Jackson Nunes da Silva

Implementação de plano de dados programável para a *eXpressive Internet Architecture* usando a Linguagem P4

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2020

Jackson Nunes da Silva

Implementação de plano de dados programável para a *eXpressive Internet Architecture* usando a Linguagem P4

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Área de Concentração: Redes de Computadores e Sistemas Distribuídos

Orientador: José Augusto Suruagy Monteiro

Recife
2020

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S586i Silva, Jackson Nunes da
Implementação de plano de dados programável para a *eXpressive Internet Architecture* usando a linguagem P4 / Jackson Nunes da Silva. – 2020.
103 f.: il., fig., tab.

Orientador: José Augusto Suruagy Monteiro.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.
Inclui referências e apêndices.

1. Redes de computadores. 2. Arquiteturas de rede. I. Monteiro, José Augusto Suruagy (orientador). II. Título.

004.6 CDD (23. ed.) UFPE - CCEN 2020 - 161

Jackson Nunes da Silva

Implementação de plano de dados programável para a *eXpressive Internet Architecture* usando a Linguagem P4

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 12/03/2020.

BANCA EXAMINADORA

Prof. Dr. Kelvin Lopes Dias
Centro de Informática / UFPE

Prof. Dr. Fabio Luciano Verdi
Universidade Federal de São Carlos / UFSCar

Prof. Dr. José Augusto Suruagy Monteiro
Centro de Informática / UFPE
(**Orientador**)

Dedico este trabalho aos meus pais, que sempre direcionaram seus esforços à formação dos filhos e são meus maiores exemplos e incentivadores.

AGRADECIMENTOS

Inicialmente, agradeço a Deus pelas bençãos na minha vida, pela saúde e força para persistir na busca por meus objetivos.

Aos meus pais, agradeço por todo o amor, dedicação e por sempre priorizarem a educação dos filhos; aos meus irmãos, pelo apoio em momentos importantes; e aos demais familiares, pela torcida para que eu obtivesse sucesso nessa trajetória.

Agradeço à minha esposa, Isadora, por todo o apoio durante a realização do mestrado, pela compreensão diante de algumas privações e pelo constante companheirismo, estando comigo em todos os momentos (principalmente os difíceis) que levaram a esta conquista.

Ao meu orientador, professor Dr. José Augusto Suruagy Monteiro, agradeço imensamente pelas oportunidades no decorrer do curso, pela confiança em mim depositada, por promover meu contato com o projeto de pesquisa FIXP e pelos direcionamentos no desenvolvimento da dissertação. Além disso, sou grato pela sua atenção e por todos os ensinamentos, sejam relacionados à realização do trabalho ou como exemplos de ser humano e profissional.

Meu agradecimento aos professores e alunos que integram o projeto FIXP, especialmente a José A. T. Gavazza, pelas indicações e esclarecimentos acerca da linguagem P4. Agradeço também aos professores, técnicos administrativos e colegas que tive a oportunidade de conhecer no Centro de Informática da UFPE. Assim como, à equipe da Diretoria de TI do IFPE, da qual faço parte, pelas palavras de incentivo.

Agradeço a todos os meus amigos, em especial: a Marco Eugênio, pelo apoio; a Carlos Gomes e Henrique Santos, pela colaboração; e a Bruno Viana, Dário Primo, Gilmar Brito, Manoel João, Meuse Nogueira e Paula Viviane, por sempre me encorajarem a fazer o mestrado.

Por fim, agradeço a todos aqueles que contribuíram, de forma direta ou indireta, para a concretização desta conquista.

"Os elementos arquitetônicos da Internet que levaram a seu grande sucesso são agora, paradoxalmente, a fonte de seus problemas mais graves." (KESHAV, 2018)

RESUMO

As pesquisas voltadas a propostas de Arquiteturas para a Internet do Futuro (*Future Internet Architecture* - FIA) visam minimizar os problemas e limitações da arquitetura atual, otimizar o atendimento às crescentes demandas da rede mundial e tratar de forma mais eficiente os desafios relacionados a questões como segurança, mobilidade e desempenho. Atualmente, há mais de uma dezena de projetos de novas arquiteturas para a *Internet*, que podem ser classificadas de acordo com a sua origem ou conforme a orientação principal na qual são baseadas (conteúdo, serviços, usuários, etc). Também é possível dividi-las entre as que buscam substituir totalmente a rede atual, a partir de uma proposta única, e as que têm como objetivo a coexistência com outros projetos de FIA. Nos últimos anos, aponta-se uma tendência de que essas propostas sejam estimuladas a atuar simultaneamente, em um contexto conhecido como pluralista. Com isso, as possibilidades de experimentação em grande escala oferecidas por *testbeds*, aliada ao crescimento de funções de virtualização e de alternativas para criação de dispositivos de rede, convergem para que novas arquiteturas possam conviver em paralelo e com a *Internet* atual. Nesse cenário de integração e perspectivas relacionadas às propostas que sugerem um redesenho da rede mundial está o projeto FIXP, que busca alavancar a Internet do Futuro no Brasil através da coexistência e interconexão de múltiplas arquiteturas de rede. Visando contribuir com o projeto FIXP, mediante a inclusão de uma nova arquitetura de rede no ambiente multi FIAs da iniciativa, este trabalho implementa o plano de dados de um elemento programável de encaminhamento de pacotes, utilizando a linguagem *Programming Protocol-independent Packet Processors* (P4) para aplicar os conceitos da *eXpressive Internet Architecture* (XIA). Assim, o trabalho demonstra a viabilidade do envio de pacotes XIP através de um dispositivo virtual de rede que possibilita o roteamento flexível de diferentes "*principals*", validando os processos de encaminhamento característicos do plano de dados da arquitetura XIA.

Palavras-chave: Internet do Futuro. Arquiteturas de Rede. Plano de Dados. XIA. P4.

ABSTRACT

Researches focused on proposals for the Future Internet Architectures (FIA) aim at minimizing problems and limitations of the current architecture, optimizing the service to the worldwide network growing demands as well as dealing more efficiently with challenges related to issues such as security, mobility and performance. At present, there are more than ten new architecture projects for the Internet, which can be classified according to their origin or to the principal they are based upon (content, services, users, etc.). It is also possible to group them into those that seek to completely replace the current network, based on a single proposal, and those which aim at the coexistence with other FIA projects. In recent years, there is a tendency to encourage these proposals to act simultaneously, in a context known as pluralist. Thereby, the possibilities for large-scale experimentation offered by the testbeds, combined with the growing number of virtualization functions and new options for the creation of network devices, converge so that new architectures can coexist, including the current Internet. Within this scenario of integration and expectations related to proposals that suggest a redesign of the worldwide network, there is the FIXP project, which seeks to leverage the Future Internet in Brazil through the coexistence and interconnection of multiple network architectures. Aiming at contributing with the FIXP project, through the inclusion of a new network architecture in the initiative multi-FIA environment, this work implements the data plane of a packet forwarding programmable element by using the Programming Protocol-independent Packet Processors (P4) language in order to apply the concepts of the eXpressive Internet Architecture (XIA). Thus, this work demonstrates the viability of sending XIP packets through a virtual network device that allows flexible routing of different principals, validating those typical forwarding procedures of the XIA architecture data plane.

Keywords: Future Internet. Network Architectures. Data Plane. XIA. P4.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

AD	*Autonomus Domain*
API	*Application Programming Interface*
ASIC	*Application Specific Integrated Circuits*
BMv2	*Behavioral Model version 2*
CCN	*Content-Centric Networking*
CID	*Content Identifier*
DAG	*Directed Acyclic Graphs*
DDoS	*Distributed Denial of Service*
ETArch	*Entity Title Architecture*
EXT4	*Fourth Extended Filesystem*
FIA	*Future Internet Architecture*
FIBRE	*Future Internet Brazilian Environment for Experimentation*
FIRE	*Future Internet Research and Experimentation*
FIXP	*Future Internet eXchange Point*
FPGA	*Field Programmable Gate Array*
GENI	*Global Environment for Network Innovations*
HID	*Host Identifier*
INATEL	Instituto Nacional de Telecomunicações
IoT	*Internet of Things*
IP	*Internet Protocol*
IPv4	*Internet Protocol version 4*
IPv6	*Internet Protocol version 6*
LTS	*Long Term Support*
MAC	*Media Access Control*
NAT	*Network Address Translation*
NDN	*Named Data Networking*
NFV	*Network Functions Virtualization*
NG	NovaGenesis
NID	*Network Identifier*
NSF	*National Science Foundation*

ONF	*Open Networking Foundation*
OSI	*Open System Interconnection*
P4	*Programming Protocol-independent Packet Processors*
P4C	*Programming Protocol-independent Packet Processors Compiler*
PISA	*Protocol-Independent Switch Architecture*
RINA	*Recursive InterNetwork Architecture*
RNP	Rede Nacional de Ensino e Pesquisa
SDN	*Software Defined Networking*
SDX	*Software Defined eXchange*
SID	*Service Identifier*
TCP	*Transmission Control Protocol*
UFPE	Universidade Federal de Pernambuco
UFSCar	Universidade Federal de São Carlos
UFU	Universidade Federal de Uberlândia
VETH	*Virtual Ethernet*
VM	*Virtual Machine*
XIA	*eXpressive Internet Architecture*
XID	*eXpressive Identifier*
XIP	*eXpressive Internet Protocol*

SUMÁRIO

1 INTRODUÇÃO

A *Internet* estabeleceu um marco na história da humanidade, alterando a maneira como as informações são disponibilizadas, diversificando significativamente as possibilidades de negócios, expandindo os meios de comunicação e modificando a cultura da sociedade. O crescimento exponencial do tráfego na *Internet* tem sido impulsionado, principalmente, pela popularização de dispositivos móveis, a disseminação do uso de sensores e a ascensão de conceitos como *Internet of Things* (IoT). De acordo com o último relatório publicado pela Cisco® (CISCO, 2020), o número de usuários da *Internet* em 2018 era de 3,9 bilhões e existiam 18,4 bilhões de dispositivos conectados. Sendo que a projeção para 2023 é de 5,3 bilhões de usuários e 29,3 bilhões de dispositivos conectados.

A arquitetura de rede utilizada hoje tem um grande desafio para atender às demandas geradas por este crescimento. Há mais de uma década, pesquisadores da área de redes questionam se a *Internet* atual possui os requisitos necessários para acompanhar essa evolução e, assim, têm investido em iniciativas que levem à implementação de redes com maiores recursos. MOREIRA et al. (2009), afirmam que a simplicidade e os problemas estruturais do modelo atual da *Internet* dificultam o atendimento às demandas relacionadas a escalabilidade, mobilidade, segurança e gerenciamento. Segundo PAUL et al. (2011), para resolver esses problemas, a *Internet* precisa ser redesenhada de modo que atenda às necessidades atuais e seja flexível o suficiente para responder aos requisitos futuros.

1.1 Contextualização

As propostas de Arquiteturas para a Internet do Futuro (*Future Internet Architecture* - FIA) estão cada vez mais presentes nas discussões e pesquisas da área de redes. Elas apontam alternativas à *Internet* atual, um projeto concebido há cinco décadas, que obteve sucesso na sua expansão e adesão, mas que não trazia em seu escopo original a intenção de compartilhar conteúdo para destinos diversos ou móveis, endereçar bilhões de *hosts* ou proteger dados em uma rede em expansão e cada vez mais importante para a sociedade.

O desenho inicial da *Internet*, que mantém a simplicidade da rede e deixa as tarefas complexas de processamento para os *hosts*, faz com que a camada de rede não acompanhe o desenvolvimento da camada de aplicações (HU et al., 2011). Desde a sua concepção, novas funções precisaram ser implementadas por meio de *patches*[1] sobre a arquitetura existente, tornando difícil o suporte às crescentes demandas por segurança, desempenho, distribuição de conteúdo e mobilidade através dessas mudanças incrementais (PAN et al., 2011).

[1] Em inglês, significa literalmente “remendo”. Programa de computador criado para atualizar ou corrigir um software de forma a melhorar sua usabilidade ou performance.

À medida que novas demandas surgiram, essas funções foram adicionadas à pilha da *Internet* para tratar as lacunas do projeto inicial, entre as quais estão NAT, *Firewall* e IPv6. Mais recentemente, como tentativas de torná-la mais flexível e programável, a partir do paradigma de *Software Defined Networking* (SDN), que permite a programação do plano de controle de dispositivos de rede (KREUTZ et al., 2015). Contudo, as limitações na base da arquitetura atual prejudicam a eficácia dessas soluções. Os elementos arquitetônicos da atual rede mundial, que levaram ao seu crescimento, são agora a fonte de seus problemas mais graves. Resolvê-los exigirá uma nova arquitetura da *Internet* (KESHAV, 2018).

Na última década, alguns projetos apresentaram propostas de novas arquiteturas para a *Internet*. Essas iniciativas, com abordagens ou metodologias diferentes e com origem em diversas partes do mundo, buscam melhorias em relação ao modelo atual baseado em TCP/IP. Dentre elas pode-se destacar, *MobilityFirst*[2] (RAYCHAUDHURI et al., 2012), *Named Data Networking* - NDN[3] (ZHANG et al., 2014) e *eXpressive Internet Architecture* - XIA[4] (NAYLOR et al., 2014), que compõem a segunda fase (FIA-NP) do projeto financiado pela *National Science Foundation* (NSF), entidade de fomento à pesquisa dos Estados Unidos (AMBROSIN et al., 2018). Além da *Recursive InterNetwork Architecture* - RINA[5], baseada nos princípios apresentados em DAY (2008); e das brasileiras *Entity Title Architecture* - ETArch[6] (PEREIRA et al., 2011) e NovaGenesis - NG[7] (ALBERTI; SINGH, 2014).

Dentre as possibilidades de classificação dessas proposições, podemos dividi-las em "puristas", que defendem uma arquitetura uniforme e flexível, para que possa atender a novos requisitos por um longo período de tempo, e "pluralistas", que desejam melhorias a curto prazo com a coexistência de diversas arquiteturas de rede (MOREIRA et al., 2009). Além disso, também é possível destacá-las em relação à orientação principal para a qual foram idealizadas (*hosts*, serviços, conteúdo, usuários, etc).

Apesar da existência de abordagens distintas e de alternativas em relação ao direcionamento para chegar a um redesenho do modelo atual, PAN et al. (2011) enumeraram três etapas para conduzir a uma futura arquitetura da Internet: (i) inovações em vários aspectos da arquitetura atual; (ii) a inclusão de inovações provenientes de vários projetos para uma arquitetura de rede única; (iii) *testbeds*[8] para experimentação em larga escala.

Após quase uma década da sugestão dessas etapas para o desenvolvimento de FIAs, observa-se o seguinte: (i) um redesenho da *Internet* atual vem se tornando imperativo, pois, suas dificuldades em tratar questões relacionadas à segurança, mobilidade, distribuição de conteúdo e escalabilidade ficam cada vez mais evidentes; (ii) a convergência das inovações relacionadas à FIA, para uma proposta única, não evoluiu como o melhor caminho para o desenvolvimento de

[2] http://mobilityfirst.winlab.rutgers.edu/
[3] https://named-data.net/
[4] http://www.cs.cmu.edu/~xia/
[5] http://www.pouzinsociety.org/
[6] http://mehar.facom.ufu.br/projects/etarch/
[7] https://www.inatel.br/novagenesis/
[8] Redes para experimentação em larga escala, sobrepostas (ou não) à Internet.

um redesenho da *Internet*; (iii) redes de experimentação em larga escala, como os *testbeds Global Environment for Network Innovations* - GENI[9] (nos Estados Unidos) e *Future Internet Brazilian Environment for Experimentation* - FIBRE[10] (desenvolvida originalmente através de uma parceria entre Brasil e Europa), buscam contribuir para a validação de projetos de inovação. Sendo que, em relação às iniciativas de Arquiteturas de Internet do Futuro, a existência de dispositivos para testes reais, sem o uso de simuladores e emuladores de rede, pode ser analogamente relevante para a realização de experimentos que visam legitimar o seu funcionamento.

A partir dessas observações e dadas as particularidades e abordagens distintas dos projetos de FIA ativos atualmente, há uma tendência de que essas propostas sejam estimuladas a funcionar simultaneamente. BALASUBRAMANIAM et al. (2011), ressaltam que uma nova arquitetura da *Internet* deve ser projetada para ser dinâmica, modular e adaptável. E que esses requisitos podem ser observados ou são característicos em processos de cooperação e ambientes de coexistência.

Um ou mais projetos podem proporcionar o redesenho da *Internet* atual. Contudo, com a disponibilidade de *testbeds* e a adoção cada vez maior de ferramentas de virtualização, crescem as possibilidades para que novas arquiteturas possam conviver em paralelo e com a *Internet* atual (FISHER, 2014).

1.2 Motivação e Justificativa

Os problemas de evolução ou engessamento da arquitetura da *Internet* durante as últimas décadas têm como uma das suas principais causas a dificuldade de modificação da sua camada de rede. Buscando-se resolver este problema através de uma proposta de redesenho da arquitetura, é importante que a solução propicie um núcleo de rede mais programável e permita a coexistência de vários projetos independentes (REXFORD; DOVROLIS, 2010). Para WANG et al. (2017), a presença simultânea de diversas propostas, com interoperabilidade na camada de rede, será um princípio na evolução das pesquisas por uma futura arquitetura da *Internet*.

Neste cenário de integração e perspectivas relacionadas à FIA, está o projeto FIXP (*Future Internet eXchange Point*) — "Alavancando a Internet do Futuro no Brasil através da coexistência e interconexão de múltiplas arquiteturas". Esta iniciativa tem como participantes a Universidade Federal de Pernambuco (UFPE), a Universidade Federal de São Carlos (UFSCar), a Universidade Federal de Uberlândia (UFU) e o Instituto Nacional de Telecomunicações (INATEL). O projeto tem como objetivo inicial estudar cenários de uso conjunto das arquiteturas NovaGenesis, RINA e ETArch, especificando e implementando pontos de interconexão chamados *Future Internet Exchange Points*. Também possui como finalidade a implementação de ambiente multi FIAs em um ponto de presença distribuído. E, por fim, testar a interconexão dessas novas arquiteturas de rede, utilizando o *testbed* FIBRE (MONTEIRO et al., 2018).

[9] https://www.geni.net/
[10] https://www.fibre.org.br/

Visando o crescimento do projeto FIXP, a inclusão de novas Arquiteturas de Internet do Futuro pode trazer contribuições para o desenvolvimento do ambiente multi FIAs proposto nessa iniciativa. Para tal, é razoável a opção por arquiteturas relevantes no cenário acadêmico de FIA ou apoiadas por organizações com destaque nesse segmento. Como exemplos de FIAs com essas características, é possível citar as propostas financiadas pela NSF (*MobilityFirst*, NDN e XIA). O fato de terem sido selecionadas para a segunda fase do programa FIA-NP, da NSF, assim como as evoluções registradas em cada um dos projetos nos últimos anos, demonstram um amadurecimento dessas propostas (FISHER, 2014).

Características como flexibilidade de endereçamento e suporte inerente à comunicação entre diferentes paradigmas (incluindo *hosts*, serviços e conteúdo) foram determinantes para a escolha da XIA para este trabalho. Além disso, o fato de ser extensível, possibilitando a inclusão de novos *principals* na sua estrutura, apresenta-se como uma característica promissora para uma arquitetura de rede implementada em um projeto que visa a integração de FIAs.

Em um ambiente multi FIAs, que tenha a intenção de aplicar os processos de cada arquitetura em um contexto mais próximo possível de um cenário de uso real, deve-se observar se as escolhas de tecnologias ou ferramentas contribuem para esse objetivo. Por exemplo, a utilização de *testbeds* é importante em relação à escala que o experimento pode atingir e devido ao desejado isolamento da rede. O uso de elementos de virtualização pode acelerar ou facilitar a inclusão de componentes de rede, como na implementação de dispositivos.

A virtualização em redes é uma das tecnologias que mais podem apoiar o amadurecimento de um projeto de Internet do Futuro (FISCHER et al., 2013). Pesquisadores defendem um modelo pluralista para a nova *Internet*, onde pilhas de protocolo de arquiteturas diferentes estejam em execução simultaneamente. Uma forma de implementar esse modelo é virtualizando dispositivos de rede, para que sejam a base de interligação entre essas redes distintas. Isso adicionaria flexibilidade ao ambiente de implementação, podendo ser o conceito de virtualização a chave para o crescimento de um cenário pluralista (MATTOS et al., 2012).

Até o início desta década, quando a maioria das FIA atualmente em evidência começaram a se desenvolver, as opções para criação de dispositivos de rede para novas arquiteturas eram bastante escassas ou complexas. Assim, a linguagem de alto nível para a programação de plano de dados, *Programming Protocol-independent Packet Processors* (P4), apresentada em BOSSHART et al. (2014), desponta como um avanço e traz possibilidades para o desenvolvimento dos experimentos direcionados às Arquiteturas para a Internet do Futuro.

Nesse contexto de coexistência de FIAs e, primordialmente, como proposta vinculada ao projeto FIXP, o trabalho de GAVAZZA et al. (2019) propõe um *switch*, usando a linguagem P4, com suporte às FIAs ETArch e NovaGenesis, além da inclusão do protocolo IP, pensando no suporte ao legado da atual arquitetura.

Diante dessas considerações, possibilidades e da implementação apresentada, este trabalho tem como motivação contribuir para o projeto FIXP, promovendo a inclusão de uma nova FIA no seu escopo atual. A escolha da arquitetura XIA para esta implantação também é um fator

motivador, pois o trabalho propõe a primeira iniciativa, até o momento da sua elaboração, de implementação do plano de dados desta FIA, em *switch* virtual, usando a linguagem P4.

1.3 Objetivos

Esta dissertação pretende contribuir com o projeto FIXP, visando incluir uma nova arquitetura de rede no ambiente multi FIAs do projeto. Para tal, busca implementar um dos requisitos obrigatórios para a incorporação de uma nova FIA: o seu plano de dados. Assim, a seguir estão listados os objetivos do trabalho.

O objetivo geral é a implementação de plano de dados programável para a Arquitetura de Internet do Futuro *eXpressive Internet Architecture* (XIA), utilizando a linguagem de programação *Programming Protocol-independent Packet Processors* (P4) para aplicar os conceitos desta FIA em dispositivo virtual de rede.

Como objetivos específicos propõe-se:

- desenvolver código P4 para a implementação do plano de dados da XIA em um dispositivo virtual de rede;
- validar a aplicação dos conceitos, componentes e processos de encaminhamento da arquitetura no dispositivo de rede implementado.

1.4 Metodologia

Tendo em vista o objetivo desta dissertação, que define a implementação do plano de dados da XIA, utilizando a linguagem P4, o conteúdo desta Seção visa destacar as etapas para atingir este propósito. Inicialmente, deve-se observar as questões relativas ao projeto FIXP, proposta com a qual este trabalho deseja contribuir. Assim, é necessário estudar as ferramentas e tecnologias utilizadas no projeto, verificar o status atual da pesquisa e as perspectivas para a expansão do cenário multi FIAs proposto pela iniciativa.

Para proporcionar o embasamento teórico, é preciso realizar estudos sobre Arquiteturas para a Internet do Futuro, aprofundando-se no plano de dados da FIA escolhida para integrar o projeto e na linguagem P4, assim como, nas ferramentas essenciais para a realização do trabalho. É importante estabelecer uma relação entre as características do plano de dados da arquitetura XIA e as possibilidades técnicas das ferramentas já definidas para a sua implementação. Além disso, deve-se realizar um levantamento técnico ou bibliográfico para selecionar adequadamente as demais tecnologias necessárias.

A arquitetura XIA possui particularidades relacionadas ao seu plano de dados que são determinantes para definir os métodos através dos quais ele deve ser implementado. Para atender esses requisitos, é preciso estudá-los profundamente e analisar os recursos apresentados nas documentações da linguagem P4. Assim, deve-se avaliar exemplos, esboçar cenários de

implementação e iniciar o desenvolvimento de códigos distintos, baseados nas possibilidades identificadas. Após descartar os cenários e tentativas que não atendem às necessidades apresentadas, deve-se eleger um arquétipo para ser utilizado na implementação.

Após a finalização do desenvolvimento do plano de dados da arquitetura XIA, deve-se testá-lo para validar a implementação. É necessário instalar um ambiente para experimentação que, devido às características do trabalho, deve ser virtualizado. Contudo, a escolha acerca da realização do experimento em uma máquina virtual ou com dispositivos (ou *hosts*) independentes deve basear-se na análise de cenários equivalentes, em possíveis limitações e no levantamento das necessidades para a execução dos testes.

Na execução dos experimentos, deve-se utilizar *switch* implementado em *software* que permita a aplicação do código P4, assim como um sistema que possua recursos para controle do plano de dados desenvolvido. Também é preciso montar o pacote XIP e, para isso, é importante utilizar ferramenta que permita a implementação do cabeçalho do protocolo, além da geração e envio do pacote. Para verificar o resultado do processamento dos pacotes no *switch*, é necessário o uso de um analisador para captura e exibição dos pacotes de rede.

O experimento deve ter como principais critérios de execução as possibilidades de endereçamento e processos de encaminhamento da arquitetura, além do suporte a pacotes com tamanhos distintos de cabeçalho. Para validar os testes realizados, deve-se comparar os resultados obtidos com as configurações e cenários propostos.

1.5 Estrutura da Dissertação

Este Capítulo de Introdução apresentou a contextualização acerca do tema proposto, destacou os motivos e justificativa para a realização do trabalho, seus objetivos (geral e específicos) e os métodos utilizados para sua execução. Nos capítulos posteriores, o conteúdo está organizado conforme descrito a seguir.

O Capítulo 2, Referencial Teórico, exibe o resultado das pesquisas bibliográficas realizadas para fundamentar e aprofundar o conhecimento em relação aos temas, ferramentas e tecnologias utilizadas para a efetivação do projeto. Também apresenta trabalhos relacionados ao objetivo desta dissertação.

No Capítulo 3, apresenta-se a implementação do plano de dados da XIA usando P4. Para isso, definem-se os requisitos da arquitetura e o escopo da execução, evidenciando o processo de desenvolvimento do código P4 a partir dos componentes principais da linguagem.

No Capítulo 4, Testes e Resultados, após definir os objetivos para a realização dos testes, apresentam-se as ferramentas utilizadas, ambiente e cenários de experimentação, além da exibição dos testes executados e dos resultados obtidos, visando validar o plano de dados implementado.

Por fim, o Capítulo 5 (Conclusão) destaca as contribuições, exibe as limitações e apresenta sugestões para trabalhos futuros, finalizando a dissertação.

2 REFERENCIAL TEÓRICO

Este Capítulo apresenta os conceitos e descrições acerca dos principais temas abordados nesta dissertação. Inicia com uma explanação sobre Internet do Futuro, que evidencia os problemas da arquitetura atual e aponta as soluções sugeridas pela comunidade acadêmica para uma mudança iminente na estrutura principal da rede. Em seguida, apresenta o detalhamento da arquitetura *eXpressive Internet Architecture* (XIA). Na sequência, discorre a respeito de Programabilidade de Plano de Dados, sobre a linguagem *Programming Protocol-independent Packet Processors* (P4) e apresenta o projeto FIXP. Por fim, exibe os trabalhos relacionados aos assuntos centrais desta proposição.

2.1 Internet do Futuro

O sucesso da *Internet* gerou grandes expectativas para a evolução de aplicativos e serviços que ela pode oferecer. Porém, a arquitetura de rede atual pode não ser capaz de atender a alguns desses anseios, tendo em vista os níveis de confiabilidade, disponibilidade e interoperabilidade exigidos. Além disso, o crescente número de terminais (computadores, dispositivos móveis, sensores, etc.) ativos na rede, com previsão de expansão para dezenas de bilhões, potencializa os problemas existentes (ZAHARIADIS et al., 2011). Devido às limitações decorrentes do seu projeto inicial, a atual rede mundial tem encontrado grandes desafios para alcançar o desenvolvimento desejado nas últimas décadas. Assim, para superar os problemas decorrentes das dificuldades no seu aperfeiçoamento, a comunidade acadêmica vem propondo novas arquiteturas de rede para a *Internet* (JIANPING et al., 2016).

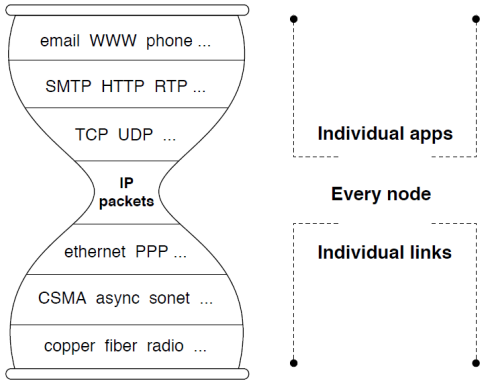
2.1.1 Problemas da arquitetura da Internet atual

A *Internet* tornou-se uma infraestrutura de abrangência mundial que disponibiliza uma grande diversidade de serviços, com níveis distintos de complexidade, que fazem com que a rede desempenhe papel importante na vida de bilhões de pessoas (KESHAV, 2018). A adesão da sociedade a novos serviços levou ao crescimento da Web e à alteração da natureza do seu tráfego ao longo dos anos, sendo hoje uma rede mais voltada a compartilhamento e consumo de conteúdo, como em plataformas de *streaming* (TARAN; LAVROV, 2016).

O crescimento da *Internet* atual, assim como o seu sucesso, tem como um dos principais motivos o modelo de “cintura estreita” (Figura 1) da sua arquitetura, que identifica a funcionalidade mínima comum que um dispositivo precisa para ter capacidade de comunicação. O *design* de ampulheta da *Internet*, que simplifica o núcleo da rede, permitiu a rápida integração de aplicativos, tendo diversos protocolos acima do IP e o suporte às camadas físicas, abaixo dele.

Contudo, o modelo da arquitetura atual dificulta consideravelmente a inclusão de novos recursos na rede, o que paralisa a sua evolução (NAYLOR et al., 2014).

