva ou iterativa. Em consultas recursivas, uma solicitação de resolução é enviada ao FIC, que responde com o mapeamento solicitado ou uma mensagem de erro. Portanto, se o FIC não possui as informações necessárias, solicita recursivamente a outros FICs até obter as informações desejadas ou até que a consulta falhe. Em consultas iterativas, se o FIC não possui as informações necessárias, ele responde com uma referência a outro FIC que pode ter as informações.

Para solicitar a resolução de um determinado URI externo em sua forma original,

o FIXP aciona uma mensagem de resolução contendo o URI externo em direção ao seu FIC local, que solicita a resolução para o FIC da arquitetura de rede. Se o FIC no nível de *Arquitetura de Rede* não for conhecido, a solicitação será encaminhada para um FIC *Raiz* que retornará a referência a ele. Nesse caso, o FIC de *Arquitetura de Rede* é selecionado com base no esquema do URI externo.

Para suportar os procedimentos de controle entre as entidades que compõem a solução *FIFu*, definiu-se um protocolo de controle, denominado *Protocolo FIFu*, que permite a comunicação entre FIC e FIXP ou entre FICs.

O objetivo deste protocolo é permitir o gerenciamento das instâncias do FIXP, incluindo o monitoramento e a configuração de vários parâmetros operacionais e das conexões entre eles, permitindo também a resolução de URIs externos em sua forma original ou vice-versa.

Como este trabalho, o *Framework* FIFu também se propõe a oferecer uma solução baseada nos princípios de SDN, permitindo assim a coexistência entre diferentes arquiteturas de Internet sob a infraestrutura da rede atual, possibilitando a criação de uma rede heterogênea formada por múltiplas arquiteturas de rede. Entretanto, o *FIFu* e a proposta deste trabalho divergem no objetivo final de suas propostas, enquanto o FIFu visa realizar a interconexão de ilhas de redes de arquiteturas diferentes através da criação de um *framework* que possibilite a interoperabilidade entre arquiteturas, a proposta apresentada tem como objetivo oferecer uma solução para IXPs que fornece ao Plano de Controle a capacidade de suportar diferentes arquiteturas de Internet e com um Plano de Dados composto por comutadores definidos por *software*, que possibilita o encaminhamento de pacotes entre entidades de mesma arquitetura, dentre essas diferentes arquiteturas. Assim, a proposta também proporciona a coexistência de diferentes arquiteturas de rede sob a infraestrutura de rede atual, possibilitando porém apenas a comunicação entre entidades de uma mesma arquitetura. Vale a pena ressaltar que, diferente do termo *FIXP* apresentado pelo *FIFu*, o termo *FIXP* apresentado neste trabalho realmente visa a implementação de uma solução em um IXP, permitindo a este passar a realizar o controle de tráfego de novas Arquiteturas de Internet do Futuro, tornando o IXP um Ponto de Troca de Tráfego de Internet do Futuro, portanto um *Future Internet Exchange Point* ou FIXP.

2 Arquiteturas de Internet do Futuro

Neste capítulo é apresentado o que é uma Arquitetura de Internet do Futuro e qual o objetivo do estudo de novas arquiteturas de Internet. Também são descritas as características e as estruturas das duas arquiteturas de Internet do Futuro abordadas e pela solução proposta, a NovaGenesis e a ETArch.

2.1 Arquiteturas de Internet do Futuro

Como forma de solucionar as limitações que surgiram com a evolução da Internet, pesquisas baseadas em novos princípios, com o objetivo de propor novas arquiteturas para uma nova Internet foram iniciadas em diversos países, incluindo Brasil, Estados Unidos, União Europeia, Japão e China. O paradigma *clean-slate* têm sido utilizado pelos pesquisadores dessas iniciativas como modelo para a especificação de novas arquiteturas de Internet, diferente da proposta incremental de solução, onde mudanças são adicionadas através de pacotes de correção na arquitetura ou aos seus protocolos. Este modelo propõem a definição e o projeto de uma nova arquitetura de Internet a partir do zero, não levando em consideração as restrições e limitações da arquitetura atual, proporcionando um funcionamento similar ao da arquitetura atual, utilizando para isso novas tecnologias e pesquisas na área para a definição de novas formas de solução para as demandas atuais. Surgem então as pesquisas de Arquiteturas de Internet do Futuro.

Uma Arquitetura de Internet do Futuro tem como objetivo, propor o redesenho de toda a arquitetura e não apenas a implementação de melhoria ou correção de pontos específicos. A aplicação da metodologia *clean-slate* em um tópico específico assume que as demais demandas da arquitetura possam ser corrigidas e inalteradas, mas a aplicação de diferentes soluções *clean-slate* aplicadas em diferentes demandas não implica necessariamente na criação de uma nova arquitetura de Internet. Contudo, a maioria dos projetos *clean-slate* anteriores se focavam em tópicos individuais. Outro objetivo importante da pesquisa de Arquiteturas de Internet do Futuro é a possibilidade de utilização de bancos de testes, desta forma, as novas arquiteturas podem ser testadas, validadas e aprimoradas executando-as nestes ambientes de teste antes de serem implantadas no mundo real. Nos últimos anos, as pesquisas em Internet do Futuro ganharam enorme impulso, como evidenciado pelo grande número de projetos de pesquisa nessa área (PAN; PAUL; JAIN, 2011).

Dentre os diversos tópicos que envolvem a criação de uma Arquitetura de Internet, destacam-se como os mais abordados pelos pesquisadores o estudo de *paradigmas orientados a dados ou conteúdo*, com o objetivo de criar novos protocolos que possibilitem

a distribuição de conteúdo e dados, as pesquisas voltadas à *mobilidade e acesso onipresente a redes*, pois, a Internet tem vivenciado a mudança de computação baseada em computadores pessoais para dispositivos móveis, levantando assim questões de escalabilidade, segurança e privacidade da informação, trabalhos envolvendo *arquiteturas centradas em computação em nuvem*, que demandam a criação de serviços e aplicações que proporcionem a distribuição de recursos em escala global e a *Segurança*, para definir e criar diferentes níveis de segurança através de criptografia, autenticação e autorização, garantindo assim a confiança entre entidades.

Assim, as pesquisas de novas Arquiteturas de Internet do Futuro tem como objetivo superar as limitações estruturais da arquitetura de Internet atual, usando para este fim o paradigma *clean-slate* nas pesquisa de projetos em novas arquiteturas de Internet do Futuro, aliado a possibilidade de realização de testes em bancos de testes, favorecendo assim a definição e implantação das novas arquiteturas, permitindo um crescimento escalável e evolucionário destas novas arquiteturas de Internet.

As iniciativas de pequisas em Arquiteturas de Internet do Futuro nos Estados unidos têm sido administradas pela *National Science Foundation* (NSF) (NATIONAL SCIENCE FOUNDATION, 2020). O programa *Future Internet Architecture* (FIA) (NSF FIA PROJECT, 2020) da NSF é baseado no programa anterior, *Future Internet Design* (FIND) (NSF NETS FIND INITIATIVE, 2020) que financiou aproximadamente cinquenta projetos de pesquisa relacionados a Internet do Futuro, sendo o FIA a continuação das propostas em novas arquiteturas, financiando quatro projetos de Arquitetura de Internet do Futuro, o *Named Data Networking* (NDN), o *MobilityFirst*, o *NEBULA* e o *Expressive Internet Architecture* (XIA).

Outro programa financiado pela NSF é o *Global Environment for Network Innovations* (GENI) (GLOBAL ENVIRONMENT FOR NETWORKING INNOVATIONS, 2020). Iniciado em 2005, o GENI é um projeto focado em prover um banco de testes de escala global para testes e validações de Arquiteturas de Internet do Futuro e têm atraído o interesse e participação acadêmica e da industria. O GENI se diferencia de outros bancos de teste por ser uma instalação de proposito geral de larga escala, que a princípio não impõem limites nas validações de arquiteturas, serviços e aplicações, permitindo a projetos *clean-slate* a realização de testes em condições reais com usuários reais em condições reais. Para isto o GENI disponibiliza múltiplas fatias virtualizadas para compartilhamento de recursos para a realização dos experimentos.

A União Européia também inicializou um conjunto de projetos de pesquisa em Arquiteturas de Internet do Futuro denominados *Framework Projects*. Estes projetos visam incentivar e aprimorar as atividades de pesquisa, desenvolvimento e inovação em diversos setores de ciência em geral, tanto no setor acadêmico como no setor industrial europeu, para o desenvolvimento de novas tecnologias. A partir do *Framework*

Project 6 que financiou projetos de 2002 a 2006, iniciativas de pesquisas em Internet do Futuro começam a ser apresentadas, como o desenvolvimento do banco de testes *One Lab* (ONE LAB, 2020), uma instalação experimental com o objetivo de realizar experimentos e testes de validação de novas arquiteturas. O *Framework Project 7*, financiou aproximadamente cento e cinquenta projetos entre 2007 a 2013, sendo que estes projetos abrangiam redes do futuro, Internet de serviço, computação em nuvem, confiabilidade de informação, tecnologias de comunicação, mídias em rede e sistemas de busca. Dentre as diversas propostas de pesquisa de Internet do Futuro, podem ser citadas como as mais representativas deste programa a Arquitetura de Internet do Futuro *4WARD* e o banco de testes *Future Internet Research and Experimentation* (FIRE), podendo ser comparado como a versão européia do GENI americano. Iniciado em 2006 com o FP6, o FIRE foi a continuação do projeto *Gigabit European Academic Networking Technology* (GEANT), também tendo participação dos setores acadêmico e industrial e tendo como seu maior objetivo a unificação de diferentes bancos de testes europeus por um controle centralizado sob um conjunto comum de objetivos.

No Japão, houve ampla participação em projetos americanos e europeus relacionados a pesquisas em Arquiteturas de Internet do Futuro, como no projeto norte-americano *Planet Lab* (PLANET LAB, 2020) e o G-Lab (GERMAN LAB, 2020) alemão. O programa de pesquisa japonês em Arquiteturas de Internet do Futuro é denominado *New Generation Network* (NWGN), patrocinado pelo *Instituto Nacional Japonês da Informação e Tecnologia da Comunicação* (NICT), considerando as arquiteturas *clean-slate* como uma nova geração de soluções, enquanto que a evolução da arquitetura IP é denominada de próxima geração (*Next Genereation Network* ou NXGN). Assim como nos Estados Unidos e na União Européia, os projetos NWGN consistem de diversos subprojetos com colaboração acadêmica e da industria japonesa em projetos de novas arquiteturas, bancos de testes, laboratórios de virtualização, comunicação sem fio, redes orientadas a serviço, mobilidade em redes virtualizadas e computação verde. Dentre os diversos projetos propostos se destacam os projetos *AKARI* (HARAI, 2009), que é o maior projeto de pesquisa de novas arquiteturas do Japão e os projetos *Japan Gigabit Network 2 plus* (JGN2 plus) e *Japan Gibabit Network Extreme* (JGN-X), que são pesquisas de bancos de testes, como o GENI e o FIRE.

Na China, os projetos de pesquisa em Internet do Futuro também foram desenvolvidos e financiados por um conjunto de programas, o *Programa 863* de 1986 a 2016, o *Programa 973* de 1997 e o *12º Projeto de Planejamento de 5 anos* de 1011 a 1015, administrados pelo *Ministério da Ciência e Tecnologia Chinês* (MoST), tendo sido desenvolvidos diversos projetos como o *New Generation Trustworthy Networks* (NGTN) de 2007 a 2010, o *New Generation Network Architectures* (NGNA) de 2009 a 2013, o *Future Internet Architectures* de 2011 a 2015 e o banco de testes *Chine Next Generation Internet* (CNGI), baseado no projetos de bancos de testes anteriores *Chinese Education and Research Network*

Projetos de Arquitetura de Internet do Futuro	
Projeto	Referência
Data Oriented Network Architecture	(KOPONEN et al., 2007)
Node Identity Internetworking Architecture	(AHLGREN et al., 2006)
Entity Title Architecture	(SILVA et al., 2012)
NovaGenesis	(ALBERTI et al., 2014)
Recursive InterNetwork Architecture	(MATTA; BOSTON, 2010)
eXpressive Internet Architecture	(ANAND et al., 2011)
NEBULA Future Internet Architecture	(ANDERSON et al., 2010)
Named Data Networking	(ZHANG et al., 2010)
Mobility First Future Internet Architecture	(SESKAR et al., 2011)
4WARD Future Internet Architecture	(NIEBERT et al., 2008)
TurfNet Architecture	(SCHMID et al., 2004)
IP Next Layer	(FRANCIS; GUMMADI, 2001)
Rendezvous, Topology, Forwarding and physical Media Architecture	(SÄRELÄ; RINTA-AHO; TARKOMA, 2008)
TRIAD Internet Architecture	(CHERITON; GRITTER, 2000)
Plutarch Internet Architecture	(CROWCROFT et al., 2003)
FAR Architecture	(CLARK et al., 2003)
SILO Architecture	(DUTTA et al., 2007)
CoLoR Architecture	(LUO et al., 2014)
New Interdomain Routing Architecture	(YANG; CLARK; BERGER, 2007)
Accountable Internet Protocol	(ANDERSEN et al., 2008)
Scalable and Adaptive Internet Solutions	(EDWALL; TREMBLAY, 2011)
CONET Architecture	(DETTI et al., 2011)
MIRAI	(WU; MIZUNO; HAVINGA, 2002)
Global Virtualization Architecture	(MARTINEZ-JULIA; SKARMETA; GALIS, 2013)
Smart Identifier Network	(ZHANG et al., 2016)

Tabela 1 – Exemplos de projetos de Arquiteturas de Internet do Futuro.

(CERNET) e *Chinese Science and Technology Network* (CSTNET).

A Tabela 1 apresenta a listagem de algumas outras iniciativas de projetos de pesquisa em Arquiteturas de Internet do Futuro propostos ao longo destes anos ao redor do mundo.

2.2 ETArch

A *ETArch* (*Entity Title Architecture*) (SILVA et al., 2012), é uma Arquitetura de Internet do Futuro, proposta em 2012, baseada no paradigma *clean-slate* e no conceito de SDN, com a separação do Plano de Controle do Plano de Dados, sendo uma implementação do *modelo Título Entidade*, onde a nomeação e o endereçamento são baseados em uma forma independente de topologia denominada *título* (*title*), que identifica uni-

camente uma entidade. Outra característica importante para a *ETArch* é a definição de um canal comum que possibilita a comunicação entre múltiplas entidades, denominado *workspace*. Diversas universidades têm trabalhado com a *ETArch* de forma a adicionar extensões à arquitetura, como mobilidade, *multicast*, qualidade de serviço, e roteamento.

Para a *ETArch*, uma entidade pode ser definida como qualquer dispositivo ou processo que possua a capacidade de se comunicar, disponibilizando ou trocando informações com outras entidades, como computadores, dispositivos móveis, sensores ou aplicações. Um título pode ser interpretado como sendo um identificador único que é relacionado a cada entidade. A comunicação entre entidades na *ETArch* ocorre através da criação de *workspaces* nos domínios, por uma entidade que deseja disponibilizar informações ou realizar a troca de informações com outras entidades. Um *workspace* é um canal lógico definido por uma entidade onde outras entidades que desejem acessar ou trocar informações com essa entidade necessitam se registrar para participar da comunicação com as demais entidades registradas. O comportamento de um *workspace* é inspirado diretamente pela tecnologia *multicast*, onde os dados e informações são enviados uma vez pela entidade fonte ao *workspace* e este se encarrega de redistribuir os dados e informações a todas as entidades associadas a este *workspace*.

Um componente chave da estrutura da *ETArch* é o *Domain Title System* (DTS), que lida com todos os aspectos de controle da rede. O DTS é um sistema distribuído pela rede, composto por *Domain Title System Agents* (DTSA), ou seja, serviços agentes que são responsáveis por manter a informação sobre as entidades registradas no domínio e no *workspace* nos quais estas entidades estão inscritas, se focando na configuração dos dispositivos de rede para possibilitar a implementação dos *workspaces* e para permitir que os dados e informações sejam encaminhados para cada entidade inscrita. Um DTSA também é responsável pela comunicação entre DTSAs, utilizando Openflow para controlar as entidades da rede.

ETArch possui dois protocolos principais que utilizam o modelo requisição/resposta. O primeiro é o *Entity Title Control Protocol* (ETCP), que é utilizado na comunicação entre entidades e o DTSA. O segundo é o *Domain Title System Control Protocol* (DTSCP), que é responsável pela comunicação entre DTSAs dentro de um mesmo DTS. A Tabela 2 descreve as operações do ETCP, enquanto a Tabela 3 apresenta as operações do DTSCP.

A *ETArch* utiliza a família de protocolos IEEE 802 para a formação dos pacotes de dados, possibilitando implantar a arquitetura em dispositivos usando acesso com e sem fio. Na proposta apresentada por este trabalho, os pacotes *ETArch* são formados utilizando o quadro 802.3, onde, o campo *Ethertype* é configurado como valor 0x0880 para indicar que a arquitetura do pacote é ETArch, o campo MAC de destino do quadro 802.3 contém a *hash* do título do *workspace* de destino, enquanto no campo de carga útil *payload* do quadro são encaminhados os campos do cabeçalho do pacote ETArch,

Entity Title Control Protocol	
Operação	Descrição
ENTITY_REGISTER	Registra uma entidade no DTS. Para ser registrada, uma entidade deve apresentar seu título, capacidades e requisitos de comunicação.
WORKSPACE_CREATE	Cria um *workspace* localmente em um DTSA, se o *workspace* possuir acesso público, o DTSA anunciará o *workspace* inserindo uma entrada no Banco de Dados de *workspace*.
WORKSPACE_ATTACH	Anexa uma entidade a um *workspace* e atualiza as portas de saída para incluir a nova entidade.
ENTITY_UNREGISTER	Remove uma entidade do DTS e atualiza as tabelas de fluxo.
WORKSPACE_DEATTACH	Remove uma entidade do *workspace* e atualiza as tabelas de fluxo.
WORKSPACE_DELETE	Deleta o *worspace* e realiza a limpeza necessária no elemento de rede do DTSA.

Tabela 2 – Operações do *Entity Title Control Protocol*.

Domain Title System Control Protocol	
Operação	Descrição
WORKSPACE_LOOKUP	Enviado por um DTSA para outro DTSA.
WORKSPACE_ADVERTISE	Insere, deleta ou atualiza o Banco de Dados de *workspace*.
DTS_MESSAGE	Realiza a comunicação entre diferentes DTSAs dentro de um mesmo DTS.

Tabela 3 – Operações do *Domain Title System Control Protocol*.

que é formado por um campo de tamanho, que indica a quantidade de *bytes* utilizados pela primitiva de controle, uma primitiva de controle (*Control Primitive*), formada pelos campos tipo, que identifica o tipo da primitiva e o campo identificador, que é um valor inteiro sequencial utilizado para identificar as primitivas enviadas, um campo de tamanho que indica a quantidade de *bytes* utilizados pela primitiva e por fim a primitiva (*Primitive*), que possui as informações. A representação do pacote ETArch pode ser visualizada na Figura 13.