Figura 1 – Modelo de ampulheta ("cintura estreita") da *Internet* atual



Fonte: ZHANG et al. (2014)

A simplicidade do núcleo de rede da arquitetura baseada em TCP/IP permitiu o seu desenvolvimento, mas também é a fonte de muitas das suas dificuldades de modificação e evolução para atender às demandas atuais e futuras (HU et al., 2011). Assim, as premissas sobre as quais a *Internet* foi projetada sofreram algumas alterações no decorrer das últimas décadas. Foram incluídas soluções incrementais decorrentes de requisitos específicos como escalabilidade, mobilidade, qualidade de serviço e, principalmente, segurança. Contudo, devido às limitações estruturais da camada de rede, esses *patches* sobrepostos têm eficácia limitada e muitas vezes são altamente ineficientes (PAUL et al., 2011).

Soluções adicionadas ao projeto original da *Internet* (NAT, IPv6, IP Móvel, IPsec[11], DNSsec[12], *Firewall*, etc) apenas amenizam, ou tratam de forma insatisfatória os problemas que surgiram com o seu crescimento. No caso do IPv6, proposto devido ao esgotamento da quantidade de endereços da versão 4 do IP, ele aumenta o espaço de endereços e pode oferecer identificadores exclusivos para os dispositivos. Isso possibilita o aprimoramento de questões de segurança (autenticação, por exemplo), porém, ainda assim, enfrenta outros problemas como os relacionados a *flooding*[13] ou mobilidade (DING et al., 2016).

Além disso, no modelo TCP/IP, o endereço IP é o identificador da *interface* do *host*, assim, o IPv6 também não resolve o fato de que uma rede centrada em *hosts* não está em conformidade com comunicações largamente distribuídas. E a multiplicação de serviços como comércio eletrônico, mídias digitais e redes sociais, levaram a um uso dominante da *Internet* como uma rede de distribuição de conteúdo (DING et al., 2016).

Algumas das dificuldades e limitações da arquitetura atual foram destacadas em JAIN (2006). Esses problemas estruturais também são apontados em trabalhos mais recentes, como

[11] IPsec (*IP Security Protocol*) promove o uso de criptografia para o IP visando tunelamento e autenticação.

[12] DNSsec (*Domain Name System Security Extensions*) é o nome dado às extensões de segurança criadas para proteger e autenticar o tráfego DNS.

[13] Ataques por “inundação”, que geram um grande volume de tráfego a partir do envio constante de mensagens.

em JIANPING et al. (2016), que expõem questões relacionadas à mobilidade e segurança na *Internet*. Nele, ressalta-se que a utilização do protocolo IP para o endereçamento dos *hosts* pode provocar interrupções das conexões durante a movimentação, influenciando negativamente a experiência do usuário. Também destaca-se o fato dos protocolos atuais de roteamento não validarem endereços de origem, possibilitando a ocorrência de eventos de segurança, como ataques DDoS[14] (*Distributed Denial of Service*), *spam*[15] e *worms*[16].

Uma das questões primordiais quando se trata dos problemas da arquitetura baseada em TCP/IP é a segurança. Para TARAN e LAVROV (2016), a falta de segurança é o maior problema da *Internet*, pois o fato de adicionar mecanismos de proteção separadamente nas camadas da rede não a torna protegida de uma forma generalizada. Algumas iniciativas como IPsec e DNSsec, por exemplo, foram idealizadas para salvaguardar protocolos existentes, mas, ainda assim, possuem fragilidades ou limitações relacionadas à proteção na rede. A estrutura inicial da *Internet* não previa recursos de segurança em sua arquitetura, o que dificulta a defesa contra ataques cada vez mais sofisticados.

Conforme destaca PAN et al. (2011), a segurança precisa ser inerente a uma arquitetura de rede que busque atender à crescente demanda por conectividade. Aliado a isso, e também como um dos requisitos principais para essa evolução, é preciso alterar o atual paradigma da *Internet*, que deve mudar de uma rede de encaminhamento de pacotes baseada em *hosts* para uma estrutura mais abrangente orientada a dados, conteúdo e usuários. Premissas como essas estimularam pesquisas direcionadas a propostas de novas arquiteturas para a rede mundial.

2.1.2 Arquiteturas de Rede para a Internet do Futuro

Para superar as dificuldades de aperfeiçoamento da *Internet* atual, parte da comunidade acadêmica responsável pelas pesquisas na área de redes vem propondo novas soluções para o futuro da *Internet* há aproximadamente uma década. Esses projetos podem ser classificados em dois tipos de abordagem: evolutiva ou limpa (*clean-slate*) (JIANPING et al., 2016). As pesquisas evolutivas buscam compreender os requisitos e o funcionamento da *Internet* para sanar os problemas existentes de forma incremental, trilhando um caminho para o futuro, mas mantendo a arquitetura e os aplicativos atuais (TARAN; LAVROV, 2016). Enquanto a abordagem *clean-slate* prevê a arquitetura projetada "do zero" para atender adequadamente às demandas atuais e requisitos futuros (KHONDOKER et al., 2014).

Muitos pesquisadores acreditam que é impossível resolver os problemas da *Internet* sem uma mudança na arquitetura, pois as inconsistências entre o *design* da estrutura atual e o uso real requerem um redesenho da rede a partir de uma abordagem *clean-slate* (DING et al., 2016). Na última década houve um crescimento significativo de esforços com este objetivo, devido ao

[14] Ataque que busca a negação do serviço através da sobrecarga dos servidores.
[15] Mensagens de e-mail não solicitadas, geralmente enviados para um grande número de destinatários.
[16] Programa semelhante ao vírus, mas que gera cópias de si mesmo para saturar computadores e redes.

desenvolvimento de projetos voltados a novas arquiteturas para a *Internet* em diversas partes do mundo, incluindo Estados Unidos, União Europeia, Ásia e Brasil (PAN et al., 2011).

Nos Estados Unidos, a *National Science Foundation* (NSF) financiou os projetos *Future Internet Design* (FIND), *Future Internet Architecture* (FIA) e FIA - *Next Phase* (FIA-NP), para promover pesquisas voltadas à *Internet* do futuro (LUTZ, 2016). Em 2010, a NSF convidou equipes de pesquisa a apresentarem propostas de arquiteturas para a Internet do Futuro, selecionando quatro iniciativas para o projeto FIA (FISHER, 2014). Esse programa tinha a previsão de 5 anos, mas em 2015 a NSF criou uma nova fase do projeto (FIA-NP), que enfatiza a avaliação por meio de protótipos, *testbeds*, implantações e experimentação das arquiteturas remanescentes. Foram selecionadas para esta nova fase as três arquiteturas mais promissoras: *MobilityFirst*, *Named Data Networking* (NDN) e *eXpressive Internet Architecture* (XIA) (AMBROSIN et al., 2018).

Também com origem nos Estados Unidos, outra arquitetura de rede com notoriedade nas pesquisas relacionadas à *Internet do futuro* é a *Recursive InterNetwork Architecture* (RINA). Os atributos da RINA foram apresentados por John Day, em seu livro "*Patterns in Network Architecture: A return to Fundamentals* (DAY, 2008), onde ele propõe um redesenho da *Internet* baseando-se nos princípios fundamentais de redes de computadores (GRASA et al., 2014).

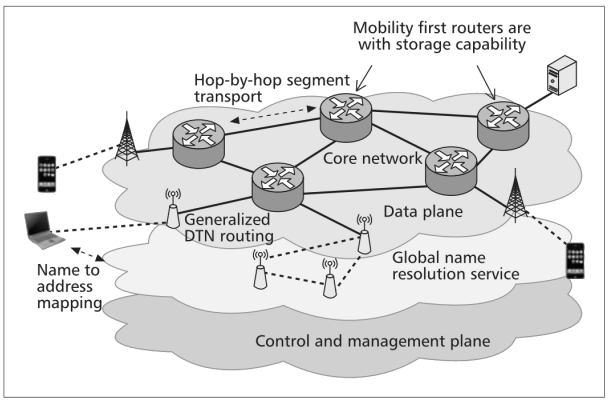
Na União Europeia, com o incentivo da *European Future Internet Assembly*, uma colaboração entre projetos de pesquisa para a *Internet* do futuro, sobressaiu-se a proposta 4WARD - *Architecture and Design for the Future Internet* (BRUNNER et al., 2010). Na Ásia, destacou-se o programa de pesquisa japonês *New Generation Network* (NWGN), onde o projeto AKARI - *Architecture Design for New Generation Network* (HARAI, 2009) alcançou maior evidência (PAN et al., 2011). Nos últimos anos, duas iniciativas obtiveram destaque no Brasil: a *Entity Title Architecture* — ETArch (PEREIRA et al., 2011), criada na Universidade Federal de Uberlândia (UFU) e a iniciativa NovaGenesis (ALBERTI; SINGH, 2014), desenvolvida no Instituto Nacional de Telecomunicações (INATEL).

Além das FIAs citadas anteriormente, pode-se destacar as seguintes arquiteturas de rede: *Content-Centric Networking* - CCN (JACOBSON et al., 2009); *Couple service Location with inter-domain Routing* - CoLoR (CHEN et al., 2013); *Data-Oriented Network Architecture* - DONA (KOPONEN et al., 2007); NEBULA *Future Internet Architecture* (ANDERSON et al., 2013), *Service-Oriented Future Internet Architecture* - SOFIA (LAN et al., 2015). Devido à relevância em relação aos paradigmas que representam, ao destaque no cenário acadêmico dos últimos anos ou pela relação direta com o projeto FIXP, as arquiteturas *MobilityFirst*, NDN, XIA, RINA, ETArch e NovaGenesis são resumidamente descritas a seguir.

A FIA *MobilityFirst* tem como principal argumento o fato da *Internet* ter sido projetada para a interconexão de *hosts* fixos na rede. O modelo atual não consegue atender adequadamente às demandas decorrentes do crescimento de dispositivos e serviços móveis, o que a proposta vê como fator primordial para ter a mobilidade como ponto central da arquitetura (PAN et al., 2011). O objetivo é que dispositivos possam alternar entre redes após a conexão, e esta deve manter-se resiliente. Para isso, a *MobilityFirst* considera como *principals*: dispositivos, conteúdo,

interfaces, serviços e usuários. Cada um deles pode ser reconhecido como unidade endereçável e deve ter identificador único, *Global Unique Identifier* (GUID), para que possa ser reconhecido na rede à qual está conectado, que também possui um identificador (AMBROSIN et al., 2018).

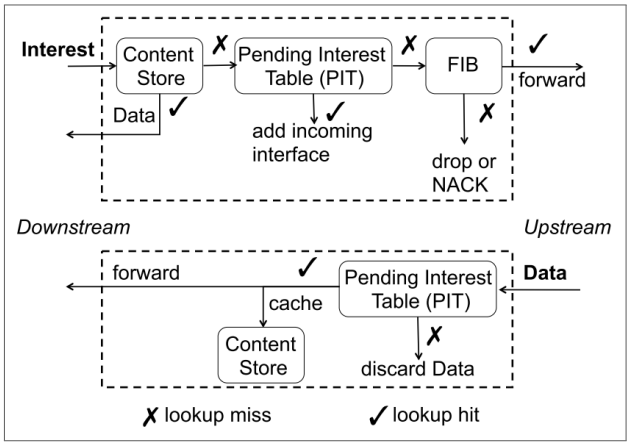
Figura 2 – Estrutura da arquitetura MobilityFirst



Fonte: PAN et al. (2011)

A *Named Data Networking* (NDN) é um dos três projetos do FIA-NP, financiado pela NSF. A NDN teve origem a partir do projeto *Content-Centric Networking* (CCN) e propõe a mudança da arquitetura da *Internet* atual, centrada em *hosts*, para um modelo orientado a dados nomeados. A camada de rede NDN usa nomes hierarquicamente estruturados para endereçar diretamente cada conteúdo. Para a requisição e encaminhamento destes, define um pacote de interesse e um pacote de dados. Cada nó NDN contém os seus três componentes principais (Figura 3): *Content Store* (CS), *cache* usado para armazenamento e recuperação de conteúdo a partir de um nó da rede; *Forwarding Interest Base* (FIB), tabela de encaminhamento com prefixos de nomes e interfaces de saída correspondentes; e *Pending Interest Table* (PIT), tabela com nomes de interesse pendentes e interfaces de entrada correspondentes (ZHANG et al., 2014).

Figura 3 – Processo de encaminhamento em um nó NDN

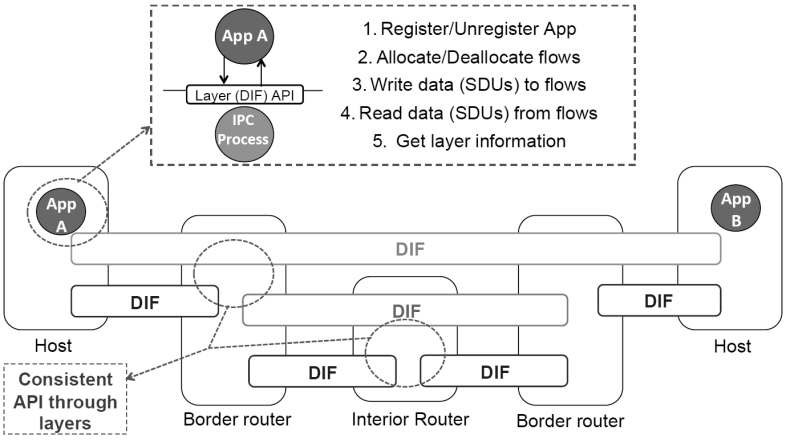


Fonte: ZHANG et al. (2014)

A *eXpressive Internet Architecture* (XIA) tem como objetivo principal ser uma arquitetura segura e permitir evoluções na sua camada de rede. Isso é possível devido à flexibilidade do seu principal protocolo, o XIP, que provê o suporte a diferentes *principals*, cada um com suas características específicas no núcleo da rede. Esses *principals* são implementados a partir de identificadores exclusivos chamados XID, sendo que, inicialmente, a XIA já oferece suporte para rede, conteúdo, *host* e serviço (NID, CID, HID e SID, respectivamente) e permite a inclusão de outros. Para promover a segurança intrínseca, o XIP vincula criptograficamente o nome de cada XID a alguma propriedade, como *hash* de chave pública/privada (MACHADO, 2014). Por ser objeto de estudo desta dissertação, a XIA é apresentada de forma detalhada na Seção 2.2.

A *Recursive InterNetwork Architecture* (RINA) tem como fundamento a ideia de que a rede é a comunicação entre processos (Inter-Process Communication - IPC) e esta é a estrutura necessária para o redesenho da *Internet*. Ela provê uma camada chamada *Distributed IPC Facility* (DIF), que pode ser repetida, de acordo com escopo da rede (GRASA et al., 2014). A DIF contém todas as funções necessárias para fornecer serviços IPC para os aplicativos ou para DIFs de um nível superior. Cada camada repetida oferta serviços IPC em um determinado escopo (Figura 4), por exemplo, uma DIF entrega serviços para uma LAN, outra para uma rede de campus, ISP ou a rede mundial. Assim, a DIF funciona como uma estrutura que disponibiliza uma API para oferecer serviços IPC a seus usuários. A recursividade é caraterizada na arquitetura ao criar DIFs de escopo amplo sobre as de escopo menor (MAFFIONE et al., 2016).

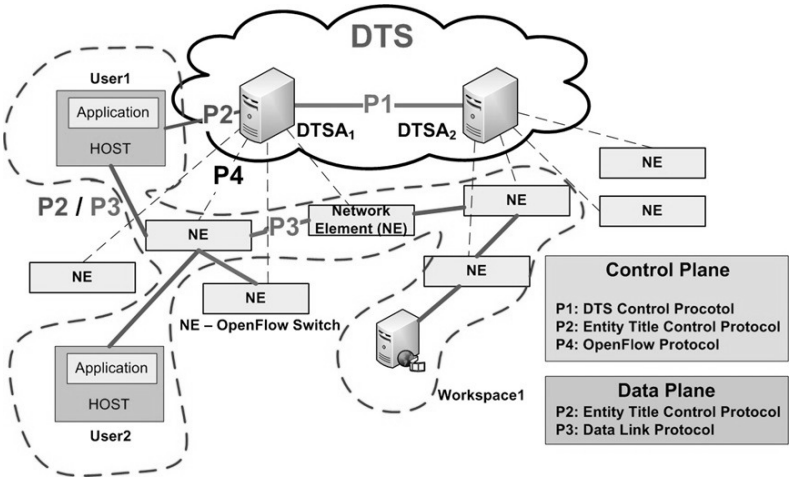
Figura 4 – Exemplo de estrutura da RINA



Fonte: MAFFIONE et al. (2016)

A *Entity Title Architecture* (ETArch) é uma FIA que tem relação estreita com SDN, pois além de promover a separação de plano de controle e plano de dados, usa o protocolo *Openflow* para que os DTSAs, agentes distribuídos que compõem o seu plano de controle (DTSA), administrem os Elementos de Rede (NE). Entre os conceitos principais que definem a arquitetura, estão: Entidade, que representa tudo com capacidade de comunicação (hosts, NEs, usuários, aplicativos, etc); e Título, que é um identificador exclusivo (THEODORO et al., 2015).

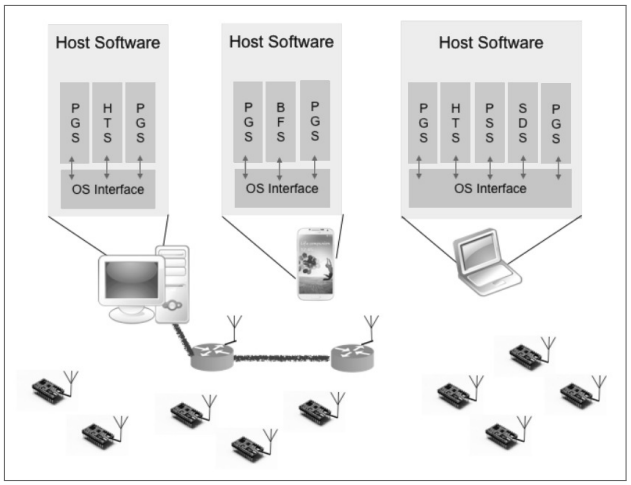
Figura 5 – Estrutura da FIA ETArch



Fonte: SILVA et al. (2012)

A arquitetura NovaGenesis (NG) foi criada tendo como escopo *Internet of Things* (IoT) para a *Internet* do Futuro. NG é um agrupamento de sistemas distribuídos onde os processamentos de informações são tratados como serviço, inclusive as funcionalidades de rede. Comunicação, processamento e armazenamento são orientados por nome, onde *Natural Language Names* (NLNs) representam os nomes de linguagem natural e *Self-Certified Names* (SCNs) os nomes autocertificados. Entre os componentes principais da NG estão: Proxy/Gateway System (PGS), que representa recursos como interfaces e sensores sem fio, atuando no encaminhamento para outras tecnologias de rede; *Hash Table System* (HTS), sistema de tabela *hash* que registra os SCNs; *Binding Forwarding System* (BFS), responsável pela escolha do HTS que armazena a ligação de nomes; e *Publish/Subscribe System* (PSS), sistema de publicação/assinatura. A combinação de HTS, BFS e PSS constitui um sistema de resolução de nomes distribuídos (ALBERTI; SINGH, 2014).

Figura 6 – Componentes da arquitetura NovaGenesis



Fonte: ALBERTI e SINGH (2014)

A Tabela 1 exibe as FIAs citadas nesta Subseção, destacando a distribuição geográfica e a diversidade de paradigmas entre as propostas apresentadas. Além de demonstrar, com essa amostragem, a disposição das propostas ao longo de uma década.

Tabela 1 – Exemplos de Arquiteturas de Rede para a Internet do Futuro

FIA	**Orientação Principal**	**Publicação**	**Origem**
4WARD	Conteúdo	2010	União Europeia
AKARI	-	2009	Japão
CCN	Conteúdo	2009	Estados Unidos
CoLoR	Serviços	2013	China
DONA	Conteúdo	2007	União Europeia
ETArch	Conteúdo	2011	Brasil
MobilityFirst	Mobilidade	2010	Estados Unidos
NDN	Conteúdo	2010	Estados Unidos
NEBULA	Computação em nuvem	2010	Estados Unidos
NovaGenesis	Serviços	2014	Brasil
RINA	Processos	2008	Estados Unidos
SOFIA	Serviços	2015	China
XIA	Conteúdo, *hosts*, serviços	2010	Estados Unidos

Fonte: O Autor (2020)

2.1.3 Interconexão de Arquiteturas de Rede (cenário pluralista)

Nos últimos anos, diversos projetos de arquiteturas *clean-slate* para a Internet do Futuro foram apresentados. Essas propostas, com requisitos e princípios distintos, surgem como opção de melhoria frente ao modelo atual. Contudo, a cada dia torna-se mais difícil definir se uma delas substituirá completamente a *Internet*.

Diante dessas possibilidades, destacam-se dois cenários relacionados a visões diferentes de como projetar essa evolução. O "purista", defendendo que atributos de diferentes propostas poderiam ser integrados a uma única arquitetura, flexível o suficiente para atender aos novos requisitos e aplicações. E o "pluralista", com o argumento de que essas arquiteturas específicas podem ser instanciadas simultaneamente através de redes interconectadas ou multiplexadas, utilizando recursos de virtualização (MOREIRA et al., 2009).

A substituição da arquitetura atual da *Internet* tende a ser um processo difícil e demorado. Atualmente, as diferenças entre as iniciativas que propõem novas arquiteturas de rede aumentam as disparidades, o que compromete um possível compartilhamento eficiente de recursos e torna improvável a integração de todos os aspectos em uma única proposta. Assim, espera-se uma *Internet* do Futuro heterogênea, de forma que várias arquiteturas de rede possam coexistir simultaneamente, incluindo a *Internet* atual (GUIMARÃES et al., 2019).

Em CHEN et al. (2018), destaca-se que a coexistência de vários paradigmas de rede será um requisito para a evolução da *Internet* do Futuro. Também salienta que mecanismos de interoperabilidade que possibilitem, por exemplo, que uma nova arquitetura possa obter conteúdo hospedado em um servidor web da arquitetura atual devem ser estimulados.

Uma maneira de promover um ambiente pluralista é virtualizar os dispositivos de rede e permitir que várias instâncias desses compartilhem a mesma infraestrutura. Dessa forma, a arquitetura atual da *Internet* pode ser executada em paralelo com outras redes virtuais, associadas a novas arquiteturas. Assim, a virtualização aponta como um conceito chave para o desenvolvimento de um cenário pluralista para a *Internet* do futuro (MATTOS et al., 2012).

2.1.4 *Testbeds* e Virtualização de Dispositivos de Rede

A *Internet* atual é uma rede comercial que atende a diversos interesses e necessidades da sociedade. Apesar da exigência de evolução, não é viável expor esses serviços a riscos decorrentes de experimentos de inovação. Portanto, um dos requisitos importantes para o desenvolvimento de pesquisas relacionadas a FIA são os ambientes virtualizados para experimentação em larga escala (*testbeds*). Dessa forma, utilizando uma base para experimentos independente e confiável, novas arquiteturas podem ser testadas, validadas e aprimoradas (PAN et al., 2011).

No cenário acadêmico da área de redes de computadores pode-se destacar os seguintes *testbeds*: *Global Environment for Network Innovations* (GENI), nos Estados Unidos; *Future Internet Research and Experimentation* (FIRE[17]), na Europa; e *Future Internet Brazilian Environment for Experimentation* (FIBRE), no Brasil. Eles fornecem infraestrutura de rede para experimentação em larga escala, a fim de contribuir com projetos de inovação, validar novos protocolos ou propostas voltadas a arquiteturas de rede.

O GENI (ELLIOTT, 2008) é um projeto financiado pela NSF e possui uma infraestrutura de rede que permite a realização de experimentos em larga escala com foco na inovação em redes de comunicação. Segundo PAN et al. (2011), ele se diferencia de outros *testbeds* por disponibilizar seus recursos em larga escala e sem restrições relativas a arquiteturas, serviços ou aplicativos de rede. O objetivo é permitir que propostas de novas arquiteturas possam realizar testes com usuário reais, num cenário equivalente ao de produção. O GENI não suporta apenas projetos existentes em uma rede de *backbone*[18] dedicada à infraestrutura, mas também orienta outras plataformas, pois sua estrutura consiste em vários subsistemas e entidades básicas (TARAN; LAVROV, 2016).

FIRE é um dos projetos de pesquisa relacionados à experimentação na União Europeia. Ele envolve esforços da indústria e da academia, tendo como objetivos: apoiar pesquisas de longo prazo sobre novos paradigmas, conceitos e arquiteturas para a *Internet* do Futuro; construir uma

[17] https://www.fed4fire.eu/

[18] Significa "espinha dorsal". É o termo utilizado para identificar as principais estruturas de rede da *Internet*, de dimensões continentais e alto desempenho, por onde trafegam os dados de todos os usuários.

instalação de experimentação em larga escala, promover federação com os *testbeds* existentes e futuros (PAN et al., 2011).

O *testbed* FIBRE nasceu como um ambiente compartilhado, voltado à Internet do Futuro, para apoiar a experimentação conjunta de pesquisadores europeus e brasileiros (ABELEM et al., 2011). Sua abordagem de federação permite que duas ou mais instituições disponibilizem parte de seus recursos a um objetivo compartilhado, que tem os critérios e níveis de conectividade estabelecidos através de um conjunto de padrões e políticas (ABELEM et al., 2013). Atualmente, a infraestrutura da FIBRE possui 11 ilhas (nós de experimentação), onde cada nó possui um conjunto de dispositivos de rede para suportar os experimentos, que são conectados por uma rede de sobreposição no *backbone* da Rede Nacional de Ensino e Pesquisa (RNP).

A ideia de virtualização para isolar experimentos de rede em execução, a partir de uma infraestrutura compartilhada, tem grande relevância para as iniciativas relativas à *Internet* do futuro. No entanto, os *testbeds* executam sobrepostos às redes baseadas em IP, limitando algumas experiências no nível da rede. Desse modo, deve-se buscar o isolamento ponta a ponta, promovendo a virtualização de *hosts* finais e dispositivos de rede (PAUL et al., 2011).

A criação de dispositivos para novas arquiteturas pode ser promovida a partir da virtualização de elementos programáveis de rede. Isso permite definir o comportamento do processamento de pacotes, incluindo tipo, sequência e semântica de operações. Dessa forma, o uso de dispositivos programáveis de rede destaca-se como um elemento crucial para a próxima geração da *Internet* (BIFULCO; RETVARI, 2018).

2.2 *eXpressive Internet Architecture* (XIA)

Esta seção é destinada à *eXpressive Internet Architecture* (XIA), assim, apresenta suas funcionalidades e recursos, destacando seus princípios fundamentais e a estrutura da arquitetura. Também ressalta os protocolos e serviços que a integram, evidenciando o protocolo XIP, que permite o suporte a diferentes *principals*, apresenta alternativas de encaminhamento e promove a segurança intrínseca. Em seguida, destaca os tipos de identificadores para os *principals*, as possibilidades de endereçamento e o formato de cabeçalho da arquitetura. Com isso, detalha o processo de encaminhamento, evidenciando os conceitos da XIA (ANAND et al., 2011).

2.2.1 Princípios Fundamentais da XIA

A XIA foi proposta com a intenção de resolver os problemas da *Internet* atual a partir da ampliação das possibilidades da camada de rede (hoje bastante limitadas no modelo TCP/IP). Os objetivos estão relacionados à segurança inerente, evolução contínua e suporte adequado às demandas de comunicação atuais e futuras. Atualmente, a principal demanda de comunicação na *Internet* está relacionada com a recuperação de conteúdo, porém, outros paradigmas podem

tornar-se prioridade no futuro (NAYLOR et al., 2014). A arquitetura XIA busca tratar estas e outras questões, tendo como pilares os seus princípios fundamentais.

A primeira premissa da arquitetura XIA é **a comunicação entre diferentes *principals***. Os aplicativos devem poder expressar a intenção de acionar funcionalidades para uma ou mais orientações. Cada *principal* define o seu próprio estilo de comunicação, sua interface com aplicações e características particulares de roteamento. A XIA não possui um limite para o número de *principals* que pode representar, sendo assim, aliado à simplificação do processo de implantação, este atributo destaca-se como a unidade de evolução da arquitetura. Inicialmente já existe suporte para o paradigma em uso (baseado em *hosts*), ao tipo de comunicação predominante hoje (conteúdo) e às orientadas a serviço (MACHADO, 2014).

Endereçamento flexível é o segundo princípio da XIA. A arquitetura deve permitir que exista ao menos um endereço alternativo para a efetivação da comunicação. Isso também é importante para sua evolução, pois novos paradigmas ou funcionalidades podem ser implantadas gradualmente. Dessa forma, mesmo que os roteadores não conheçam todas as informações de destino incluídas no pacote, a comunicação pode ser concretizada devido à flexibilidade do endereçamento alternativo. O processo de endereçamento que melhor representa este atributo é chamado de *fallback* (NAYLOR et al., 2014).

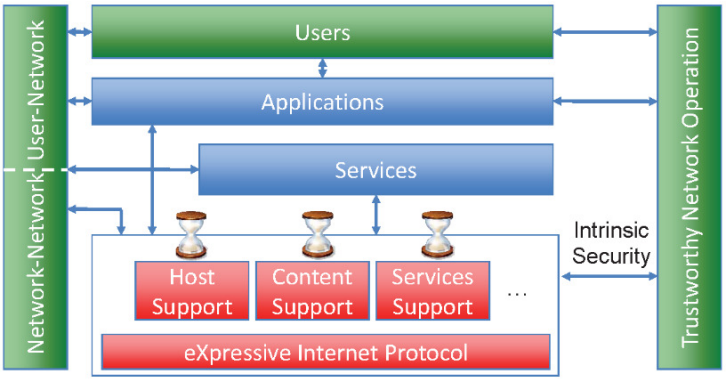
A arquitetura XIA tem como seu terceiro conceito fundamental, a **segurança intrínseca**. Assim, os *principals* suportados devem ser intrinsecamente seguros, o que é obtido através da autocertificação dos seus identificadores. Isso contribui para a confiabilidade das comunicações finais, acesso a serviços e transferências de dados na rede. Porém, não há rigidez quanto ao tipo de certificação utilizada, sendo esta atribuída de acordo com os requisitos do *principal* (DING et al., 2016).