2.3 NovaGenesis

NovaGenesis (NG) é uma Arquitetura de Internet do Futuro, apresentada em 2008 com a proposta de utilizar as melhores tecnologias atuais, visando a criação de uma arquitetura de informação convergente definida por serviços identificados por

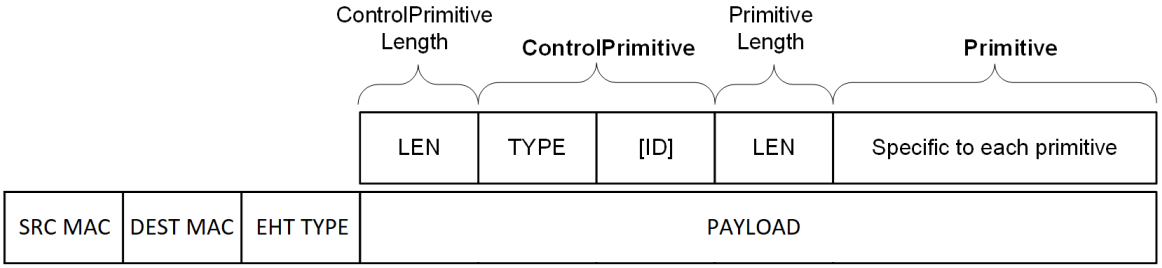


Figura 13 – Estrutura de um pacote ETArch.

nomes, que integram a troca, o armazenamento e o processamento de dados. Para isso, NG propõem o conceito de protocolos implementados como serviço (*Protocols-Implemented-as-a-Service* ou PIaaS), que são utilizados por entidades que implementam o padrão *publish/subscribe*.

A NG baseia-se em quatro conceitos principais. A *nomeação, nomes auto-verificáveis, resolução de nomes e cache de rede*, NG trabalha com nomes de linguagem natural e nomes simples. Um nome em linguagem natural (*Natural Language Name* ou NLN) é um nome que possui significado para as pessoas, já um nome simples (*Flat Name*) é obtido como o resultado da aplicação de um NLN através da função *hash* MurMurHash 3, gerando um nome auto-verificável (*Self-Verifying Name* ou SVN). A NG faz uso de NLNes e SVNes na resolução de nomes, para assim identificar entidades alvos de comunicação. A *identificação e localização*, um identificador é um nome que especifica unicamente uma entidade entre outras, enquanto um localizador é um nome que indica uma posição em que uma entidade pode ser encontrada, denotando uma noção de distância entre as entidades de um mesmo espaço de nomes. Os *serviços e contratos*, um serviço é qualquer entidade conectada à arquitetura que visa processar, trocar ou armazenar informações, um contrato é o nome dado a um contrato de nível de serviço (SLA) entre serviços, que define os limites, responsabilidades, cláusulas a serem respeitadas, critérios para conclusão e outros aspectos. Por fim, *Proxies, Gateways e Controladores*, um *proxy* é um serviço que representa outras entidades para composição dinâmica de recursos, um *gateway* é um tradutor ou um ponto de entrada/saída entre diferentes tecnologias e um controlador é um serviço que implementa e executa a tomada de decisão em relação às configurações de recursos físicos ou outros serviços, utilizando assim do conceito de SDN em empregar controladores de software para configurar outras funcionalidades que não o encaminhamento de quadros.

As mensagens NG podem ser encapsuladas no quadro IEEE 802.3, neste caso, o campo *Ethertype* é configurado com o valor 0x1234 para indicar que a arquitetura do pacote é NG, enquanto a *camada de adaptação* e a *mensagem* da NG são encaminhado no campo de carga útil (*payload*) do quadro.

A *camada de adaptação* é formada por 4 campos. O primeiro campo, os *bytes*

reservados, são formado por 4 *bytes*, onde os dois primeiros não são utilizados, o terceiro é utilizado pelo comutador FIXP para informar à NG a porta de entrada do pacote e o quarto *byte* indica se o fragmento atual é o último fragmento do pacote. O valor 0x46 aponta o último fragmento, enquanto o valor 0x45 indica que ainda existem fragmentos a ser encaminhados. O segundo campo é o *identificador de mensagem*, que apresenta uma *hash* de 32 *bits*, randomicamente gerada, que identifica cada mensagem. O terceiro campo é a *sequência de fragmentos*, de 4 *bytes*, este campo informa o número do fragmento atual do pacote. Tanto o primeiro fragmento de um pacote fragmentado como um pacote não fragmentado apresentam valor igual a 0x00000000. E o quarto campo é o *tamanho da mensagem*, de 8 bytes, que informa o tamanho em *bytes* da *mensagem NovaGenesis*. A Figura 14 apresenta uma representação dos campos do pacote da NovaGenesis.

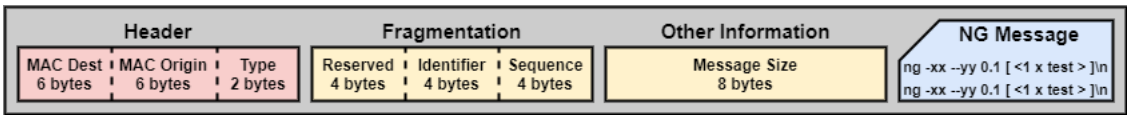


Figura 14 – Estrutura de um pacote NovaGenesis (ALBERTI et al., 2018).

Uma mensagem NG é formada por *linhas de comando* e caso necessário, de um campo de carga de dados.

Uma *linha de comando* é um texto em ASCII formado em uma única linha, com parâmetros separados por espaços em branco, formada por um cabeçalho composto por um *identificador da arquitetura*, formado pelas letras *ng*, pelo *nome do comando*, precedido por um sinal de menos, pelas *alternativas de comando*, precedidas por dois sinais de menos e pela *versão do comando* e por uma seção de argumentos, encapsulada por colchetes. Cada argumento é encapsulados pelos sinais de menor e maior, e possuem obrigatoriamente 2 parâmetros e um ou mais elementos. O primeiro parâmetro indica a quantidade de elementos do argumento, o segundo parâmetro indica o tipo dos elementos e por fim tem-se a quantidade de elementos definidos pelo primeiro parâmetro.

A primeira linha de comando de um pacote, e apenas a primeira linha de comando do primeiro fragmento, em caso de um pacote fragmentado, é responsável pelo transporte das informações do emissor e o destinatário da mensagem. Esta linha de comando recebe o nome de *linha de comando de roteamento*, sendo composta por três argumentos. O primeiro é formado pelo nome de auto-verificação do domínio do emissor. O segundo argumento é composto por 4 nomes auto-verificáveis, um para o *hardware*, um para o sistema operacional, um para o processo e um para o bloco, relacionados ao emissor do pacote. E o último argumento, também composto de quatro elementos com os mesmos significados, porém relacionados ao destinatário. Quando a mensagem não possui um destino específico para algum destes valores, é usado o valor 0xFFFFFFFFFFFF para indicar *broadcast*. A estrutura de uma *linha de comando de roteamento* pode ser observada na Figura 15.

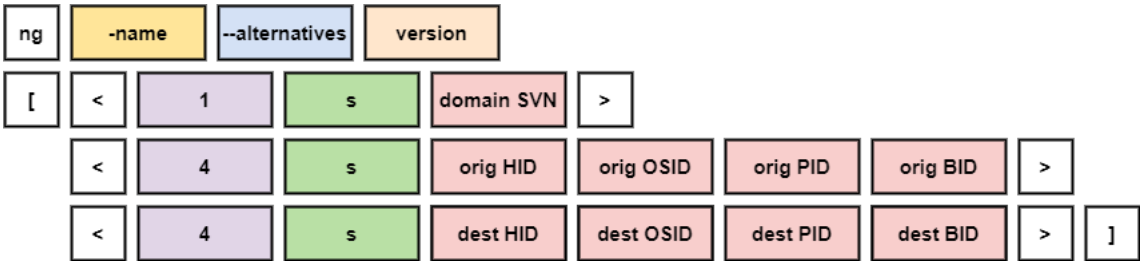


Figura 15 – Formato de uma linha de comando de roteamento.

3 Detalhamento da Proposta

A criação de um ambiente heterogêneo que propicie a coexistência de novas arquiteturas de Internet, através da infraestrutura de rede atual, apresenta-se como um solução plausível como forma de evolução para a Internet, pois mantém a atual infraestrutura, existente há cinquenta anos e ainda possibilitando a utilização desta mesma infraestrutura para o tráfego de pacotes de outras arquiteturas hoje em desenvolvimento, segmentadas e ilhadas.

Dessa forma, é apresentada neste trabalho uma proposta que aplicada em IXPs, por estes já possuírem toda uma infraestrutura de rede para a realização de encaminhamento, e que já conecta diversas redes, possibilita aos mesmos realizar o encaminhamento de pacotes de Arquiteturas de Internet do Futuro, funcionalidade hoje não existente nos IXPs. Com isso, entidades pertencentes a uma mesma arquitetura de Internet, podem se comunicar, contribuindo assim para o surgimento de um novo ambiente heterogêneo, composto pela arquitetura atual e por diversas novas Arquiteturas de Internet do Futuro.

Com a aplicação desta proposta em um Ponto de Troca de Tráfego de Internet convencional, este poderia passar ser denominado por *Ponto de Troca de Tráfego de Internet do Futuro* (*Future Internet Exchange Point* ou FIXP), como apresentado primeiramente em (MONTEIRO, 2015), designando assim o termo FIXP a um IXP que passa a realizar, além do encaminhamento convencional de pacotes IP, o encaminhamento de pacotes de Arquiteturas de Internet do Futuro.

A proposta para coexistência de múltiplas arquiteturas de Internet apresentada neste trabalho, se utiliza dos princípios de SDN, separando o *Plano de Controle* do *Plano de Dados* e utilizando controladores implementados em *software* para realizar o controle das regras de encaminhamento das arquiteturas suportadas, sendo cada arquitetura controlada por um controlador específico. Porém, apenas isso não é o suficiente para permitir que a proposta suporte o processamento de pacotes de novas arquiteturas. Assim, utilizando a linguagem P4 que apresenta a *independência de protocolo* como uma das suas principais caraterísticas, foi criado um comutador programável, que suporta o processamento de pacotes de múltiplas arquiteturas de Internet.

Cada Arquitetura de Internet do Futuro é projetada e desenvolvida com motivações e objetivos distintos uma das outras. Assim, cada uma define e utiliza um cabeçalho diferente das demais em formato, quantidade de campos, forma de endereçamento, entre outras características. Dessa forma, uma maneira padronizada de comunicação entre os controladores das arquiteturas e o comutador programável se fez necessária.

Para isso, um *Protocolo de Controle* foi definido.

A atual versão desta proposta não contempla a realização de tratamentos contra falhas, tais como verificação de *timeout* e reenvio de pacotes perdidos, bem como a descoberta e configuração dinâmica dos elementos presentes na proposta, como o relacionamento de interfaces de rede os controladores de quais arquiteturas a elas conectados, na *Camada de Abstração*, sendo esta configuração realizada de forma manual na versão atual.

A proposta utiliza o quadro 802.3 como protocolo base para a geração e tráfego dos pacotes de dados e de controle. Assim, a proposta para coexistência apresentada engloba as Arquiteturas de Internet do Futuro hoje baseadas em 802.3 ou que possam ter seus cabeçalhos adaptados para serem transmitidos no campo de carga útil do quadro 802.3. Dessa forma, a proposta é formada por um conjunto de processos distribuídos em três camadas denominadas *Camada de Controle*, *Camada de Abstração*, *Camada de Comutação* e por um novo *Protocolo de Controle*, que serão detalhados nas Seções a seguir. A Figura 16 apresenta uma representação das três camadas, os processos existentes em cada camada e o fluxo dos pacotes de dados e de controle.

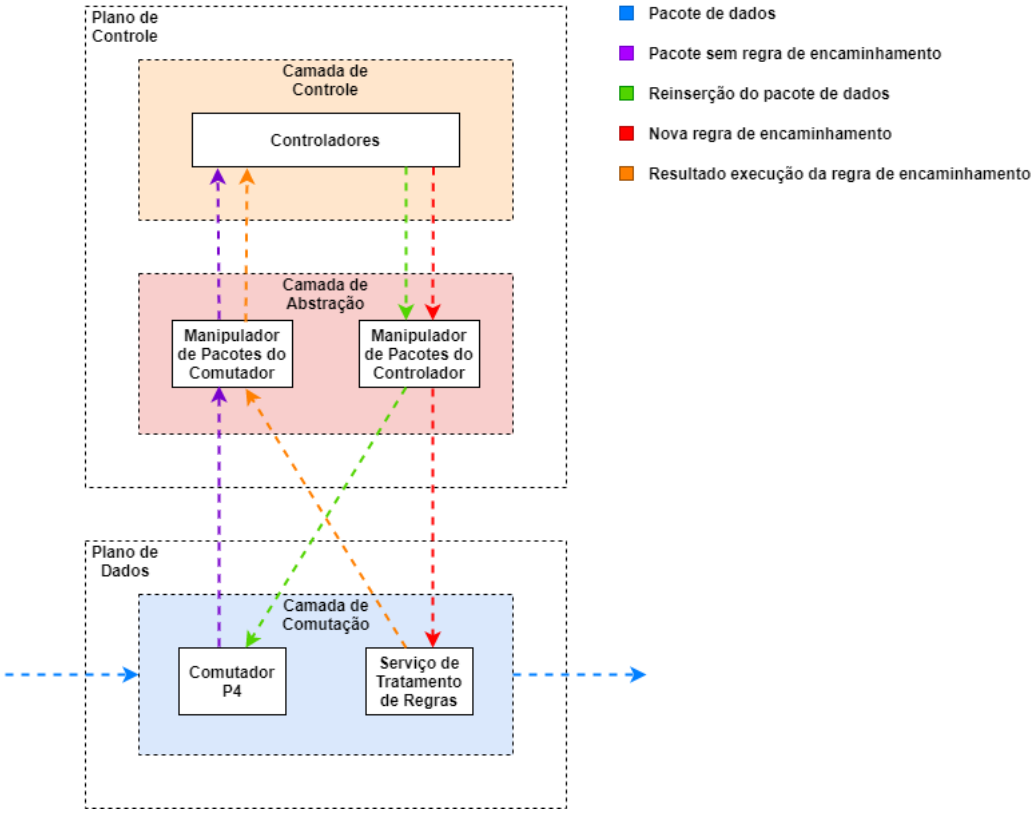


Figura 16 – Visão geral da proposta e seus componentes.

3.1 Camada de Controle

A *Camada de Controle*, é a denominação dada ao conjunto de servidores que executam os controladores de cada uma das arquitetura suportada pela proposta. Um controlador é uma aplicação responsável pelo controle das regras de encaminhamento de uma arquitetura específica no comutador programável multi arquitetura e também é responsável pela reinserção de pacotes de dados originalmente sem regras de encaminhamento, enviado pelo comutador programável multi arquitetura ao controlador responsável. Ambas as operações são realizadas através do envio de pacotes de controle dos controladores ao comutador programável multi arquitetura.

Os controladores podem ser disponibilizados em um servidores físicos ou virtuais, permitindo assim que cada controlador seja executado ou interrompido individualmente, não interferindo na operação dos demais controladores existentes. Cada controlador possui uma interface de rede pela qual se conecta à *Camada de Abstração* para realizar a troca de pacotes com o comutador programável multi arquitetura, através da *Camada de Abstração*.

Independente da arquitetura de Internet, para operar conforme esta proposta, um controlador necessita ser capaz de receber e armazenar temporariamente pacotes de dados, identificar o tipo da arquitetura do pacote recebido e no caso de um tipo de arquitetura válida, obter os dados de endereçamento de destino do pacote para gerar um pacote de controle de *Adição de Regra de Encaminhamento*, para que uma nova regra de encaminhamento seja adiciona na tabela de regras do comutador.

O controlador necessita também, ao receber um pacote de controle de *Resultado de Adição de Regra de Encaminhamento*, identificar e armazenar as informações do *status* e do identificador da regra executada pelo *Serviço de tratamento de Regras* e por fim, gerar um pacote de controle de *Reinserção de Pacote de Dados*, com o conteúdo do pacote de dados recebido inicialmente. A Figura 17 apresenta o diagrama de atividades das funcionalidades básicas de um controlador em relação à proposta.

3.2 Camada de Abstração

A *Camada de Abstração* é representada por um servidor físico ou virtual e tem por objetivo, possibilitar a utilização de apenas uma interface de rede do comutador programável multi arquitetura como meio de comunicação padrão com os controladores da *Camada de Controle* e realizar o encaminhamento de pacotes entre controladores e comutador programável. Caso contrário, o comutador programável multi arquitetura necessitaria utilizar uma interface de rede para cada controlador existente, consumindo assim interfaces de rede que deveriam ser destinadas ao uso de dispositivos, para a troca

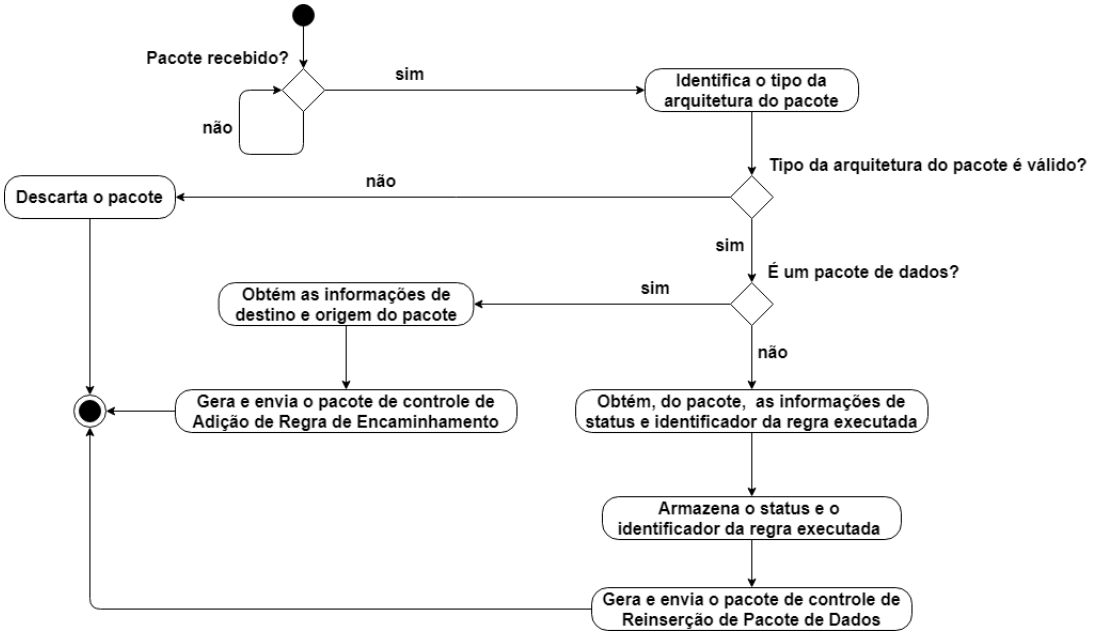


Figura 17 – Diagrama de atividades das funcionalidades básicas de um controlador.

de tráfego e não para conexão com controladores, necessitando também implementar a lógica de encaminhamento dos pacotes aos seus respectivos controladores.

Esta camada define interfaces de rede suficiente para permitir a conexão com cada um dos controladores existentes na *Camada de Controle* e uma interface de rede para realizar a conexão desta camada ao comutador programável na *Camada de Comutação*. A *Camada de Abstração* funciona como um roteador, realizando o encaminhando de pacotes entre controladores e comutador programável multi arquitetura, porém, apresentando a característica peculiar de utilizar o valor do campo *Ethertype* dos pacotes como parâmetro chave nas regras de roteamento.

Para a realização desses roteamentos, esta camada apresenta dois processos. O primeiro é o *Manipulador de Pacotes do Comutador*, que aguada o recebimento de pacotes enviados pelos processos do comutador programável multi arquitetura, da *Camada de Comutação*. Quando um pacote é recebido, o tipo da arquitetura do pacote é verificado e caso seja uma arquitetura válida, este é enviado para a interface de rede do controlador da arquitetura correspondente, caso seja um tipo de arquitetura inválida, o pacote é descartado. A Figura 18 apresenta as atividades realizadas pelo *Manipulador de Pacotes do Comutador*.

O segundo processo é o *Manipulador de Pacotes do Controlador*, que permanece aguardando a chegada de pacotes através de qualquer uma das interface de rede dos controladores. Assim que um pacote é recebido, é verificado o tipo da arquitetura do pacote e sendo uma arquitetura válida, o pacote é enviado para a interface de rede do comutador, na *Camada de Comutação*, caso o tipo da arquitetura seja inválido. o pacote é descartado. A Figura 19 apresenta as atividades realizadas pelo *Manipulador*

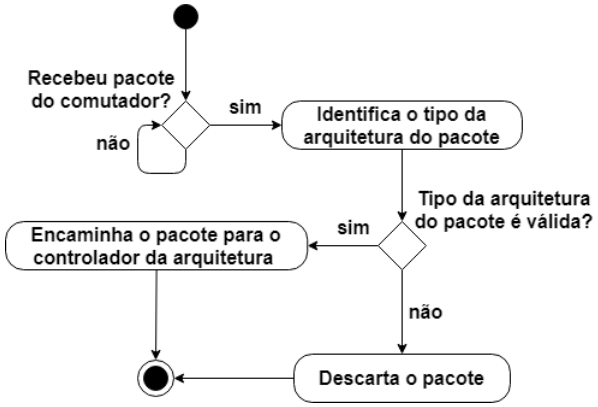


Figura 18 – Diagrama de atividades do *Manipulador de Pacotes do Comutador*.

de Pacotes do Controlador.

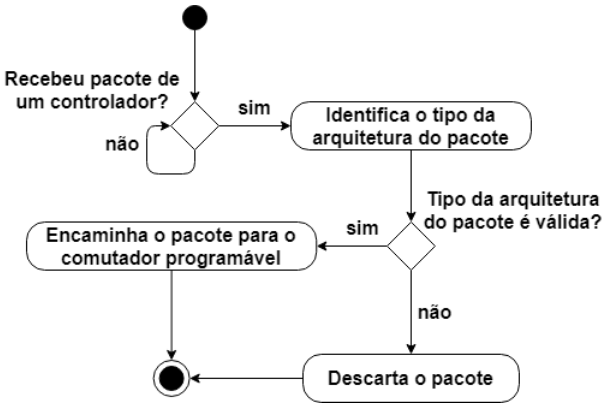


Figura 19 – Diagrama de atividades do *Manipulador de Pacotes do Controlador*.

3.3 Camada de Comutação

A *Camada de Comutação*, é o nome dado à camada onde se encontra o servidor físico ou virtual que atua como comutador programável multi arquitetura. O comutador programável define interfaces de rede para realizar conexão com as entidades que terão seus pacotes processados e encaminhados, definindo também uma interface de rede conectada à *Camada de Abstração*, para a realização da troca de pacotes com os controladores.

O comutador programável multi arquitetura é um servidor que realiza o recebimento de pacotes, a validação do tipo da arquitetura dos pacotes, o descarte de pacotes inválidos, o encaminhamento de pacotes de dados válidos aos seus destinos, o envio dos pacotes válidos sem regra de encaminhamento ao controlador responsável para a geração de um novo pacote de controle de regra de encaminhamento e o processamento de pacotes de controle com regras de encaminhamento.

Para isso, são implementados dois processos, o *Serviço de Tratamento de Regras* e o *Comutador P4*. O *Serviço de Tratamento de Regras*, é responsável pela aplicação de regras de encaminhamento enviadas pelos controladores, atualizando as tabelas de regras de encaminhamento do *Comutador P4*. Também é responsável por enviar o pacote de controle de *Resultado de Adição de Regra de Encaminhamento*, notificando o controlador sobre o resultado da aplicação da regra executada. A Figura 20 apresenta a sequência das atividades realizadas pelo *Serviço de Tratamento de Regras*.

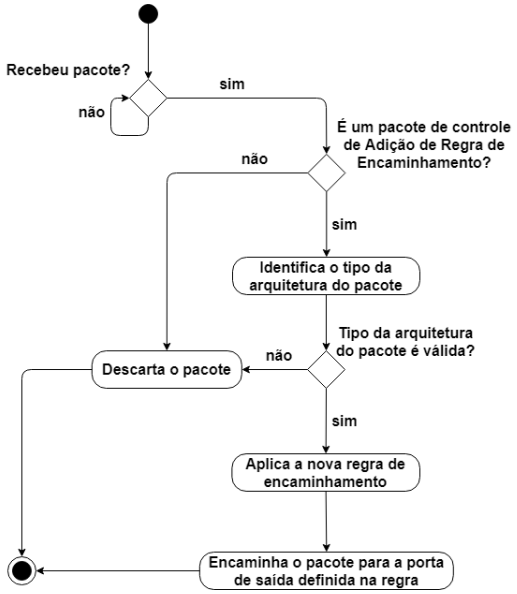


Figura 20 – Diagrama de atividades do *Serviço de Tratamento de Regras*.

O *Comutador P4*, é um comutador multi arquitetura definido por software, implementado em linguagem P4, responsável pela identificação do tipo da arquitetura de um pacote e pelo encaminhamento de pacotes de controle e de dados a seus destinos ou ao controlador da respectiva arquitetura, no caso do pacote não possuir regra de encaminhamento na tabela de regras de encaminhamento da sua arquitetura. A Figura 21 apresenta a sequência das atividades realizadas pelo *Comutador P4*.

3.4 Protocolo de Controle

O *Protocolo de Controle* é utilizado pelos controladores para realizar, de forma padronizada e independente da arquitetura, a adição de novas regras de encaminhamento a serem aplicadas no comutador e a reinserção de pacotes de dados originalmente sem regras de encaminhamento na tabela de regras do comutador. Ele também é utilizado pelo comutador programável multi arquitetura para encaminhar ao controlador que enviou o pacote de adição de uma nova regra de encaminhamento, um outro pacote de controle contendo resultado da aplicação da nova regra de encaminhamento na tabela de regras do comutador, após a aplicação da regra e encaminhamento pelo *Serviço de*

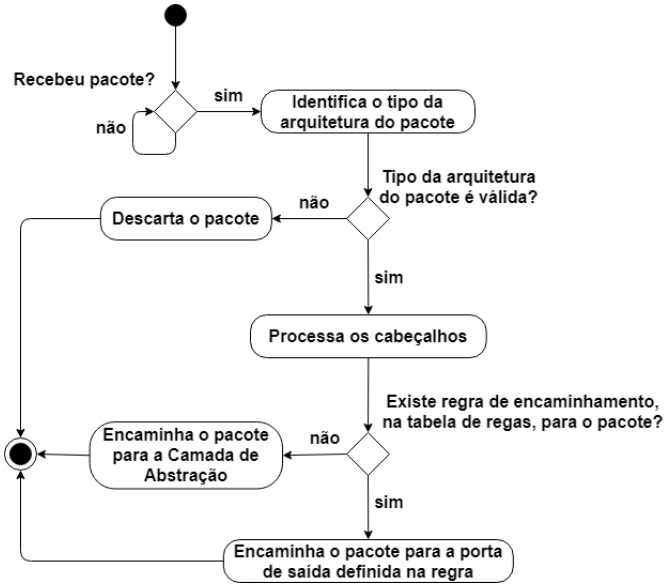


Figura 21 – Diagrama de atividades do *Comutador P4*.

Tratamento de Regras. Atualmente o *Protocolo de Controle* implementa três operações de controle. A primeira é a *Adição de Regra de Encaminhamento*, a segunda é a operação de *Resultado de Adição de Regra de Encaminhamento* e a terceira é a *Reinserção de Pacote de Dados*.

A estrutura do *Protocolo de Controle* é baseada no quadro Ethernet 802.3, com os campos MAC de destino, MAC de origem, *Ethertype* e o campo de carga útil (*payload*). Os campos MAC de destino e MAC de origem são utilizados com diferentes propósitos para cada um dos pacotes de controle definidos, sendo seus usos detalhada a seguir. Para os pacotes do *Protocolo de Controle*, o campo *Ethertype*, é sempre configurado com o valor hexadecimal 0x0900, para indicar que este é um pacote de controle.

O campo de carga útil de cada pacote de controle é responsável pelo transporte das informações utilizadas para a execução de cada operação de controle. Assim, cada campo de carga é composto por uma quantidade diferente de campos. A quantidade de campos que compõem a carga útil de cada um dos pacotes de controle e suas descrições são detalhadas a seguir. A representação da estrutura básica do protocolo de controle é apresentada na Figura 22.

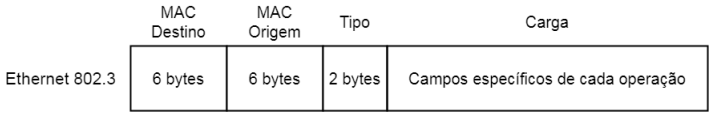


Figura 22 – Representação da estrutura do Protocolo de Controle.

3.4.1 Adição de Regra de Encaminhamento

A primeira operação de controle é a *Adição de Regra de Encaminhamento*. O comutador programável multi arquitetura, ao receber um pacote de dados, realiza a

identificação do seu tipo de arquitetura do pacote e verifica a existência de regras de encaminhamento para o pacote em sua tabela de regras de encaminhamento a fim de obter a informação da porta do comutador pela qual o pacote deve ser encaminhado.

Porém, pacotes de dados sem regras na tabela de regras de encaminhamento são, por padrão, encaminhados através da *Camada de Abstração* ao controlador responsável, para que este obtenha o endereço de destino do pacote e crie um novo pacote de controle de *Adição de Regra de Encaminhamento*, que será enviado através da *Camada de Abstração* ao comutador programável. Ao chegar no comutador programável, o *Serviço de Tratamento de Regras* recebe o pacote, verifica se é um pacote de controle de *Adição de Regra de Encaminhamento*, identifica o tipo da arquitetura, informada no pacote de controle, e aplica a regra de encaminhamento na tabela de regras de encaminhamento da arquitetura informada do pacote de controle. A Figura 23 apresenta o diagrama de atividade do fluxo da operação de *Adição de Regra de Encaminhamento*.

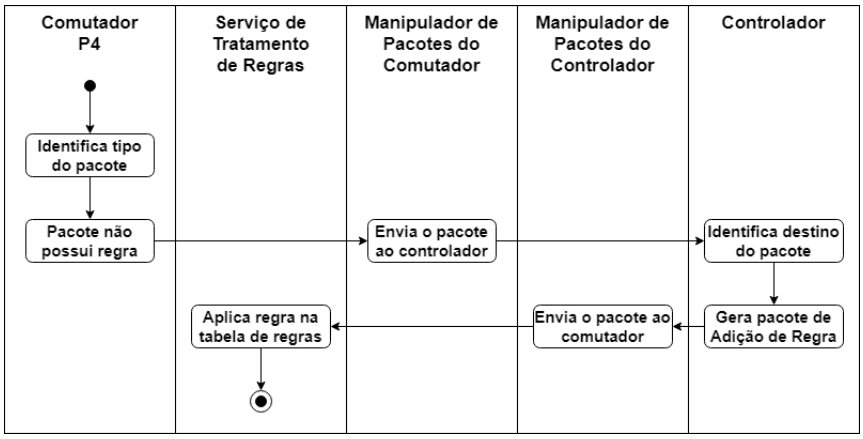


Figura 23 – Diagrama de atividade da operação *Adição de Regra de Encaminhamento*.

Para o pacote de controle de *Adição de Regra de Encaminhamento*, o MAC de destino é setado com o valor hexadecimal **0x464958500000**, que representa os *bytes* dos caracteres *F*, *I*, *X* e *P*, o MAC de origem não precisa ser preenchido e o campo *Ethertype* é configurado com o valor hexadecimal **0x0900**, indicando um pacote de controle. Desta forma, o processo *Manipulador de Pacotes do Controlador* da *Camada de Abstração* identifica o pacote de controle, direcionando-o para a *Camada de Comutação*.

O campo de carga útil do pacote de controle de *Adição de Regra de Encaminhamento* é formado por um objeto do tipo lista da linguagem Python, convertido em uma *string* padrão JSON, tendo por fim todos seus *bytes* adicionados ao campo. O objeto do tipo lista que compõem a carga útil do pacote de *Adição de Regra de Encaminhamento* é composto por dez campos, com as seguintes informações:

- **Arquitetura Encapsulada**: sequência de um a seis dígitos, em formato decimal, que informa o tipo da arquitetura do pacote que originou esta operação. Por

exemplo, para indicar o tipo IP, é informado o valor *2048* (20148 é a representação decimal do valor hexadecimal 0x0800).

- **Identificador de Sequência dos Pacotes**: um dígito em formato decimal, que informa o número de controle da sequência dos pacotes de controle. Por definição, o número de sequência é iniciado a partir do valor **1**.

- **Identificador do Comutador**: sequência de caracteres iniciada por **"**, seguida por um caractere **s**, dois dígitos em formato decimal e por ouro caractere **"**. Informa ao *Manipulador de Pacotes do Controlador*, a identificação do comutador ao qual este pacote deve ser encaminhado. Na versão atual, este campo deve ser configurado com **"s01"**.

- **Porta do Servidor Thrift**: sequência de até seis dígitos, em formato decimal, que representa o endereço da porta do servidor Thrift. A porta padrão do Servidor Thrift é **9090**.

- **Identificador de Comando**: um dígito em formato decimal que indica qual o comando o *Serviço de Tratamento de Regras* deve executar. Atualmente é implementada apenas a adição, representada pelo valor **1**.

- **Nome da Tabela de Encaminhamento**: sequência de caracteres iniciada e terminada por **"**, que informa o nome da seção de ingresso, seguido por um caractere **.** e pelo nome da tabela de encaminhamento a ser utilizada. Como por exemplo a sequência de caracteres **"FIXP_Switch_Ingress.ipv4_table"**.

- **Lista de Chaves**: um conjunto de um ou mais valores em formato decimal ou caracteres em formato *ASCII*, limitado entre aspas, separados por vírgula, definido entre colchetes. Estes valores representam os endereços de destino dos pacotes de dados da arquitetura, podendo ser um MAC, um IP ou uma *hash*.

- **Lista de Ações**: um conjunto de um ou mais valores em formato decimal ou caracteres em formato *ASCII*, limitado entre aspas, separados por vírgula, definido entre colchetes. Este campo informa o nome das ações de encaminhamento executadas quando da execução da regra de encaminhamento a ser adicionada.

- **Lista de Parâmetros das Ações**: um conjunto, definido entre colchetes, de um ou mais conjuntos de valores em formato decimal, separados por vírgula, definidos entre colchetes. Estes parâmetros são utilizados pelas ações de encaminhamento quando for realizado o encaminhamento de um pacote. Atualmente este parâmetros representam as portas de saída por onde o pacote a ser encaminhado deve ser enviado.

- **Opção da Operação**: um dígito em formato decimal que informa ao *Serviço de Tratamento de Regras* se a operação de adição insere uma nova regra, ou atualiza uma regra já existente. Atualmente é implementada apenas a opção de adição, representada pelo valor **0**.

A Figura 24 apresenta um exemplo de pacote de controle de *Adição de Regra de Encaminhamento*.

MAC origem	MAC origem	Tipo	Carga útil
46:49:58:50:00:00	00:00:00:00:00:00	0x0900	[2048, 1, "s01", 9090, 1, "FIXP_Switch_Ingress.ipv4", ["192.168.221.102"], ["FIXP_Switch_Ingress.ipv4.forward"], [[7]], 0]

Figura 24 – Exemplo de pacote de controle de *Adição de Regra de Encaminhamento*.

3.4.2 Resultado de Adição de Regra de Encaminhamento

A segunda operação de controle é o *Resultado de Adição de Regra de Encaminhamento*. Assim que o pacote de controle de *Adição de Regra de Encaminhamento* é recebido e processado pelo *Serviço de Tratamento de Regras* da *Camada de Comutação*, são obtidos o status e o identificador da operação realizada na tabela de regras de encaminhamento do comutador. Assim o *Serviço de Tratamento de Regras* gera um pacote de controle de *Resultado de Adição de Regra de Encaminhamento*, para informar ao controlador responsável o resultado da operação realizada, retornado o status e um identificador da operação realizada. Este identificador é necessário para realizar uma futura remoção da regra adicionada na tabela. A Figura 25 apresenta o diagrama de atividade do fluxo da operação de *Resultado de Adição de Regra de Encaminhamento*.

No pacote de controle de *Resultado de Adição de Regra de Encaminhamento*, o MAC de origem é setado com o valor hexadecimal **0x464958500000**, o MAC de destino não é configurado e o campo *Ethertype* é configurado com o valor hexadecimal **0x0900**, para indicar um pacote de controle da proposta. Com isso, o processo *Manipulador de Pacotes do Comutador* da *Camada de Abstração* identifica o pacote de controle, direcionando-o para o controlador correto na *Camada de Controle*. O campo de carga útil do pacote de controle de *Resultado de Adição de Regra de Encaminhamento*, também é formado por um objeto do tipo lista da linguagem Python, convertido em uma *string* padrão JSON, tendo por fim todos seus *bytes* adicionados ao campo. O objeto do tipo lista que compõem a carga útil do pacote de *Adição de Regra de Encaminhamento* é composto por sete campos, com as seguintes informações:

- **Arquitetura Encapsulada**: sequência de um a seis dígitos, em formato decimal, que informa o tipo da arquitetura do pacote que originou esta operação. Por

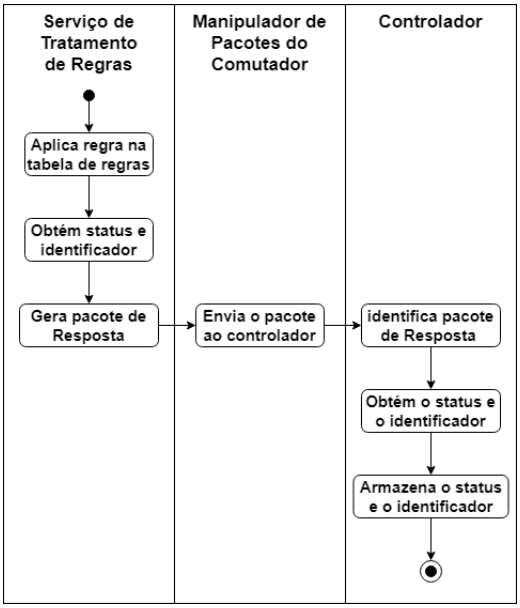


Figura 25 – Diagrama de atividade da operação *Resultado de Adição de Regra de Encaminhamento*.

exemplo, para indicar o tipo IP, é informado o valor *2048* (20148 é a representação decimal do valor hexadecimal 0x0800).