2.2.2 Estrutura da arquitetura XIA

A XIA mantém o desenho de "cintura estreita"(Figura 1) da *Internet* atual, contudo, a nova arquitetura modifica e amplia a camada de rede para atingir os objetivos previstos nos seus princípios fundamentais. Com o suporte a diversos *principals*, cada um possui seu modelo de ampulheta próprio (Figura 7), assim, pode estabelecer mecanismos específicos de segurança e definir como processar o seu roteamento. Já os aplicativos, podem apontar para uma ou mais *principals* ao expressar diretamente sua intenção de acessar ou disponibilizar funcionalidades (XIA_OVERVIEW, 2016).

Além do suporte a múltiplas orientações, a Figura 7 exibe os demais componentes da XIA: seu protocolo principal, o *eXpressive Internet Protocol* (XIP), responsável por prover a base para a concretização dos princípios fundamentais da arquitetura, além de dar suporte a serviços de comunicação e rede (transporte, mobilidade, etc). Esses serviços, por sua vez, fazem a ligação com aplicativos e *hosts* finais. Também são apresentadas as estruturas complementares responsáveis por gerenciamento, controle e usabilidade da rede.

Figura 7 – Estrutura da arquitetura XIA



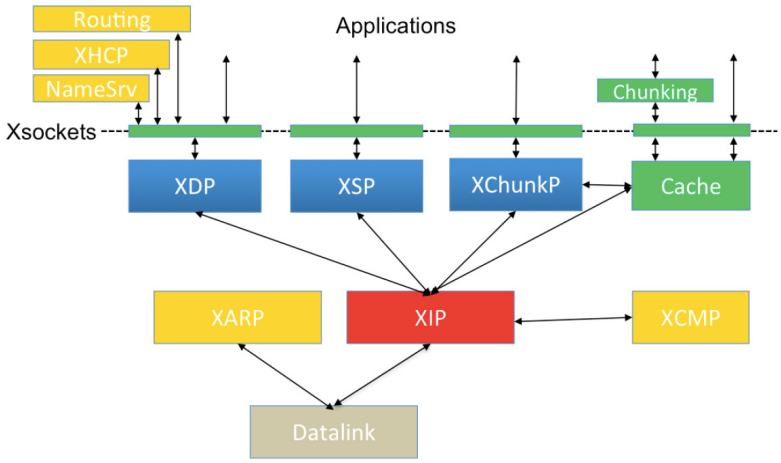
Fonte: XIA_OVERVIEW (2016)

2.2.3 Protocolos XIA

A Figura 8 apresenta a cadeia de protocolos XIA, onde destaca-se o núcleo da arquitetura, o *eXpressive Internet Protocol* (XIP). Esse protocolo de camada de rede define um formato comum de cabeçalho, esquemas de endereçamento e o comportamento de todos os nós ao processar pacotes de entrada e saída. Além disso, dá suporte a diferentes tipos de *principals*, que podem ser associadas à arquitetura XIA por um novo identificador chamado *Expressive Identifier* - XID. Ele permite a cada *principal* adequar funções como encaminhamento e segurança, de acordo com seus requisitos específicos (HAN et al., 2012).

Como principal protocolo da rede, as especificações e funcionalidades vinculadas ao XIP determinam o funcionamento do plano de dados da arquitetura XIA. Assim, os identificadores, o processo de endereçamento e o padrão do cabeçalho definidos por ele são detalhados a seguir, nas Subseção 2.2.4, Subseção 2.2.5 e Subseção 2.2.6, respectivamente.

Figura 8 – Cadeia de protocolos da arquitetura XIA



Fonte: NAYLOR et al. (2014)

Além do XIP, a Figura 8 exibe também os demais protocolos ou serviços que compõem a

arquitetura XIA (BARRETT, 2017):

- *X-Datagram Protocol* (XDP): protocolo não orientado a conexão, baseado em mensagens;
- *X-Stream Protocol* (XSP): protocolo orientado a conexão, que exige um processo de *handshake* para configurar a comunicação;
- *X-Chunk Protocol* (XChunkP): protocolo de transporte confiável, para recuperação de conteúdo na rede (em cache);
- *Content Cache*: facilita a recuperação de conteúdo na rede, permitindo que as solicitações de conteúdo sejam atendidas por roteador intermediário;
- *X-Address Resolution Protocol* (XARP): protocolo que resolve XIDs de destino em endereços da camada de link (endereços MAC, por exemplo);
- *X-Control Message Protocol* (XCMP): protocolo XIA equivalente ao ICMP;
- *X-host configuration protocol* (XHCP): protocolo de configuração de *hosts* em uma rede XIA;
- *Name Service* (NameSrv): serviço de rede para o registro e resolução de nomes.

2.2.4 *Principals* e Segurança Intrínseca

A especificação de um *principal* para a arquitetura XIA deve incluir a semântica da comunicação com as orientações do tipo proposto, um *XIDType* exclusivo, um método para prover segurança intrínseca e o processamento necessário para encaminhar pacotes com endereços do novo *principal* (NAYLOR et al., 2014).

Cada XID é o emparelhamento de um tipo de orientação principal e um nome (ou ID), com tamanho igual a 160 *bits* (MACHADO, 2015). Os XIDs são mapeados para as devidas localizações utilizando um serviço de resolução de nomes (equivalente ao DNS) ou com base em um mapeamento local, no caso de endereços internos, como um endereço MAC para um dispositivo *bluetooth* (TUNCER et al., 2012). A segurança intrínseca nos XIDs é implementada a partir de *hash* de chave pública, que permite obter a chave correspondente a partir de qualquer fonte, confiável ou não (MACHADO, 2015).

A arquitetura XIA inicialmente definiu quatro tipos básicos de identificadores (XID), que correspondem a *host*, serviço, conteúdo e domínio autônomo (ou rede):

- HID (XID de *host*): é gerado usando o *hash* de chave pública do *host* que ele nomeia. Ao contrário da *Internet* atual, cada *host* XIA possui um HID exclusivo, independente da interface pela qual esteja se comunicando. Esse recurso ajuda a suportar a mobilidade do *host*;
- SID (XID de serviço): os serviços no XIA são equivalentes a aplicativos na *Internet* de hoje. Dessa forma, um SID pode ser especificado como intenção para um processo de aplicação relativo a um determinado serviço. Os SIDs são gerados computando um *hash* da chave pública do serviço identificado por ele (AMBROSIN et al., 2018);
- CID (XID de conteúdo): esse tipo de orientação principal indica a intenção do usuário de recuperar um conteúdo. O envio de um pacote de intenção a um CID inicia a recuperação do conteúdo a partir de um *host*, um cache de rede ou outra fonte que possa vir a armazená-lo

futuramente. Os CIDs são gerados com base em *hash* criptográfico do conteúdo do arquivo que ele nomeia, vinculando o conteúdo ao nome;

- AD (domínio autônomo) ou NID (XID de rede): especifica um domínio autônomo ou rede que permite que uma entidade verifique se está se comunicando com a rede pretendida. Também é gerado usando o *hash* de chave pública da rede identificada por ele (NAYLOR et al., 2014).

XIA usa, prioritariamente, o NID para o endereçamento global. Após o envio para o domínio de rede específico, os pacotes são encaminhados ao *host* pelo HID, que especifica o destinatário dos pacotes. Com isso, um par NID – HID pode ser usado para garantir a acessibilidade global. O domínio encaminha o pacote para o host e este, para o serviço desejado.

Na *Internet* atual, os verdadeiros pontos finais de comunicação são, normalmente, processos de aplicativos. Na arquitetura XIA, essa noção de processos como o destino verdadeiro pode ser explicitada através do SID. Ele também pode ser usado para efetuar comunicações no estilo *anycast*, quando pacotes são enviados para vários servidores solicitando um serviço específico.

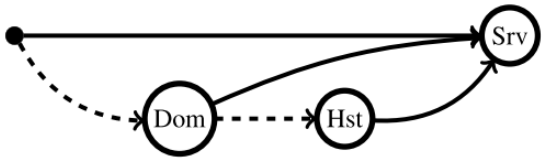
O CID permite que os aplicativos possam expressar sua intenção de recuperar o conteúdo sem levar em consideração sua localização. Em uma rede XIA ele pode ser usado para recuperar conteúdo disponível em qualquer destino, inclusive em roteadores intermediários da rede, com o auxílio do suporte a cache da arquitetura (MENG et al., 2017).

2.2.5 Endereçamento XIP

Um endereço XIP é um Grafo Acíclico Dirigido (*Directed Acyclic Graph* - DAG). Por definição, um DAG exibe o relacionamento entre variáveis (chamadas de nós no contexto de grafos) e suas conexões assumem a forma de linhas. Essas linhas são direcionadas, o que no grafo é representado por uma seta indicando seu efeito (BARRETT, 2020).

Para ilustrar esse conceito, no contexto de endereçamento da arquitetura XIA, deve-se observar o DAG da Figura 9, que indica a intenção de alcançar um serviço (“Srv”). Contudo, os roteadores podem encaminhar o pacote para o *host* “Hst” através da rede de destino “Dom”, caso eles não conheçam o endereço relacionado ao serviço “Srv” (GRANDL et al., 2012).

Figura 9 – Exemplo de endereçamento DAG

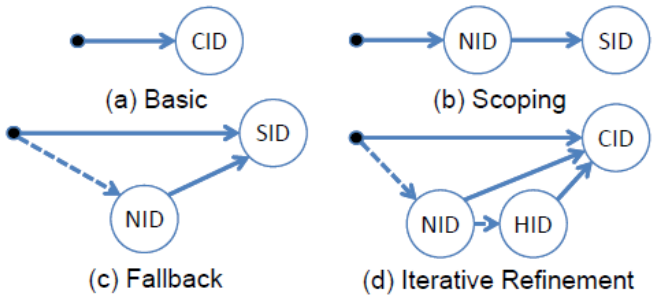


Fonte: GRANDL et al. (2012)

Ao contrário dos endereços IP do CIDR de hoje, que possuem um único identificador, os endereços XIP normalmente incluem vários identificadores (ANAND et al., 2011). Isso é possível devido à flexibilidade proporcionada com o uso do DAG para o endereçamento do

protocolo XIP. Fornecer os caminhos apropriados (DAG) para o correto encaminhamento através da rede XIA é responsabilidade das aplicações (AMBROSIN et al., 2018). Os dispositivos de rede XIA são obrigados a encaminhar os pacotes de acordo com a intenção expressa no DAG (MACHADO et al., 2015).

Figura 10 – Endereçamento DAG para a arquitetura XIA



Fonte: NAYLOR et al. (2014)

A Figura 10 demostra os tipos de endereçamento do protocolo XIP (básico, escopo, *fallback* e refinamento iterativo):
(a) Básico: é o endereçamento mais simples da arquitetura XIA, pois possui apenas a indicação da intenção (nó coletor). Neste caso, acesso direto a partir do CID. Esse tipo de endereçamento é útil em alguns ambientes limitados, como redes locais;
(b) Escopo: nesse exemplo, o pacote precisa ser entregue primeiro ao domínio (NID) e depois é encaminhado para o serviço desejado. Isso é adequado para alcançar endereços que não são globalmente roteáveis;
(c) *Fallback*: esse é o tipo de endereçamento que demonstra a flexibilidade do protocolo XIP, pois permite alternativas de roteamento expressas em um endereço DAG. O *fallback* é representado pela linha pontilhada, que aponta para uma opção de roteamento diferente caso o caminho direto informado no DAG esteja inacessível (MENG et al., 2017);
(d) Refinamento Iterativo: a união dos modelos Escopo e *Fallback* dá origem ao Refinamento Iterativo. No caso de XIDs não roteáveis globalmente, seria necessário passar obrigatoriamente por um domínio (NID ou AD) global, para depois utilizar as possibilidades do *fallback* a partir do primeiro salto da rede (NAYLOR et al., 2014)

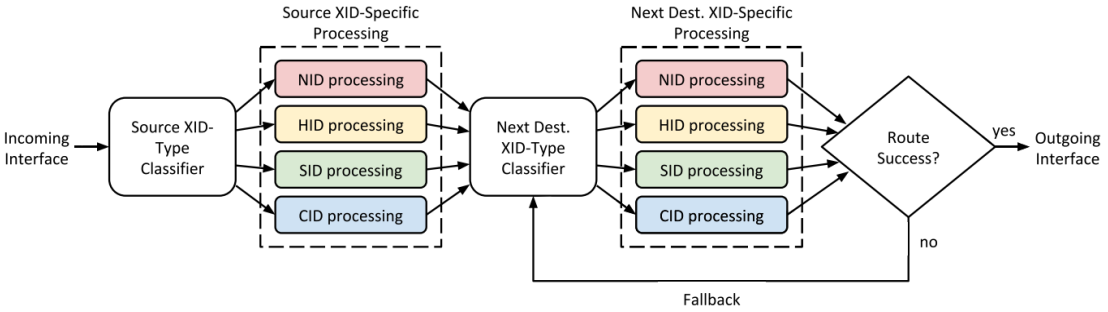
O endereçamento DAG do protocolo XIP possui as seguintes propriedades (ANAND et al., 2011):

- Cada nó do DAG possui um XID exclusivo;
- O roteamento começa com o nó de entrada, que não possui um XID definido, e termina em um nó XID (nó coletor);
- A estrutura de endereçamento não precisa ser alterada para acomodar novos *principals*;
- Cada componente possui exatamente um nó de entrada e pelo menos um coletor;
- Cada nó é um XID que possui possibilidades de encaminhamento para o próximo salto. Estas serão verificadas na ordem em que estão listadas, até não restarem opções.

2.2.6 Cabeçalho XIP

Uma das principais características do processo de endereçamento XIP é a flexibilidade de definir caminhos alternativos como destinos para um pacote. Essa propriedade, chamada *fallback*, diminui o tempo de inatividade e de interrupção do serviço (AMBROSIN et al., 2018). A Figura 11 exibe um diagrama de roteamento que demonstra a utilização do *fallback*. A sua compreensão é importante para proporcionar um melhor entendimento da função de cada campo que compõe o cabeçalho do protocolo XIP.

Figura 11 – Diagrama de roteamento usando *fallback*



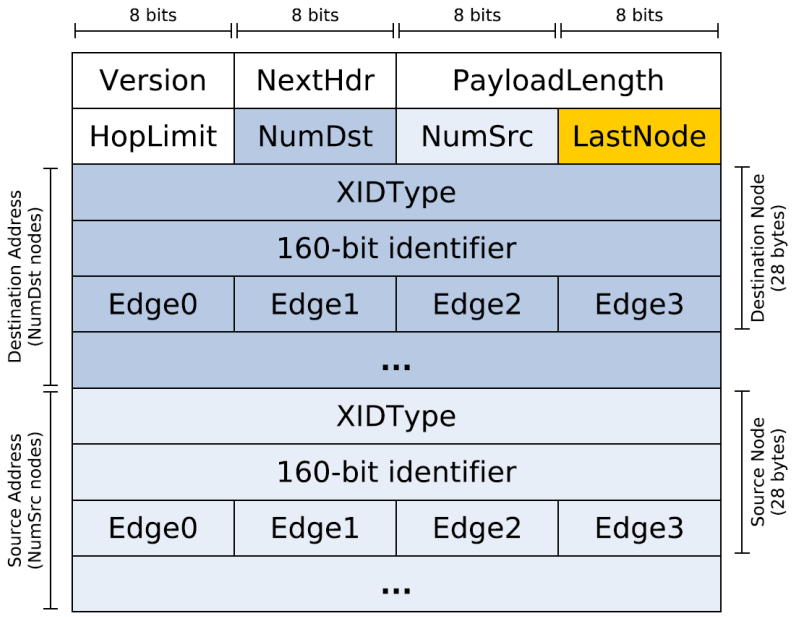
Fonte: AMBROSIN et al. (2018)

O endereçamento *fallback* permite que as partes que se comunicam especifiquem caminhos alternativos (no DAG) para o caso dos roteadores não conseguirem encontrar a intenção principal. Assim, suas arestas de saída são verificadas na ordem em que estão listadas. Se o roteador não puder avançar usando a primeira aresta (*edge0*), ele tentará a próxima (*edge1*), repetindo esse processo até encontrar o próximo salto ou ficar sem arestas (ao chegar a *edge3*), o que implica que nenhum nó é acessível (ANAND et al., 2011). Diante dessas afirmativas, deve-se descrever os campos que integram o cabeçalho XIP, presentes na Figura 12, para identificar os atributos responsáveis pelo funcionamento dessas propriedades.

O cabeçalho do protocolo XIP codifica um endereço de destino e um endereço de origem. Na arquitetura XIA, um pacote de intenção é enviado a partir do DAG de destino. Dessa forma, obtém-se como resposta outro pacote, de mesmo formato, com os dados solicitados e tendo utilizado o caminho inverso do DAG de destino para chegar à origem. *NumDst* e *NumSrc* indicam a quantidade de nós que compõem esses endereços de destino e de origem. Como resultado, os endereços possuem tamanho variável. A seguir, estão listados os campos e agrupamentos que constituem o cabeçalho do protocolo XIP:

- *Version (8 bits)*: define a versão do protocolo XIP;
- *NextHdr (8 bits)*: indica o próximo tipo de cabeçalho (de um protocolo de transporte, por exemplo);
- *PayloadLenght (16 bits)*: define o tamanho dos dados;
- *HopLimit (8 bits)*: determine o limite de saltos através dos roteadores da rede;

Figura 12 – Cabeçalho do protocolo XIP



Fonte: BYERS et al. (2014)

- *NumDst (8 bits)*: indica o número de nós de destino do cabeçalho. Este valor refere-se à quantidade de XIDs informados no DAG definido para que este pacote chegue ao seu destino;
- *NumSrc (8 bits)*: indica o número de nós de origem do cabeçalho;
- ***LastNode (8 bits)***: ponteiro armazenado pelo cabeçalho do pacote, que registra o último *node* válido (encaminhado) do DAG do endereço de destino. Dessa forma, o dispositivo de rede saberá a partir de onde começar as consultas para realizar o encaminhamento. Isso torna o processamento por salto mais eficiente, permitindo que os roteadores processem um DAG parcial em vez de um DAG completo (HAN et al., 2012). As tentativas de encaminhamento para o próximo salto da rede são realizadas a partir das arestas (*edges*) do *LastNode*;
- *XIDType (32 bits)*: tipo de XID que pode corresponder a uma rede (ou domínio), *host*, serviço ou conteúdo. A Tabela 2 exibe os valores dos *principals* que podem ser usados no *XIDType*;
- *Identifier - ID (160 bits)*: identificador criptografado vinculado ao *XIDType*;
- *Edge (8 bits)*: é uma aresta (ou borda) que faz referência a um dos nós registrados no cabeçalho do pacote. Quando o último nó válido é referenciado e indicado como *LastNode*, o roteador tenta encaminhar o XID correspondente a cada nó apontado pelas arestas, na ordem de prioridade definida pela sequência (*Edge0*, *Edge1*, *Edge2* e *Edge3*);
- *Destination Node*: agrupamento dos campos *XIDType*, *ID*, *Edge0*, *Edge1*, *Edge2* e *Edge3*, que corresponde a um nó de destino;
- *Source Node*: agrupamento dos campos *XIDType*, *ID*, *Edge0*, *Edge1*, *Edge2* e *Edge3*, que corresponde a um nó de origem;
- *Destination Address*: agrupamento de nós de destino que tem sua quantidade definida pelo campo *NumDst*. Esses nós correspondem aos XIDs informados no DAG;
- *Source Address*: agrupamento de nós de origem que tem sua quantidade definida em *NumSrc*.

Tabela 2 – Valores constantes na XIA

Valor	Descrição	Tipo
0xC0DE	Ethernet type XIA	*Ethertype*
0x7f	Empty edge	DAG
0x7e	Entry node index	DAG
0x10	Autonomous Domain (AD ou NID)	*Principal*
0x11	Host (HID)	*Principal*
0x12	Content (CID)	*Principal*
0x13	Service (SID)	*Principal*
0x14	United 4ID	*Principal*
0x15	XIP over IPv4	*Principal*
0x16	XIP over UDP over IPv4	*Principal*
0x17	eXpressive Datagram Protocol	*Principal*
0x18	Serval's serviceID	*Principal*
0x19	Serval's flowID	*Principal*
0x20	zFilter (multicast)	*Principal*
0x21	Longest Prefix Matching	*Principal*

Fonte: O Autor (2020)

2.3 Programabilidade de Plano de Dados

As redes de computadores podem ser divididas em três planos de funcionalidade: dados, controle e gerenciamento. O plano de dados refere-se aos dispositivos que realizam o encaminhamento dos dados na rede. O plano de controle está relacionado aos protocolos usados para preencher as tabelas de encaminhamento dos elementos do plano de dados. E o plano de gerenciamento inclui os serviços de *software* de gerência (KREUTZ et al., 2015).

Nas redes IP convencionais, os planos de controle e dados estão juntos, associados nos mesmos dispositivos de rede. Essa disposição foi eficaz durante alguns anos de existência da *Internet*, contudo, tornou-se um entrave para a evolução da rede, tendo em vista a crescente demanda por novas funcionalidades. Desse modo, assegurar um núcleo da rede mais programável é um começo promissor para resolver o problema de rigidez ou dificuldade de mudança da arquitetura atual (REXFORD; DOVROLIS, 2010).

A aspiração por redes mais programáveis teve início ainda no final do século passado, como nas propostas de redes ativas (TENNENHOUSE et al., 1997; ALEXANDER et al., 1998). Contudo, foi nos anos 2000 que este propósito progrediu significativamente, com a separação do plano de controle e do plano de dados, a partir do desenvolvimento do paradigma de *Software Defined Networking* (SDN) e da proposição do protocolo *Openflow* (MCKEOWN et al., 2008).

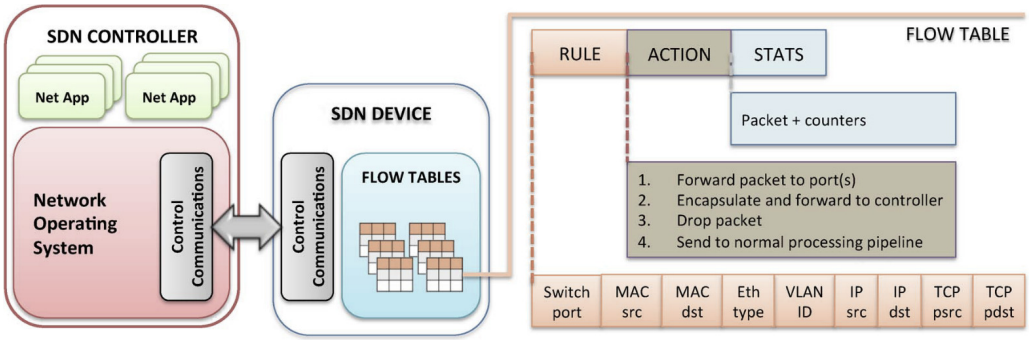
Com a mudança provocada por SDN, a inteligência da rede passa a ser centralizada no plano de controle, composto por diferentes controladores de rede, que podem integrar um ambiente centralizado ou distribuído (KALJIC et al., 2019). Os dispositivos no plano de dados

realizam o encaminhamento de pacotes com base nas regras inseridas pelos controladores. Esses, no plano de controle, supervisionam a rede subjacente e oferecem interface flexível para implementação de aplicações de serviços de rede (XIA et al., 2015).

SDN e *Openflow* são propostas originalmente acadêmicas que obtiveram grande projeção na indústria nos últimos anos. Evidência disso é que os grandes fabricantes de dispositivos de rede possuem equipamentos comerciais com suporte a *Openflow*. Também há o fato de que algumas das maiores empresas de tecnologia da informação do mundo (Google, Microsoft, Yahoo, etc) contribuíram para a criação da *Open Networking Foundation* (ONF[19]), que tem como objetivo principal promover e adotar o SDN por meio do desenvolvimento de padrões abertos (KREUTZ et al., 2015).

Uma circunstância fundamental para expandir as possibilidades de programação de dispositivos de rede foi a existência de padrão aberto que provê a comunicação entre os planos de controle e de dados (BIFULCO; RETVARI, 2018). É incontestável o destaque do *Openflow* neste contexto de dissociação e programabilidade. Seu modelo de processamento (Figura 13) permite que controladores interajam com dispositivos em toda a rede para inserir, modificar ou remover entradas da tabela do plano de dados (HANCOCK; MERWE, 2016).

Figura 13 – Dispositivo SDN com suporte a *Openflow*



Fonte: KREUTZ et al. (2015)

Nos últimos anos, as demandas por crescimento do *Openflow* o tornaram mais complexo, devido à adição de um conjunto cada vez maior de tipos de cabeçalho a que ele deve dar suporte. Essas extensões sucessivas da especificação tendem a continuar, pois certamente novos recursos serão desejados (por redes de data center, operadoras de telefonia, etc), tendo em vista que novas funcionalidades são inseridas mais rapidamente através do uso de *switches* implementados em *software* (HANCOCK; MERWE, 2016).

Contudo, mesmo com as extensões do *Openflow*, o plano de dados permanece um tanto rígido, pois é modificado de acordo com a evolução das versões do protocolo. O que tem se mostrado limitado do ponto de vista de resposta às inovações. Dessa forma, almeja-se que os *switches* deem suporte a mecanismos flexíveis de análise e combinação de campos de cabeçalho, possibilitando ao controlador acessar esses recursos através de uma interface aberta comum.

[19] https://www.opennetworking.org/

A partir disso, as intenções foram voltadas também para a ampliação das possibilidades de programação do plano de dados (BOSSHART et al., 2014).

Como resultado de um estudo sobre flexibilidade e programação do plano de dados em SDN, no trabalho de KALJIC et al. (2019) concluiu-se que a evolução do plano de dados passa por um desvio gradual da arquitetura original do *Openflow*. E que os requisitos para avançar no desenvolvimento de plano de dados apontam para soluções que abordem adequadamente os problemas de programação e flexibilidade. Ressalta também que uma linguagem de programação independente de plataforma e uma estrutura baseada em estados, representam direções futuras para a programabilidade de plano de dados.

Nessa direção, foi apresentado em 2013 o projeto *Reconfigurable Match Table* - RMT (BOSSHART et al., 2013), que propõe primitivas de ação que especificam como lidar com o cabeçalho do pacote e permite o processamento independente de protocolo (KALJIC et al., 2019). O trabalho demonstrou que um plano de dados programável é possível mesmo em *hardware* ASIC[20]. Esse resultado inspirou ferramentas como *Programming Protocol-independent Packet Processors* (P4), *Protocol Oblivious Forwarding* (POF) e o comutador de *software* PISCES (HANCOCK; MERWE, 2016), destacando-se a linguagem P4 e os projetos vinculados a ela, que têm sido vistos com crescente entusiasmo pela academia (BIFULCO; RETVARI, 2018).

2.4 *Programming Protocol-independent Packet Processors* (P4)

Programming Protocol-independent Packet Processors (P4) é uma linguagem de programação para expressar como os pacotes são processados pelo plano de dados de um elemento de encaminhamento programável (BOSSHART et al., 2014). A linguagem pode ser implementada em *switches* (*hardware* ou *software*), placas de rede, roteadores ou dispositivos programáveis do tipo FPGA[21] (GARCIA et al., 2018). P4 (P4-16) prevê a independência de protocolo e de plataformas, obtidas através da introdução de reconfigurabilidade na análise e processamento de pacotes, além da abstração do *hardware* (KALJIC et al., 2019).

Ser independente de protocolo significa que não se estabelece previamente o formato de um pacote. Isso permite a definição de novos protocolos, de acordo com a necessidade, e elimina as restrições de como pacotes individuais podem ser correlacionados. Além disso, P4 também permite a alocação flexível de memória do dispositivo (HILL; GROSSO, 2018).

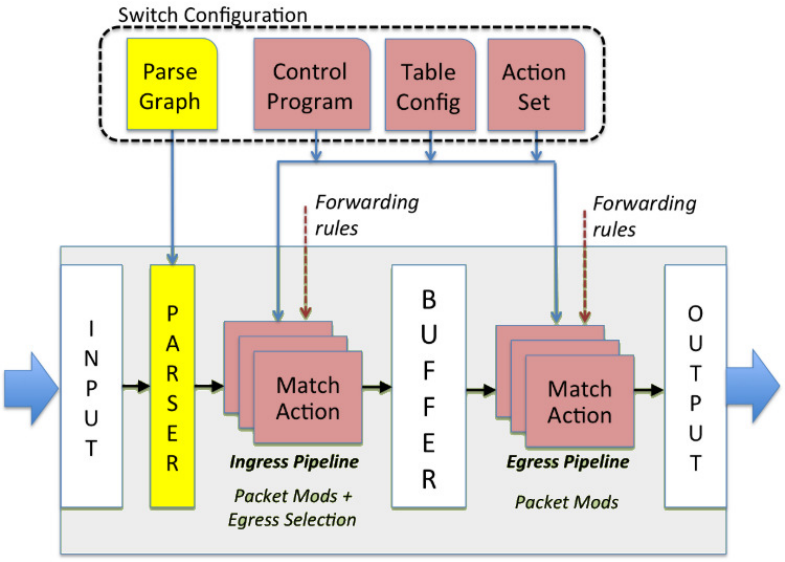
Baseado na arquitetura *Protocol-Independent Switch Architecture* (PISA), P4 possui um modelo abstrato de encaminhamento a partir do qual é possível expressar a configuração de um dispositivo e como os pacotes serão processados (GARCIA et al., 2018). As principais abstrações fornecidas pela linguagem P4 são: as declarações de cabeçalho (*headers*), definindo estrutura e tamanho dos campos; o analisador de pacotes (*parser*), que determina a sequência de

[20] Application-Specific Integrated Circuit: chip customizado visando propósito específico.

[21] *Field Programmable Gate Array* (ou matriz de portas programáveis) é um dispositivo lógico programável que suporta a implementação de circuitos digitais.

diferentes cabeçalhos; as tabelas (*match-action*), que associam chaves definidas pelo usuário a ações de correspondência; o fluxo de controle (*control flow*); o suporte a objetos externos (*externs*) e o uso de estruturas de dados (*metadata*). A Figura 14 demonstra o modelo abstrato de encaminhamento de pacotes da linguagem P4.