- **Identificador de Sequência dos Pacotes**: um dígito em formato decimal, que informa o número de controle da sequência dos pacotes de controle. Por definição, o número de sequência é iniciado a partir do valor **1**.

- **Identificador do Comutador**: sequência de caracteres iniciada por **"**, seguida por um caractere **s**, dois dígitos em formato decimal e por ouro caractere **"**. Informa ao *Manipulador de Pacotes do Controlador*, a identificação do comutador ao qual este pacote deve ser encaminhado. Na versão atual, este campo deve ser configurado com **"s01"**.

- **Porta do Servidor Thrift**: sequência de até seis dígitos, em formato decimal, que representa o endereço da porta do servidor Thrift. A porta padrão do Servidor Thrift é a **9090**.

- **Identificador do Comando Realizado**: um dígito em formato decimal que indica qual o comando foi realizado pelo *Serviço de Tratamento de Regras*. Atualmente é implementado apenas o comando de adição, representada pelo valor **1**.

- **Lista de Identificadores**: um conjunto de um ou mais valores em formato decimal, separados por vírgula, definido entre colchetes. Este campo informa os valores dos códigos de identificação das operações de adição realizadas pelo *Serviço de Tratamento de Regras*, sendo necessárias para futuramente realizar a remoção de uma regra da tabela de regas de encaminhamento.

- **Status**: um dígito em formato decimal que informa o status da operação de *Adição de Regra de Encaminhamento* executada pelo *Serviço de Tratamento de Regras*. O valor **0** indica sucesso, enquanto o valor **1** indica falha.

A Figura 26 apresenta um exemplo de pacote de controle de *Resultado de Adição de Regra de Encaminhamento*.

MAC origem	MAC origem	Tipo	Carga útil
00:00:00:00:00:00	46:49:58:50:00:00	0x0900	[2048, 1, "s01", 9090, 1, [0], 0]

Figura 26 – Exemplo de pacote de controle de *Resultado de Adição de Regra de Encaminhamento*.

3.4.3 Reinserção de Pacote de Dados

A terceira operação de controle é a *Reinserção de Pacote de Dados*. O controlador, ao receber o pacote com o *Resultado de Execução de Regra de Encaminhamento* com status positivo, gera o pacote de controle de *Reinserção de Pacote de Dados*, para reenviar à *Camada de Comutação* o pacote originalmente direcionado ao controlador por não possuir regra de encaminhamento na tabela, para que este seja processado pelo comutador e encaminhado ao seu destino. A Figura 27 apresenta o diagrama de atividade do fluxo da operação de *Reinserção de Pacote de Dados*.

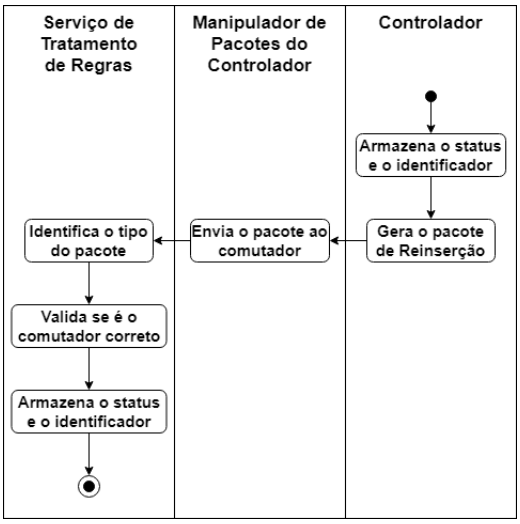


Figura 27 – Diagrama de atividade da operação *Reinserção de Pacote de Dados*.

O pacote de controle de *Reinserção de Pacote de Dados*, os campos MAC de origem e destino são configurados com os respectivos valores de MAC de origem e destino do pacote de dados original e o campo *Ethertype* é setado com o valor hexadecimal **0x0900**. Assim, o processo *Manipulador de Pacotes do Controlador* da *Camada de Abstração* identifica

o pacote de controle, direcionando-o para a *Camada de Comutação*. O campo de carga útil do pacote de controle de *Reinserção de Pacote de Dados* é formado pela concatenação dos dois bytes do tipo de arquitetura em formato hexadecimal, com o identificador do comutador, formado pelo caractere **s** seguido de dois dígitos decimais, com o valor da porta de saída informada na adição, em formato hexadecimal de dois *bytes* e por fim, o campo de carga útil do pacote de dados original. A Figura 28 apresenta um exemplo de pacote de controle de *Reinserção de Pacote de Dados*.

MAC origem	MAC origem	Tipo	Carga útil
00:00:00:00:00:28	00:00:00:00:00:26	0x0900	\x08\x00s01\x00\x07<bytes do campo de carga útil do pacote original>

Figura 28 – Exemplo de pacote de controle de *Reinserção de Pacote de Dados*.

Um exemplo de fluxo de encaminhamento de pacotes entre duas entidades *A* e *B* é visto na Figura 29 que apresenta o diagrama de sequência do fluxo de pacotes, com o comutador apresentando as tabelas de regras de encaminhamento inicialmente sem regras.

O próximo pacote de dados enviado da entidade *A* para a entidade *B* será recebido pelo processo *Comutador P4* na *Camada de Comutação* e, após ter sua arquitetura verificada, será encaminhado para interface de rede da entidade *B*, sem qualquer intervenção do Plano de Controle.

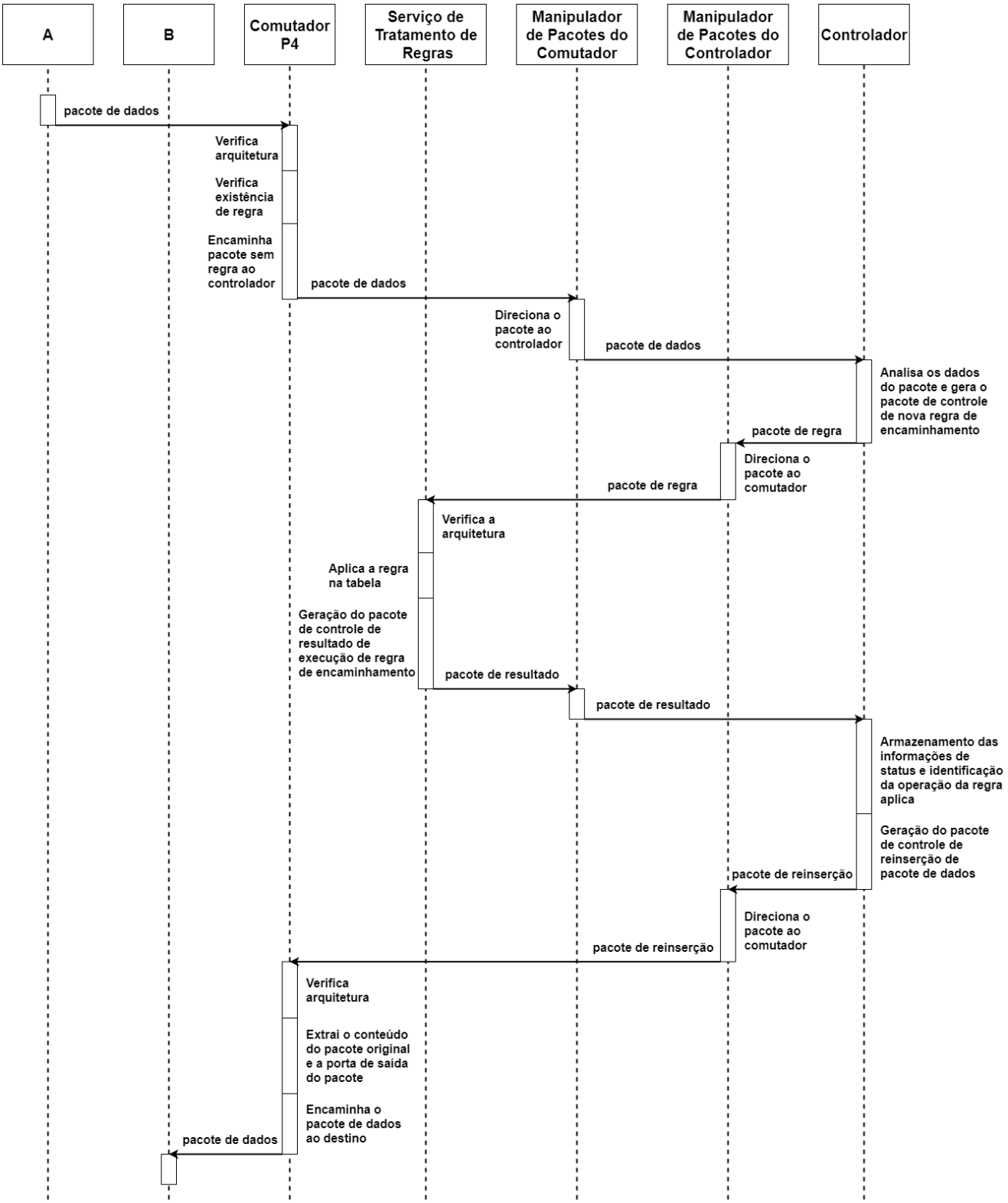


Figura 29 – Diagrama de Sequência do fluxo de pacotes.

4 Implementação e Avaliação

Para demonstrar o funcionamento da proposta apresentada, implementou-se uma prova de conceito, com as três camadas, seus respectivos processos e o Protocolo de Controle, com funcionalidades básicas que permitem a realização de encaminhamento de pacotes e a aplicação de regras de encaminhamento das arquiteturas NovaGenesis, ETArch e do protocolo IP.

Na Seção 4.1, são apresentados os detalhes de implementação, destacando pontos específicos referentes aos cabeçalhos das arquiteturas suportadas e dos processos existentes nas camadas de comutação, abstração e controle, que permitem à prova de conceito realizar a aplicação de regras de encaminhamento e o controle de tráfego de pacotes das diferentes arquiteturas suportadas.

Na Seção 4.2, são apresentados a configuração, o ambiente, a topologia e os testes realizados para validar o tratamento da fragmentação de pacotes da NovaGenesis e da prova de conceito com a arquitetura ETArch e o protocolo IP. Por fim, sendo executados os testes envolvendo os tempos gastos pelo comutador programável multi arquitetura com o processamento de pacotes de dados, de controle e com o tempo completo do fluxo de pacotes.

4.1 Implementação

Nesta seção, serão apresentados os detalhes de implementação dos processos *Comutador P4* e *Serviço de Tratamento de Regras*, da *Camada de Comutação*, do *Manipulador de Pacotes do Controlador* e *Manipulador de Pacotes do Comutador*, da *Camada de Abstração* e, finalmente, alguns detalhes sobre os controladores da *Camada de Controle*.

4.1.1 Camada de Comutação

A *Camada de Comutação*, é definida por um servidor virtual, com 10 gigabytes de espaço em disco, 4 gigabytes de memória, com Sistema Operacional Linux Ubuntu Server 16.04 de 64 bits, sendo esta versão de Sistema Operacional a indicada para a compilação e execução do Behavior Model version 2 (BMv2) (BEHAVIORAL MODEL, 2020). Nesta máquina virtual foram instaladas a linguagem Python 2.7.5, a biblioteca Scapy (SCAPY-PACKET CRAFTING FOR PYTHON2 AND PYTHON3, 2020), para geração e manipulação de pacotes em Python, a *API* P4-Runtime (P4 Runtime, 2019), que implementa o controle dos elementos do Plano de Dados de um dispositivo programado em P4, o Behavioral Model version 2, que implementa em C++ o comportamento de

processamento de pacotes definido por um programa feito em P4, o compilador P4C, que é o compilador para a linguagem de programação P4.

Foram configuradas oito interfaces de rede, sendo duas para conexão com *hosts* IP, duas para conexão com *hosts* ETArch, duas para conexão com *hosts* NovaGenesis, uma interface de rede para conexão com a *Camada de Abtração* e uma interface de rede para realização de conexão ssh através do *host* para troca de arquivos com a máquina virtual do comutador. A Figura 30, é uma representação da *Camada de Comutação*, apresentando as interfaces de rede e os processos existentes nesta camada.

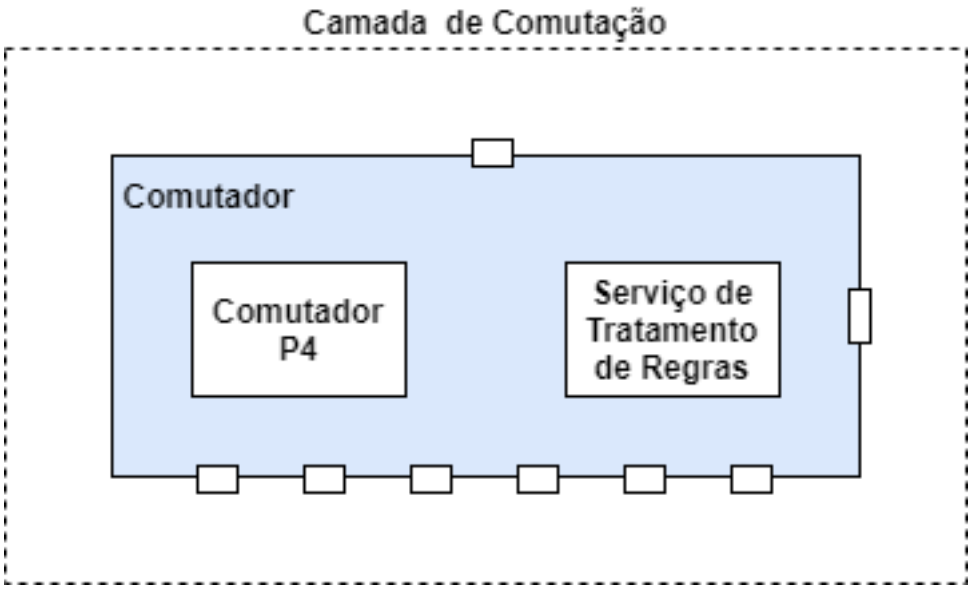


Figura 30 – Representação da *Camada de Comutação*.

Esta camada possui dois processos, o *Comutador P4*, que realiza a identificação do tipo da arquitetura e realiza o encaminhamento de pacotes e o processo *Serviço de Tratamento de Regras*, responsável pela aplicação de regras de encaminhamento e envio do pacote de controle *Resultado de Adição de Regra de Encaminhamento*.

O *Comutador P4*, é um programa criado utilizando a linguagem de processamento de pacotes $P4_{16}$, que define o processo de análise e encaminhamento de pacotes das arquiteturas de Internet ETArch, NovaGenesis e do protocolo IP. Para isso, são definidos os cabeçalhos dos protocolos *Ethernet*, IP e das Arquiteturas de Internet do Futuro ETArch e NovaGenesis, além das três seções funcionais: o *Analisador*, responsável pela identificação do tipo da arquitetura, o *Ingresso*, com a definição das tabelas de encaminhamento e das ações de encaminhamento de cada arquitetura e o *Agrupador*, que remonta os campos dos cabeçalhos de cada protocolo no pacote a ser encaminhado à porta de saída correta.

O primeiro cabeçalho a ser definido foi o *Ethernet*, pois, as informações dos cabeçalhos IP, ETArch e NovaGenesis (ou futuras arquiteturas suportadas pela proposta), são trafegados através do 802.3, em seu campo de carga útil. O cabeçalho IP

é definido como uma sequência de campos diretamente correspondentes à definição do protocolo IP, para permitir o tratamento individual de cada campo do cabeçalho. Para a NovaGenesis, são definidos os cabeçalhos *novagenesis_hasDHID_t*, que apresenta a *linha de comando de roteamento* com o campo de destino *dhid* e o cabeçalho *novagenesis_hasntDHID_t*, que não apresenta a *linha de comando de roteamento*. Esta abordagem é parte da solução para encaminhamento de pacotes fragmentados da NovaGenesis. Para a ETArch são definidos três cabeçalhos. O cabeçalho *etarch_t*, que é utilizado para definir pacotes de dados e pacote de controle destinados ao controlador e os cabeçalhos *etarch_switch_t* e o *etarch_switch_2_t*, utilizados para definir o pacote de controle de resposta internos da própria ETArch. Os cabeçalhos *etarch_t* e *etarch_switch_2_t* são definidos como uma sequência de *bits*, pois não apresentam campos que precisam ser processados, possuindo apenas informação a ser encaminhada.

Como parte da implementação da solução para o encaminhamento de pacotes fragmentados da NovaGenesis, utilizou-se uma estrutura do tipo *register* para armazenar temporariamente a informação de destino do pacote, o *dhid*, enviado apenas no primeiro fragmento de pacotes fragmentados. Assim, durante o processamento dos demais fragmentos, que não possuem a informação de destino, essa informação é obtida da estrutura e recuperada, possibilitando assim o encaminhamento de pacotes fragmentados.

O valor do campo *msgid* é utilizado como parâmetro da função *hash* para a geração de um índice, onde o valor do campo *dhid*, do cabeçalho *novagenesis_hasDHID_t*, será armazenado/recuperado na estrutura *register*. Atualmente, a função *hash* é a única forma de obtenção de um índice em P4. Para a geração do índice, foi utilizado o método *crc32*. Uma característica interessante da função de *hash* utilizada é a realização do mapeamento do valor obtido em um intervalo informado. Dessa forma, foram utilizados os valores 0 e 1023 como limites inferior e superior do intervalo, definindo um intervalo de 1024 valores possíveis, sendo o suficiente para a realização dos testes.

Assim, para a prova de conceito, a estrutura *register* foi definida para suportar o armazenamento de 1024 valores diferentes de informações de destino, sendo este valor relacionado à quantidade de índices gerados pela função *hash*. A definição da estrutura temporária é mostrada no Código-Fonte 5.

No *Analisador*, é implementada a extração do protocolo Ethernet e a verificação dos valores dos campos: *Ethertype*, MAC de destino e MAC de origem dos cabeçalhos. O valor do campo *Ethertype* é utilizado para identificar a arquitetura do pacote. Para um *Ethertype* com valor igual a 0x0800, identifica-se um pacote da arquitetura IP, para o valor 0x0880, um pacote da ETArch e para o valor temporário 0x1234, um pacote da NovaGenesis. Caso o *Ethertype* apresente o valor 0x0900, este é um pacote de controle FIXP.

Para pacotes de controle FIXP, o MAC de destino com valor 0x464958500000 (valores em hexadecimal dos caracteres F,I,X e P) indica um pacote de controle de *Adição de*

Regra de Encaminhamento, enquanto um pacote de controle FIXP com o MAC de origem com valor 0x464958500000 indica um pacote de controle de *Resposta de Adição de Regra de Encaminhamento*. Por fim, pacotes ETArch com o MAC de destino que apresentam o valor 0x445453000000, representam pacotes de controle ETArch com destino ao DTS, enquanto pacotes ETArch com MAC de destino com valor 0xffffffffffff, representam pacotes de controle ETArch de resposta. A Figura 31 apresenta um diagrama de atividades com o fluxo de identificação das arquiteturas implementado pelo *Analisador*. A codificação da extração do protocolo *Ethernet* e da verificação do campo *Ethertype* é apresentada no Código-Fonte 6.

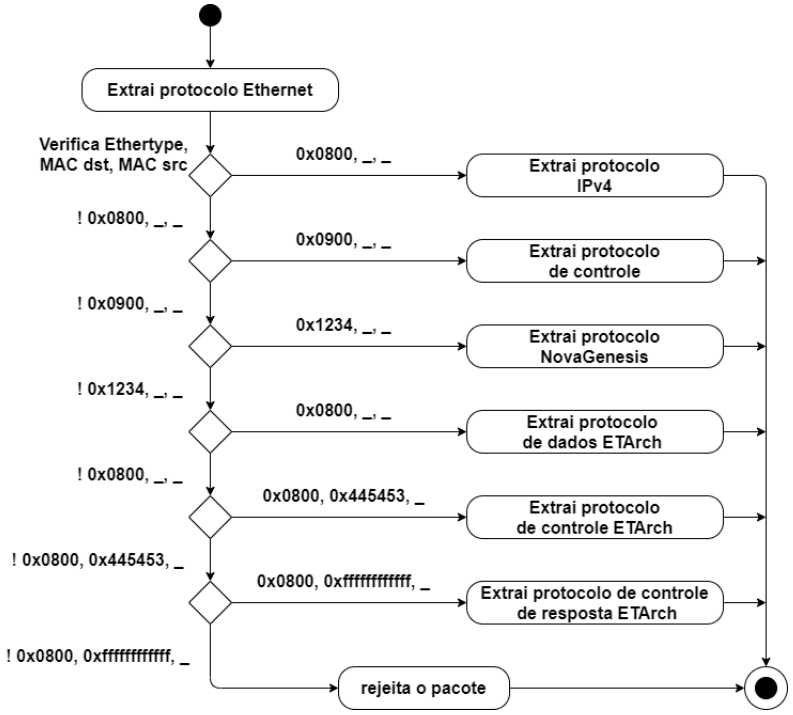


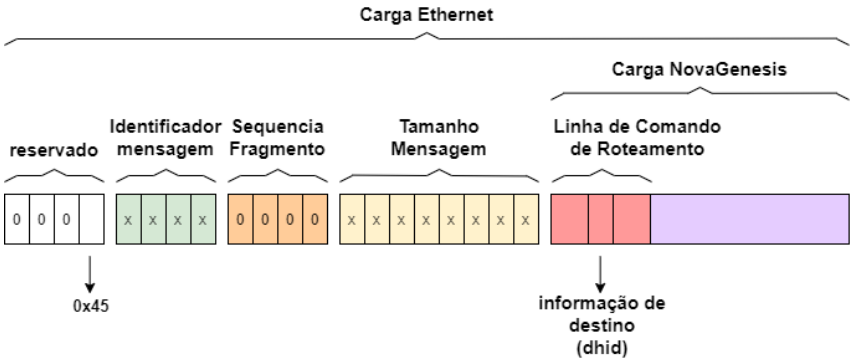
Figura 31 – Diagrama de Atividades da identificação da arquitetura de um pacote.

No caso de pacotes ETArch, se o campo MAC de destino apresenta o valor 0x445453000000, que representa os valores hexadecimais dos caracteres D,T e S (DTS, acrônimo de Domain Title System), isto indica que este é um pacote de controle interno da ETArch e que este pacote foi enviado de uma entidade ETArch ao DSTA, como uma operação de *WORKSPACE_CREATE* ou qualquer outra descrita na Tabela 2 da Seção 2.2. Se o campo MAC de destino apresentar o valor 0xFFFFFFFFFFFF, indica que estes são um segundo tipo de pacotes de controle internos da ETArch. Estes porém, são pacotes que realizam o *handshake* de controle na ETArch.

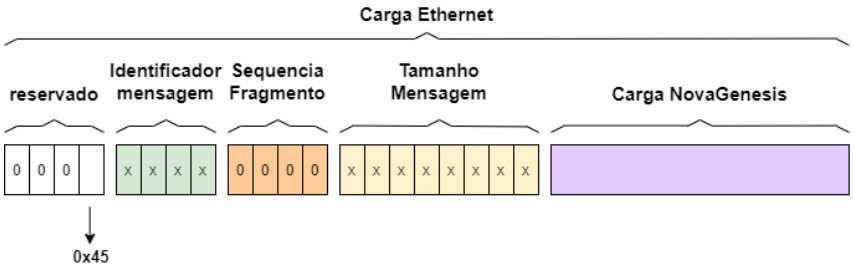
No caso de pacotes NovaGenesis, o *Analisador* necessita ainda identificar qual dos dois tipos de cabeçalho é utilizado no pacote. Esta etapa é necessária pois, o cabeçalho dos pacotes contendo a *linha de comando de roteamento*, presente em um pacote completo ou no primeiro fragmento de um pacote, como visto na Figura 32 (a), é diferente do

cabeçalho dos demais fragmentos de um pacote, que não apresentam a *linha de comando de roteamento*, como representado na Figura 32 (b).

Para identificar se o fragmento possui a informação de destino do pacote (fragmento com *sequência de fragmentos* apresentando valor igual a 0), é necessário verificar o valor presente nos *bytes* do campo *sequência de fragmentos*. Para isso, é utilizado o método *lookahead* do P4, que realiza a leitura dos próximos *n bits* do cabeçalho. Neste caso, são lidos os próximos 96 *bits* (4 *bytes* do *reservado*, 4 *bytes* do *identificador de mensagem* e 4 *bytes* da *sequência de fragmentos*) a partir do campo *reservado*. Assim, os *bytes* dos campos *reservado* e *identificador da mensagem* são desprezados, restando os últimos 4 *bytes* do campo *sequência de fragmentos*, que são verificados. Se este campo possuir valor igual a 0, então este é um pacote completo ou o primeiro fragmento de um pacote fragmentado NovaGenesis, apresentado então a *linha de comando de roteamento*.



(a) Primeiro fragmento do pacote NovaGenesis.



(b) Segundo fragmento do pacote NovaGenesis.

Figura 32 – Formatos dos dois tipos de fragmentos de pacote NovaGenesis.

A Figura 33 mostra a representação do diagrama de estados do *Analisador*. O Código-Fonte 7 apresenta a implementação do *Analisador* que realiza a verificação dos *bits* e define o tipo de cabeçalho correto a ser utilizado.

Com a arquitetura do pacote identificada, a etapa seguinte é a extração dos campos do protocolo identificado, finalizando assim a seção de análise e com o fluxo de execução seguindo para a Seção de *Ingresso*.

Na Seção de *Ingresso*, são definidas as tabelas de encaminhamento para cada arquitetura, cada qual configurada com sua chave de encaminhamento, seus atributos de

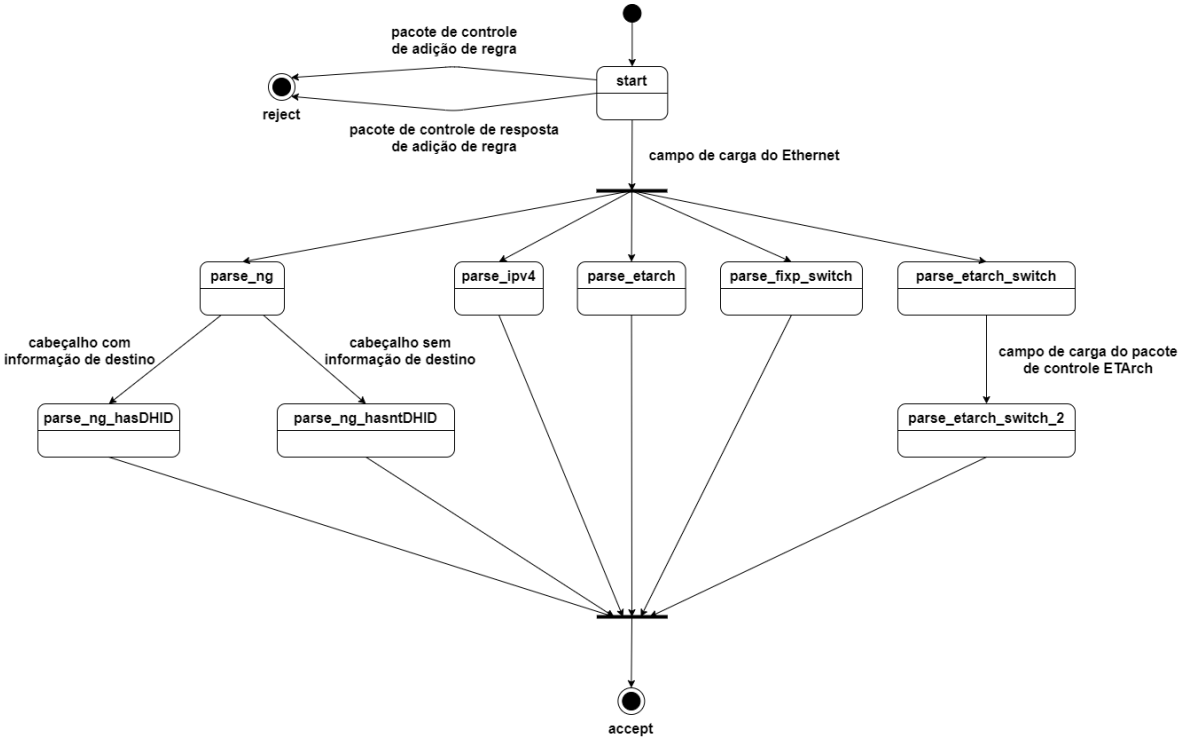


Figura 33 – Diagrama de Estados do Analisador.

tamanho da tabela, ação padrão, ações de encaminhamento e seu programa de controle.

As *ações* P4 são como funções da linguagem de programação C, sendo onde o desenvolvedor implementa as instruções com a finalidade de manipular e processar os cabeçalhos, estruturas e metadados, a fim de realizar o encaminhamento de pacotes. As *ações toControllerIP*, *toControllerETArch* e *toControllerNG* são utilizadas, por suas respectivas arquiteturas, para especificar a porta de saída para a *Camada de Abstração*, através da porta 1 do comutador. Esta porta foi definida como a porta padrão de encaminhamento de pacotes sem regra de encaminhamento aos controladores. As *ações ng_SetSpec* e *ipv4_SetSpec* definem a porta de saída do pacote para o seu destino. A *ação etarch_SetSpec_Group*, se assemelha às ações *ng_SetSpec* e *ipv4_SetSpec*, pois também possui a finalidade de definir portas de saída. Porém, esta *ação* permite à ETArch, através da criação de grupos na tabela de regras de encaminhamento, definir uma única porta ou um conjunto de portas de saída, realizando assim *unicast* ou *multicast*. A *ação toMakePacketOut*, realiza a reinserção de pacotes de qualquer uma das arquiteturas suportadas. E a *ação drop* é definida para realizar o descarte de pacotes. A implementação das *ações* descritas é apresentada no Código-Fonte 8.

O campo de endereço IP de destino foi definido como chave da tabela de encaminhamento para o protocolo IP. Para a ETArch, definiu-se como chave da tabela de encaminhamento o campo endereço de MAC de destino. E para a NovaGenesis, foi definido o campo da *linha de comando de roteamento* denominado *dhid* (Destination Host Id). As chaves das três tabelas foram definidas com o tipo *exact*. O tipo *exact* foi

utilizado para permitir compatibilidade futura com o dispositivo NetFPGA-SUME, cuja implementação atual do P4 suporta este tipo.

As três tabelas definem o atributo *size*, descrito na Seção 1.1.2, com o valor igual a 1023, especificando assim, a quantidade máxima de entradas de regras de encaminhamento suportadas pelas tabelas. As tabelas também definem o atributo *default_action*, que especifica a ação a ser executada em pacotes sem regra de encaminhamento. No caso da ETArch, o *default_action* foi definido com a ação *toControllerEtarch*. Esta ação verifica se o pacote é uma resposta de pacote de controle ETArch, redirecionando-o para a porta da entidade a receber o pacote ou se é um pacote de dados ETArch sem regra de encaminhamento, encaminhando-o para o controlador ETArch através da *Camada de Abstração*, pela porta 1 do comutador. Na NovaGenesis, foi definida a ação *toControllerNG*, que atribui, ao terceiro byte reservado do cabeçalho NoveGenesis, o número da porta de ingresso do pacote no comutador, encaminhando então o pacote para o controlador NovaGenesis através da *Camada de Abstração*, pela porta 1. E para o IP, foi atribuído ao atributo *default_action* a ação *toControllerIP* que encaminha o pacote para o controlador IP pela *Camada de Abstração*, através da porta 1.

Por fim, em cada tabela é definido o atributo *actions*, com a lista de nomes das ações utilizadas pela arquitetura. Para o IP, *actions* é definido com as ações *ipv4_SetSpec* e *toControllerIP*. Para a ETarch, são definidas as ações *etarch_SetSpec_Group* e *toControllerETArc* no atributo. E para a NovaGenesis, *action* é definido com as ações *ng_SetSpec* e *toControllerNG*. O Código-Fonte 9 apresenta a definição das tabelas das arquiteturas suportadas pela solução.

O *Programa de Controle* é definido dentro do *apply*, no *Ingresso*. Na figura 34 é apresentando o diagrama de atividades com o fluxo de controle do *Programa de Controle*.

O Código-Fonte 10 apresenta como as regras do *Programa de Controle* foram implementadas no *apply* do *Ingresso* em P4.

Por fim, é executada a reconstrução de pacote pelo *Agrupador*, que reagrupa os campos dos protocolos extraídos pelo *Analisador*, em um novo pacote a ser enviado pela porta de saída determinada pela ação de encaminhamento. A implementação do reagrupamento dos campos dos protocolos é mostrada no Código-Fonte 11.

Ainda na *Camada de Comutação*, também é implementado em Python o *Serviço de Tratamento de Regras*. Este processo tem como objetivo: receber pacotes de controle de *Adição de Regra de Encaminhamento*, encaminhados pelos controladores; aplicar as regras de encaminhamento na tabela de regras de encaminhamento da arquitetura específica e gerar e encaminhar ao controlador que enviou o pacote de *Adição de Regra de Encaminhamento*, um pacote de controle de *Resultado de Adição de Regra de Encaminhamento*, informando o controlador do *status* e do *identificador* da operação realizada na

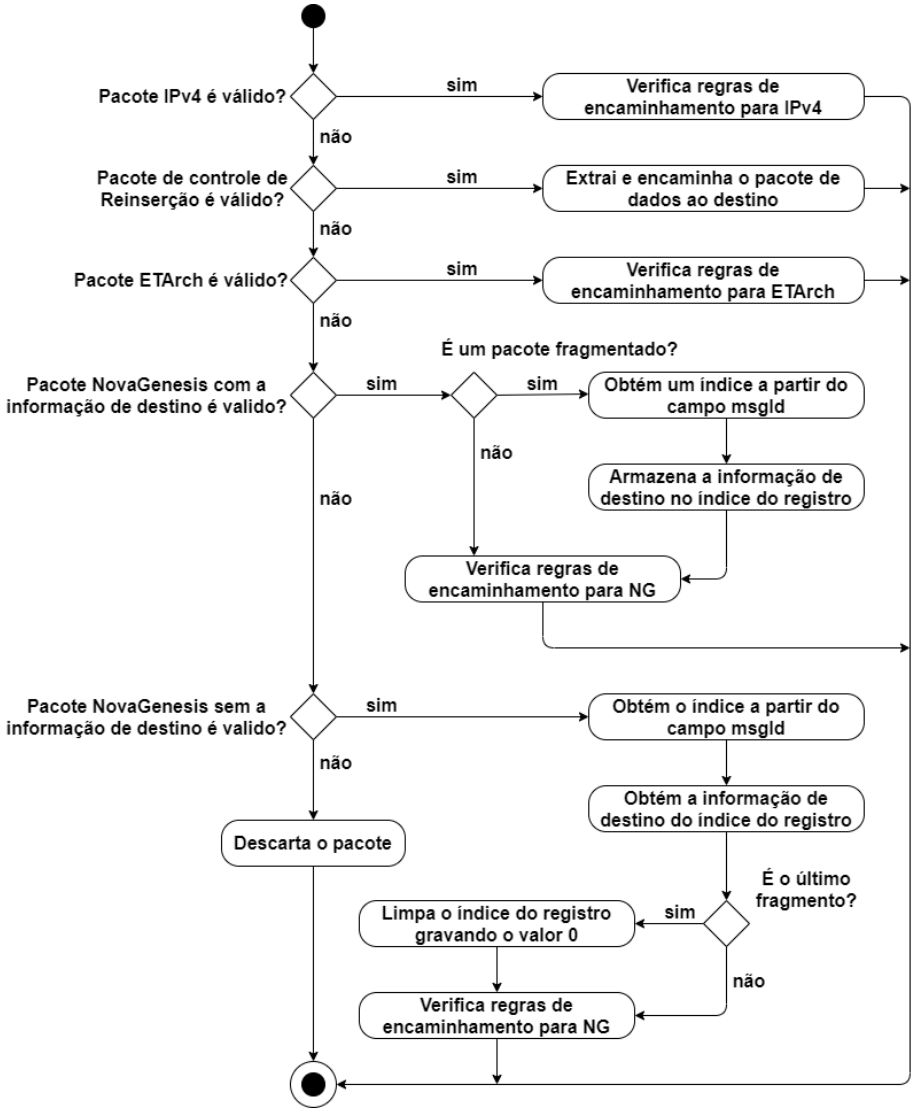


Figura 34 – Diagrama de Atividades do Programa de Controle do comutador.

tabela de regras de encaminhamento. O *Serviço de Tratamento de Regras* executa o método *sniff* da biblioteca Scapy do Python, ouvindo a interface de rede definida por *I_FACE*. Assim que um pacote é recebido na interface de rede esperada, é executando método *handle_pkt*, que realiza a validação do pacote através da chamada do método *valida-ArquiteturaEncapsulada*. Com o pacote validado positivamente, é executado o método *runCommandP4*, que cria uma nova regra de encaminhamento, no padrão do *P4 Runtime* e executa, através de processo do sistema operacional, o programa *pi_CLI_bmv2*, com os parâmetros informados pelo controlador, para realizar a aplicação da nova regra de encaminhamento. Após a execução da aplicação da regra de encaminhamento, é executado o método *primitiveSend*, que cria um novo pacote de controle FIXP de *Resultado de Adição de Regra de Encaminhamento* com as informações da execução do *pi_CLI_bmv2*, enviando o pacote de *Resultado de Adição de Regra de Encaminhamento* pela interface de rede definida por *I_FACE*. A implementação Python dos principais métodos do *Serviço de Tratamento de Regras* é apresentada no Código-Fonte 12.