Figura 14 – Modelo abstrato de encaminhamento usando P4



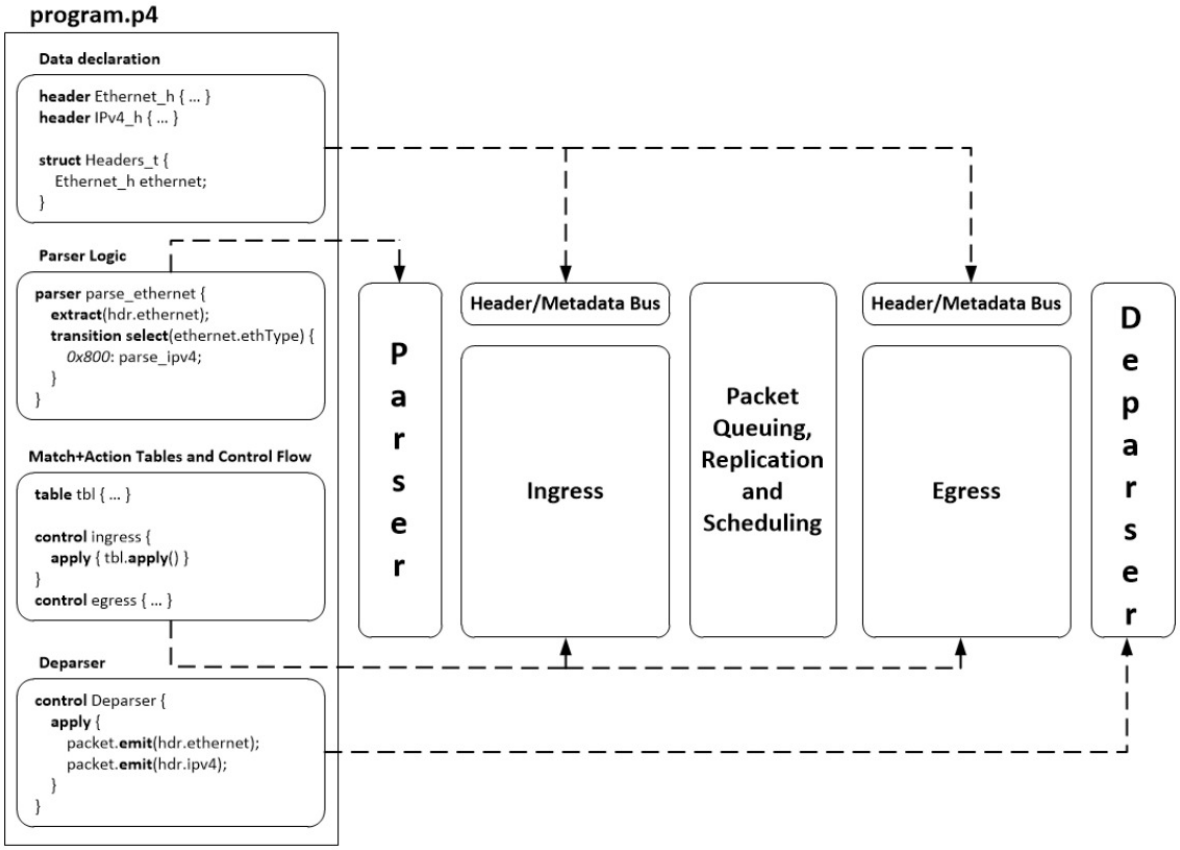
Fonte: BOSSHART et al. (2014)

Um programa P4 começa analisando um pacote com base em cabeçalhos definidos pelo usuário. A natureza independente do protocolo, possibilita que esses cabeçalhos sejam referentes a um protocolo conhecido, como o IP, ou algo completamente novo. Depois que os cabeçalhos são analisados, as ações são executadas de acordo com resultados de pesquisas realizadas na tabela de encaminhamento. O P4 permite correspondências da tabela com campos dos cabeçalhos de pacotes ou com metadados adicionados ao código (HILL et al., 2019). A Figura 15 apresenta a estrutura de um programa P4, exibindo os seus componentes principais.

Cada entrada em uma tabela especifica uma ação a ser executada. Sem esse registro, uma ação padrão pode ser configurada. As ações que podem ser implementadas pelo usuário são baseadas em um determinado conjunto de primitivas estabelecidas na especificação P4. Elas permitem modificar campos, adicionar ou remover cabeçalhos e clonar de pacotes. Embora o P4 permita a análise, pesquisas em tabela e a execução de ações no plano de dados, as modificações nas tabelas devem ser feitas pelo plano de controle. Contudo, a especificação do P4 não define a interface entre o plano de dados e o plano de controle (P4-CONSORTIUM, 2019).

Para fazer a interface do plano de controle com um plano de dados escrito em P4, pode-se utilizar o *P4Runtime*. Aliado ao compilador P4C e ao *switch* virtual *Behavioral Model version 2* (BMv2), permitem um fluxo de execução completo de um programa P4, que pode ser observado na Figura 16, destacando-se esses três componentes.

Figura 15 – Estrutura de um programa P4



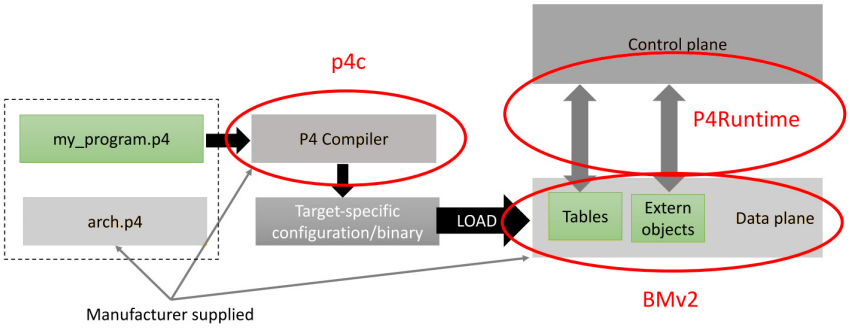
Fonte: OSINSKI (2020)

P4C é o compilador para a linguagem de programação P4, que suporta as versões P4-14 e P4-16. Possui um processo simples de compilação que gera um arquivo JSON (*JavaScript Object Notation*), através do módulo *p4c-bm*. Esse arquivo é usado como entrada do código P4 no *switch*. *Behavioral Model version 2 (BMv2)* é o comutador em *software* usado como referência para P4. Ele recebe o arquivo originado na compilação e realiza a sua interpretação para implementar o comportamento do processamento de pacotes definidos pelo programa P4 (GARCIA et al., 2018).

O BMv2 não tem a intenção de ser uma opção de *switch* implementado em *software* a nível de produção. O objetivo é ser uma ferramenta para desenvolvimento, teste e depuração de códigos P4 criados para o planos de dados, assim como para a solução voltada ao plano de controle. Do ponto de vista de desempenho (taxa de transferência e latência), o BMv2 é inferior a um *switch* em *software* usado no nível de produção, como o *Open vSwitch* (BAS, 2019).

P4Runtime é uma interface de plano de controle que administra o comportamento do encaminhamento de pacotes em tempo de execução. Ele preenche as tabelas de encaminhamento e manipula seu estado a partir de um programa P4. Não é vinculado a qualquer tipo de *hardware*, portanto, a estrutura do *P4Runtime* não sofre alterações, independente de quais protocolos e recursos o plano de dados suporte. Assim, a mesma API pode ser usada para controlar vários tipos de dispositivos diferentes (OCONNOR et al., 2019).

Figura 16 – Fluxo de execução de um programa P4



Fonte: GARCIA et al. (2018)

Para MCKEOWN et al. (2017), o *P4Runtime* é uma nova forma do *software* responsável pelo plano de controle manipular o plano de dados de um *switch*, roteador, *firewall*, balanceador de carga, etc. Também considera que seu aspecto mais inovador é permitir o controle de uma variedade de planos de encaminhamento, independente de ser construído a partir de uma função fixa, *switch* programável ASIC, um FPGA, ou um comutador em *software* executando em um servidor x86. Dessa forma, o plano de controle pode ser uma pilha de protocolos executando localmente no sistema operacional do *switch* local ou remotamente em um servidor.

Antes do *P4Runtime*, os dispositivos desenvolvidos em P4 eram controlados através de APIs de plano de controle personalizadas e proprietárias. Essas soluções próprias são incompatíveis e não podem ser portadas para outro sistema. O *P4Runtime* tenta resolver isso definindo uma API padrão, aberta e independente de plataforma (BAS; MOHSIN, 2019).

2.5 Projeto FIXP

O projeto *Future Internet Exchange Point* (FIXP) busca alavancar a *Internet* do futuro no Brasil através da coexistência e interconexão de múltiplas arquiteturas de rede. A iniciativa tem como participantes a Universidade Federal de Pernambuco (UFPE), a Universidade Federal de São Carlos (UFSCar), a Universidade Federal de Uberlândia (UFU) e o Instituto Nacional de Telecomunicações (Inatel).

A implementação mais recente da proposta do projeto FIXP é apresentada em GAVAZZA et al. (2020). Nesse trabalho, considera-se que promover um ambiente em que diferentes arquiteturas de rede sejam interconectadas e executadas paralelamente na mesma infraestrutura representa a melhor abordagem para uma *Internet* do Futuro. Sendo assim, esta interconexão é realizada através de um *Future Internet Exchange Point* (FIXP).

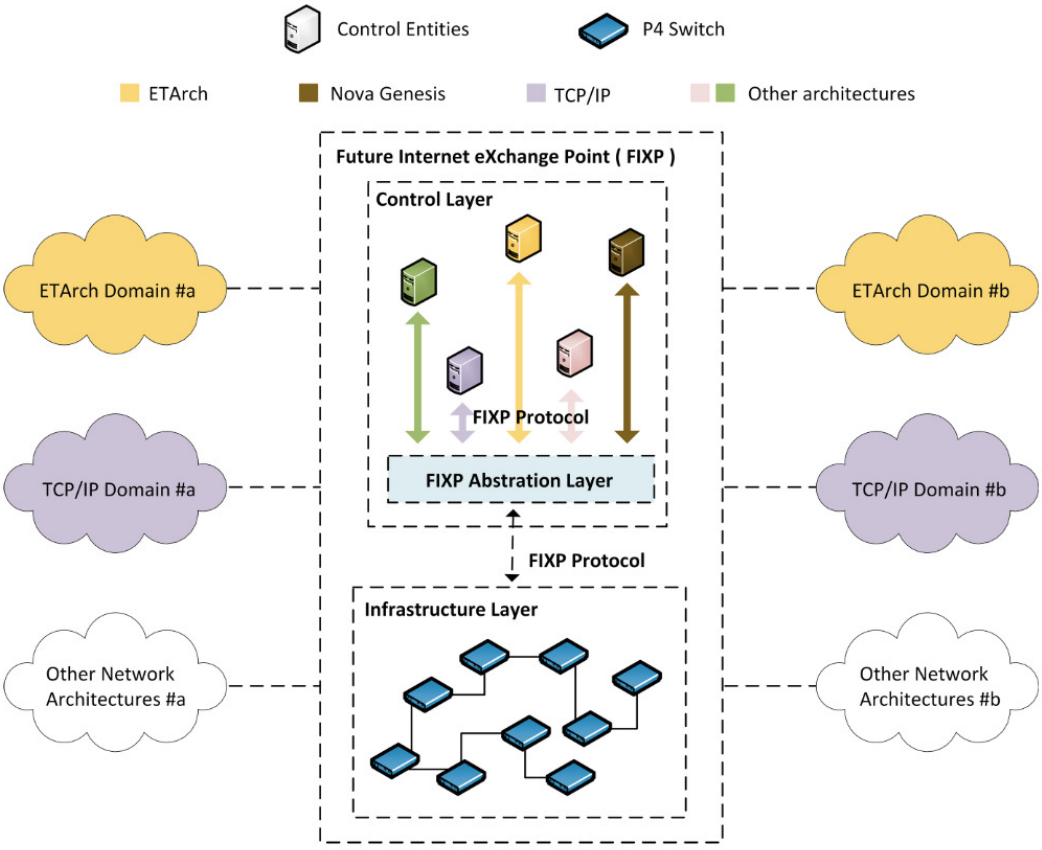
O FIXP visa promover a interconexão de FIAs fisicamente separadas, permitindo a troca de tráfego entre elas e também com a arquitetura atual da *Internet*. A iniciativa tem como pilares para o seu desenvolvimento os conceitos de *Software Defined Networking* (SDN), *Network Functions Virtualization* — NFV (HAN et al., 2015), *Software Defined eXchange* — SDX (GUPTA et al., 2014) e *Cloud Computing*. Dessa forma, visa promover a troca de tráfego entre

diferentes FIAs implantadas em *Internet Service Providers* (ISPs) diversos. Essa possibilidade de interconectar domínios de FIAs com propostas distintas permite às aplicações focar no que cada arquitetura oferece de melhor.

A implementação atual do projeto utiliza duas arquiteturas: a FIA ETArch e o modelo tradicional baseado em TCP/IP. A Figura 17 exibe um exemplo com as arquiteturas de rede presentes em domínios diferentes e a interconexão realizada pelo FIXP. O tráfego resultante da interconexão de diferentes roteadores ocorre na sua Camada de Infraestrutura, composta por comutadores desenvolvidos em P4 que contêm as informações relacionadas às arquiteturas, como tabelas de correspondência e cabeçalhos.

Em relação à interconexão das FIAs, o FIXP recebe os pacotes, identifica a arquitetura (através do *ethertype*) e os envia para a FIA correspondente, conforme regras de encaminhamento estabelecidas. A Camada de Controle armazena as entidades de controle de cada arquitetura suportada, que populam os planos de dados correspondentes com as regras de encaminhamento, sendo que a comunicação entre as camadas de Infraestrutura e Controle é realizada pelo Protocolo FIXP. A Camada de Abstração é responsável por inspecionar e encaminhar, às entidades de controle correspondentes, as primitivas de controle encapsuladas e enviadas pelo Protocolo FIXP.

Figura 17 – Arquitetura do FIXP



Fonte: GAVAZZA et al. (2020)

2.6 Trabalhos Relacionados

Esta Seção tem como objetivo apresentar trabalhos relacionados a esta dissertação. Nas pesquisas realizadas na literatura acadêmica, não foi encontrada outra proposta voltada à implementação do plano de dados da arquitetura XIA, utilizando P4. Também não foi identificado projeto desenvolvido com alguma outra linguagem pertencente ao mesmo segmento e voltado à mesma FIA. Dessa forma, descrevem-se a seguir algumas soluções implementadas em pesquisas que se assemelham ou possuem temas comuns aos deste trabalho.

Em MARQUES (2017) apresenta-se a implementação de um roteador para a arquitetura de *Internet* do futuro *Named Data Networking* (NDN), utilizando a linguagem P4-16. Ele desenvolve os principais componentes da arquitetura NDN: *Pending Interest Table* (PIT), *Forwarding Interest Base* (FIB) e *Content Store* (CS). Com destaque para este último, um *cache* para recuperação de conteúdo em roteadores intermediários da rede. Na solução apresentada, o CS pode ser disponibilizado tanto nos registros, quanto no *switch* P4.

A arquitetura NDN é baseada em nome, sendo que os conteúdos são identificados através de nomeação hierárquica, semelhante a uma URL (ex. a/b/c/d). Essa característica dificulta a implementação do seu plano de dados utilizando P4. A especificação da versão utilizada da linguagem define que não é possível lidar com *strings* em campos do cabeçalho, para que estes possam ser verificados pelo analisador (*parser*). Assim, para contornar essa dificuldade, criou-se uma estrutura de partição contendo vários blocos de mesmo tamanho, que acomodam cada uma das partes (separadas por "/") do nome hierarquicamente atribuído. Esses segmentos do nome são convertidos em *hash* para que possam ser manipulados no código P4.

Com a utilização de função de *hash* para representar o nome, foi possível utilizar os atributos dos componentes da arquitetura (PIT, FIB e CS), que armazenam esses endereços para a recuperação e encaminhamento do pacotes através dos nós da rede. Contudo, como os blocos da partição têm tamanho definido, há uma limitação em relação à extensão de nomes que este endereço pode expressar.

Além disso, o endereçamento do NDN é formado pela aninhamento de campos no formato *Type-Length-Value* (TLV), sendo alguns deles de tamanho variável. A linguagem P4 possui um suporte limitado a campos de tamanho variável. Não permitindo, por exemplo, a utilização de campos de cabeçalho com comprimento indefinido como chave para verificação nas tabelas de encaminhamento. Assim, no trabalho de MARQUES (2017), o endereçamento não pôde ser implementado em sua totalidade, sendo necessário definir tamanhos fixos para realizar o roteamento dos pacotes.

A relação do trabalho de MARQUES (2017) com esta dissertação evidencia-se no uso da linguagem P4 para a implementação do plano de dados de uma FIA e pela particularidade de ambas (NDN e XIA) possuírem campos de tamanho variável no seu cabeçalho. Com a diferença que, no caso da XIA, o endereço possui um limite de tamanho. Isso possibilitou a devida implementação do endereçamento e a realização efetiva dos processos de roteamento da arquitetura.

O trabalho de MACHADO et al. (2015) propõe o Linux XIA, uma implementação nativa da *eXpressive Internet Architecture* (XIA) executada a partir do *kernel* do Linux. O projeto defende o que chama de meta-arquitetura de rede, para promover a coexistência de projetos *clean-slate*. Assim, ao invés de projetar mais uma nova arquitetura, optou-se pela utilização da XIA para esse fim, sobretudo pelas suas características de suporte a novos *principals*.

Para demonstrar que o Linux XIA é uma plataforma viável para promover a interconexão de designs *clean-slate*, as arquiteturas Serval (NORDSTRÖM et al., 2012) e zFilter (JOKELA et al., 2009) foram adicionadas à estrutura implementada com a XIA. Também é provido o suporte ao IP, pois a proposta defende que qualquer projeto que pretenda promover um redesenho da *Internet* deverá conviver por anos com a arquitetura atual. O *Serval* adiciona uma camada orientada a serviços, acima da camada de rede. E o *zFilter* é baseado em encaminhamento *multicast* em larga escala.

Com o objetivo de viabilizar a implementação da arquitetura atual, criou-se um conjunto de *principals* XIA chamado 4IDs, que permitem que *hosts* habilitados para XIA se comuniquem como uma rede IPv4. Assim, buscando promover a integração com o IPv4, foram criados os XIDs: X4ID, para a compatibilidade com o TCP/IP; U4ID, equivalente ao UDP; e XPM, semelhante ao recurso de LPM. Para agregar o Serval, foram gerados os identificadores ServalID (usado no roteamento XIA) e FlowID (aplicado em um contexto mais interno à arquitetura). No caso do zFilter, utiliza-se um *Link ID*, que integra a sua estrutura e identifica os pacotes correspondentes. Contudo, para a realização do encaminhamento, ele usa os XIDs da XIA (AD e HID) e do IPv4 (4IDs), através de redirecionamentos.

Por fim, o trabalho implementa um aplicativo *multicast*, capaz de enviar conteúdo de vídeo através de uma rede heterogênea, combinando os XIDs XIA, IP e zFilter. Também expõe proposta para a incorporação da arquitetura *Named Data Networking* (NDN) ao projeto de coexistência do Linux XIA. Para MACHADO et al. (2015), prover uma forma de introduzir novas funcionalidades na camada de rede, de forma incremental, permite que propostas existentes atualmente possam ser testadas ou comparadas. O trabalho destaca que a implementação da XIA no Linux aprimora ainda mais a vocação evolutiva da FIA, pois permite adicionalmente a interoperação de arquiteturas em um ambiente comum. O uso da XIA, juntamente com a ideia de coexistência ou integração de arquiteturas de rede, apontam a principal relação entre o "Linux XIA" e esta dissertação.

3 IMPLEMENTAÇÃO DO PLANO DE DADOS DA XIA

Este Capítulo apresenta a implementação do plano de dados da *eXpressive Internet Architecture* (XIA), usando a linguagem P4 para aplicá-lo em um dispositivo de rede programável. Para isso, inicialmente, apresentam-se os requisitos para a implementação do plano de dados da arquitetura XIA. Na sequência, define-se o escopo para a desenvolvimento do plano de encaminhamento da XIA usando P4. Nas Seções seguintes, detalha-se o desenvolvimento do código P4, destacando seu processo de construção de acordo com os componentes principais da linguagem.

3.1 Requisitos para a implementação do plano de dados da XIA

A arquitetura XIA possui algumas características no seu plano de dados que devem ser observadas para a realização da sua implementação, independente da linguagem ou ferramenta utilizada para este fim. Essas propriedades são baseadas nos seus princípios fundamentais: *principals* diversos, flexibilidade de endereçamento e segurança intrínseca. Os conceitos estão descritos na Subseção 2.2.1 do trabalho.

A comunicação de rede na XIA é realizada a partir do seu protocolo principal, o XIP, que utiliza XIDs para identificar os diversos *principals* para os quais oferece suporte, representando orientações distintas (rede, *host*, serviço e conteúdo). Portanto, para a definição do plano de dados XIA, o principal requisito é implementar o tratamento do cabeçalho XIP. Nele é possível identificar algumas características necessárias como: tamanho de endereço variável; suporte a campos com até 160bits; segurança implementada no identificador de cada *node* que compõe o endereço, usando elementos de criptografia e chave pública.

Entre as características citadas do cabeçalho XIP, o tamanho variável do endereço é a que mais influencia nas suas possibilidades de endereçamento e oferece alternativas de encaminhamento do pacote a partir do seu plano de dados. O endereço XIP, representado por um DAG, pode ser formado por um número diferente de *nodes*. Cada node representa um XID e possui até quatro arestas (*edges*) que indicam as possibilidades de encaminhamento, referenciando outros *nodes* presentes no cabeçalho. Implementar esses campos e garantir o fluxo mencionado são requisitos essenciais para o efetivo funcionamento do plano de dados da XIA

Uma outra questão que deve ser observada refere-se à tabela de encaminhamento do plano de dados. A chave da tabela *match-action* da arquitetura XIA é composta por dois campos (*XIDType* e *ID*), portanto, deve-se confirmar a correspondência apenas se os dois forem válidos para um mesmo *node*. Além disso, a implementação precisa permitir que, ao consultar um *node* na tabela de encaminhamento, em que não haja correspondência, a consulta possa ser realizada novamente com outro *node* do mesmo pacote. Isso possibilita a verificação das alternativas de encaminhamento características da arquitetura.

3.2 Escopo da implementação usando P4

Para estabelecer o escopo da implementação, observou-se requisitos para a compatibilidade com o projeto FIXP, principalmente a linguagem P4, utilizada para o desenvolvimento dos planos de dadas das FIAs que integram o projeto. Também foram analisadas as características da arquitetura XIA, objeto de estudo desta dissertação, observando as possibilidades de implementação das suas propriedades utilizando a linguagem P4.

As metas para desenvolvimento do código foram definidas com base nos princípios fundamentais da arquitetura XIA (*principals* diversos, flexibilidade de endereçamento e segurança intrínseca). Desses, estão no escopo da implementação o suporte a diferentes orientações principais (rede, serviço, *host*, conteúdo) e a configuração das possibilidades de endereçamento, a partir das quais os processos de encaminhamento da arquitetura são realizados.

Contudo, a implementação não contempla a inclusão de propriedades para promover a segurança intrínseca. A ausência desse recurso não interfere na incorporação das demais características, e também não exigirá mudanças na estrutura principal do código ao adicioná-la. Prover a segurança na XIA implica na aplicação de *hash* de chave pública/privada nos identificadores XID (HID, SID e CID). O que, no contexto deste trabalho, será realizado futuramente a partir de elementos externos (*externs*) da linguagem P4.

Portanto, optou-se por priorizar os dois primeiros princípios fundamentais da arquitetura (citados na Subseção 2.2.1), para atender aos requisitos relacionados à diversidade de *principals* e aos processos de encaminhamento da XIA. Dessa forma, o plano de dados implementado deve permitir um número variável de (*nodes*) no cabeçalho do pacote e possibilitar a busca por alternativas de correspondência entre os *nodes* de destino presentes no mesmo. Essas premissas estão de acordo com as regras de cabeçalho apresentadas na Subseção 2.2.6.

O protocolo XIP possui uma estrutura de campos no seu cabeçalho que permite uma quantidade variável de *nodes* que compõem os endereços, contudo, existe um número máximo de ocorrências de *nodes* no cabeçalho. MACHADO (2015) aponta o tamanho máximo de nove *nodes* para o endereço do cabeçalho XIP, porém, salienta que quatro *nodes* atenderiam a maioria dos casos de uso da XIA. Diante disso, para este trabalho, adotou-se a quantidade máxima de quatro *nodes* por endereço do cabeçalho XIP (origem ou destino).

3.3 Definição de cabeçalhos (*headers*)

Um algoritmo desenvolvido em P4 começa com a definição dos cabeçalhos, onde são especificados os tipos, tamanhos e sequência dos campos. Eles são organizados sequencialmente e em bloco, de acordo com a ordem em que devem ser verificados pelo analisador. Conforme descrito na Seção 2.5, na definição da arquitetura do projeto FIXP, para identificar o destino de um pacote utiliza-se o *ethertype* correspondente à arquitetura. Dessa forma, o primeiro cabeçalho implementado é o *Ethernet*, pois o tráfego de pacotes XIP é realizado através dele.

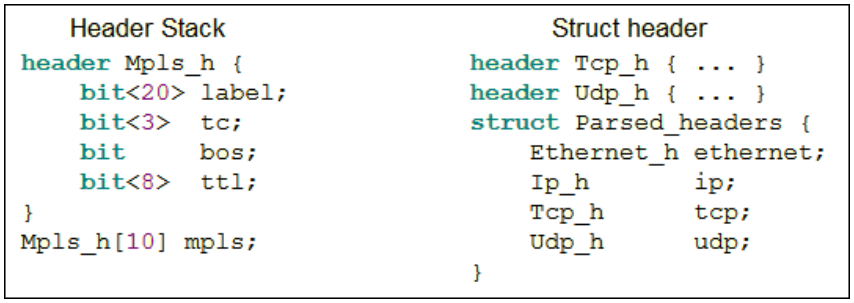
O Código-fonte 1 contém a definição do cabeçalho *Ethernet*. Na linha 1, registra-se o *ethertype* da XIA como uma constante (*ETHERTYPE_XIA*), além do tamanho do campo que ele representa. Esse valor foi apresentado na Tabela 2 (Subseção 2.2.6), que exibe as constantes da arquitetura. Na linha 2, o comprimento do campo de endereço do cabeçalho *Ethernet* é referenciado em *EthernetAddr*. Das linhas 4 a 8, define-se o cabeçalho *Ethernet*, incluindo três campos: endereço de destino (*dst_addr*), endereço de origem (*src_addr*) e *Ethertype*.

Código-fonte 1 – Definição do cabeçalho *Ethernet*

```
1 const   bit<16> ETHERTYPE_XIA = 0xc0de;   //Ethernet Type
2 typedef bit<48> EthernetAddr;
3
4 header Ethernet_t {
5     EthernetAddr    dst_addr;
6     EthernetAddr    src_addr;
7     bit<16>         ethertype;
8 }
```

O cabeçalho do protocolo XIP, núcleo de rede da arquitetura XIA, é representado na Figura 12. Para implementá-lo, é necessário analisar sua composição e características particulares. Devido ao tamanho variável dos seus endereços de destino e origem, com até quatro *nodes* cada, conforme detalhado na Subseção 2.2.6, sua implementação não pode ser realizada em apenas um bloco de campos sequenciais (como no *Ethernet*). Assim, analisando os recursos disponíveis na linguagem, através de documentos oficiais, tutoriais e exemplos, chegou-se a duas possibilidades: o uso de pilhas ou de estruturas de cabeçalho, conforme exemplos retratados na Figura 18.

Figura 18 – Comparativo entre *header stack* e *struct header*



Fonte: Adaptada de P4-CONSORTIUM (2019)

A pilha de cabeçalhos (*Header Stack*) representa uma matriz. Desse modo, os campos referentes ao *node* do cabeçalho XIP poderiam compor um cabeçalho separado e este seria representado como *XIP_node[4]*, indicando a presença de quatro cabeçalhos. Já o componente *Struct header*, pode agrupar cabeçalhos e campos. Assim, de forma análoga à matriz, seria um bloco contendo os campos de um *node* XIP. Contudo, não poderia representá-lo apenas uma vez como o *header stack*, sendo necessária a repetição dessa estrutura para definir a quantidade máxima de *nodes*. Após a realização de testes preliminares, optou-se por referenciar a parte dos

campos do XIP relativas ao *node* utilizando a propriedade *Struct header*. Com isso, os campos foram divididos entre os que possuem tamanho fixo e os de comprimento variável, sendo que um agrupamento de *nodes* dá origem a um endereço de tamanho variável.

Código-fonte 2 – Definição dos campos do cabeçalho XIP

```
1  header XIP_fixedfields_t {
2      bit<8>      version;
3      bit<8>      next_hdr;
4      bit<16>     payload_len;
5      bit<8>      hop_limit;
6      bit<8>      num_dst;
7      bit<8>      num_src;
8      bit<8>      last_node;
9  }
10
11 struct XIP_Node_t {
12     bit<32>     xid_type;
13     bit<160>    id;
14     bit<8>      edge0;
15     bit<8>      edge1;
16     bit<8>      edge2;
17     bit<8>      edge3;
18 }
```

No Código-fonte 2 é possível observar a separação realizada a fim de definir os campos do XIP. O cabeçalho *XIP_fixedfields_t* (linha 3) representa os campos que não possuem variação de tamanho, enquanto que a estrutura *XIP_Node_t* (linha 13) referencia um (*node*) do XIP. Cada endereço do protocolo (origem ou destino) é composto por até quatro *nodes*, assim, esse é o elemento de comprimento variável da XIA. A partir dessa estrutura (*XIP_Node_t*), são definidos os cabeçalhos que correspondem aos endereços de tamanhos variáveis da arquitetura (Código-fonte 3):

- *One_Node_t*: Cabeçalho composto por um *node* XIP (*xid_type*, *id*, *edge1*, *edge2*, *edge3*, *edge4*);
- *Two_Nodes_t*: Cabeçalho que contém dois *nodes* XIP;
- *Three_Nodes_t*: Cabeçalho que contém três *nodes* XIP;
- *Four_Nodes_t*: Cabeçalho que contém quatro *nodes* XIP.