4.1.2 Camada de Abstração

A *Camada de Abstração*, foi definida em um servidor virtual, com 10 gigabytes de espaço em disco, 256 megabytes de memória, com Sistema Operacional Linux Debian 10 Server de 64 bits. Nesta máquina virtual foram instaladas a linguagem Python 2.7.5 e a biblioteca Scapy (SCAPY-PACKET CRAFTING FOR PYTHON2 AND PYTHON3, 2020), para geração e manipulação de pacotes em Python.

Nesta máquina virtual foram configuradas cinco interfaces de rede, uma para conexão com o controlador IP, uma para conexão com o controlador ETArch, uma para conexão com o controlador NovaGenesis, uma para conexão com a *Camada de Comutação* e uma para realização de conexão ssh através do *host* para troca de arquivos com a máquina virtual da *Camada de Abstração*.

Esta camada foi definida com o objetivo de intermediar a troca de pacotes entre as camadas de *Comutação* e de *Controle*, agindo como um roteador, possibilitando assim a utilização de apenas uma interface de rede do comutador programável multi arquitetura para a troca de pacotes com os controladores.

Para realizar esse roteamento, a camada possui dois processos, o *Manipulador de Pacotes do Comutador*, que recebe os pacotes através da interface de rede com o comutador, identifica o tipo da arquitetura do pacote e direciona o pacote para a interface de rede do controlador da arquitetura. E o processo *Manipulador de Pacotes do Controlador*, responsável pelo recebimento de pacotes das interfaces de rede dos controladores, identificando e validando o tipo da arquitetura dos pacotes, encaminhando-os para a interface de rede do comutador.

Para realizar a tarefa de redirecionamento de pacotes vindos da interface de rede conectada ao comutador, implementou-se, em Python, o processo denominado *Manipulador de Pacotes do Comutador* que, através do método *sniff* da biblioteca Scapy do Python, ouve a interface de rede conectada ao comutador e ao receber um pacote, invoca o método *handle_pkt* que realiza a identificação da arquitetura do pacote e o direcionamento deste à interface de rede onde se encontra conectado o controlador da respectiva arquitetura. A implementação do *Manipulador de Pacotes do Comutador* é mostrada no Código-Fonte 13.

Para redirecionar os pacotes recebidos pelas diferentes interfaces de rede dos controladores, também foi implementado em Python o processo *Manipulador de Pacotes do Controlador*, que executa o método *sniff*, informando ao parâmetro *iface* uma lista de interfaces de rede, possibilitando assim ouvir as interfaces de rede de cada um dos controladores existentes. Assim que um pacote é recebido por qualquer uma das interfaces monitoradas, é validado o tipo da arquitetura do pacote e em caso de um pacote válido, o mesmo é encaminhado para a interface de de rede conectada à *Camada*

de Comutação. A implementação deste processo é apresentada no Código-Fonte 14. A Figura 35, é uma representação da *Camada de Abstração*, apresentando as interfaces de rede e os processos existentes nesta camada.

A versão atual da prova de conceito, define diretamente no código, o relacionamento entre o nome das interfaces de rede e o tipo da arquitetura a ela conectada. Por exemplo, é definido na implementação dos processos desta camada que a interface de rede *eth1* se conecta a interface de rede do controlador de tipo 0x0800 (valor hexadecimal do tipo do IP). Também são definidas no código estruturas de lista, da linguagem Python, que informam a relação de interfaces redes utilizadas pelos controladores.

Assim, novas versões da implementação da proposta podem trazer a implementação de estruturas dinâmicas para os processos de manipulação de pacotes, além da especificação e implementação de novas operações de controle do *Protocolo de Controle*, a fim de possibilitar a realização da configuração dinâmica dos relacionamentos entre nomes de interfaces de rede e o tipo da arquitetura a ela conectada, bem como a construção dinâmica das listas de interfaces de redes conectadas aos controladores.

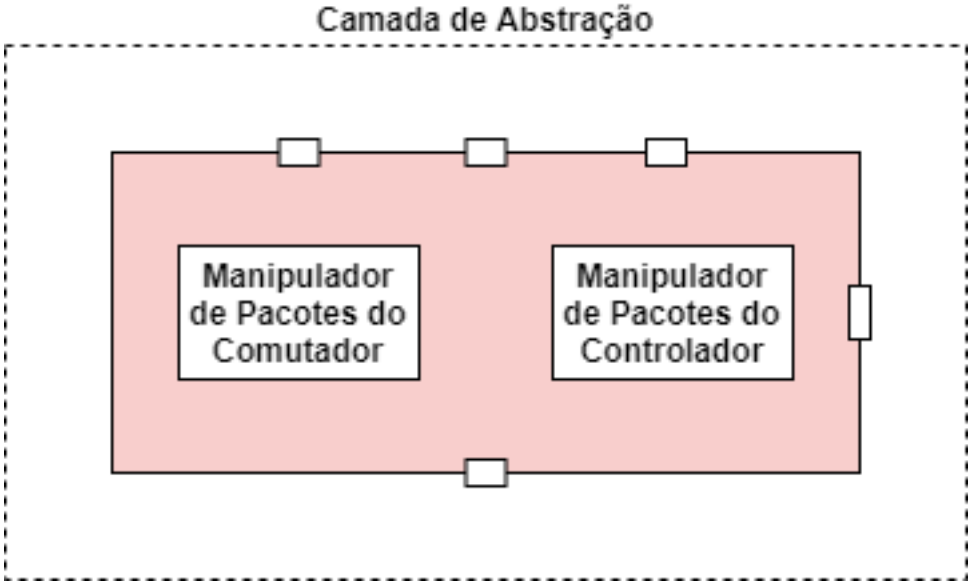


Figura 35 – Representação da *Camada de Abstração*.

4.1.3 Camada de Controle

A *Camada de Controle*, é o nome dado ao conjunto de máquinas virtuais definidas para realizar a execução de um controlador de uma das arquitetura de Internet suportadas. Três servidores virtuais foram criados, um para ETArch, um para NovaGenesis (que acabou não sendo utilizado por seu controlador ainda estar em desenvolvimento) e um para o IP, cada uma com 10 gigabytes de espaço em disco, 256 Megabytes de memória e com Sistema Operacional Linux Debian 10 Server de 64 bits. Nestas máquinas virtuais foram instaladas a linguagem Python 2.7.5 e a biblioteca Scapy (SCAPY-PACKET

CRAFTING FOR PYTHON2 AND PYTHON3, 2020), para geração e manipulação de pacotes em Python.

Em cada um dos servidores virtuais, foram configuras duas interfaces de rede, uma para conexão com a *Camada de Abstração*, para troca de pacotes com o comutador e uma interface de rede para realização de conexão ssh através do *host* para troca de arquivos com a máquina virtual do controlador.

Esta camada foi definida com o objetivo de intermediar a troca de pacotes entre as camadas de *Comutação* e *Controle*, como um roteador, possibilitando assim a utilização de apenas uma interface de rede do comutador para a troca de pacotes com os controladores.

O controlador da ETArch e o controlador do protocolo IP, criados para a execução desta prova de conceito, implementam as funcionalidades básicas para recebimento de pacotes, descritos na Seção 3.1, através do método *sniff*, da biblioteca Scapy do Python, para o recebimento de pacotes pela interface de rede, da geração de pacotes de controle de *Adição de Regra de Encaminhamento*, realizada pelo método *fixpPrimitiveSend* e a geração de pacotes de controle de *Reinserção de Pacote de Dados*, realizado pelo método *fixpPrimitiveSend*. O Código-Fonte 15 apresenta a implementação dos métodos descritos. A Figura 36, uma representação da *Camada de Comutação*, apresentando as interfaces de rede e os processos existentes nesta camada.

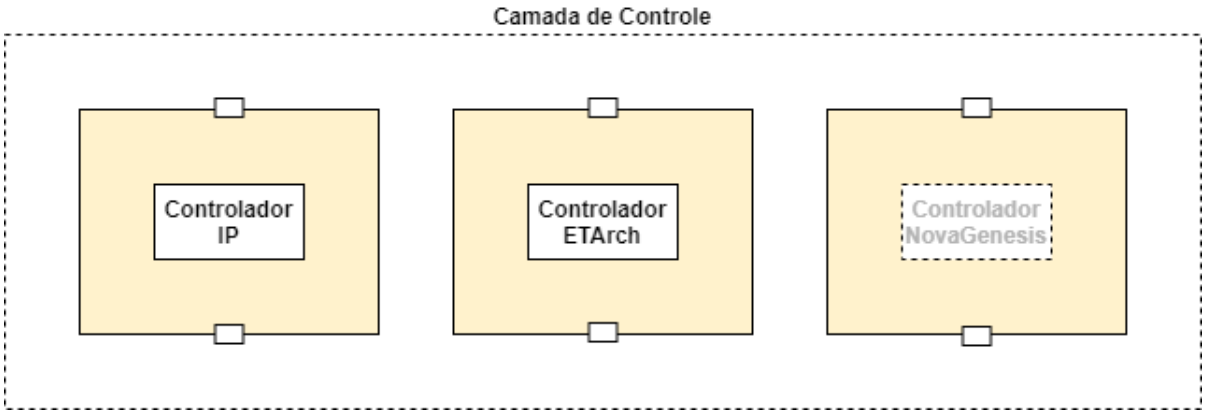


Figura 36 – Representação da *Camada de Controle*.

4.2 Validação e testes

Nesta seção, é apresentada a configuração do ambiente de testes, sua topologia, a especificação do hardware utilizado, o processo de validação do tratamento da fragmentação de pacotes da NovaGenesis, o processo de validação da prova de conceito e os resultados obtidos da execução de testes de tempo de processamento de pacotes de dados e de pacotes de controle de (*Adição de Regra de Encaminhamento*).

A NovaGenesis, mesmo não sendo utilizada na validação da prova de conceito e nos testes finais, teve seus dois cabeçalhos e o tratamento para o encaminhamento de pacotes fragmentados implementados no comutador programável multi arquitetura utilizado na prova de conceito e testes. Para a validação do tratamento da fragmentação de pacotes da NovaGenesis, um novo programa em Python foi criado, o *send.py*, que realizar o envio de pacotes com os dois tipos de cabeçalhos da NovaGenesis, um contendo a *linha de comando de roteamento*, com informação de destino do pacote e outro cabeçalho sem a informação de destino. Antes do início desta validação, o comutador programável apresenta a existência de uma regra de encaminhamento na tabela de regras da arquitetura NovaGenesis para o destino utilizado nos testes.

Ao executar o programa *send.py* em um computador virtual com interface de rede conectada ao comutador, informando como argumento na linha de execução o valor 0, é gerado e enviado um pacote NovaGenesis com o cabeçalho que possui a *linha de comando de roteamento*, que apresenta a informação de destino do pacote. Este pacote é então recebido e processado pelo *Comutador P4*, que armazena o valor do campo *dhid* na estrutura *register* do comutador e encaminha o pacote à interface de rede conectada ao destino.

Ao executar novamente o programa *send.py*, informando agora como argumento na linha de execução o valor 1, é gerado e enviado um pacote NovaGenesis com o cabeçalho que não apresenta a *linha de comando de roteamento*. Desta forma, este pacote é recebido e processado pelo *Comutador P4*, que recupera a informação de destino do pacote na estrutura *register* do comutador e encaminha o pacote à interface de rede conectada ao destino.

Dessa forma, foi possível observar o correto processamento e encaminhamento dos dois tipos de fragmentos de pacotes da NovaGenesis, realizado pelo comutador programável, apresentando assim suporte à arquitetura NovaGenesis.

A avaliação da prova de conceito e os testes foram realizados em um cenário virtualizado em que cada computador, controladores, a *Camada de Abstração* e o comutador programável multi arquitetura, foram executados em máquinas virtuais separadas, de forma individual.

O ambiente de validação e testes é composto por oito máquinas virtuais executadas pelo Oracle VM Virtual Box versão 6.0.14, rodando em um computador Dell Inspiron, com processador Intel i5-7400 de 3GHz, com 4 núcleos, 1 processo por núcleo, com 8 Gigabytes de memoria RAM e com Sistema Operacional Windows 10 Home de 64 *bits*. A topologia do ambiente pode ser observada na Figura 37.

O processo para validação da prova de conceito consistiu no envio de 1000 pacotes de dados de um computador *Computador A* para um *computador B*, com o

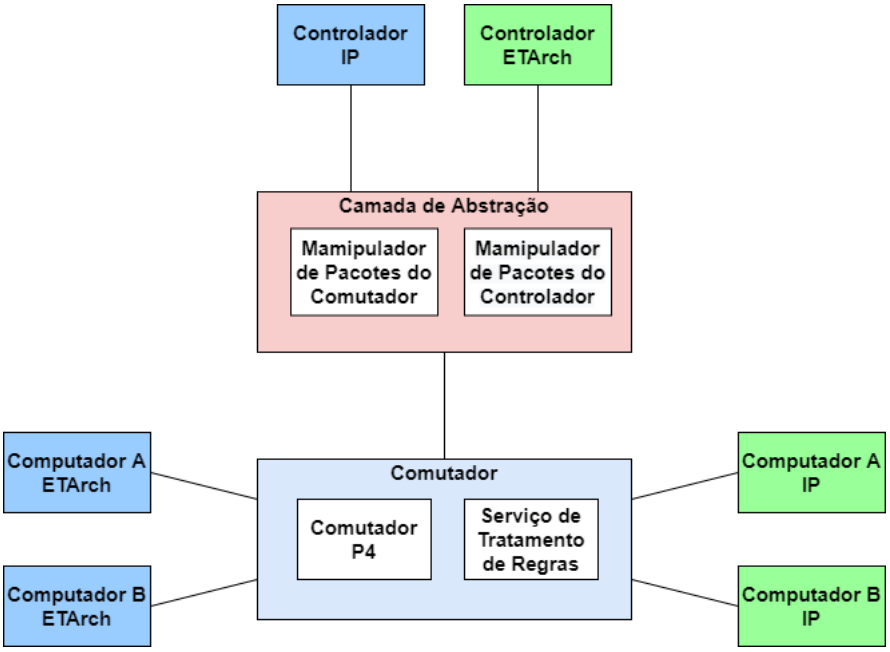


Figura 37 – Topologia de rede utilizada na validação e testes.

ambiente em um estado inicial onde o comutador têm as suas tabelas de regras de encaminhamento vazias, sem nenhuma regra de encaminhamento qualquer. Dessa forma, ao receber o primeiro pacote de dados, o comutador programável multi arquitetura identifica o tipo da arquitetura do pacote e verifica a inexistência de regras de encaminhamento para o pacote. Com a inexistência de regra de encaminhamento para o pacote, o comutador encaminha o pacote através da *Camada de Abstração* para o controlador responsável, para que o controlador gere um novo pacote de controle de *Adição de Regra de Encaminhamento* que será enviado através da *Camada de Abstração* ao comutador para que este aplique a regra em sua tabela de regras de encaminhamento. A partir do segundo pacote ao milésimo, como há regra de encaminhamento na tabela de regra de encaminhamento, os pacotes são recebidos pelo comutador, processados e encaminhados à interface de rede conectada ao destinatário.

A execução da validação da prova de conceito da proposta foi realizada em dois momentos. Inicialmente, foi realizado o envio de pacotes, de forma individual para o IP e depois para a ETArch. Em seguida, foram realizados envios de pacotes IP e ETArch de forma simultânea. A arquitetura NovaGenesis não foi utilizada durante o processo de validação e dos testes em razão do controlador da arquitetura NovaGenesis estar em desenvolvimento durante o período da realização da validação e da execução dos testes. A Figura 38 apresentada o fluxo dos pacotes durante o processo de validação da prova de conceito. A seguir, é descrita cada etapa do fluxo:

1. O *Computador A* envia um pacote de dados, com destino ao *Computador B*, ao comutador programável multi arquitetura.

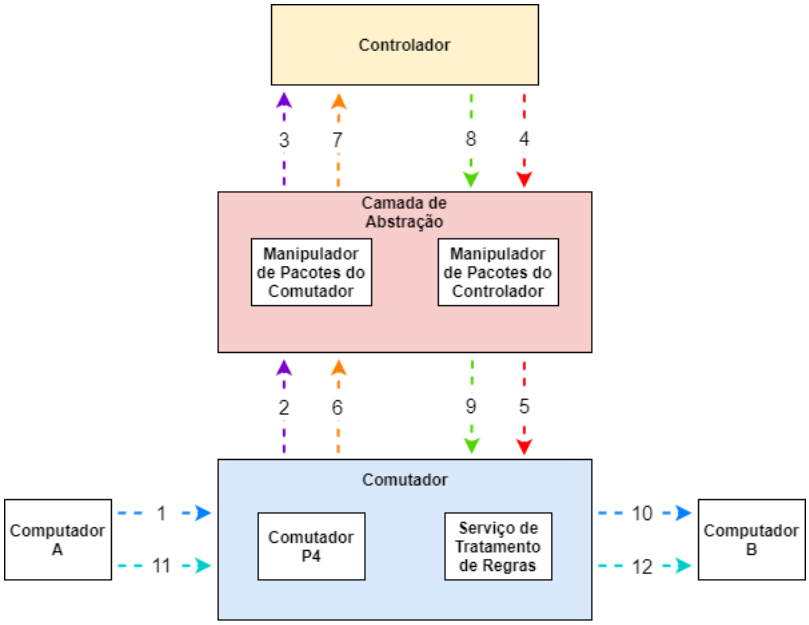


Figura 38 – Fluxo de pacotes durante a validação da prova de conceito.

2. O *Comutador P4* recebe e verifica o tipo da arquitetura do pacote e o encaminha aos controladores, através da interface de rede conectada à *Camada de Abstração*, por não possuir regra de encaminhamento;

3. O pacote é recebido pelo *Manipulador de Pacotes do Comutador* na *Camada de Abstração*, que identifica e valida o tipo da arquitetura do pacote e o direciona à interface de rede do controlador responsável pela arquitetura.

4. O controlador recebe o pacote de dados sem regra de encaminhamento e gera um novo pacote de controle de *Adição de Regra de Encaminhamento*, enviando-o ao comutador programável, através da *Camada de Abstração*.

5. O *Manipulador de Pacotes do Controlador* recebe o pacote de controle de *Adição de Regra de Encaminhamento*, identifica o tipo do pacote e o direciona à interface de rede do comutador;

6. O *Serviço de Tratamento de Regras* recebe o pacote, identifica o seu tipo e processa o pacote de controle de *Adição de Regra de Encaminhamento*. Um novo pacote de controle de *Resultado de Adição de Regra de Encaminhamento* é gerado pelo *Serviço de Tratamento de Regras* que o envia ao controlador, através da interface de rede conectada à *Camada de Abstração*.

7. O processo *Manipulador de Pacotes do Comutador* recebe o pacote de controle de *Resultado de Adição de Regra de Encaminhamento*, identifica o tipo do pacote e o direciona ao controlador da arquitetura.