Código-fonte 3 – Criação de cabeçalhos com número variável de *nodes*

```
1  header One_Node_t {
2      XIP_Node_t  node1;
3  }
4
5  header Two_Nodes_t {
6      XIP_Node_t  node1;
7      XIP_Node_t  node2;
8  }
9
10 header Three_Nodes_t {
```

```
11      XIP_Node_t  node1;
12      XIP_Node_t  node2;
13      XIP_Node_t  node3;
14  }
15
16  header  Four_Nodes_t {
17      XIP_Node_t  node1;
18      XIP_Node_t  node2;
19      XIP_Node_t  node3;
20      XIP_Node_t  node4;
21  }
```

Todos os cabeçalhos definidos são inseridos em uma estrutura (*struct_headers*), que determina que eles poderão ser analisados no *parser*. O Código-fonte 4 exibe os cabeçalhos implementados neste trabalho, na ordem em que foram definidos no código: *ethernet*, *fixedfields* (lista os campos de tamanho fixo do XIP), quatro possibilidades de cabeçalho para o endereço de destino (relacionadas às quantidades de *nodes*) e mais quatro relativas ao endereço de origem do protocolo principal da XIA.

Código-fonte 4 – Estrutura com cabeçalhos implementados

```
1  struct headers_t {
2      Ethernet_t          ethernet;
3      XIP_fixedfields_t   fixedfields;
4      One_Node_t          dst_one_node;
5      Two_Nodes_t         dst_two_nodes;
6      Three_Nodes_t       dst_three_nodes;
7      Four_Nodes_t        dst_four_nodes;
8      One_Node_t          src_one_node;
9      Two_Nodes_t         src_two_nodes;
10     Three_Nodes_t       src_three_nodes;
11     Four_Nodes_t        src_four_nodes;
12 }
```

3.4 Analisador de cabeçalhos (*parser*)

No componente *parser*, realiza-se a extração dos pacotes definidos em *struct headers_t* e determina-se quais campos serão analisados. Desse modo, o primeiro cabeçalho verificado é o *Ethernet*. O Código-fonte 5 demonstra esse procedimento, onde: na linha 1, inicia-se o *parser*; a linha 2 mostra a extração do cabeçalho; a linha 3 define o campo a ser analisado (*ethernet.ethertype* corresponde a "cabeçalho.campo"), que também será usado para determinar a transição para o próximo *parser*; e, na linha 4, verifica-se o conteúdo do campo *ethertype* do cabeçalho, neste caso, o *ethertype* XIA (0xc0de — definido como uma constante no Código-fonte 1). Se validado, a transição para o *parser* do próximo cabeçalho (*parse_xip_fixedfields*) é realizada.

Código-fonte 5 – Análise do cabeçalho *Ethernet*

```
1     state parse_ethernet {
2         packet.extract(hdr.ethernet);
3         transition select (hdr.ethernet.ethertype) {
4             ETHERTYPE_XIA: parse_xip_fixedfields;
5             default: accept;
6         }
7     }
```

Feita a transição para o *parse_xip_fixedfields*, o cabeçalho que contém os campos de tamanho fixo do protocolo XIP é extraído, conforme observado na linha 2 do Código-fonte 6. Em seguida, realiza-se a transição para o *parse_select_xip_dst_nodes*. Nesse momento, nenhum campo do cabeçalho *fixedfields* é analisado, pois eles serão verificados a partir do *parser* relativo ao endereço de destino.

Código-fonte 6 – Extração do cabeçalho com campos de tamanho fixo do XIP

```
1     state parse_xip_fixedfields {
2         packet.extract(hdr.fixedfields);
3         transition parse_select_xip_dst_nodes;
4     }
```

O cabeçalho *fixedfields* é composto pelos seguintes campos: *version*, *next_hdr*, *payload_len*, *hop_limit*, *num_dst*, *num_src* e *last_node*. Os componentes *NumDst* e *NumSrc* do XIP, representados no cabeçalho *fixedfields* como *num_dst* e *num_src*, determinam a quantidade de *nodes* que os endereços de destino e origem possuem. Portanto, a análise desses campos define quais cabeçalhos são extraídos para representar esses endereços, de acordo com o número de *nodes* referenciados.

Os endereços de destino e origem do XIP foram implementados, no código P4, com quatro possibilidades de cabeçalho, baseados na quantidade de *nodes*, sendo que apenas uma dessas alternativas é utilizada para cada pacote recebido. No Código-fonte 7, observa-se o analisador *parse_select_xip_dst_nodes*, que realiza a transição a partir da verificação do campo *num_dst* do cabeçalho *fixedfields*, conforme mostra a linha 2 do código. Assim, o cabeçalho que será extraído na sequência depende do valor registrado no campo *num_dst*, conforme descrito a seguir e visualizado no Código-fonte 7:

- ***num_dst* = 1**: transição realizada para *parse_dst_one_node* (linhas 3 e 10), que extrai os campos do cabeçalho *dst_one_node* (linha 11) e faz a transição para o *parser* que analisa o endereço de origem (linha 12);
- ***num_dst* = 2**: transição realizada para *parse_dst_two_nodes* (linhas 4 e 14), que extrai os campos do cabeçalho *dst_two_nodes* (linha 15) e faz a transição para o *parser* que analisa o endereço de origem (linha 16);
- ***num_dst* = 3**: transição realizada para *parse_dst_three_nodes* (linhas 5 e 18), que extrai os

campos do cabeçalho *dst_three_nodes* (linha 19) e faz a transição para o *parser* que analisa o endereço de origem (linha 20);

- ***num_dst* = 4**: transição realizada para *parse_dst_four_nodes* (linhas 6 e 22), que extrai os campos do cabeçalho *dst_four_nodes* (linha 23) e faz a transição para o *parser* que analisa o endereço de origem (linha 24).

Código-fonte 7 – Definição do cabeçalho que representa o endereço de destino XIP

```
1     state parse_select_xip_dst_nodes {
2         transition select (hdr.fixedfields.num_dst) {
3             1: parse_dst_one_node;
4             2: parse_dst_two_nodes;
5             3: parse_dst_three_nodes;
6             4: parse_dst_four_nodes;
7             default: accept;
8         }
9     }
10    state parse_dst_one_node {
11        packet.extract(hdr.dst_one_node);
12        transition parse_select_xip_src_nodes;
13    }
14    state parse_dst_two_nodes {
15        packet.extract(hdr.dst_two_nodes);
16        transition parse_select_xip_src_nodes;
17    }
18    state parse_dst_three_nodes {
19        packet.extract(hdr.dst_three_nodes);
20        transition parse_select_xip_src_nodes;
21    }
22    state parse_dst_four_nodes {
23        packet.extract(hdr.dst_four_nodes);
24        transition parse_select_xip_src_nodes;
25    }
```

Com a definição do endereço de destino, efetua-se a transição e o próximo *parser* analisa o campo *src_num* do cabeçalho *fixedfields*, realizando assim, o mesmo procedimento executado anteriormente, contudo, agora executado para selecionar o cabeçalho referente ao endereço de origem. Por se tratar do mesmo procedimento realizado para o endereço de destino, a seleção do cabeçalho que contém os campos relativos ao endereço de origem não é detalhada nesta Seção. No entanto, toda a implementação do componente de análise de cabeçalhos (*parser*) pode ser verificada no Apêndice A deste trabalho.

3.5 Controle de fluxo e tabelas *match-action*

O componente *control* de um código P4 inicia com a definição das ações (*actions*) a serem executadas a partir de uma correspondência (*match*) na tabela de encaminhamento do plano de dados desenvolvido. Ao implementar as ações para a tabela *match-action* do plano de dados XIA, deve-se observar, além do encaminhamento do pacote para o próximo salto da rede,

os campos do cabeçalho XIP que sofrerão alterações a partir de uma correspondência nas chaves da tabela, assim como a definição do comportamento para pacotes sem validação.

Conforme apresentado na Subseção 2.2.6, que detalha o cabeçalho XIP, além dos campos relacionados aos endereços de destino e origem, dois outros são ligados à realização do encaminhamento do pacote: *HopLimit*, que estabelece o número máximo de saltos na rede para que um pacote chegue ao seu destino; e *LastNode*, que funciona como um ponteiro que indica qual o último *node* do DAG (endereço) validado na tabela de um comutador e, portanto, encaminhado através da rede.

No Código-fonte 8 é possível visualizar a ação implementada para a realização do encaminhamento, *to_port_action* (linha 5), assim como os procedimentos executados, nesta ação, a partir de uma correspondência válida na tabela *match-action*: registrar o *node* válido como *last_node*, no campo homólogo do cabeçalho *fixedfields*, a partir de um metadado (linha 6); e decrementar o valor do campo *hop_limit*, do cabeçalho *fixedfields*, relativo ao número limite de saltos na rede (linha 7).

Código-fonte 8 – Ações para a tabela *match-action*

```
1     action drop_action() {
2         mark_to_drop(standard_metadata);
3         dropped = true;
4     }
5     action to_port_action (bit<9> port) {
6         hdr.fixedfields.last_node = meta.last_node;
7         hdr.fixedfields.hop_limit = hdr.fixedfields.hop_limit - 1;
8         standard_metadata.egress_spec = port;
9     }
```

O metadado utilizado no Código-fonte 8, assim como outros referenciados ao longo da implementação, são declarados em uma estrutura (*struct metadata_t*) logo após a definição dos cabeçalhos, em *struct headers_t*. Eles devem ter o tipo e tamanho idênticos aos dos campos que representam. A lista de metadados pode ser vista na versão completa do código (Apêndice A).

Com as ações já estabelecidas, a tabela *match-action* deve ser implantada. Para isso, além das ações, é necessário especificar a chave (ou chaves) da tabela, através da qual um endereço válido é identificado. Na arquitetura XIA, os *nodes* presentes no cabeçalho do pacote são identificados pelo seu XID. Conforme apresentado na Subseção 2.2.4, cada XID é composto de um tipo e um identificador. Desse modo, a chave da tabela *match-action* da XIA possui dois campos: *XIDType* e ID.

O Código-fonte 9 apresenta a implementação da tabela *xia1*, onde a chave é definida com os dois campos citados anteriormente: o tipo (linha 3), referenciado pelo metadado *meta.dst_xid_type*; e o identificador (linha 4), declarado através do metadado *meta.dst_id*. Para que haja correspondência, é necessário que os dois valores sejam válidos para um mesmo *node*. A tabela também indica as ações (linhas 7 e 8), previamente implementadas, que têm como

efeitos descartar o pacote (*drop_action*) ou encaminhá-lo para a porta de saída (to_port_action). Além disso, a tabela também estabelece o número máximo de regras de encaminhamento (linha 10) e a ação padrão para pacotes sem correspondência (linha 11).

Código-fonte 9 – Tabela *match-action*

```
1     table xia1 {
2
3         key = {
4             meta.dst_xid_type : exact;
5             meta.dst_id : exact;
6         }
7
8         actions = {
9             drop_action;
10            to_port_action;
11        }
12
13        size = 1024;
14        default_action = drop_action;
15    }
```

A arquitetura XIA possui endereço de tamanho variável e alternativas de encaminhamento baseadas no *node* selecionado do endereço (DAG). Desse modo, para representar o cabeçalho do protocolo XIP no código P4, este foi dividido, possibilitando a definição de opções de cabeçalho referentes às quantidades factíveis de *nodes*. Assim, para que um *node* seja verificado na tabela *match-action*, é necessário primeiro identificar a quantidade de *nodes* presentes no pacote. Em seguida, realizam-se os testes de correspondência na tabela de encaminhamento, de acordo com a prioridade estabelecida nas arestas (*edges*) do *node* indicado.

No componente *control* da linguagem P4, esta averiguação é implementada a partir da opção *apply*, onde são analisadas as condições para as consultas à tabela de encaminhamento. A linguagem de programação P4 não possui opções de *loops*, chamadas recursivas ou ponteiros (FINGERHUT, 2018), portanto, as implementações de verificação realizadas devem levar em conta suas características ou possíveis limitações. Diante disso, essas consultas foram desenvolvidas a partir de estruturas condicionais aninhadas.

O Código-fonte 10 evidencia uma estrutura condicional que realiza consultas para: identificar a quantidade de *nodes* presentes no pacote; revelar o último *node* válido (*last_node*); e atribuir os valores dos *edges*, do *xid_type* e do *id* aos seus metadados correspondentes. Essa porção do código, representada no Código-fonte 10, demonstra as verificações realizadas para a existência de um (linha 1) ou dois (linha 5) *nodes*. Como é possível demonstrar a lógica da implementação utilizando este fragmento do código, pois os processos executados para a ocorrência de dois *nodes* (linhas 6-19) se repetem para as consultas relacionadas às demais quantidades, os procedimentos para três e quatro *nodes* não são detalhados nesta Seção. Contudo, a implementação completa pode ser verificada no Apêndice A desta dissertação.

Código-fonte 10 – Quantidade de *nodes* e atribuição de valores do XID para metadados

```
1     if (hdr.fixedfields.num_dst == 1) {
2         meta.dst_node1_xid_type = hdr.dst_one_node.node1.xid_type;
3         meta.dst_node1_id = hdr.dst_one_node.node1.id;
4     }
5     else if (hdr.fixedfields.num_dst == 2) {
6         if (hdr.fixedfields.last_node == 1) {
7             meta.dst_edge0 = hdr.dst_two_nodes.node1.edge0;
8             meta.dst_edge1 = hdr.dst_two_nodes.node1.edge1;
9         }
10        else if (hdr.fixedfields.last_node == 2) {
11            meta.dst_edge0 = hdr.dst_two_nodes.node2.edge0;
12            meta.dst_edge1 = hdr.dst_two_nodes.node2.edge1;
13        }
14        meta.dst_node1_xid_type = hdr.dst_two_nodes.node1.xid_type;
15        meta.dst_node1_id = hdr.dst_two_nodes.node1.id;
16        meta.dst_node2_xid_type = hdr.dst_two_nodes.node2.xid_type;
17        meta.dst_node2_id = hdr.dst_two_nodes.node2.id;
18    }
19    else if (hdr.fixedfields.num_dst == 3) {
20    ...
21    }
22    else if (hdr.fixedfields.num_dst == 4) {
23    ...
24    }
```

Os procedimentos realizados no Código-fonte 10 são descritos a seguir:

- **linhas 1, 5, 19 e 22**: são processadas as consultas para identificar a quantidade de *nodes* no pacote. Isso é feito através da verificação do campo *num_dst* do cabeçalho *fixedfields*, que pode conter os valores de 1 a 4;
- **linhas 6 e 10**: verifica-se o campo *last_node* do cabeçalho *fixedfields*, que registra o último *node* válido. Essa consulta não é necessária na ocorrência de apenas um *node* no pacote;
- **linhas 7 e 8; 11 e 12**: identificado o último *node* válido, os valores das suas arestas (*edges*) são atribuídos aos metadados correspondentes. No exemplo com dois *nodes*, *edge0* e *edge1*. Na ocorrência de 4 *nodes*, registram-se os valores de *edge0*, *edge1*, *edge2* e *edge3*;
- **linhas 2 e 3; 14 e 15; 16 e 17**: dependendo do número de *nodes*, os valores referentes a *xid_type* e *id* (chave da tabela *match-action*), de cada *node*, são registrados em metadados.

Os processos do Código-fonte 10 têm como objetivo identificar o *XIDType* e o *ID* do *node* a ser verificado na tabela *match-action*. No Código-fonte 11, utiliza-se como exemplo o final da estrutura condicional referente à opção com quatro *nodes*. Dessa forma, é possível visualizar a atribuição dos valores *xid_type* e *id*, de cada um dos quatro *nodes* que compõem o endereço de destino, para seus metadados correspondentes: *meta.dst_nodeN_xid_type* e *meta.dst_nodeN_id*. Onde *N*, nesta descrição, representa o número do *node* referenciado no Código-fonte 11.

Código-fonte 11 – Utilização de metadados para referenciar *xid_type* e *id*

```
1     ...
2     meta.dst_node1_xid_type = hdr.dst_four_nodes.node1.xid_type;
3     meta.dst_node1_id = hdr.dst_four_nodes.node1.id;
```

```
4       meta.dst_node2_xid_type = hdr.dst_four_nodes.node2.xid_type;
5       meta.dst_node2_id = hdr.dst_four_nodes.node2.id;
6       meta.dst_node3_xid_type = hdr.dst_four_nodes.node3.xid_type;
7       meta.dst_node3_id = hdr.dst_four_nodes.node3.id;
8       meta.dst_node4_xid_type = hdr.dst_four_nodes.node4.xid_type;
9       meta.dst_node4_id = hdr.dst_four_nodes.node4.id;
10      }
```

Identificados os valores dos campos que constituem a chave de consulta da tabela *match-action*, é preciso descobrir a ordem em que os XIDs dos *nodes* serão enviados para verificação nessa tabela de encaminhamento. A sequência é determinada pelas arestas (*edges*) do *node* identificado como *last_node*. Esse processo caracteriza o endereçamento do tipo *fallback* da arquitetura XIA, que oferece alternativas de encaminhamento a partir do seu plano de dados.

Tomando como exemplo, novamente, a opção de cabeçalho com quatro *nodes* e com o campo *last_node* já identificado, conforme procedimento descrito no Código-fonte 10, o *node* assinalado possui quatro arestas como possibilidades de encaminhamento (Figura 19). No Código-fonte 12 é possível observar o processo para a identificação do valor da aresta (*edge*). Nele são exibidos os procedimentos apenas para o campo *edge0*, porém, o método é análogo para todos os demais (*edge1*, *edge2* e *edge3*).

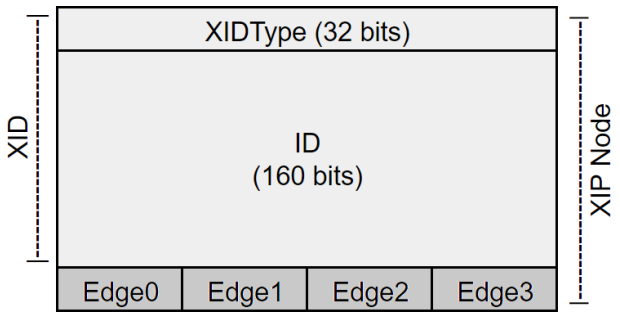
Código-fonte 12 – Identicando a prioridade das arestas (*edges*) do *LastNode*

```
1       meta.edge = meta.dst_edge0;
2
3       if (meta.edge == meta.dst_edge0) {
4           if (meta.dst_edge0 == 1) {
5               meta.dst_xid_type = meta.dst_node1_xid_type;
6               meta.dst_id = meta.dst_node1_id;
7               meta.last_node = 1;
8               xia1.apply();
9           }
10          else if (meta.dst_edge0 == 2) {
11              meta.dst_xid_type = meta.dst_node2_xid_type;
12              meta.dst_id = meta.dst_node2_id;
13              meta.last_node = 2;
14              xia1.apply();
15          }
16          else if (meta.dst_edge0 == 3) {
17              meta.dst_xid_type = meta.dst_node3_xid_type;
18              meta.dst_id = meta.dst_node3_id;
19              meta.last_node = 3;
20              xia1.apply();
21          }
22          else if (meta.dst_edge0 == 4) {
23              meta.dst_xid_type = meta.dst_node4_xid_type;
24              meta.dst_id = meta.dst_node4_id;
25              meta.last_node = 4;
26              xia1.apply();
27          }
28          meta.edge = meta.dst_edge1;
29      }
```

Os procedimentos executados no Código-fonte 12 para identificar o valor de cada *edge* e realizar a consulta do *node* correspondente são descritos a seguir:

- **linha 1**: inicialmente, é criado um metadado referente ao *edge* atual (*meta.edge*). Atribui-se a este, como valor, um outro metadado que representa o *edge0* do *node* de destino selecionado (*meta.dst_edge0*). Isso força a consulta a iniciar na primeira aresta;
- **linha 2**: verifica se o *edge* atual corresponde ao *edge0*;
- **linhas 3, 9, 15 e 21**: Identifica se o valor do *edge0* é igual a 1, 2, 3 ou 4, respectivamente. Sendo que apenas uma das opções verificadas é válida;
- **linhas 4, 10, 16 e 22**: atribui ao metadado *meta.dst_xid_type*, o *xid_type* correspondente ao *node* em que a verificação do valor do *edge* for válida;
- **linhas 5, 11, 17 e 23**: registra no metadado *meta.dst_id*, o *id* referente ao *node* em que a averiguação do *edge* retornar o valor correto;
- **linhas 6, 12, 18 e 24**: atribui ao metadado *meta.last_node* valor equivalente ao do *edge* em que a verificação foi validada;
- **linhas 7, 13, 19 e 25**: aplica os valores armazenados nos metadados à tabela *xia1*, realizando a consulta do *node* apontado pelo *edge* válido. Portanto, os campos referentes a XIDType (*meta.dst_xid_type*), ID (*meta.dst_id*) e LastNode (*meta.last_node*) são utilizados na consulta da tabela *match-action*;
- **linha 27**: atribui o metadado que corresponde ao *edge1*, para verificação da próxima aresta.

Figura 19 – Representação do XIP *Node*



Fonte: Adaptada de BYERS et al. (2014)

O Código-fonte 12 demonstra o processo para a identificação do *node* que corresponde ao *edge0* e o envio deste *node* para consulta na tabela de encaminhamento, buscando uma correspondência válida. Assim, é preciso realizar o mesmo processo nas arestas seguintes. Para isso, é necessário que, ao final da consulta de uma aresta, o valor do metadado que corresponde ao *edge* atual seja alterado para o próximo. Esta ação é realizada na linha 27 do Código-fonte 12, o que permite que o processo se repita, agora analisando o *edge1*. Em seguida, é executado novamente para a próxima aresta, até encontrar um valor correspondente na tabela ou descartar o pacote, ao finalizar a verificação do *edge3* sem uma opção válida.

Essa atribuição de um metadado, referenciando explicitamente a aresta que deve ser verificada, foi necessária para retornar ao código após uma consulta à tabela de encaminhamento, em que não houve correspondência com os campos referentes à chave. Além disso, foi necessária a implementação de uma tabela para cada *edge* a ser consultado, pois o P4 não permite que uma tabela seja verificada mais de uma vez. Portanto, para viabilizar os processos de encaminhamento da XIA, o código possui quatro tabelas idênticas, diferenciadas apenas pelo nome que as identifica (xia1, xia2, xia3 e xia4). Tanto a definição do problema, quanto a descrição da solução estão disponíveis em FINGERHUT (2019). Além disso, todo o fluxo de controle, condições para consulta e implementação de tabelas e ações podem ser verificados no código completo, disponível no Apêndice A deste trabalho.

3.6 Montagem do pacote de saída (*deparser*)

Após todo o processamento realizado no pacote de entrada, os cabeçalhos são enviados ao componente *deparser* para que sejam remontados na ordem correta, definindo, assim, o pacote de saída. Dessa forma, o pacote estará pronto para ser encaminhado como uma sequência de *bits* para o próximo salto da rede. O Código-fonte 13 apresenta o pacote XIP ordenado e montado para encaminhamento.

Código-fonte 13 – Ordem dos cabeçalhos no pacote de saída

```
1  control XIA_Deparser(packet_out packet, in headers_t hdr) {
2      apply {
3          packet.emit(hdr.ethernet);
4          packet.emit(hdr.fixedfields);
5          packet.emit(hdr.dst_one_node);
6          packet.emit(hdr.dst_two_nodes);
7          packet.emit(hdr.dst_three_nodes);
8          packet.emit(hdr.dst_four_nodes);
9          packet.emit(hdr.src_one_node);
10         packet.emit(hdr.src_two_nodes);
11         packet.emit(hdr.src_three_nodes);
12         packet.emit(hdr.src_four_nodes);
13     }
14 }
```

3.7 Considerações sobre a implementação

Este Capítulo apresenta, na Seção 3.1, características da arquitetura XIA que devem ser observadas para a implementação do seu plano de dados. Esses requisitos são apontados sem considerar a tecnologia ou linguagem de programação. Em seguida, na Seção 3.2, delimita-se a implementação a ser realizada utilizando a linguagem P4, definindo o escopo do projeto a partir de metas baseadas nos princípios da XIA e em dados técnicos da arquitetura.

Nas Seções seguintes detalha-se o desenvolvimento do código P4, evidenciando seus processos característicos, de acordo com componentes principais da linguagem. Com isso, na Seção 3.3 definem-se os cabeçalhos dos pacotes, demonstrando a segmentação realizada para representar o cabeçalho referente aos endereços de tamanho variável. Na Seção 3.4, observa-se o analisador de cabeçalhos, onde é possível visualizar a extração e a análise dos cabeçalhos, baseando-se em critérios estabelecidos nos *parsers*.

A Seção 3.5 exibe as ações configuradas para atuarem sobre os pacotes com correspondência válida na tabela de encaminhamento. Essa, por sua vez, apresenta os campos que formam a chave de verificação para a arquitetura XIA. No restante da Seção, responsável pelo controle do fluxo do pacote, destacam-se as estruturas implementadas para identificar a quantidades de *nodes* em um pacote, o último *node* válido e para percorrer suas arestas, encaminhando os *nodes* referenciados para a consulta na tabela *match-action*.

Por fim, a Seção 3.6 demonstra como os pacotes são remontados, após o processamento, para que sejam encaminhados ao próximo salto da rede. Para validar o funcionamento adequado do plano de dados implementado, deve-se organizar um ambiente de testes onde possam ser propostos cenários distintos. Neles, devem-se verificar as possibilidades de endereçamento e alternativas de encaminhamento previstas na arquitetura.

4 TESTES E RESULTADOS

Este Capítulo apresenta os testes e resultados da implementação proposta neste trabalho. Para isso, expõe os objetivos e metodologia para a realização dos testes, estabelece as ferramentas e tecnologias necessárias, descreve a configuração do ambiente de experimentação e demonstra a execução das aplicações utilizadas. Em seguida, detalha o cenário para a realização dos experimentos e realiza os testes de encaminhamento de pacotes. Os resultados obtidos visam validar a implementação do plano de dados da arquitetura XIA usando a linguagem P4.

4.1 Objetivos e Metodologia dos Testes

A realização de testes tem como objetivo demonstrar que o código desenvolvido atende aos requisitos do plano de dados da arquitetura XIA, dentro do escopo proposto neste trabalho. Assim, deve-se verificar as condições relacionadas às alternativas de encaminhamento e o número de *nodes* previstos para endereçamento no cabeçalho do protocolo principal da XIA. Também é preciso atestar a validação das chaves na tabela de encaminhamento e a execução das ações sobre o pacote processado pelo plano de dados do *switch*.

Para alcançar esses objetivos, é preciso selecionar e configurar as ferramentas necessárias, além de executar um ambiente para experimentos que permita a elaboração de cenários de testes distintos. Dessa forma, os cenários propostos devem oferecer possibilidades para a realização de encaminhamento de pacotes, de forma que seja possível viabilizar a utilização de endereços diferentes e alternativas de encaminhamento entre os *nodes* que compõem um mesmo DAG. A realização desses testes têm que verificar se as propriedades da arquitetura XIA foram implementadas corretamente no plano de dados desenvolvido e se as saídas dos pacotes processados correspondem aos resultados esperados.

4.2 Definição de ferramentas e tecnologias utilizadas

Conforme apresentado em Seções anteriores, este trabalho tem a intenção de contribuir com o projeto FIXP. Consequentemente, algumas das tecnologias escolhidas para a implementação estão relacionadas à compatibilidade com as ferramentas utilizadas nessa iniciativa. Enquadram-se nesta condição: a linguagem de programação P4, aplicada no desenvolvimento do plano de dados do dispositivo virtual de rede; e o *switch* em *software* BMv2, que permite a execução de um comutador virtual a partir de um código P4.

Devido às suas características (destacadas na Seção 2.4) e à integração com as ferramentas pré-definidas, utiliza-se também o *P4Runtime*, que disponibiliza uma API de plano de controle para interagir com elementos de plano de dados desenvolvidos em P4. Para a geração de pacotes

XIP, adotou-se a biblioteca *Scapy* (BIONDI, 2020), da linguagem *Python*. A escolha foi baseada na sua utilização no projeto FIXP e por atender adequadamente aos requisitos de geração de cabeçalhos de pacotes para novos protocolos.

No ambiente de execução do trabalho, tanto para o desenvolvimento dos códigos, quanto na realização dos testes, utiliza-se a virtualização como recurso principal. O uso de Máquina Virtual (*Virtual Machine* - VM) torna este cenário mais flexível, escalável e portável. Contudo, observou-se a necessidade de definição em relação à quantidade de VMs para a realização dos testes. Concentrando todas as ferramentas em uma VM ou utilizando várias VMs, uma para cada *host* ou *switch* que porventura fosse necessário para compor o cenário de experimentação. Considerando o escopo do trabalho, em qualquer dos cenários, nenhuma das implementações ou testes necessários seriam comprometidos.

Diante disso, definiu-se a opção por apenas uma VM, devido à simplicidade do ambiente e pela economia de recursos computacionais, além de reduzir possibilidades ou variáveis que pudessem comprometer o andamento do experimento. Portanto, foi instalada uma Máquina Virtual, com 20 GB de armazenamento, 2 GB de memória e sistema operacional Linux Ubuntu 18.04 LTS. A escolha do sistema operacional foi definida a partir dos requisitos sugeridos em documentação para a utilização do P4 (RIJSMAN, 2019).

Como o experimento utiliza apenas uma VM para a realização dos testes de encaminhamento de pacotes, torna-se necessário decidir a forma de conexão do dispositivo virtual implementado. Optou-se por dispositivos *Ethernet* Virtuais (*veth*), que são interfaces lógicas suportadas pelo *kernel* do Linux. Elas são criadas sempre em pares e conectadas por um enlace, de forma análoga a um cabo de conexão de rede. As interfaces virtuais permitem que pacotes transmitidos por um dispositivo ligado a um *veth* sejam imediatamente encaminhados para outro dispositivo conectado a este par. Isso torna os pares *veth* ideais para conectar diferentes componentes de rede virtual (DONATO, 2017). Para capturar os pacotes XIP nas interfaces virtuais (*veth*), utiliza-se o analisador de rede *tcpdump*.