8. O controlador recebe o pacote de controle de *Resultado de Adição de Regra de Encaminhamento*, armazena as informações de *status* e *identificador* da execução realizada e gera um novo pacote de controle de *Reinserção de Pacote de Dados* a partir do pacote de dados sem regra de encaminhamento recebido anteriormente.

9. O *Manipulador de Pacotes do Controlador* recebe o pacote de controle de *Reinserção de Pacote de Dados*, identifica o tipo do pacote e o direciona ao comutador programável;

10. O *Comutador P4* recebe o pacote de *Reinserção de Pacote de Dados*, identifica o tipo do pacote e encaminha o pacote para a interface de rede conectada ao *Computador B*.

11. O *Computador A* envia um novo pacote de dados ao comutador programável multi arquitetura, com destino ao *Computador B*

12. O pacote é recebido pelo *Comutador P4* que verifica o tipo da arquitetura do pacote, valida a existência de regra de encaminhamento para o pacote e o encaminha para a interface de rede conectada ao *Computador B*, que recebe o pacote de dados.

Ao final da execução da validação da prova de conceito, foi possível verificar nas três situações apresentadas, o recebimento correto de todos os 1000 pacotes de dados pelo computador de destino, confirmando assim a validação do processo de aplicação de regra de encaminhamento e de encaminhamento e pacotes da prova de conceito.

Após a execução da validação da prova de conceito, foram realizados testes para avaliar e verificar o comportamento dos pacotes IP e ETArch no ambiente. Para a execução do teste de verificação do tempo de encaminhamento de pacotes de dados, foram repetidas 30 vezes o envio de 100 pacotes de dados do IP e ETArch com de 500 *bytes*, 1000 *bytes* e 1500 *bytes*. Para a execução do teste de verificação do tempo de processamento de regra de encaminhamento, foram repetidas 30 vezes o envio de 100 pacotes de dados do IP e ETArch sem regra de encaminhamento, que resultam na geração e aplicação do pacote de dados de *Adição de Regra de Encaminhamento*.

É importante ressaltar que, não é intenção destes testes realizar comparações entre as arquiteturas de Internet, sendo o foco a verificação do desempenho do comutador programável multi arquitetura para diferentes tipos de cabeçalhos suportados testados (ETArch e IP) e tamanhos de pacote. A seguir são apresentados os resultados obtidos pelos testes.

A Figura 39 apresenta o tempo médio gasto em milissegundos pelo *Comutador P4* para realizar o processamento de encaminhamento de pacotes de dados.

Para este teste, a medição do tempo é iniciada com recebimento do pacote de dados na interface de rede, do comutador, conectada ao *Computador A* e finalizada quando o pacote é enviado pela interface de rede, do comutador, conectada ao *Computador B*.

Para a realização da medição dos tempos, foi criado um programa em Python chamado *timer.py* que utiliza biblioteca Scapy para o recebimento e manipulação de pacotes em Python. Esse programa inicializa duas linhas de execução, uma aguarda o recebimento de um pacote na interface de rede, do comutador, conectada ao *Computador A*, armazenando a informação da data e hora do recebimento em formato de milissegundos, enquanto a outra linha de execução aguarda o envio de um pacote na interface de rede, do comutador, conectada ao *Computador B*, armazenando a informação da data e hora do envio. Assim, é calculada a diferença entre a data e hora do envio com a data e hora do recebimento, obtendo o tempo gasto com o processamento do pacote, em milissegundos, que é impresso na tela.

Observando os resultados, pode-se ver que o processamento de pacotes de dados ETArch tiveram um desempenho melhor do que o processamento dos pacotes de dados IP. Provavelmente isso se deve à informação de endereçamento ETArch necessitar menor processamento, pois a informação é definida no protocolo *Ethernet*, enquanto que para o IP, o processamento de mais um protocolo, o IP, é necessário. Também pode-se concluir que, para o processamento de pacotes de dados de uma mesma arquitetura, em P4, o tamanho do pacote é irrelevante, sendo impactante no tempo de processamento a complexidade do cabeçalho, isto é, a quantidade de campos e protocolos a serem processados para a realização do encaminhamento.

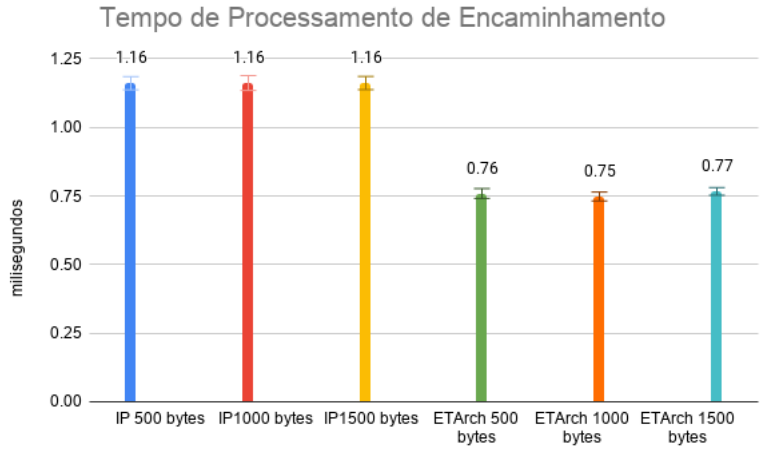


Figura 39 – Tempo de processamento de encaminhamento de pacote.

A Figura 40 apresenta o tempo médio de processamento para adicionar regras de encaminhamento no comutador programável multi arquitetura.

Para este teste, a medição do tempo é considerada a partir do momento em que

o pacote de dados sem regra de encaminhamento é recebido no comutador e finalizada após a aplicação da regra de encaminhamento pelo *Serviço de Tratamento de Regras*.

Para a medição dos tempos deste teste, também foi utilizado o programa timer.py, mas ele foi alterado para imprimir na tela o valor da data e hora, em milissegundos, do recebimento do pacote. O *Serviço de Tratamento de Regras* também foi alterado para imprimir na tela o valor da data e hora, em milissegundos, do fim da aplicação da nova regra de encaminhamento na tabela de regras. Assim, é calculada a diferença entre a data e hora do término da aplicação da nova regra de encaminhamento na tabela de regas, com a data e hora do recebimento do pacote, obtendo o tempo gasto com o processamento para adicionar novas regras de encaminhamento no comutador programável multi arquitetura.

É possível observar que o tempo médio de processamento de uma regra de encaminhamento pela ETArch é muito superior ao tempo de processamento do IP. Isso se deve ao fato de a ETArch enviar três pacotes de controle internos à arquitetura, para executar o registro da entidade em um *workspace*, enviando depois o pacote de controle de *Adição de Regra de Encaminhamento* ao *Serviço de Tratamento de Regras* para realizar a inserção da nova regra de encaminhamento.

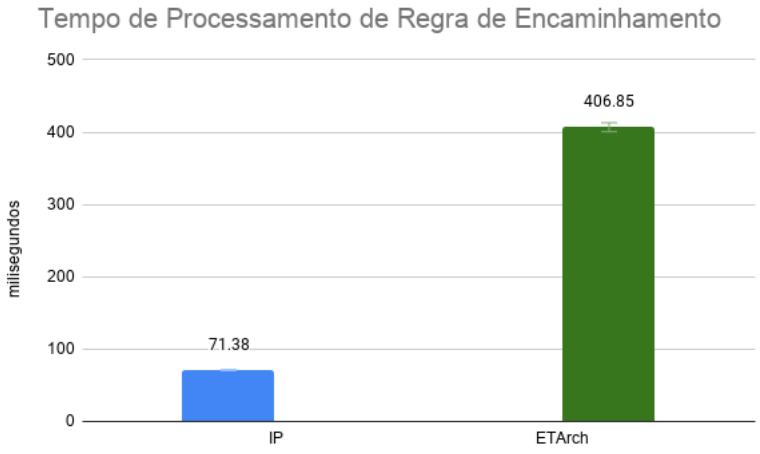


Figura 40 – Tempo de processamento de regra de encaminhamento.

Conclusão

Com a necessidade de melhorias da atual arquitetura de rede em questões como segurança, controle de tráfego, endereçamento, mobilidade, qualidade de serviço, entre outras, diversas iniciativas voltadas para as pesquisas de novas arquiteturas de Internet, denominadas Arquiteturas de Internet do Futuro, afloraram pelo globo. Devido a grande quantidade de pesquisas emergentes e de suas diferentes abordagens e objetivos, identificar e eleger uma nova arquitetura que possa vir a se tornar a nova arquitetura de Internet é uma tarefa realmente complexa. Outro ponto importante a ser considerado é a inviabilidade da substituição completa da atual infraestrutura de redes por uma nova. Dessa forma, a integração de novas arquiteturas de Internet à infraestrutura atual se apresenta como uma solução viável para a evolução da rede.

Assim, através da proposta apresentada neste trabalho, a coexistência de múltiplas Arquiteturas de Internet do Futuro com a atual arquitetura de Internet baseada em *Ethernet*, torna-se possível, possibilitando assim a comunicação entre entidades pertencentes a uma mesma arquitetura de Internet.

Para isso, o estudo e desenvolvimento de um novo comutador definido por software, que possibilita o encaminhamento de pacotes, tanto das novas Arquiteturas de Internet do Futuro quanto da arquitetura atual, foi realizado.

Em paralelo, foram realizadas pesquisas voltadas às estruturas dos cabeçalhos e do funcionamento das arquiteturas ETArch e NovaGenesis, para a implementação do encaminhamento de pacotes destas pelo novo comutador desenvolvido.

O desenvolvimento de processos complementares, como o *Serviço de Tratamento de Regras*, os *Manipuladores de Pacotes* da *Camada de Abstração*, a implementação de versões funcionais dos controladores da arquitetura ETArch e IP, além da definição de um novo protocolo de controle, possibilitaram a criação de uma prova de conceito da proposta.

Com a validação da prova de conceito da proposta, pôde-se observar a integração entre seus elementos como esperado, comprovando assim o funcionamento da proposta.

Assim, com a implementação e validação da prova de conceito da proposta, que possibilita a coexistência de múltiplas arquiteturas de Internet do Futuro com arquitetura atual, é criado o ambiente heterogêneo de arquiteturas de rede que torna possível realizar novos estudos e observações de como a interação de diferentes arquiteturas pode contribuir para a resolução dos desafios impostos pelas limitações da Internet atual e quais as mudanças devem ser realizadas por parte das operadores de redes que desejem diferenciar suas funcionalidades dos demais concorrentes de mercado

e realizar a implantação da proposta.

Até a conclusão desta dissertação, a especificação da proposta seguiu evoluindo, não sendo possível a realização da execução e testes externos ao ambiente de implementação e validação citados. Desta forma, como sugestão para trabalhos futuros, podem ser apontadas: a adição de uma versão inicial do controlador da arquitetura NovaGenesis à proposta e a execução da validação e testes no ambiente *FIBRE* (*Future Internet Brazilian Environment for Experimentation*) (FIBRE, 2020).

Publicações

Jose Gavazza, Juliano Melo, Thiago Silva, Antonio M. Alberti, Pedro Frosi Rosa, Flávio de Oliveira Silva, Fábio Luciano Verdi, Jose Augusto Suruagy Monteiro. Future Internet Exchange Point (FIXP): Enabling Future Internet Architectures Interconnection. In The Springer 34th International Conference on Advanced Information Networking and Applications (AINA-2020).

Jose Augusto T. Gavazza, Michael dos Santos, Paulo Ditarso Maciel Jr., Fábio Luciano Verdi, Jose A. S. Monteiro. Implementação de um switch em P4 com suporte simultâneo a múltiplas arquiteturas de Internet do Futuro. 10th Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF), SBRC 2019.

Referências

4D PROJECT. 2020. Http://www.cs.cmu.edu/ 4D/. [Online; acessado em 07-April-2020]. Citado na página 8.

AGER, B. et al. Anatomy of a large european ixp. In: *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. [S.l.: s.n.], 2012. p. 163–174. Citado na página 16.

AHLGREN, B. et al. A node identity internetworking architecture. In: IEEE. *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. [S.l.], 2006. p. 1–6. Citado na página 32.

ALBERTI, A. M. et al. Naming and name resolution in the future internet: Introducing the novagenesis approach. *Future Generation Computer Systems*, v. 67, p. 163 – 179, 2017. ISSN 0167-739X. Citado 2 vezes nas páginas 2 e 4.

ALBERTI, A. M. et al. Forwarding/routing with dual names: The novagenesis approach. In: SBC. *Anais do IX Workshop de Pesquisa Experimental da Internet do Futuro*. [S.l.], 2018. Citado 2 vezes nas páginas 13 e 36.

ALBERTI, A. M. et al. A novagenesis proxy/gateway/controller for openflow software defined networks. In: IEEE. *10th International Conference on Network and Service Management (CNSM) and Workshop*. [S.l.], 2014. p. 394–399. Citado na página 32.

ANAND, A. et al. Xia: An architecture for an evolvable and trustworthy internet. In: *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. [S.l.: s.n.], 2011. p. 1–6. Citado na página 32.

ANDERSEN, D. G. et al. Accountable internet protocol (aip). In: *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. [S.l.: s.n.], 2008. p. 339–350. Citado na página 32.

ANDERSON, T. et al. Nebula-a future internet that supports trustworthy cloud computing. Citeseer, 2010. Citado na página 32.

BEHAVIORAL MODEL. 2020. Https://github.com/p4lang/behavioral-model. [Online; acessado em 07-April-2020]. Citado na página 53.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, ACM, v. 44, n. 3, p. 87–95, 2014. Citado 3 vezes nas páginas 13, 10 e 11.

CAMERA, P. E.; ZANETTI, A. B. Introdução à linguagem de programação p4, o futuro das redes. 2019. Citado na página 10.

CHATZIS, N.; SMARAGDAKIS, G.; FELDMANN, A. On the importance of internet exchange points for today's internet ecosystem. *arXiv preprint arXiv:1307.5264*, 2013. Citado 3 vezes nas páginas 13, 16 e 18.

CHERITON, D. R.; GRITTER, M. Triad: A new next-generation internet architecture. Citeseer, 2000. Citado na página 32.

CLARK, D. et al. Fara: Reorganizing the addressing architecture. In: *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*. [S.l.: s.n.], 2003. p. 313–321. Citado na página 32.

CLARK, D. et al. *New arch: Future generation internet architecture*. [S.l.], 2004. Citado na página 2.

CLARK, D. et al. *Towards the Future Internet Architecture*. [S.l.], 1991. Citado na página 2.

CLARK, D. D. et al. Tussle in cyberspace: defining tomorrow's internet. *IEEE/ACM transactions on networking*, IEEE, v. 13, n. 3, p. 462–475, 2005. Citado na página 1.

CROWCROFT, J. et al. Plutarch: an argument for network pluralism. *ACM SIGCOMM Computer Communication Review*, ACM New York, NY, USA, v. 33, n. 4, p. 258–266, 2003. Citado na página 32.

DETTI, A. et al. Conet: a content centric inter-networking architecture. In: *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*. [S.l.: s.n.], 2011. p. 50–55. Citado na página 32.

DEVOLVED CONTROL OF ATM NETWORKS. 2020. Https://www.cl.cam.ac.uk/research/srg/netos/projects/archive/dcan/. [Online; acessado em 07-April-2020]. Citado na página 8.

DUTTA, R. et al. The silo architecture for services integration, control, and optimization for the future internet. In: IEEE. *2007 IEEE International Conference on Communications*. [S.l.], 2007. p. 1899–1904. Citado na página 32.

EDWALL, T.; TREMBLAY, B. Sail project. 2011. Citado na página 32.

ETHANE ARCHITECTURE. 2020. Http://yuba.stanford.edu/ethane/. [Online; acessado em 07-April-2020]. Citado na página 8.

FAPESP. 2020. Http://www.fapesp.br/. [Online; acessado em 01-July-2020]. Citado na página 4.

FEFERMAN, D. L. et al. Uma nova revolução em redes: Programação do plano de dados com p4. 2018. Citado na página 10.

FELDMANN, A. Internet clean-slate design: what and why? *ACM SIGCOMM Computer Communication Review*, ACM New York, NY, USA, v. 37, n. 3, p. 59–64, 2007. Citado na página 2.

FIA-NEXT PHASE. 2020. Https://www.nsf.gov/pubs/2013/nsf13538/nsf13538.htm. [Online; acessado em 07-April-2020]. Citado na página 2.

FIBRE. 2020. Http://fibre.org.br/. [Online; acessado em 07-April-2020]. Citado na página 72.

FRANCIS, P.; GUMMADI, R. Ipnl: A nat-extended internet architecture. In: *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. [S.l.: s.n.], 2001. p. 69–80. Citado na página 32.

GERMAN LAB. 2020. Https://www.german-lab.de/. [Online; acessado em 07-April-2020]. Citado na página 31.

GLOBAL ENVIRONMENT FOR NETWORKING INNOVATIONS. 2020. Https://www.geni.net/. [Online; acessado em 07-April-2020]. Citado na página 30.

GRASA, E. et al. Design principles of the recursive internetwork architecture (rina). 2011. Citado na página 2.

GUIMARãES, C. et al. Exploring interoperability assessment for future internet architectures roll out. *Journal of Network and Computer Applications*, Elsevier, v. 136, p. 38–56, 2019. Citado 3 vezes nas páginas 13, 24 e 25.

GUPTA, A. et al. An industrial-scale software defined internet exchange point. In: *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. [S.l.: s.n.], 2016. p. 1–14. Citado na página 20.

GUPTA, A. et al. Sdx: A software defined internet exchange. *ACM SIGCOMM Computer Communication Review*, ACM New York, NY, USA, v. 44, n. 4, p. 551–562, 2014. Citado na página 18.

GUPTA, A. et al. Sdx: A software defined internet exchange. *ACM SIGCOMM Computer Communication Review*, ACM, v. 44, n. 4, p. 551–562, 2015. Citado na página 19.

HARAI, H. Akari architecture design for new generation network. In: IEEE. *2009 IEEE/LEOS Summer Topical Meeting*. [S.l.], 2009. p. 155–156. Citado 2 vezes nas páginas 2 e 31.

KLOMP, J. v. L. J. Recursive internetwork architecture. 2016. Citado na página 2.

KOPONEN, T. et al. A data-oriented (and beyond) network architecture. In: *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*. [S.l.: s.n.], 2007. p. 181–192. Citado na página 32.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, Ieee, v. 103, n. 1, p. 14–76, 2014. Citado na página 7.

LUO, H. et al. Color: an information-centric internet architecture for innovations. *IEEE Network*, IEEE, v. 28, n. 3, p. 4–10, 2014. Citado na página 32.

MARTINEZ-JULIA, P.; SKARMETA, A. F.; GALIS, A. Towards a secure network virtualization architecture for the future internet. In: SPRINGER. *The Future Internet Assembly*. [S.l.], 2013. p. 141–152. Citado na página 32.

MATTA, A.; BOSTON, U. Recursive internetwork architecture. 2010. Citado 2 vezes nas páginas 2 e 32.

MONTEIRO, J. A. S. Leveraging the future internet in brazil through the coexistence and interconnection of multiple architectures. p. 1–19, 2015. Citado 3 vezes nas páginas 1, 4 e 39.