4.3 Ambiente de experimentação

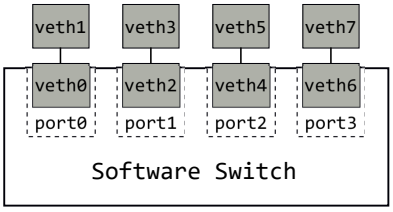
O ambiente de experimentação é composto pela máquina virtual e as aplicações descritas na Seção 4.2. Com todos os aplicativos instalados, inicia-se a configuração desse ambiente de testes com a criação das interfaces virtuais *ethernet* (*veth*). No escopo deste trabalho, as *veths* serão conectadas ao *switch* implementado em *software* BMv2, para a realização do encaminhamento dos pacotes XIP através do dispositivo. As *veths* são criadas a partir dos seguintes comandos:

```
$ ip link add name veth0 type veth peer name veth1
$ ip link set dev veth0 up
$ ip link set dev veth1 up
```

As linhas de comando destacadas configuram o par *veth0 – veth1*, além de habilitar cada uma das interfaces criadas. Outros ajustes podem ser realizados, como definir um valor para o *Maximum Transmission Unit* (MTU) ou desabilitar o suporte a IPv6. Para este trabalho, utilizando os mesmos comandos apresentados, foram habilitados mais três pares de interfaces virtuais (*veth2 – veth3*; *veth4 – veth5*; e *veth6 – veth7*). Todos os pares são conectados ao BMv2, disponibilizando quatro interfaces ligadas ao *switch*.

No cenário proposto para o experimento, mesmo com a utilização de *veths* conectadas em pares, o encaminhamento só é possível quando há rota definida na tabela *match-action* do plano de dados implementado em P4. Caso a rota não exista ou o comutador esteja inativo, o pacote não poderá ser enviado. A Figura 20 representa o *software switch* conectado às *veths*.

Figura 20 – *Software Switch* conectado às interfaces virtuais *ethernet* (*veth*)



Fonte: O Autor (2020)

Para iniciar o *switch* BMv2, além de apontar as interfaces de rede as quais será conectado, é preciso identificar o arquivo *.json*, que possui as regras de plano de dados que serão adicionadas no comutador virtual. O arquivo é gerado, juntamente com seu o diretório, a partir da compilação do código P4. Portanto, deve-se acessar seu diretório (ou informar o caminho absoluto) e executar a instrução referente ao *simple_switch*, como no exemplo a seguir:

```
$ simple_switch --interface 0@veth0 --interface 1@veth2 --interface 2@veth4
    --interface 3@veth6 xia.json
```

O comando executado associa os pares de interfaces virtuais (*veths*) às portas do *switch* BMv2 (portas 0-3), conforme exemplo da Figura 20. Além disso, insere o plano de dados da arquitetura XIA no *switch*, através do arquivo *xia.json*. A Figura 21 mostra o BMv2 sendo iniciado, onde é possível observar as portas habilitadas, assim como a indicação do arquivo *xia.json*, a partir do seu caminho absoluto.

Figura 21 – *Switch* BMv2 em execução

```
jns@p4-16:~/ufpe/xia$ sudo simple_switch --interface 0@veth0 --interface 1@veth2
 --interface 2@veth4 --interface 3@veth6 /home/jns/ufpe/xia/xia.bmv2/xia.json
Calling target program-options parser
Adding interface veth0 as port 0
Adding interface veth2 as port 1
Adding interface veth4 as port 2
Adding interface veth6 as port 3
```

Fonte: Capturada pelo Autor (2020)

Conforme apresentado na Seção 2.4, a linguagem P4 não oferece a interface entre o plano de dados e o plano de controle. Além disso, o escopo deste trabalho não prevê a implementação de um plano de controle. Dessa forma, o preenchimento da tabela de encaminhamento do plano de dados XIA foi realizado diretamente (manualmente), utilizando o *P4Runtime*, que possui uma API para controle de planos de dados desenvolvidos em P4. Mais detalhes estão disponíveis na Seção 2.4 desta dissertação.

Na Figura 22, visualiza-se a inicialização do *P4Runtime*, assim como as quatro tabelas implementadas no programa *xia.p4*, exibidas a partir da execução do comando "*show_tables*". Na imagem, observa-se também os campos que representam a chave da tabela de encaminhamento do plano de dados XIA implementado: *xid_type* (32 *bits*) e *id* (160 *bits*).

Figura 22 – P4Runtime em execução e exibindo as tabelas *match-action* da XIA

```
jns@p4-16:~/ufpe/xia$ simple_switch_CLI
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: show_tables
XIA_Ingress.xia1              [implementation=None, mk=scalars.metadata_t.dst_x
id_type(exact, 32),     scalars.metadata_t.dst_id(exact, 160)]
XIA_Ingress.xia2              [implementation=None, mk=scalars.metadata_t.dst_x
id_type(exact, 32),     scalars.metadata_t.dst_id(exact, 160)]
XIA_Ingress.xia3              [implementation=None, mk=scalars.metadata_t.dst_x
id_type(exact, 32),     scalars.metadata_t.dst_id(exact, 160)]
XIA_Ingress.xia4              [implementation=None, mk=scalars.metadata_t.dst_x
id_type(exact, 32),     scalars.metadata_t.dst_id(exact, 160)]
```

Fonte: Capturada pelo Autor (2020)

Com o *switch* em funcionamento, deve-se executar a aplicação que realiza a captura de pacotes nas interfaces de saída do comutador. Essa função é realizada pelo *tcpdump*, que captura os pacotes nas *veths* conectadas ao *switch* BMv2, em caso de correspondência com os campos chave na tabela de encaminhamento. A Figura 23 exibe três telas, que mostram a execução do *tcpdump* verificando as interfaces virtuais: *veth3*, *veth5* e *veth7*.

Figura 23 – Execução do *tcpdump* para captura de pacotes nas *veths*

```
jns@p4-16:~/ufpe/xia$ sudo tcpdump -n -i veth3
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth3, link-type EN10MB (Ethernet), capture size 262144 bytes
```

```
jns@p4-16:~/ufpe/xia$ sudo tcpdump -n -i veth5
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth5, link-type EN10MB (Ethernet), capture size 262144 bytes
```

```
jns@p4-16:~/ufpe/xia$ sudo tcpdump -n -i veth7
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth7, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Fonte: Capturada pelo Autor (2020)

Para realizar os testes de encaminhamento no ambiente de experimentação, é necessário implementar o cabeçalho do pacote XIP. Para isso, utilizou-se a biblioteca *Scapy*, da linguagem *Python*, através da qual os pacotes podem ser gerados e enviados. A Figura 24 mostra parte do cabeçalho do pacote XIP implementado, exibido a partir de uma consulta realizada com o *Scapy*. Neste trabalho, foi implementada uma versão de cabeçalho para cada quantidade de *nodes* XIP que compõem o endereço de destino (1-4) do pacote. Assim, o arquivo correspondente é referenciado no momento do envio do pacote pelo *Scapy*. A versão do cabeçalho XIP implementado com quatro *nodes* no endereço de destino está disponível no Apêndice B.

Figura 24 – Visualização do pacote XIP gerado com *Scapy*

```
>>> ls (XIP)
version    : XByteField                          = (1)
next_hdr   : XByteField                          = (0)
payload_len : XShortField                         = (0)
hop_limit  : XByteField                          = (0)
num_dst    : XByteField                          = (0)
num_src    : XByteField                          = (0)
last_node  : XByteField                          = (0)
dst_n1_xid_type : XBitField (32 bits)                 = (0)
dst_n1_id  : XBitField (160 bits)                = (0)
dst_n1_edge0 : XByteField                          = (0)
dst_n1_edge1 : XByteField                          = (0)
dst_n1_edge2 : XByteField                          = (0)
dst_n1_edge3 : XByteField                          = (0)
```

Fonte: Capturada pelo Autor (2020)

4.4 Cenário para a realização dos testes

A arquitetura XIA possui quatro tipos de endereçamento para realizar o encaminhamento de pacotes: básico, escopo, *fallback* e refinamento iterativo, conforme apresentado na Subseção 2.2.5. Sendo que os dois últimos citados permitem a execução dos principais processos para o encaminhamento de pacotes da XIA. Diante disso, propõe-se um cenário de testes que utilize uma representação baseada no tipo refinamento iterativo (que contempla o *fallback*), em que seja possível demonstrar as características da XIA em relação às possibilidades e flexibilidade do seu endereçamento.

Para a realização dos testes, é preciso executar alguns procedimentos manuais que permitirão a representação de um ambiente com um número variável de *switches*. Como o experimento é processado em apenas uma VM, utilizando interfaces *veth*, é necessário configurar o *switch* virtual implementado a cada salto, para validar esta representação. Assim, a tabela de encaminhamento é alterada de acordo com o comutador que se deseja representar. Em relação aos pacotes, eles são encaminhados manualmente utilizando o *Scapy*. São enviados para a porta 0 (*veth0–veth1*) do comutador virtual e podem ser capturados em uma das portas de saída (portas 1-3), pelo *tcpdump*. Assim, os pacotes capturados são encaminhados para o próximo salto.

A definição da porta 0 do *switch* para o recebimento do pacote de entrada é apenas para fins de representação, com o objetivo de melhorar o entendimento do cenário proposto. Não há qualquer restrição em relação à utilização das portas do *switch*, que são acionadas a partir indicação na tabela de encaminhamento. Assim, a execução de encaminhamento do comutador é comum ao de todo dispositivo deste tipo, sendo o seu diferencial definido pelo processamento do plano de dados implementado.

Para a realização dos testes, propõe-se a elaboração de endereços (DAGs) distintos. Assim, será possível acompanhar a evolução do DAG, bem como do pacote a cada salto na rede, representados por uma topologia utilizada como exemplo. Além disso, deve-se destacar o pacote XIP de entrada (gerado pelo *Scapy*), a chave utilizada para correspondência na tabela de encaminhamento e a porta por onde os pacotes da saída são direcionados.

Havendo correspondência com a chave da tabela *match-action*, o pacote é encaminhado para a porta de saída indicada, sendo capturado pelo *tcpdump*. A partir da exibição do pacote de saída, é possível identificar qual o XID (CID, HID e SID) validado na tabela, assim como os demais campos do cabeçalho XIP. A análise do pacote capturado também permite atestar que ele foi enviado através da porta indicada na tabela de encaminhamento, além de possibilitar a confirmação das alterações previstas nos campos do cabeçalho XIP implementado.

Apesar do experimento ser executado apenas em uma VM, utilizando a representação proposta nesta Seção para a realização dos testes, o ambiente permite montar cenários distintos para o encaminhamento dos pacotes. É possível utilizar alternativas diferentes, baseadas no DAG, obtendo resultados (ou caminhos) diferenciados a partir de um mesmo endereço. Assim, os cenário propostos para os testes permitem verificar os tipos de endereçamento e processos de encaminhamento da arquitetura XIA.

4.5 Testes da Implementação

Os testes realizados buscam demonstrar que o plano de dados desenvolvido em P4 é capaz de promover a flexibilidade de endereçamento, as possibilidades de encaminhamento e o suporte a endereços de tamanho variável, previstos na arquitetura XIA. Para a realização do experimento, utiliza-se o modelo de endereçamento "refinamento iterativo", que é a união do "*fallback*" com o tipo "escopo", permitindo a verificação de diferentes possibilidades de encaminhamento. Serão utilizados também os XIDs padrão da arquitetura XIA: AD (NID), HID, SID e CID, referentes aos *principals* de domínio autônomo (ou rede), *host*, serviço e conteúdo, respectivamente. Os XIDs são formados por *XIDType*, que correspondem a valores constantes da XIA (Tabela 2), e ID, definido por um *hash* de 160 *bits*.

Conforme relatado na Seção 3.2, o suporte a elementos de segurança, como *hashes*, não está no escopo da implementação deste trabalho. Portanto, para criar o identificador (ID) criptografado, utilizou-se uma ferramenta de geração manual de *hashes*, onde através do algoritmo SHA1 cria-se um *hash* de 160 *bits* a partir de um texto informado. Os IDs (*hashes*)

gerados compõem os XIDs utilizados nos testes, juntamente com os *XIDType* definidos pela arquitetura XIA. Por sua vez, os XIDs representam os *nodes* no cabeçalho XIP, e estes formam os endereços de destino e origem do pacote. Diante disso, a Tabela 3 apresenta os XIDs utilizados na realização dos testes deste trabalho.

Tabela 3 – XIDs utilizados nos testes de encaminhamento

XID	***XIDType***	***ID***
AD1	0x10	0x68eb3add703333d71b056096c6ae00cebc12b12b
AD2	0x10	0xeb19ad2c3bee46c208d6f249461da33609ad9632
HID	0x11	0xd72a10b9599e22a51d314b698edf88370daf825c
SID	0x13	0xb836a34a125e1c1d94048c50c5c8e9ae537b78d8
CID	0x12	0x832c1a31e54a836da96add18cf80f89adb51a94b

Fonte: O Autor (2020)

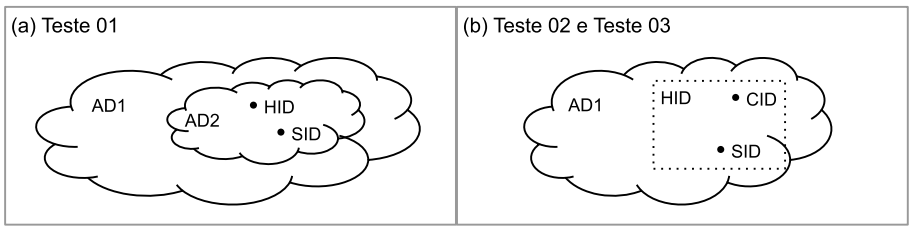
A partir das informações relativas às estruturas, componentes e configurações realizadas para promover o experimento, as Subseções a seguir apresentam os testes executados para demonstrar as principais possibilidades de encaminhamento da solução implementada. Os testes utilizam o tipo de endereçamento "refinamento iterativo", assim, a partir da origem, não é possível acessar o destino diretamente. O pacote inicialmente é encaminhado para um *switch* com correspondência válida para um *node* do DAG globalmente conhecido, por exemplo. Portanto, para os testes realizados neste trabalho, considera-se que o primeiro *switch* a receber o pacote refere-se à validação para AD1, sendo este definido como *last_node*. Com isso, os saltos seguintes são analisados, detalhando-se cada uma das iterações até o destino.

Os objetivos e a descrição dos testes realizados são listados as seguir:

- **Teste 01**: Consiste na utilização do DAG (endereço) AD1 > AD2 > HID > SID, onde os XIDs representam domínios (AD1 e AD2), *host* (HID) e serviço (SID). No teste, busca-se um serviço (SID) que não é globalmente conhecido e está associado ao *host* (HID), podendo ser alcançado no domínio 1 (AD1), através deste. Além disso, como o serviço possui identificação divulgada na rede, pode ser alcançado diretamente a partir de AD2, como alternativa de encaminhamento usando *fallback*. Assim, o objetivo deste teste é analisar o processamento de pacotes utilizando XIDs do mesmo tipo, diferenciados pelo seu ID, que representam domínios com escopos diferentes. A Figura 25(a) apresenta um exemplo da disposição dos *principals* na rede proposta;
- **Teste 02**: Utiliza o endereço AD1 > HID > SID > CID, onde os XIDs representam domínio (ou rede), *host*, serviço e conteúdo, respectivamente. Em um contexto diferente do teste anterior, neste teste busca-se um conteúdo (CID), que possui seu identificador divulgado no domínio (AD1), podendo ser alcançado alternativamente a partir dos demais *nodes* que compõem o DAG. Neste cenário, o identificador do conteúdo é conhecido no escopo do domínio 1 (AD1), assim como os demais *nodes* que compõem o endereço. Um exemplo da disposição dos *principals* na rede proposta está disponível na Figura 25(b);

- **Teste 03**: Possui o mesmo DAG do teste anterior (Teste 02), contudo, utiliza alternativas diferentes de encaminhamento para chegar ao conteúdo. O objetivo é demonstrar a flexibilidade do endereçamento característica da arquitetura XIA, utilizando o mesmo DAG para destacar possibilidades distintas de chegar ao destino;
- **Teste 04**: Tem como objetivo evidenciar que o plano de dados implementado é capaz de processar pacotes com endereços de tamanhos diferentes. Dessa forma, exibe os pacotes de entrada e saída referentes a três, dois e um *node*, respectivamente, no endereço de destino.

Figura 25 – Exemplo da disposição de domínios e demais XIDs na rede proposta



Fonte: O Autor (2020)

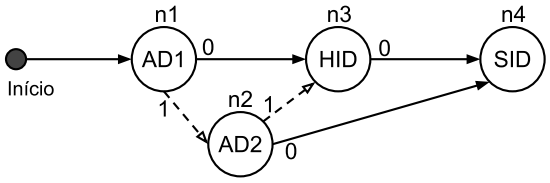
Os testes descritos são detalhados nas Subseções seguintes. Nelas são exibidas as representações dos endereços (DAGs) utilizados, assim como os rótulos inseridos para uma melhor identificação dos *nodes*, de suas arestas e do processo de encaminhamento dos pacotes utilizando esses endereços. Além disso, evidencia o pacote XIP utilizado nos testes, apontando os campos que o constituem.

Para detalhar os processos de encaminhamento nessas Subseções, são apresentadas figuras que representam cada um dos saltos do pacote na topologia utilizada como exemplo, além da evolução do DAG referente ao endereçamento. Elas também possuem imagens com representações e capturas de tela que destacam: os pacotes de entrada e saída do *switch*; a indicação da porta utilizada por ele para o envio do pacote; e sua tabela de encaminhamento, exibindo a chave com correspondência válida.

4.5.1 Teste 01 — DAG: AD1 > AD2 > HID > SID

O objetivo deste primeiro teste é verificar o processo de encaminhamento de pacotes utilizando XIDs do mesmo tipo (AD), diferenciados pelo seu identificador, que podem representar domínios com diferentes escopos. Para isso, busca-se um serviço (SID) utilizando o DAG (endereço) AD1 > AD2 > HID > SID. Ele é composto por dois domínios autônomos (AD1 e AD2) e os *principals* que correspondem a *host* e serviço. O DAG é retratado na Figura 26 e contém os XIDs (*nodes*) que constituem o endereço de destino (n1, n2, n3 e n4), referenciados através dos *nodes* adicionados no cabeçalho do pacote XIP. Além disso, os arcos direcionados indicam suas arestas, que são numeradas (0, 1, 2 e 3) de acordo com a posição do *edge* indicado no *node* do pacote do principal protocolo da XIA.

Figura 26 – DAG para o Teste 01



Fonte: O Autor (2020)

A partir do DAG da Figura 26, que apresenta os XIDs que constituem o endereço de destino do cabeçalho do pacote XIP, a Tabela 4 apresenta os *nodes* que formam esse endereço. Para cada *node* na tabela, são exibidos: o *XIDType* correspondente, o identificador (ID) e suas arestas (*edges*), que referenciam outros *nodes* do endereço como forma de promover as alternativas de encaminhamento. A Tabela 4 também mostra um *node* referente ao endereço de origem, que indica um *host*, representado na Figura 26 como o início do DAG. Conforme descrito na Subseção 2.2.6, no envio do pacote com os dados, a origem pode ser alcançada a partir do endereço do destino, fazendo o caminho reverso do DAG.

Tabela 4 – Teste 01: *Nodes* do pacote XIP utilizado

Node	***XIDType***	***ID***	***Edge0***	***Edge1***	***Edge2***	***Edge3***
dst_n1	0x10	0x68eb3add703333d71b056096c6ae00cebc12b12b	3	2	1	0
dst_n2	0x10	0xeb19ad2c3bee46c208d6f249461da33609ad9632	4	3	2	0
dst_n3	0x11	0xd72a10b9599e22a51d314b698edf88370daf825c	4	3	0	0
dst_n4	0x13	0xb836a34a125e1c1d94048c50c5c8e9ae537b78d8	4	0	0	0
src_n1	0x11	0xe86953d2ac741f6a0f9dd6023563c0b6d604dfb9	0	0	0	0

Fonte: O Autor (2020)

A Tabela 5 apresenta a evolução do pacote e do DAG durante o seu processamento. Nela, observa-se que, apesar da realização do salto na rede (1 e 2), não há alteração referente à indicação do *node* atual no endereço (DAG), demonstrando que um *node* representado no DAG pode corresponder a várias saltos do pacote através da rede. Assim, destaca a dissociação entre a evolução do DAG e o encaminhamento dos pacotes entre os *switches*. Os detalhes dos processos de encaminhamento realizados no Teste 01 estão disponíveis nas figuras a seguir (Figura 27, Figura 28, Figura 29 e Figura 30).

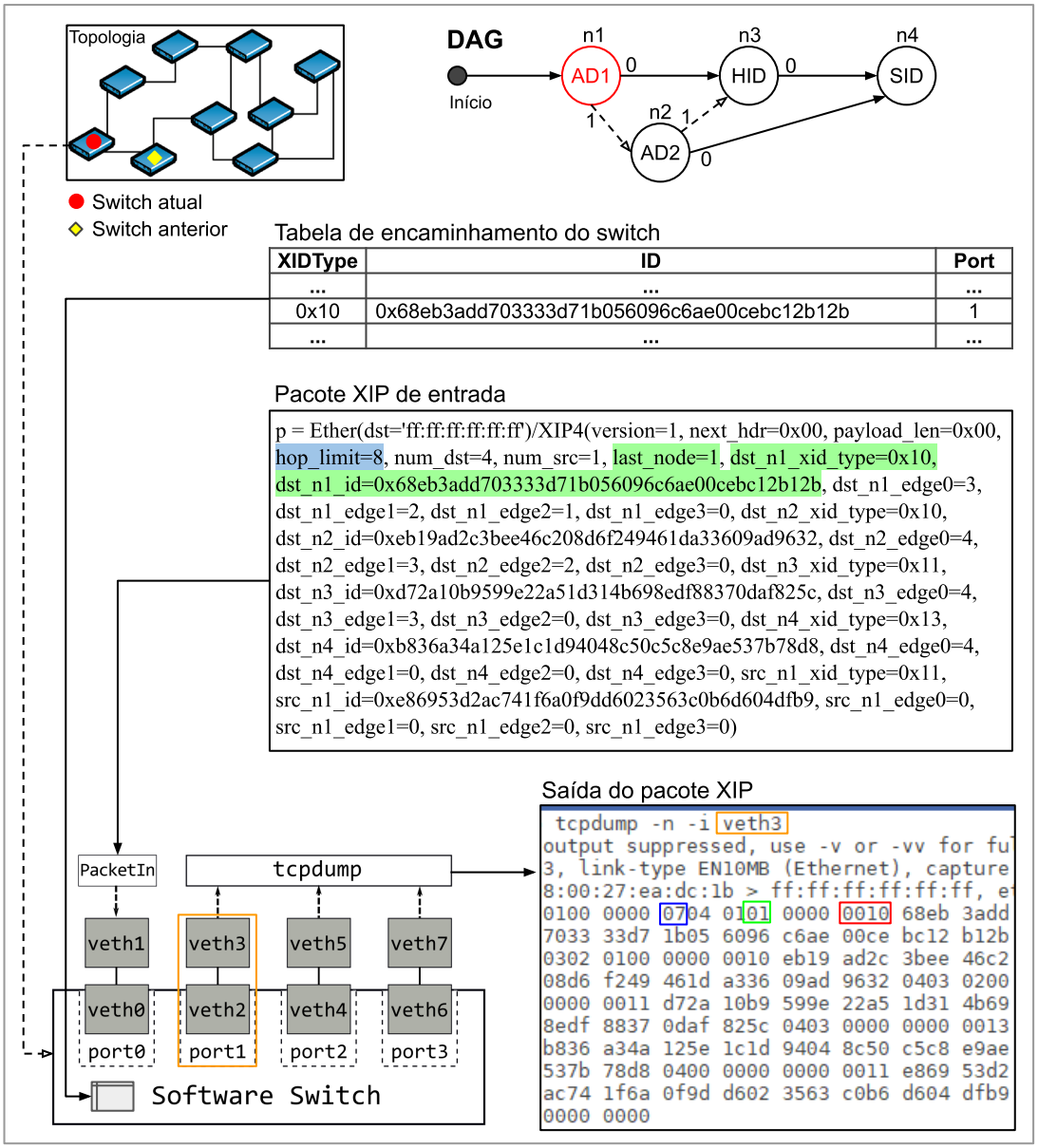
Tabela 5 – Teste 01: Evolução do pacote XIP e do DAG

Salto	***Node* atual do DAG**	***LastNode***	***HopLimit***
1	AD1	1	7
2	AD1	1	6
3	AD2	2	5
4	SID	4	4

Fonte: O Autor (2020)

Na primeira iteração analisada do Teste 01 (Figura 27), sendo "n1" (AD1) o *last_node*, envia-se o *node* referente ao seu *edge0* para consulta dos campos-chave na tabela do *switch*. Como não há validação para HID, o *node* do *edge1* é consultado na tabela, mas sem correspondência para AD2. Assim, verifica-se *xid_type* e *id* do AD1, que correspondem a uma chave registrada, mantendo-o como *last_node* e realizando o encaminhamento pela porta indicada na tabela.

Figura 27 – Teste 01: primeira iteração analisada



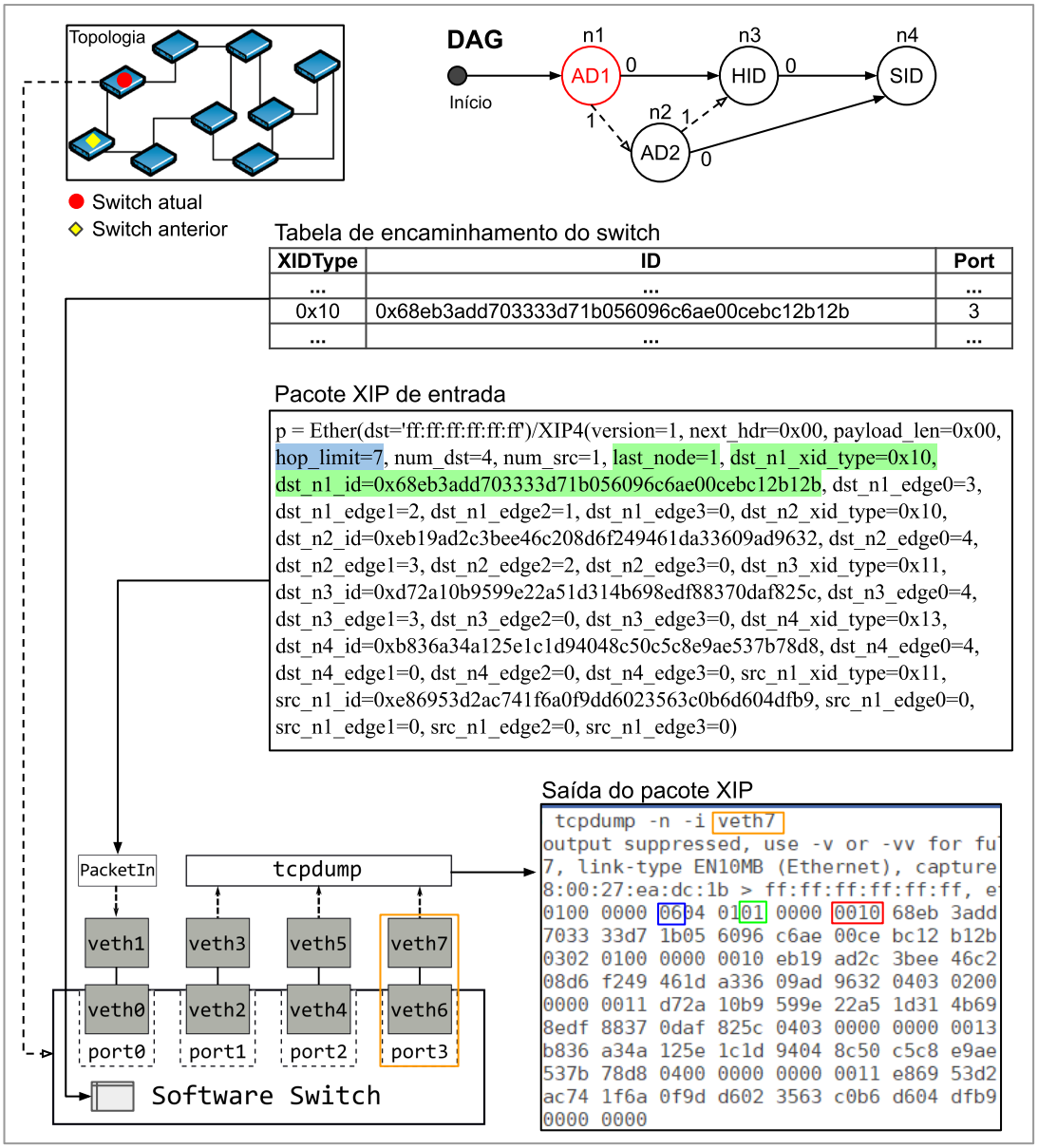
Fonte: O Autor (2020)

Com a realização do encaminhamento do pacote, observa-se na Figura 27:

- O valor do *hop_limit* foi decrementado (de 8 para 7) — destaque em azul;
- A chave identificada na tabela refere-se ao *node1* (AD1) — destaque em vermelho;
- O campo *last_node* permanece com valor 1 (*node1*) — destaque em verde;
- O encaminhamento foi realizado pela porta1 (*veth2–veth3*) — destaque em laranja.

Na segunda iteração analisada do Teste 01 (Figura 28), sendo "n1" (AD1) o *last_node*, envia-se o *node* referente ao seu *edge0* para consulta dos campos-chave na tabela do *switch*. Como não há validação para HID, o *node* do *edge1* é consultado na tabela, mas sem correspondência para AD2. Assim, verifica-se *xid_type* e *id* do AD1, que correspondem a uma chave registrada, mantendo-o como *last_node* e realizando o encaminhamento pela porta indicada na tabela.

Figura 28 – Teste 01: segunda iteração analisada



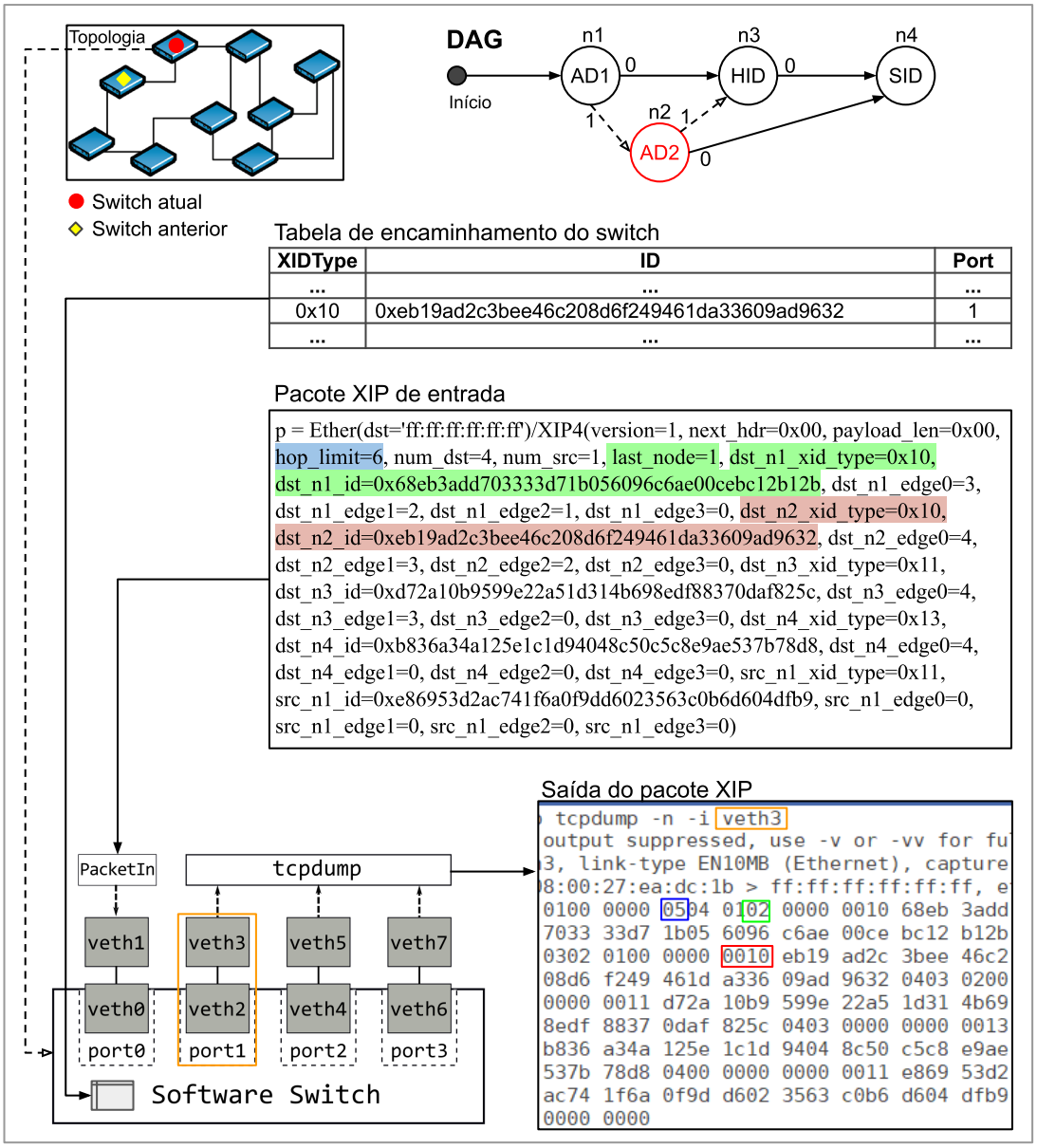
Fonte: O Autor (2020)

Com a realização do encaminhamento do pacote, observa-se na Figura 28:

- O valor do *hop_limit* foi decrementado (de 7 para 6) — destaque em azul;
- A chave identificada na tabela refere-se ao *node1* (AD1) — destaque em vermelho;
- O campo *last_node* permanece com valor 1 (*node1*) — destaque em verde;
- O encaminhamento foi realizado pela porta3 (*veth6–veth7*) — destaque em laranja.

Na terceira iteração analisada do Teste 01 (Figura 29), sendo "n1" (AD1) o *last_node*, envia-se o *node* referente ao seu *edge0* para consulta dos campos-chave na tabela do *switch*. Como não há validação para HID, o *node* relativo ao *edge1* é consultado. Assim, verifica-se *xid_type* e *id* do AD2, que correspondem a uma chave registrada, definindo este como *last_node* e realizando o encaminhamento pela porta indicada na tabela.

Figura 29 – Teste 01: terceira iteração analisada



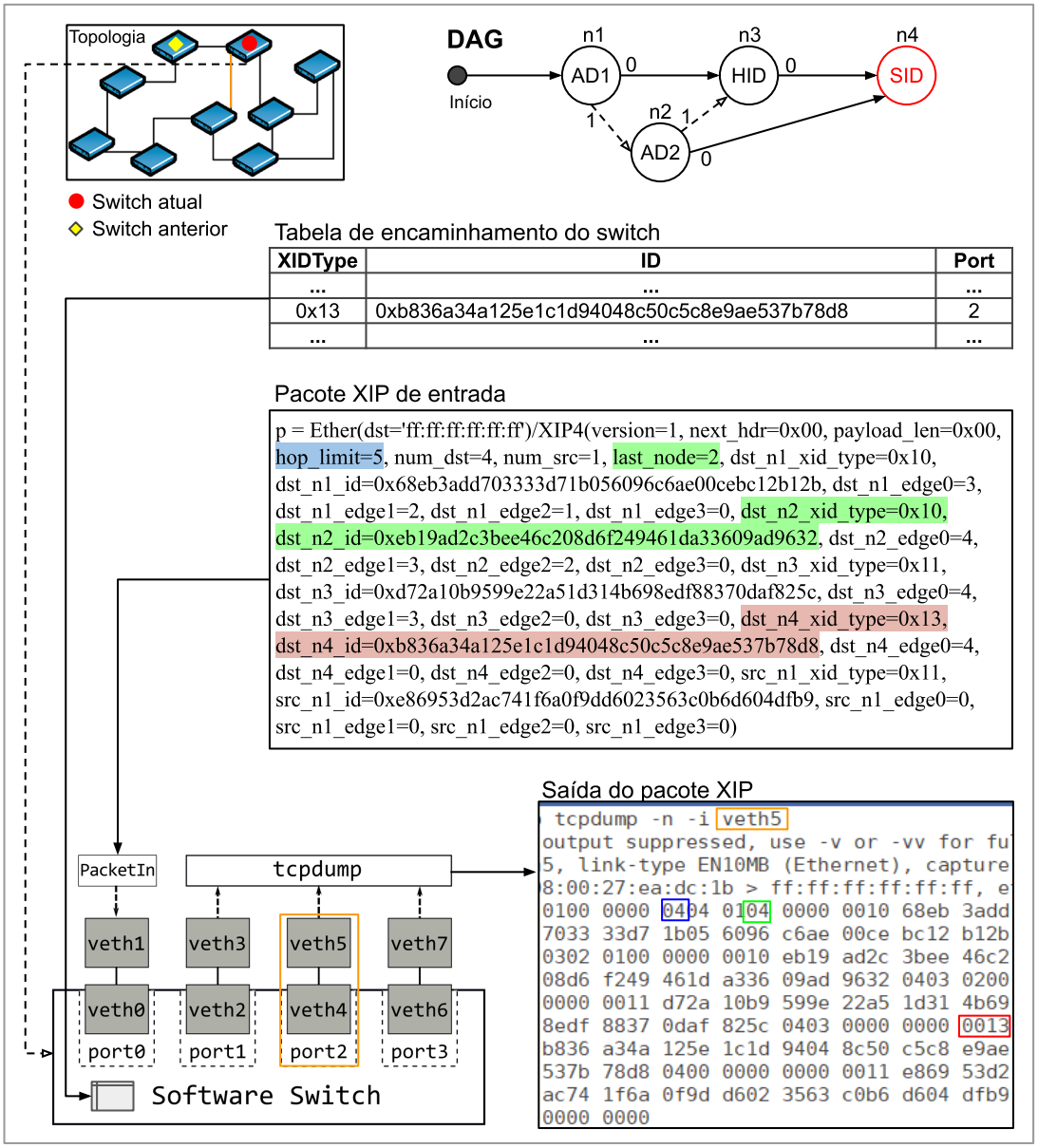
Fonte: O Autor (2020)

Com a realização do encaminhamento do pacote, observa-se na Figura 29:

- O valor do *hop_limit* foi decrementado (de 6 para 5) — destaque em azul;
- A chave identificada na tabela refere-se ao *node2* (AD2) — destaque em vermelho;
- O campo *last_node* foi alterado de 1 (*node1*) para 2 (*node2*)— destaque em verde;
- O encaminhamento foi realizado pela porta1 (*veth2–veth3*) — destaque em laranja.

Na quarta iteração analisada do Teste 01 (Figura 30), sendo "n2" (AD2) o *last_node*, envia-se o *node* referente ao seu *edge0* para consulta dos campos-chave na tabela de encaminhamento do *switch*. Como há correspondência dos campos *xid_type* e *id* do SID, define-se este como *last_node* e realiza-se o encaminhamento pela porta indicada na tabela.

Figura 30 – Teste 01: quarta iteração analisada



Fonte: O Autor (2020)

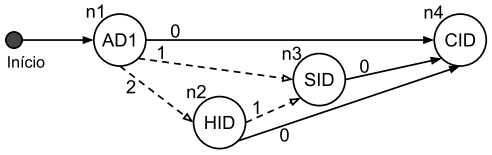
Com a realização do encaminhamento do pacote, observa-se na Figura 30:

- O valor do *hop_limit* foi decrementado (de 5 para 4) — destaque em azul;
- A chave identificada na tabela refere-se ao *node4* (SID) — destaque em vermelho;
- O campo *last_node* foi alterado de 2 (*node2*) para 4 (*node4*)— destaque em verde;
- O encaminhamento foi realizado pela porta2 (*veth4–veth5*) — destaque em laranja.

4.5.2 Teste 02 — DAG: AD1 > HID > SID > CID (cenário 1)

No segundo teste, busca-se recuperar um conteúdo (CID) utilizando o DAG composto por um domínio autônomo (AD1) e os *principals* que correspondem a *host*, serviço e conteúdo. O DAG é retratado na Figura 31 e contém os XIDs (*nodes*) que constituem o endereço de destino (n1, n2, n3 e n4). Além disso, os arcos direcionados indicam suas arestas, que são numeradas (0, 1, 2 e 3) de acordo com a posição do *edge* indicado no *node* do pacote do protocolo XIP.

Figura 31 – DAG para Teste 02 e Teste 03



Fonte: O Autor (2020)

A partir do DAG da Figura 31, que apresenta os XIDs que constituem o endereço de destino do cabeçalho do pacote XIP, a Tabela 6 apresenta os *nodes* que formam esse endereço. Para cada *node* na tabela, são exibidos: o *XIDType* correspondente, o identificador (ID) e suas arestas (*edges*), que referenciam outros *nodes* do endereço como forma de promover as alternativas de encaminhamento definidas nos princípios fundamentais da arquitetura XIA.

Tabela 6 – Teste 02: *Nodes* do pacote XIP utilizado

Node	***XIDType***	***ID***	***Edge0***	***Edge1***	***Edge2***	***Edge3***
dst_n1	0x10	0x68eb3add703333d71b056096c6ae00cebc12b12b	4	3	2	1
dst_n2	0x11	0xd72a10b9599e22a51d314b698edf88370daf825c	4	3	2	0
dst_n3	0x13	0xb836a34a125e1c1d94048c50c5c8e9ae537b78d8	4	3	0	0
dst_n4	0x12	0x832c1a31e54a836da96add18cf80f89adb51a94b	4	0	0	0
src_n1	0x11	0xe86953d2ac741f6a0f9dd6023563c0b6d604dfb9	0	0	0	0

Fonte: O Autor (2020)

A Tabela 7 apresenta a evolução do pacote e do DAG durante o seu processamento. Nela, observa-se a realização dos saltos na rede, o registro da evolução do DAG e o caminho percorrido pelo pacote até o *node* coletor (de destino). Os detalhes dos processos de encaminhamento realizados no Teste 02 estão disponíveis nas figuras a seguir (Figura 32, Figura 33 e Figura 34).

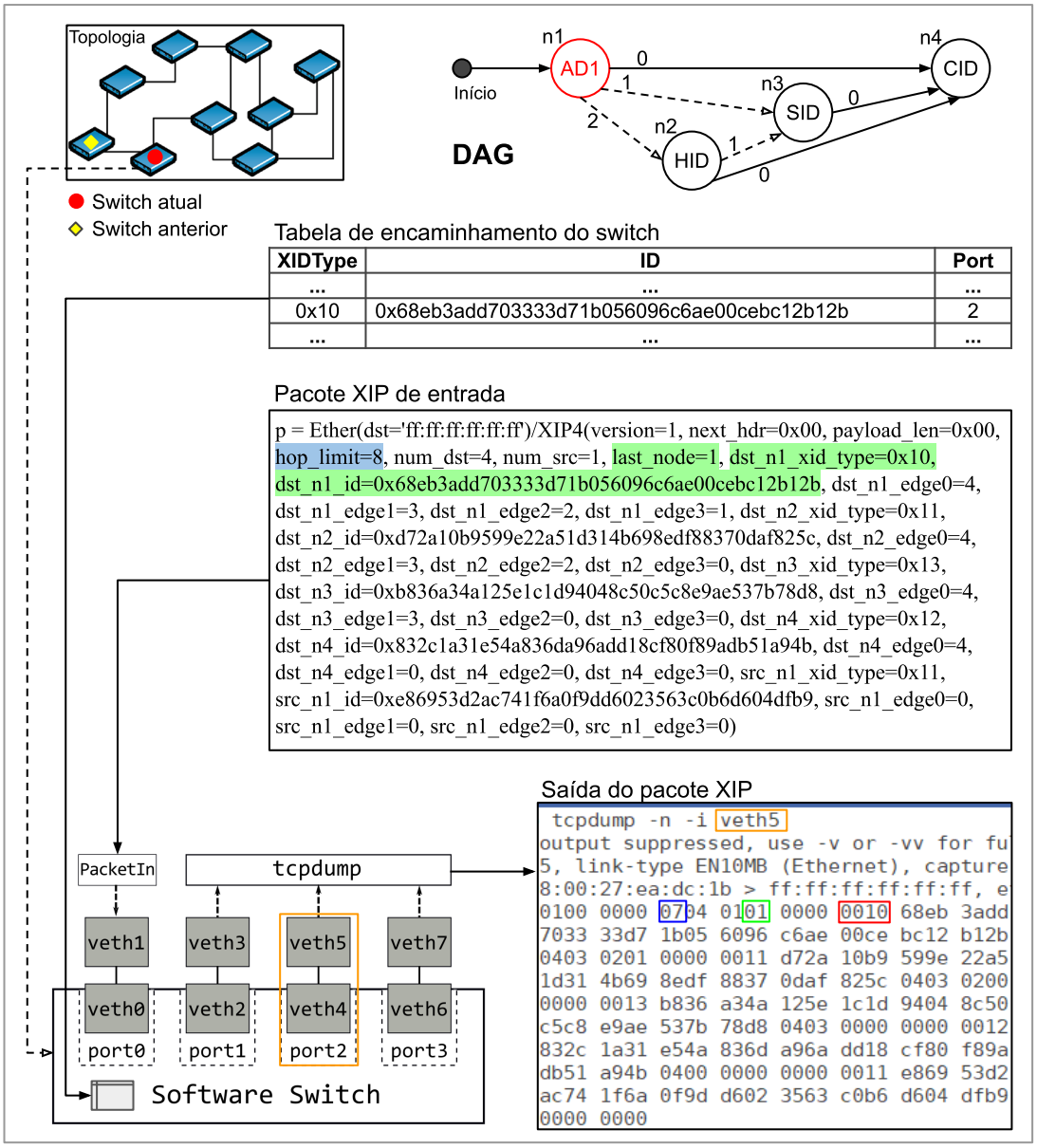
Tabela 7 – Teste 02: Evolução do pacote XIP e do DAG

Salto	***Node* atual do DAG**	***LastNode***	***HopLimit***
1	AD1	1	7
2	SID	3	6
3	CID	4	5

Fonte: O Autor (2020)

Na primeira iteração analisada do Teste 02 (Figura 32), sendo “n1” (AD1) o *last_node*, envia-se o *node* referente ao seu *edge0* para consulta na tabela do *switch*. Como não há validação para CID, o *node* de *edge1* e, em seguida, *edge2* são consultados, mas sem correspondência para SID ou HID. Assim, verifica-se *xid_type* e *id* do AD1, que correspondem a uma chave registrada, mantendo-o como *last_node* e realizando o encaminhamento pela porta indicada na tabela.

Figura 32 – Teste 02: primeira iteração analisada



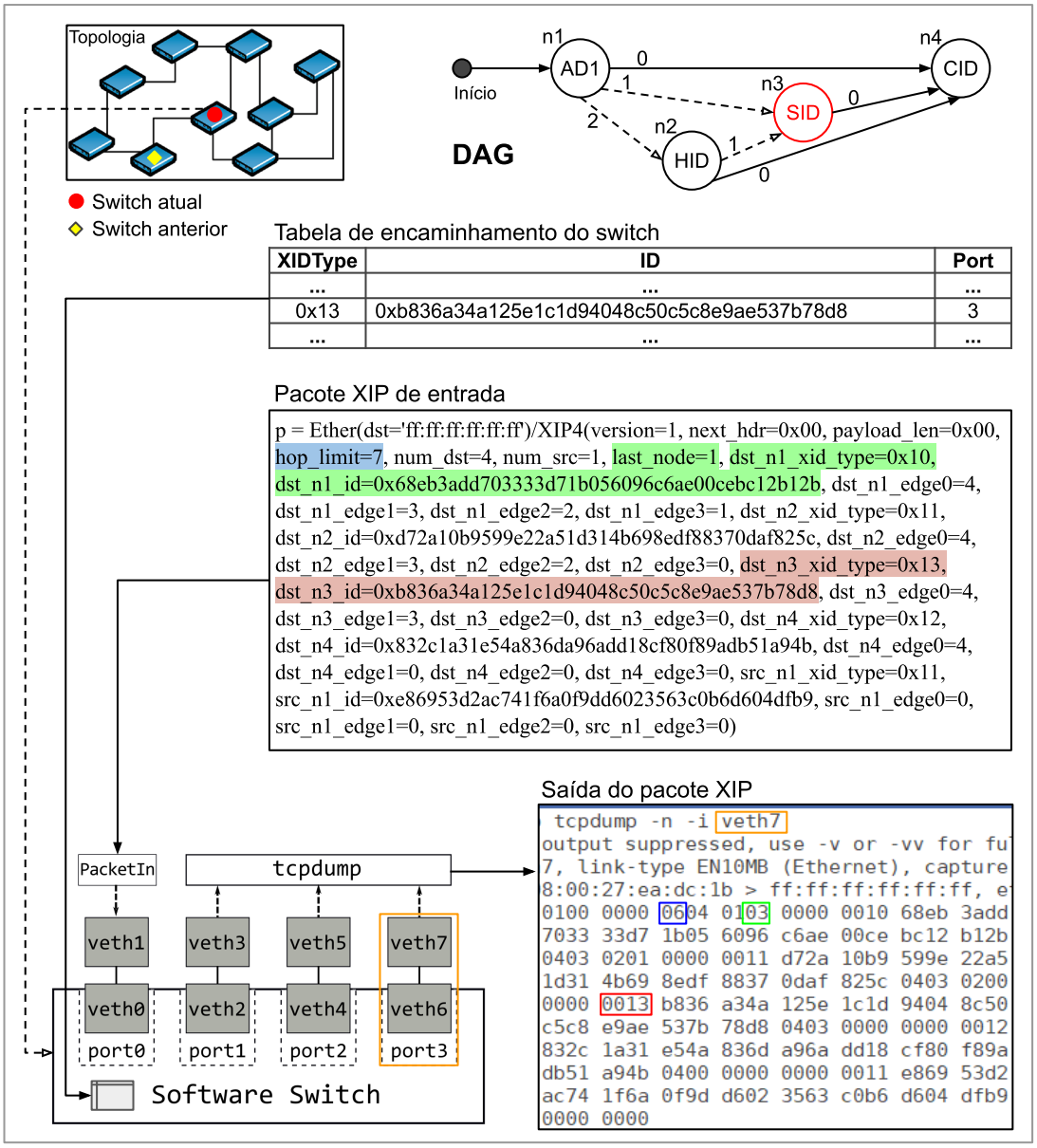
Fonte: O Autor (2020)

Com a realização do encaminhamento do pacote, observa-se na Figura 32:

- O valor do *hop_limit* foi decrementado (de 8 para 7) — destaque em azul;