MOREIRA, M. D. et al. Internet do futuro: Um novo horizonte. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC*, v. 2009, p. 1–59, 2009. Citado na página 1.

NATIONAL SCIENCE FOUNDATION. 2020. Https://www.nsf.gov/. [Online; acessado em 07-April-2020]. Citado 2 vezes nas páginas 15 e 30.

NEXT GENERATION INTERNET. 2020. Https://www.ngi.eu/. [Online; acessado em 07-April-2020]. Citado na página 2.

NIEBERT, N. et al. The way 4ward to the creation of a future internet. In: IEEE. *2008 IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications*. [S.l.], 2008. p. 1–5. Citado na página 32.

NSF FIA PROJECT. 2020. Http://www.nets-fia.net/. [Online; acessado em 07-April-2020]. Citado 2 vezes nas páginas 2 e 30.

NSF NETS FIND INITIATIVE. 2020. Http://www.nets-find.net/. [Online; acessado em 07-April-2020]. Citado 2 vezes nas páginas 2 e 30.

NUNES, B. A. A. et al. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, IEEE, v. 16, n. 3, p. 1617–1634, 2014. Citado 2 vezes nas páginas 13 e 9.

ONE LAB. 2020. Https://www.onelab.eu/. [Online; acessado em 07-April-2020]. Citado na página 31.

OPEN ALLIANCE SIG. 2020. Http://www.opensig.org/. [Online; acessado em 07-April-2020]. Citado na página 7.

OPEN NETWORKING FOUNDATION. 2020. Https://www.opennetworking.org/. [Online; acessado em 07-April-2020]. Citado na página 8.

P4 Runtime. 2019. <https://p4.org/p4-runtime/>. [Online; accessed 09-December-2019]. Citado na página 53.

PACKET CLEARING HOUSE. 2020. Https://www.pch.net/ixp/dir. Citado na página 19.

PAN, J.; PAUL, S.; JAIN, R. A survey of the research on future internet architectures. *IEEE Communications Magazine*, IEEE, v. 49, n. 7, 2011. Citado 2 vezes nas páginas 21 e 29.

PLANET LAB. 2020. Https://www.planet-lab.org/. [Online; acessado em 07-April-2020]. Citado na página 31.

REN, J. et al. Vicn: a versatile deployment framework for information-centric networks. *IEEE Network*, IEEE, v. 28, n. 3, p. 26–34, 2014. Citado na página 21.

RFC 4741-NETCONF. *Dezembro de*, 2006. Citado na página 8.

SÄRELÄ, M.; RINTA-AHO, T.; TARKOMA, S. Rtfm: Publish/subscribe internetworking architecture. *ICT Mobile Summit*, Stockholm, v. 29, p. 73–82, 2008. Citado na página 32.

SCAPY-PACKET CRAFTING FOR PYTHON2 AND PYTHON3. 2020. Https://scapy.net/. [Online; acessado em 07-April-2020]. Citado 3 vezes nas páginas 53, 61 e 63.

SCHMID, S. et al. Turfnet: An architecture for dynamically composable networks. In: SPRINGER. *Workshop on Autonomic Communication*. [S.l.], 2004. p. 94–114. Citado na página 32.

SESKAR, I. et al. Mobilityfirst future internet architecture project. In: *Proceedings of the 7th Asian Internet Engineering Conference*. [S.l.: s.n.], 2011. p. 1–3. Citado na página 32.

SILVA, F. de O. et al. On the analysis of multicast traffic over the Entity Title Architecture. In: *2012 18th IEEE International Conference on Networks (ICON)*. [S.l.: s.n.], 2012. p. 30–35. Citado 3 vezes nas páginas 2, 4 e 32.

SOUSA, N. F. S. de; ROTHENBERG, C. E. Softwarização em redes: Do plano de dados ao plano de orquestração. 2017. Citado na página 10.

WANG, S. et al. On adapting http protocol to content centric networking. In: *Proceedings of the 7th international conference on future internet technologies*. [S.l.: s.n.], 2012. p. 1–6. Citado na página 21.

WHEELER, D. C.; O'KELLY, M. E. Network topology and city accessibility of the commercial internet. *The Professional Geographer*, Taylor & Francis, v. 51, n. 3, p. 327–339, 1999. Citado na página 16.

WU, G.; MIZUNO, M.; HAVINGA, P. J. Mirai architecture for heterogeneous network. *IEEE Communications Magazine*, IEEE, v. 40, n. 2, p. 126–134, 2002. Citado na página 32.

WU, H. et al. On incremental deployment of named data networking in local area networks. In: IEEE. *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. [S.l.], 2017. p. 82–94. Citado 2 vezes nas páginas 13 e 22.

YANG, X.; CLARK, D.; BERGER, A. W. Nira: a new inter-domain routing architecture. *IEEE/ACM Transactions On Networking*, IEEE, v. 15, n. 4, p. 775–788, 2007. Citado na página 32.

ZHANG, H. et al. Smart identifier network: A collaborative architecture for the future internet. *IEEE network*, IEEE, v. 30, n. 3, p. 46–51, 2016. Citado 2 vezes nas páginas 2 e 32.

ZHANG, L. et al. Named data networking (ndn) project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, Citeseer, v. 157, p. 158, 2010. Citado na página 32.

APÊNDICE A – Códigos Fonte

```
1  header ethernet_t {
2    macAddr_t dst;
3    macAddr_t src;
4    bit<16>   type;
5  }
```

Código-Fonte 1 – Cabeçalho Ethernet.

```
1  header ipv4_t {
2    bit<4>    version;
3    bit<4>    ihl;
4    bit<8>    diffserv;
5    bit<16>   totalLen;
6    bit<16>   identification;
7    bit<3>    flags;
8    bit<13>   fragOffset;
9    bit<8>    ttl;
10   bit<8>    protocol;
11   bit<16>   hdrChecksum;
12   ip4Addr_t srcAddr;
13   ip4Addr_t dstAddr;
14 }
```

Código-Fonte 2 – Cabeçalho IP.

```
header novagenesis_hasDHID_t {
  bit<32>   rsvd;
  bit<32>   msgId;
  bit<32>   fraqSeq;
  bit<64>   msgSize;
  bit<672>  iniMsg;
  bit<64>   dhid;
  bit<1544> endMsg;
}

header novagenesis_hasntDHID_t {
  bit<32>   rsvd;
  bit<32>   msgId;
  bit<32>   fraqSeq;
  bit<64>   msgSize;
  bit<2280> msg;
}
```

Código-Fonte 3 – Cabeçalhos NovaGenesis.

```
header etarch_t {
  bit<368>  cabEtarch;
}

header etarch_switch_t {
  bit<8>    responsePrimitiveType;
  bit<16>   switchEtarch;
  bit<16>   portEtarch;
}

header etarch_switch_2_t {
  bit<328>  cabEtarch;
}
```

Código-Fonte 4 – Cabeçalhos ETArch.

```
register<bit<64>>((bit<32>)1024) hash_map;
```

Código-Fonte 5 – Estrutura para armazenamento temporário da informação de destino de pacote NovaGenesis.

```
state start {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.type,
                      hdr.ethernet.dst,
                      hdr.ethernet.src) {
        (0x800,_,_)          : parse_ipv4;
        (0x1234,_,_)         : parse_ng;
        (0x0900,0x464958500000,_) : accept;
        (0x0900,_,0x464958500000) : accept;
        (0x0900,_,_)         : parse_fixp_switch;
        (0x0880,0x445453000000,_) : parse_etarch;
        (0x0880,0xffffffffffff,_) : parse_etarch_switch;
        (0x0880,_,_)         : parse_etarch;
  }
}
```

Código-Fonte 6 – Extração do protocolo Ethernet e verificação dos campos Ethertype, MAC destino e MAC origem.

```
1  state parse_ng {
2      bit<96> hasDHID = packet.lookahead<bit<96>>();
3
4      transition select (hasDHID & 0xffffffff) {
5        0: novagenesis_hasDHID_parse;
6        default: novagenesis_hasntDHID_parse;
7      }
8    }
9
10   state novagenesis_hasDHID_parse {
11     packet.extract(hdr.novagenesis_hasDHID);
12     transition accept;
13   }
14
15   state novagenesis_hasntDHID_parse {
16     packet.extract(hdr.novagenesis_hasntDHID);
17     transition accept;
18   }
```

Código-Fonte 7 – Identificação tipo de cabeçalho NovaGenesis.

```
1  action toControllerIP() { standard_metadata.egress_spec = 1; }
2
3  action toControllerETArch() {
4    if(hdr.etarch_switch.isValid()) {
5      if((hdr.etarch_switch.responsePrimitiveType == P4_SWITCH) &&
6         (hdr.etarch_switch.switchEtarch == P4_SWITCH_ID)) {
7        standard_metadata.egress_spec = (bit <9>)(hdr.etarch_switch.
           portEtarch);
8      } else standard_metadata.egress_spec = P4_DROP_PORT;
9    } else if(hdr.etarch.isValid()) standard_metadata.egress_spec = 1;
10 }
11
12 action toControllerNG() {
13   if(hdr.novagenesis_hasDHID.isValid()) {
14     hdr.novagenesis_hasDHID.rsvd = hdr.novagenesis_hasDHID.rsvd + (
         bit<32>)standard_metadata.ingress_port + 255;
15   } else if(hdr.novagenesis_hasntDHID.isValid()) {
16     hdr.novagenesis_hasntDHID.rsvd = hdr.novagenesis_hasntDHID.rsvd +
          (bit<32>)standard_metadata.ingress_port + 255;
17   }
18
19   standard_metadata.egress_spec = 1; }
20
21 action toMakePacketOut() {
22   if((hdr.raw_switch.responsePrimitiveType == P4_SWITCH) &&
23      (hdr.raw_switch.switchEtarch == P4_SWITCH_ID)) {
24     hdr.ethernet.type = 0x0800;
25     standard_metadata.egress_spec = (bit <9>)(hdr.raw_switch.
         portEtarch);
26   } else standard_metadata.egress_spec = P4_DROP_PORT;
27 }
28
29 action etarch_SetSpec_Group(mcastGroup_t mcastGroup) {
30   standard_metadata.mcast_grp = mcastGroup; }
31
32 action ng_SetSpec(egressSpec_t port) {
33   standard_metadata.egress_spec = port; }
34
35 action ipv4_SetSpec(egressSpec_t port) {
36   standard_metadata.egress_spec = port; }
37
38 action drop() {
39   standard_metadata.egress_spec = P4_DROP_PORT; }
```

Código-Fonte 8 – Ações de regra de encaminhamento implementadas para IP, ETArch e NovaGenesis.

```
table ipv4_table {
  key = { hdr.ipv4.dstAddr: exact; }

  actions = {
    ipv4_SetSpec;
    toControllerIP;
  }

  size = 1023;

  default_action = toControllerIP();
}

table etarch_table {
  key = { hdr.ethernet.dst: exact; }

  actions = {
      etarch_SetSpec;
    etarch_SetSpec_Group;
    toControllerEtharc;
  }

  size = 1023;

  default_action = toControllerEtharc();
}

table novagenesis_table {
  key = { metadata.dhid: exact; }

  actions = {
    ng_SetSpec;
    toControllerNG;
  }

  size = 1023;

  default_action = toControllerNG();
}
```

Código-Fonte 9 – Definição das Tabelas de encaminhamento das arquiteturas IP, ETArch e NovaGenesis.

```
apply {
    bit<10> index;

  if(hdr.ipv4.isValid()) ipv4_table.apply();
  else if(hdr.raw_switch.isValid()) toMakePacketOut();
  else if((hdr.ethernet.isValid()) &&
          (hdr.ethernet.type == 0x0880)) etarch_table.apply();
  else if(hdr.novagenesis_hasDHID.isValid()) {
    if((hdr.novagenesis_hasDHID.rsvd & 0xff) == 0x45) {
      hash(index, HashAlgorithm.crc32, (bit<10>)0, {hdr.
          novagenesis_hasDHID.msgId}, (bit<10>)1023);
      hash_map.write((bit<32>)index, hdr.novagenesis_hasDHID.dhid);
    }

    novagenesis_table.apply();
  } else if(hdr.novagenesis_hasntDHID.isValid()) {
    hash(index, HashAlgorithm.crc32, (bit<10>)0, {hdr.
        novagenesis_hasntDHID.msgId}, (bit<10>)1023);
    hash_map.read(metadata.dhid, (bit<32>)(index));

    if(metadata.dhid != 0 && (hdr.novagenesis_hasntDHID.rsvd & 0xff)
        == 0x46) {
      hash_map.write((bit<32>)index, 0);
    }

    novagenesis_table.apply();
  }
  else drop();
}
```

Código-Fonte 10 – Programa de Controle.

```
apply {
  packet.emit(hdr.ethernet);
  packet.emit(hdr.ipv4);
  packet.emit(hdr.etarch_switch_2);
  packet.emit(hdr.etarch);
  packet.emit(hdr.novagenesis_hasDHID);
  packet.emit(hdr.novagenesis_hasntDHID);
}
```

Código-Fonte 11 – Remontagem do pacote de saída.

```
1  def primitiveSend(listaResposta) :
2    dadoSerializado = '/*' + json.dumps(listaResposta) + '*/'
3
4    pkt = Ether(src='FIXP\x00\x00', dst='SWITCH', type=ARQUITETURA_FIXP
       )
5    pkt = pkt / Raw(load = dadoSerializado)
6    sendp(pkt, iface=I_FACE, verbose=False)
7
8  def validaArquiteturaEncapsulada(arquiteturaEncapsuladaP) :
9    if arquiteturaEncapsuladaP in [int(ARQUITETURA_IPV4,16),
10           int(ARQUITETURA_ETARCH,16),
11           int(ARQUITETURA_FIXP_S_H,16),
12           int(ARQUITETURA_NG,16),
13           int(ARQUITETURA_NG_SIMULACAO,16)] :
14     return True
15   return False
16
17 def handle_pkt(pkt):
18   if Ether in pkt:
19     if pkt[Ether].type == ARQUITETURA_FIXP and
20        pkt[Ether].dst == '46:49:58:50:00:00' :
21       rawData = json.loads(pkt[Raw].load[(pkt[Raw].load.find('/*')+2)
           :pkt[Raw].load.find('*/')])
22
23       if len(rawData) != 0 :
24         if len(rawData[0]) != 0 :
25           if ((validaSwitch(rawData)) and (
               validaArquiteturaEncapsulada(rawData[0][0]))):
26             (status, listaResposta) = runCommandP4(rawData)
27
28             primitiveSend(listaResposta)
29
30 def main():
31   sys.stdout.flush()
32   sniff(iface=I_FACE, prn = lambda x: handle_pkt(x))
```

Código-Fonte 12 – Serviço de Tratamento de Regras.

```
1  def handle_pkt(pkt):
2    if Ether in pkt:
3      if pkt[Ether].type == ARQUITETURA_IPV4 and pkt[Ether].src in
           MACS_REGISTRADOS:
4        sendp(pkt, iface=IFACE_4, verbose=False)
5
6      elif pkt[Ether].type == ARQUITETURA_ETARCH and pkt[Ether].dst ==
           '44:54:53:00:00:00' :
7        sendp(pkt, iface=IFACE_1, verbose=False)
8
9      elif pkt[Ether].type == ARQUITETURA_NG:
10       sendp(pkt, iface=IFACE_3, verbose=False)
11
12     elif pkt[Ether].type == ARQUITETURA_FIXP and pkt[Ether].src == '
           46:49:58:50:00:00' :
13       rawData = json.loads(pkt[Raw].load[(pkt[Raw].load.find('/*')+2)
             :pkt[Raw].load.find('*/')])
14
15       if rawData[0][0] == ARQUITETURA_IPV4 :
16         ifaceChoice = IFACE_4
17
18       elif rawData[0][0] == ARQUITETURA_ETARCH :
19         ifaceChoice = IFACE_1
20
21       elif rawData[0][0] == ARQUITETURA_NG :
22         ifaceChoice = IFACE_3
23
24       sendp(pkt, iface=ifaceChoice, verbose=False)
25
26 def main():
27   sys.stdout.flush()
28   sniff(iface='eth0', prn = lambda x: handle_pkt(x))
```

Código-Fonte 13 – Manipulador de Pacotes do Comutador.

```
1  def handle_pkt(pkt):
2      if Ether in pkt:
3          if pkt[Ether].type == ARQUITETURA_IPV4_FIXP and pkt[Ether].
               src in MACS_REGISTRADOS :
4              if (pkt.sniffed_on == I_FACE_4):
5          sendp(pkt, iface=I_FACE_0, verbose=False)   %
6
7      elif pkt[Ether].type == ARQUITETURA_ETARCH and pkt[Ether].src ==
           '44:54:53:00:00:00' :
8              if (pkt.sniffed_on == I_FACE_1):
9          sendp(pkt, iface=I_FACE_0, verbose=False)
10
11     elif pkt[Ether].type == ARQUITETURA_NG:
12             if (pkt.sniffed_on == I_FACE_3):
13             sendp(pkt, iface=I_FACE_0, verbose=False)
14
15     elif pkt[Ether].type == ARQUITETURA_FIXP and pkt[Ether].dst == '
           46:49:58:50:00:00' :
16       sendp(pkt, iface=I_FACE_0, verbose=False)
17
18     elif pkt[Ether].type == ARQUITETURA_FIXP :
19       sendp(pkt, iface=I_FACE_0, verbose=False)
20
21     pkt.show2()
22
23 def main():
24   sys.stdout.flush()
25   sniff(iface=[I_FACE_1, I_FACE_3, I_FACE_4], prn = lambda x:
         handle_pkt(x))
```

Código-Fonte 14 – Manipulador de Pacotes do Controlador.

```
1  def primitiveSend(pktP, arquiteturaEncapsuladaP, switchP, portaP) : #
       PACKET_OUT
2    pktAux = Ether(dst=pktP[Ether].dst, src=pktP[Ether].src, type=
       ARQUITETURA_FIXP)
3    pktAux = pktAux / Raw(load = arquiteturaEncapsuladaP + str(switchP)
        + str(portaP))
4    pktAux = pktAux / IP(dst=pktP[IP].dst, src=pktP[IP].src, proto=pktP
       [IP].proto)
5    pktAux = pktAux / UDP(sport=pktP[UDP].sport, dport=pktP[UDP].dport)
6    pktAux = pktAux / Raw(load = str(pktP[Raw].load))
7
8    pktP.show2()
9    pktAux.show2()
10
11   sendp(pktAux, iface=IFACE_P, verbose=False)
12
13 def fixpPrimitiveSend(sendDataP) :
14   jasonString = '/*' + json.dumps(sendDataP) + '*/'
15
16   pkt = Ether(dst='FIXP\x00\x00', src='IPCONT', type=ARQUITETURA_FIXP
       )
17   pkt = pkt / Raw(load = jasonString)
18
19   sendp(pkt, iface=IFACE_P, verbose=False)
```

Código-Fonte 15 – Métodos dos controladores para geração dos pacotes de *Adição de Regra de Encaminhamento* e *Reinserção de Pacote de Dados*.