- A chave identificada na tabela refere-se ao *node1* (AD1) — destaque em vermelho;
- O campo *last_node* permanece com valor 1 (*node1*) — destaque em verde;
- O encaminhamento foi realizado pela porta2 (*veth4–veth5*) — destaque em laranja.

Na segunda iteração analisada do Teste 02 (Figura 33), sendo "n1" (AD1) o *last_node*, envia-se o *node* referente ao seu *edge0* para consulta dos campos-chave na tabela do *switch*. Como não há validação para CID, o *node* relativo ao *edge1* é consultado. Assim, verifica-se *xid_type* e *id* do SID, que correspondem a uma chave registrada, definindo este como *last_node* e realizando o encaminhamento pela porta indicada na tabela.

Figura 33 – Teste 02: segunda iteração analisada



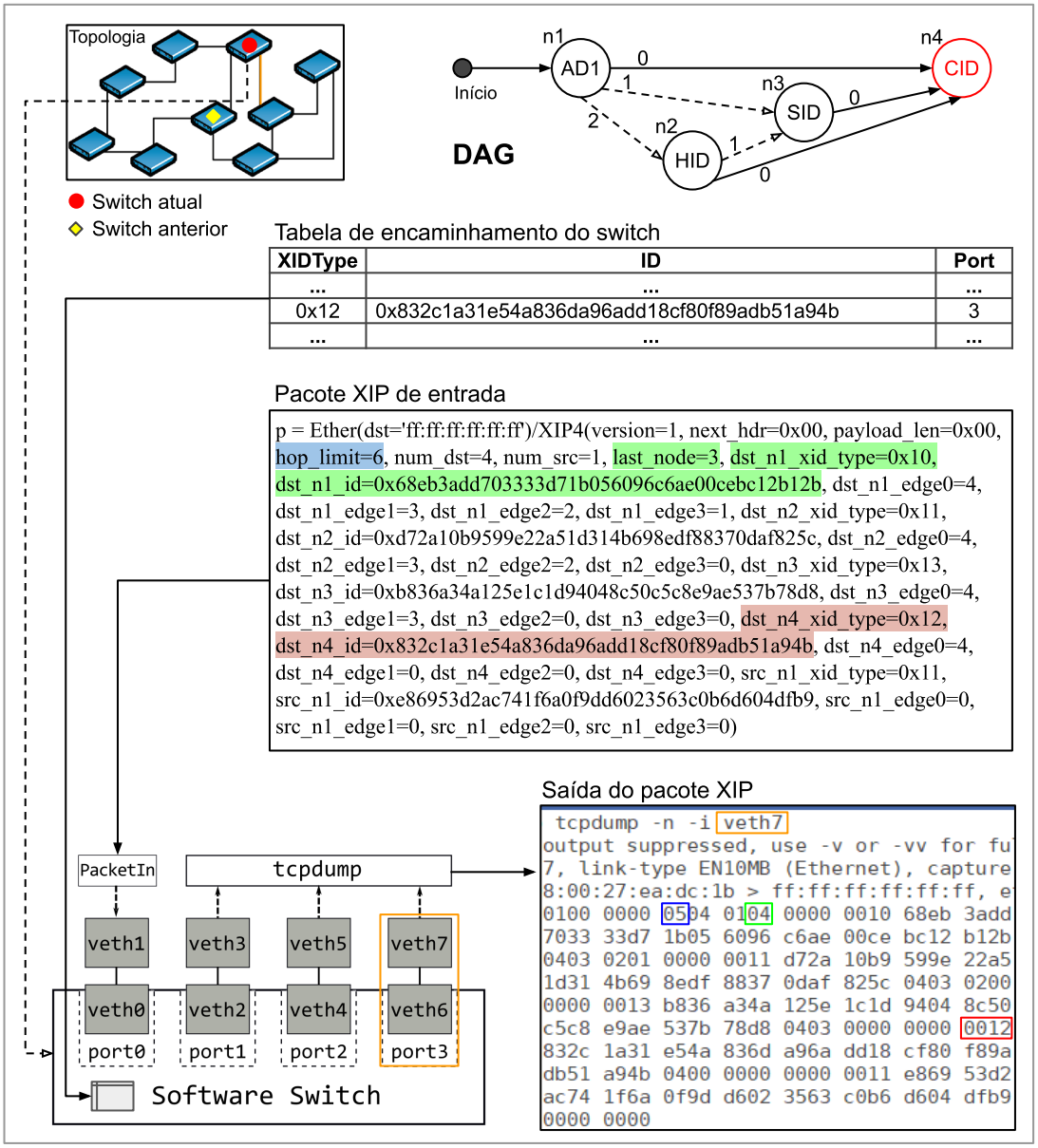
Fonte: O Autor (2020)

Com a realização do encaminhamento do pacote, observa-se na Figura 33:

- O valor do *hop_limit* foi decrementado (de 7 para 6) — destaque em azul;
- A chave identificada na tabela refere-se ao *node3* (SID) — destaque em vermelho;
- O campo *last_node* foi alterado de 1 (*node1*) para 3 (*node3*)— destaque em verde;
- O encaminhamento foi realizado pela porta3 (*veth6–veth7*) — destaque em laranja.

Na terceira iteração analisada do Teste 02 (Figura 34), sendo “n3” (SID) o *last_node*, envia-se o *node* referente ao seu *edge0* para consulta dos campos-chave na tabela de encaminhamento do *switch*. Como há correspondência dos campos *xid_type* e *id* do CID, define-se este como *last_node* e realiza-se o encaminhamento pela porta indicada na tabela.

Figura 34 – Teste 02: terceira iteração analisada



Fonte: O Autor (2020)

Com a realização do encaminhamento do pacote, observa-se na Figura 34:

- O valor do *hop_limit* foi decrementado (de 6 para 5) — destaque em azul;
- A chave identificada na tabela refere-se ao *node4* (CID) — destaque em vermelho;
- O campo *last_node* foi alterado de 3 (*node3*) para 4 (*node4*)— destaque em verde;
- O encaminhamento foi realizado pela porta3 (*veth6–veth7*) — destaque em laranja.

4.5.3 Teste 03 — DAG: AD1 > HID > SID > CID (cenário 2)

No terceiro teste do experimento deste trabalho, define-se um cenário que utiliza o mesmo endereço (DAG) apresentado na Subseção 4.5.2, retratado pela Figura 31. Assim, o DAG é formado por um domínio autônomo (AD1) e os *principals* que correspondem a *host*, serviço e conteúdo (HID, SID e CID). Os XIDs são representados através dos *nodes* adicionados no cabeçalho XIP implementado. Dessa forma, a Tabela 8 apresenta os *nodes* que constituem o cabeçalho utilizado neste terceiro teste.

Tabela 8 – Teste 03: *Nodes* do pacote XIP utilizado

Node	***XIDType***	***ID***	***Edge0***	***Edge1***	***Edge2***	***Edge3***
dst_n1	0x10	0x68eb3add703333d71b056096c6ae00cebc12b12b	4	3	2	1
dst_n2	0x11	0xd72a10b9599e22a51d314b698edf88370daf825c	4	3	2	0
dst_n3	0x13	0xb836a34a125e1c1d94048c50c5c8e9ae537b78d8	4	3	0	0
dst_n4	0x12	0x832c1a31e54a836da96add18cf80f89adb51a94b	4	0	0	0
src_n1	0x11	0xe86953d2ac741f6a0f9dd6023563c0b6d604dfb9	0	0	0	0

Fonte: O Autor (2020)

A Tabela 9 apresenta a evolução do pacote e do DAG durante o seu processamento. Nela, observa-se a realização dos saltos na rede, o registro da evolução do DAG e o caminho percorrido pelo pacote até o *node* coletor (de destino). Os detalhes dos processos de encaminhamento realizados no Teste 03 estão disponíveis nas figuras a seguir (Figura 35, Figura 36 e Figura 37).

Tabela 9 – Teste 03: Evolução do pacote XIP e do DAG

Salto	***Node* atual do DAG**	***LastNode***	***HopLimit***
1	AD1	1	9
2	HID	2	8
3	CID	4	7

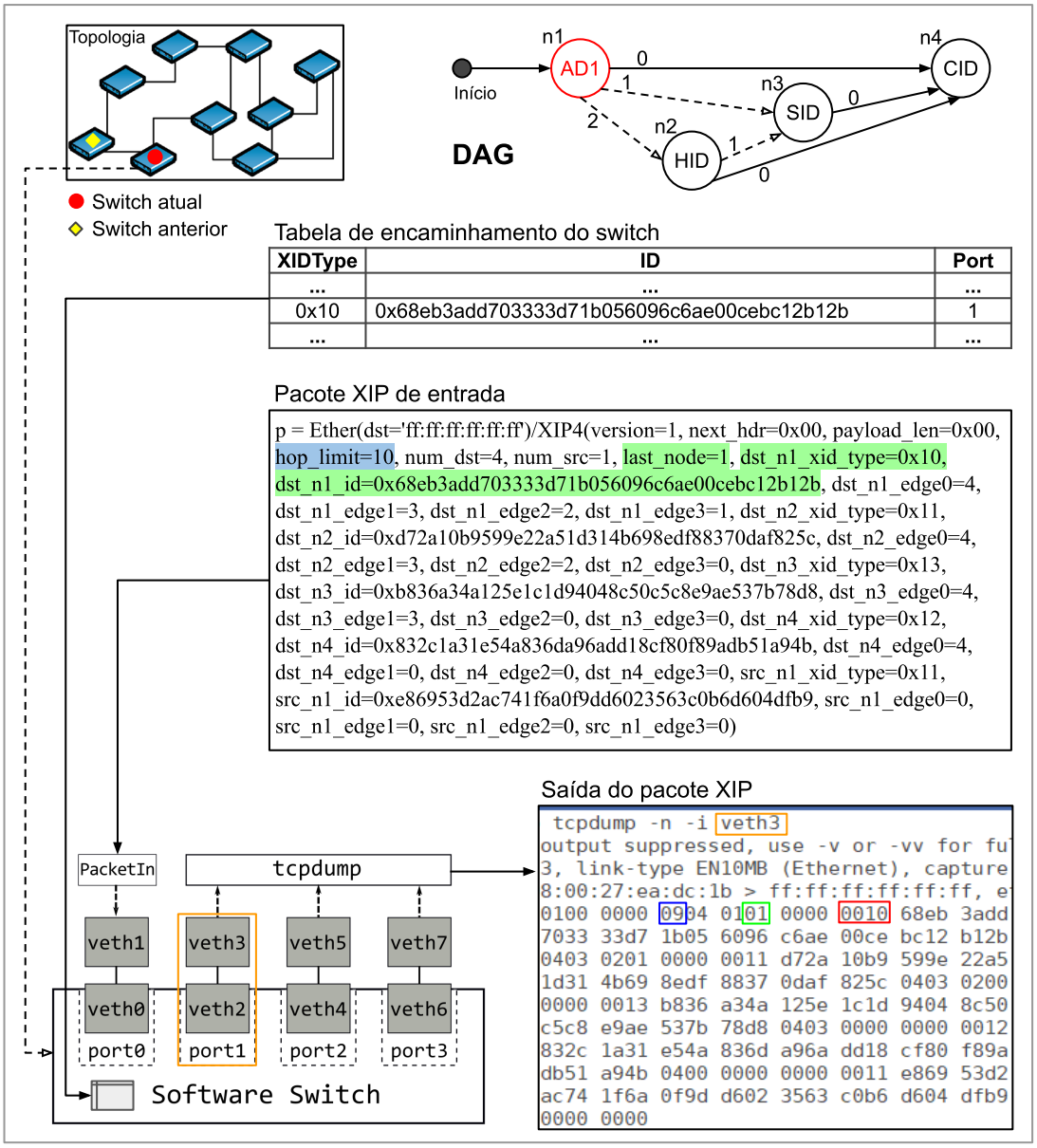
Fonte: O Autor (2020)

O objetivo de propor dois cenários de testes que utilizam o mesmo DAG para realizar o processamento dos pacotes é demonstrar as alternativas de encaminhamento da arquitetura XIA a partir de um mesmo endereço. Além disso, ao comparar os processos de encaminhamento realizados na Subseção 4.5.2 e nesta Subseção, é possível observar a dissociação entre o DAG e o caminho percorrido pelo pacote na topologia da rede.

Esses cenários distintos também podem ser usados para exemplificar casos de uso em uma rede real. Devido ao caminho percorrido pelo pacote no Teste 02 (Subseção 4.5.2), este poderia ter origem em uma aplicação de serviço de *streaming*, que recupera o conteúdo desejado a partir do *cache* em um dos *switches* da rede. No teste realizado nesta Subseção, o pacote poderia estar relacionado a um aplicativo que recupera o conteúdo de um *host* que hospeda um serviço *web*.

Na primeira iteração analisada do Teste 03 (Figura 35), sendo "n1" (AD1) o *last_node*, envia-se o *node* referente ao seu *edge0* para consulta na tabela do *switch*. Como não há validação para CID, o *node* de *edge1* e, em seguida, *edge2* são consultados, mas sem correspondência para SID ou HID. Assim, verifica-se *xid_type* e *id* do AD1, que correspondem a uma chave registrada, mantendo-o como *last_node* e realizando o encaminhamento pela porta indicada na tabela.

Figura 35 – Teste 03: primeira iteração analisada



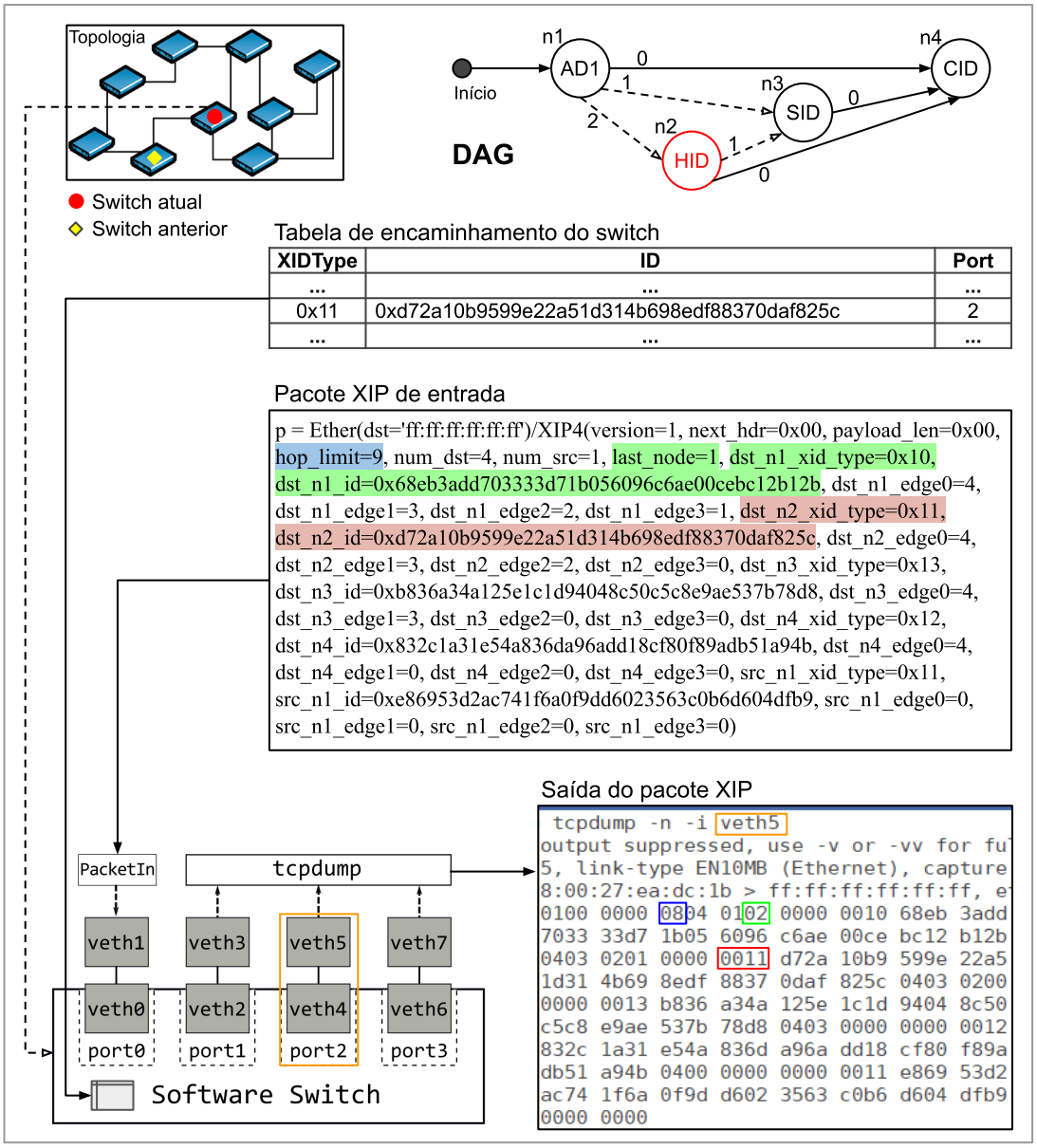
Fonte: O Autor (2020)

Com a realização do encaminhamento do pacote, observa-se na Figura 35:

- O valor do *hop_limit* foi decrementado (de 10 para 9) — destaque em azul;
- A chave identificada na tabela refere-se ao *node1* (AD1) — destaque em vermelho;
- O campo *last_node* permanece com valor 1 (*node1*) — destaque em verde;
- O encaminhamento foi realizado pela porta1 (*veth2–veth3*) — destaque em laranja.

Na segunda iteração analisada do Teste 03 (Figura 36), sendo "n1" (AD1) o *last_node*, envia-se o *node* referente ao seu *edge0* para consulta na tabela do *switch*. Como não há validação para CID, o *node* relativo a *edge1* é verificado, mas sem correspondência para SID. Em seguida, o *node* do *edge2* é consultado, com correspondência do *xid_type* e *id* do HID, definindo este como *last_node* e realizando o encaminhamento pela porta indicada na tabela.

Figura 36 – Teste 03: segunda iteração analisada



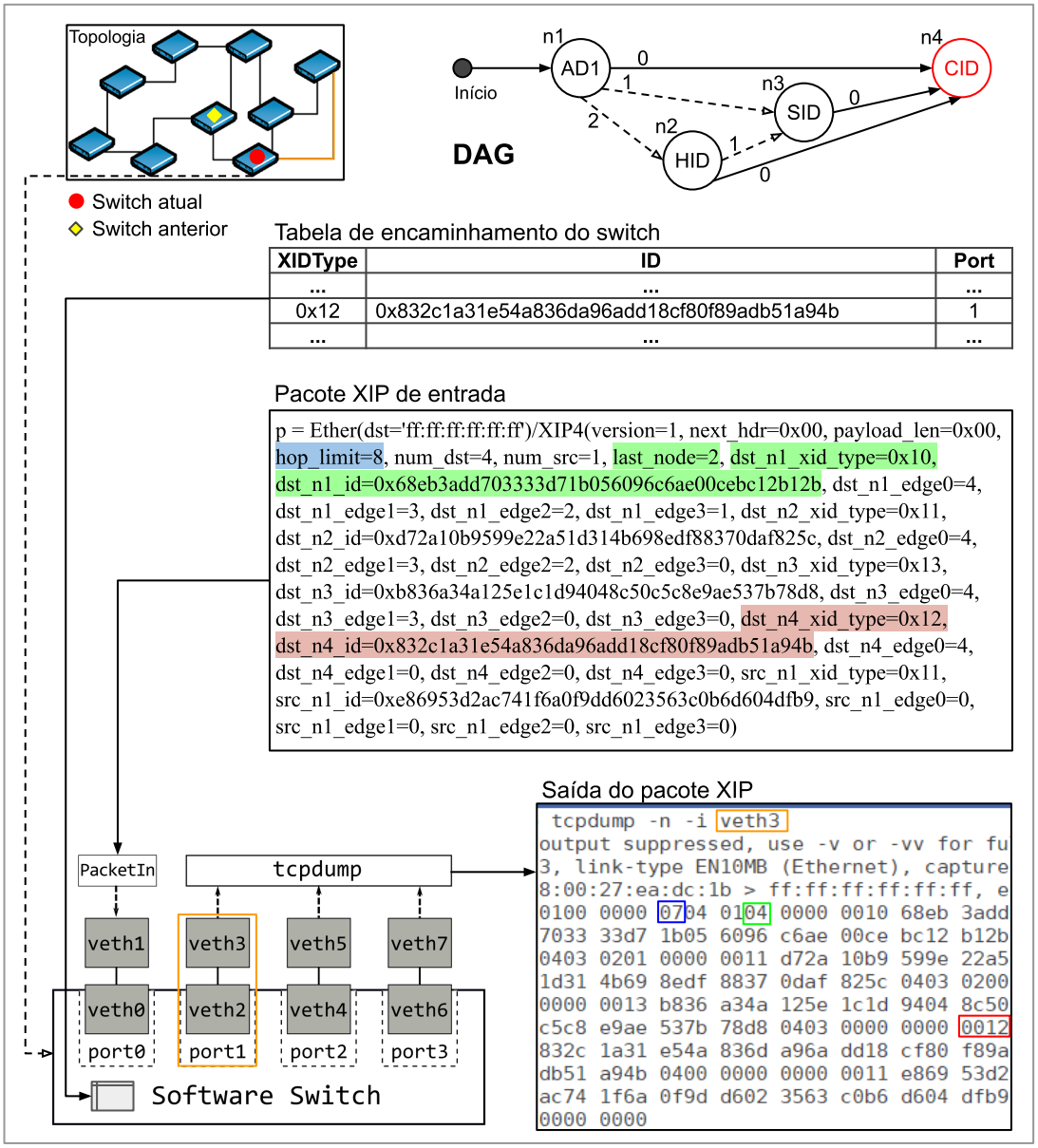
Fonte: O Autor (2020)

Com a realização do encaminhamento do pacote, observa-se na Figura 36:

- O valor do *hop_limit* foi decrementado (de 9 para 8) — destaque em azul;
- A chave identificada na tabela refere-se ao *node2* (HID) — destaque em vermelho;
- O campo *last_node* foi alterado de 1 (*node1*) para 2 (*node2*)— destaque em verde;
- O encaminhamento foi realizado pela porta2 (*veth4–veth5*) — destaque em laranja.

Na terceira iteração analisada do Teste 03 (Figura 37), sendo “n2” (HID) o *last_node*, envia-se o *node* referente ao seu *edge0* para consulta dos campos-chave na tabela de encaminhamento do *switch*. Como há correspondência dos campos *xid_type* e *id* do CID, define-se este como *last_node* e realiza-se o encaminhamento pela porta indicada na tabela.

Figura 37 – Teste 03: terceira iteração analisada



Fonte: O Autor (2020)

Com a realização do encaminhamento do pacote, observa-se na Figura 37:

- O valor do *hop_limit* foi decrementado (de 8 para 7) — destaque em azul;
- A chave identificada na tabela refere-se ao *node4* (CID) — destaque em vermelho;
- O campo *last_node* foi alterado de 2 (*node2*) para 4 (*node4*)— destaque em verde;
- O encaminhamento foi realizado pela porta1 (*veth2–veth3*) — destaque em laranja.

4.5.4 Teste 04: Processamento de endereços com tamanhos diferentes

Os testes anteriores foram realizados com pacotes XIP utilizando quatro *nodes* no endereço de destino, para promover as alternativas de encaminhamento previstas nos princípios da arquitetura XIA. Contudo, na sua especificação define-se que o endereço que compreende o cabeçalho do pacote possui tamanho variável. O comprimento do endereço XIP depende da quantidade de *nodes* que o compõem, sendo que, neste trabalho, ele corresponde a um valor entre 1 e 4 *nodes*. Dessa forma, o Teste 04 tem como objetivo demonstrar o processamento de pacotes com tamanhos diferentes do endereço de destino. Essas variações não foram utilizadas nos testes anteriores por reduzirem as alternativas de encaminhamento.

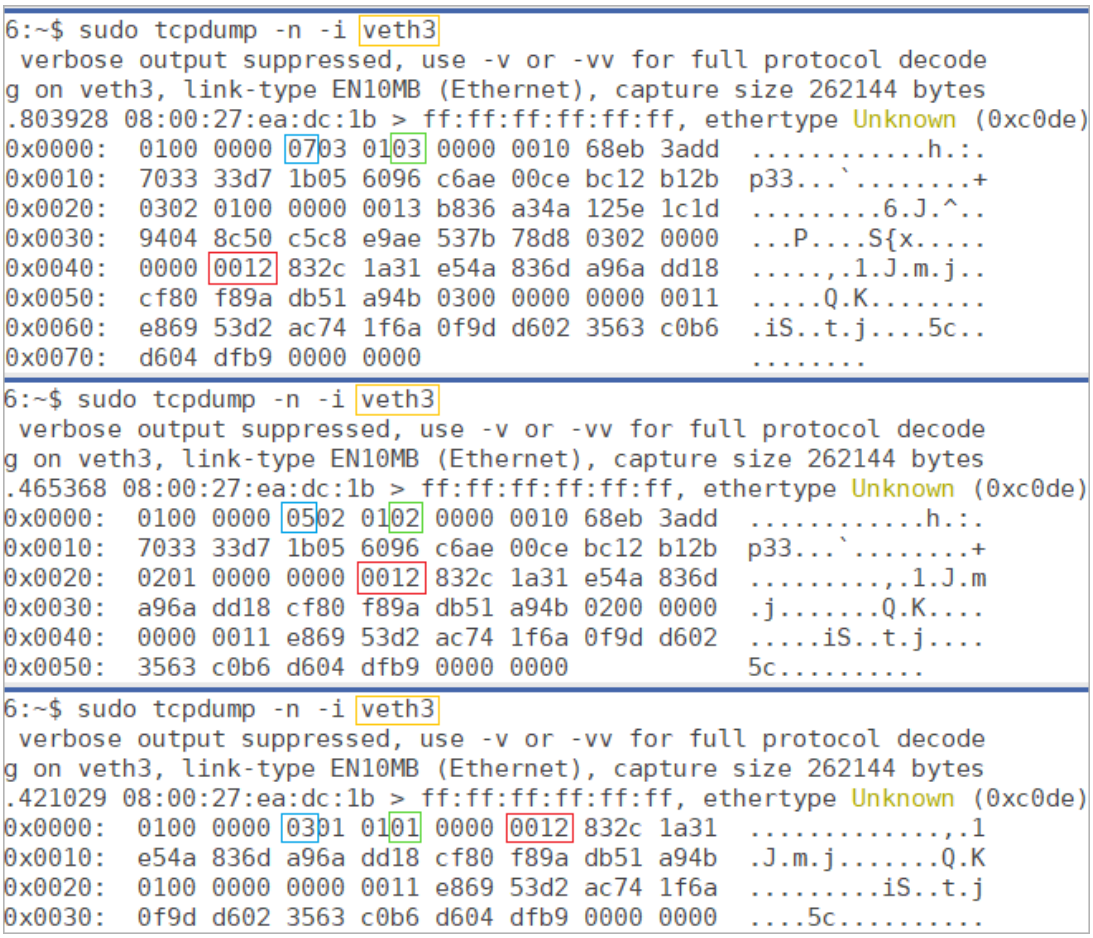
A Tabela 10 apresenta os pacotes de entrada utilizados para a realização do Teste 04. Esses pacotes, enviados ao *switch* que contém o plano de dados da XIA implementado em P4, são compostos por 3 *nodes*, 2 *nodes* e 1 *node*, respectivamente, no endereço de destino do seu cabeçalho XIP. A tabela de encaminhamento do *switch* possui os seguintes dados, na linha referente à correspondência válida para os pacotes utilizados no teste: ***xid_type*** 0x12; **id** 0x832c1a31e54a836da96add18cf80f89adb51a94b; e **porta** 1 (*veth2–veth3*).

Tabela 10 – Teste 04: Pacotes de entrada com quantidades diferentes de *nodes*

Nº de Nodes	**Pacote**
03 *nodes*	p = Ether(dst='ff:ff:ff:ff:ff:ff')XIP3(version=1, next_hdr=0x00, payload_len=0x00, hop_limit=8 , num_dst=3, num_src=1, last_node=2 , dst_n1_xid_type=0x10, dst_n1_id=0x68eb3add703333d71b056096c6ae00cebc12b12b, dst_n1_edge0=3, dst_n1_edge1=2, dst_n1_edge2=1, dst_n1_edge3=0, dst_n2_xid_type=0x13, dst_n2_id=0xb836a34a125e1c1d94048c50c5c8e9ae537b78d8, dst_n2_edge0=3, dst_n2_edge1=2, dst_n2_edge2=0, dst_n2_edge3=0, dst_n3_xid_type=0x12, dst_n3_id=0x832c1a31e54a836da96add18cf80f89adb51a94b, dst_n3_edge0=3, dst_n3_edge1=0, dst_n3_edge2=0, dst_n3_edge3=0, src_n1_xid_type=0x11, src_n1_id=0xe86953d2ac741f6a0f9dd6023563c0b6d604dfb9, src_n1_edge0=0, src_n1_edge1=0, src_n1_edge2=0, src_n1_edge3=0)
02 *nodes*	p = Ether(dst='ff:ff:ff:ff:ff:ff')XIP2(version=1, next_hdr=0x00, payload_len=0x00, hop_limit=6 , num_dst=2, num_src=1, last_node=1 , dst_n1_xid_type=0x10, dst_n1_id=0x68eb3add703333d71b056096c6ae00cebc12b12b, dst_n1_edge0=2, dst_n1_edge1=1, dst_n1_edge2=0, dst_n1_edge3=0, dst_n2_xid_type=0x12, dst_n2_id=0x832c1a31e54a836da96add18cf80f89adb51a94b, dst_n2_edge0=2, dst_n2_edge1=0, dst_n2_edge2=0, dst_n2_edge3=0, src_n1_xid_type=0x11, src_n1_id=0xe86953d2ac741f6a0f9dd6023563c0b6d604dfb9, src_n1_edge0=0, src_n1_edge1=0, src_n1_edge2=0, src_n1_edge3=0)
01 *node*	p = Ether(dst='ff:ff:ff:ff:ff:ff')/XIP1(version=1, next_hdr=0x00, payload_len=0x00, hop_limit=4 , num_dst=1, num_src=1, last_node=1 , dst_n1_xid_type=0x12, dst_n1_id=0x832c1a31e54a836da96add18cf80f89adb51a94b, dst_n1_edge0=1, dst_n1_edge1=0, dst_n1_edge2=0, dst_n1_edge3=0, src_n1_xid_type=0x11, src_n1_id=0xe86953d2ac741f6a0f9dd6023563c0b6d604dfb9, src_n1_edge0=0, src_n1_edge1=0, src_n1_edge2=0, src_n1_edge3=0)

Fonte: O Autor (2020)

Figura 38 – Teste 04: Pacotes de saída com quantidades diferentes de *nodes*



Fonte: O Autor (2020)

A Figura 38 exibe três telas agrupadas, com os pacotes de saída capturados através do *tcpdump*, que correspondem ao resultado do processamento dos pacotes de entrada, enviados através do *switch*. Dessa forma, a partir dos destaques realizados nos pacotes de entrada e saída, conclui-se o seguinte:

- O valor do *hop_limit* foi decrementado de 8 para 7, no pacote com três *nodes*; de 6 para 5, no pacote com dois *nodes*; e de 4 para 3, no pacote com um *node* — destaque em azul;
- A chave identificada na tabela refere-se ao *node3*, no pacote com 3 *nodes*; ao *node2*, no pacote com 2 *nodes*; e ao *node1*, no pacote com 1 *node*. Todos representam o CID — destaque em vermelho;
- O campo *last_node* foi alterado de 1 para 3, no pacote com três *nodes*; de 1 para 2, no pacote com dois *nodes* e manteve-se com valor 1, no pacote com um *node* — destaque em verde;
- O encaminhamento foi realizado pela porta1 (*veth2–veth3*) para os testes referentes aos três tamanhos de pacotes — destaque em laranja.

A saída correta desses pacotes valida mais uma funcionalidade da XIA, implementada no plano de dados desenvolvido em P4.

4.5.5 Considerações sobre os testes e resultados

Os testes realizados obtiveram os resultados previstos, de acordo com o ambiente e cenários de experimentação propostos. A realização dos experimentos para validar a implementação, executados em apenas uma VM, não comprometeu a aferição dos resultados, tendo em vista que o objetivo é atestar que as características do plano de dados da arquitetura XIA foram devidamente implementadas no código P4 desenvolvido. Em relação à execução dos testes, deve-se fazer algumas observações ou considerações acerca dos procedimentos realizados.

Os pacotes utilizados no experimento possuem um endereço *ethernet* de *broadcast* (ff:ff:ff:ff:ff:ff) antes dos campos que correspondem ao XIP. Indicar um endereço *ethernet* (não exatamente o de *broadcast*) é necessário para que o pacote possa ser enviado utilizando o *Scapy*, tendo em vista que o XIP trafega entre as interfaces utilizando o protocolo *ethernet*. Contudo, o fato de usar o endereço de *broadcast* não resulta no envio do pacote para todas as interfaces da rede, pois o *Scapy* indica explicitamente o nome da interface de rede para qual o pacote será encaminhado. Como no exemplo "*sendp(p, iface="veth0")*", que envia os pacotes para a interface virtual *veth0*. Além disso, ao processar o pacote, o *switch* só o encaminhará para a porta indicada na sua tabela *match-action*.

Os testes foram realizados utilizando dois endereços (DAGs) distintos, cinco XIDs e quatro interfaces de rede. No decorrer do experimento, os pacotes foram enviados de forma a validar o funcionamento correto do plano de dados. Desse modo, observa-se que um mesmo XID (*node*), validado na tabela de encaminhamento, mas em testes distintos, teve os seus pacotes correspondentes enviados através de portas diferentes. Assim como, a partir da validação de chaves de *nodes* distintos, verifica-se o encaminhamento de pacotes utilizando uma mesma porta do *switch*. Esses procedimentos buscaram validar o funcionamento correto do plano de dados implementado e do encaminhamento dos pacotes, baseando-se exclusivamente nos dados referentes à chave e porta indicadas na tabela do *switch*.

5 CONCLUSÃO

Os problemas ou limitações da *Internet* atual incentivaram pesquisadores da área de redes, de diferentes partes do mundo, a buscarem um novo modelo de arquitetura para a rede mundial. Esta deverá atender de maneira adequada às demandas, atuais e futuras, relacionadas à segurança, desempenho, mobilidade, escalabilidade e gerenciamento. Após aproximadamente uma década de pesquisas relacionadas a Arquiteturas para a Internet do Futuro (*Future Internet Architecture* - FIA), alguns projetos se destacaram e têm mostrado, em sua maioria, avanços relevantes em relação ao modelo atual da *Internet*.

Contudo, a mudança não seria um processo simples, e está cada vez mais difícil chegar à conclusão de qual arquitetura irá substituir a *Internet*, ou se isso é possível utilizando apenas uma proposta. Assim, cresce entre os pesquisadores a ideia de que arquiteturas distintas possam coexistir, sendo utilizadas em paralelo e com a *Internet* atual. Este cenário, chamado de pluralista, estimula iniciativas como o projeto FIXP (*Future Internet Exchange Point*), proposto para alavancar a *Internet* do Futuro no Brasil a partir da coexistência e interconexão de múltiplas arquiteturas de rede.

Com a intenção de contribuir com o projeto FIXP, adicionando uma nova arquitetura de rede à sua proposta multiFIAs, esta dissertação apresenta a implementação de plano de dados programável para a FIA *eXpressive Internet Architecture* (XIA). O desenvolvimento foi realizado utilizando a linguagem *Programming Protocol-independent Packet Processors* (P4), para aplicar os conceitos da XIA em dispositivo virtual de rede. O trabalho implementou dois dos princípios fundamentais da XIA, com o suporte a diferentes orientações principais (rede, *host*, serviço e conteúdo) e às possibilidades de endereçamento características da arquitetura.

Os testes de encaminhamento foram realizados através da execução do ambiente de experimentação e da definição de cenários distintos. A partir deles, observou-se que o plano de dados implementado em P4 atende às principais caraterísticas descritas nas especificações da XIA. Estas estão relacionadas a endereçamento flexível, alternativas de encaminhamento baseada em opções do protocolo XIP e validação de chave com a combinação de dois campos do cabeçalho. Além disso, os testes validam o comportamento do *switch*, que envia corretamente os pacotes para o próximo salto na rede, a partir de chave e porta indicadas na sua tabela de encaminhamento. Diante disso, conclui-se que o objetivo apresentado no Capítulo 1 deste trabalho foi alcançado, conforme demonstram os testes realizados.

5.1 Contribuições

Com a confirmação de que os objetivos propostos no trabalho foram alcançados, verificados a partir dos testes realizados, destaca-se como principal contribuição desta dissertação a implementação do plano de dados da arquitetura eXpressive Internet Architecture (XIA), em

dispositivo virtual de redes, usando a linguagem P4. A partir desta implementação, inicia-se o processo de inclusão da arquitetura XIA no projeto FIXP — "Alavancando a Internet do Futuro no Brasil através da coexistência e interconexão de múltiplas arquiteturas".

Nas pesquisas realizadas pelo autor, na literatura acadêmica, não foram encontrados trabalhos anteriores que relacionem a arquitetura XIA e a linguagem de programação P4. Portanto, até o momento da sua elaboração, esta dissertação apresenta-se como a primeira proposta de implementação da arquitetura XIA usando P4.

5.2 Limitações

Como principal limitação deste trabalho, destaca-se a ausência dos mecanismos para implantação do conceito de segurança intrínseca aos identificadores (XID) da XIA, que utilizam *hash* de chave pública-privada para prover segurança às suas comunicações. Esta limitação é destacada por tratar-se do único, dos três princípios fundamentais da arquitetura XIA, não implementado na proposta atual.

5.3 Trabalhos Futuros

A conclusão deste trabalho motiva a realização de estudos que deem continuidade às implementações com P4 e relacionadas à arquitetura XIA, no cenário de Internet do Futuro. Assim, são itens para o desenvolvimento de trabalhos futuros:

- A realização de testes da solução proposta, utilizando VMs distintas para a execução de *switches* e *hosts*, visando a avaliação de desempenho da implementação;
- A adição de mecanismos para promover a segurança intrínseca nos identificadores da arquitetura XIA já utilizados na implementação atual;
- O desenvolvimento de um plano de controle para integrar-se ao plano de dados implementado para a arquitetura XIA;
- Um estudo comparativo com arquiteturas que implementem segurança no núcleo da rede.

REFERÊNCIAS

ABELEM, A. et al. Advances in developing a Future Internet testbed in Brazil. p. 1–24, 2011.

ABELEM, A. et al. FIT@BR - a Future Internet Testbed in Brazil. *Proceedings of the Asia-Pacific Advanced Network*, v. 36, p. 1–8, 12 2013. ISSN 2227-3026.

ALBERTI, A. M.; SINGH, D. Developing a NovaGenesis architecture model for service oriented future Internet and IoT: An advanced transportation system scenario. In: *2014 IEEE World Forum on Internet of Things (WF-IoT)*. [S.l.: s.n.], 2014. p. 359–364.

ALEXANDER, D. S. et al. The SwitchWare Active Network Architecture. *IEEE Network*, v. 12, n. 3, p. 29–36, 1998. ISSN 08908044.

AMBROSIN, M.; COMPAGNO, A.; CONTI, M.; GHALI, C.; TSUDIK, G. Security and Privacy Analysis of National Science Foundation Future Internet Architectures. *IEEE Communications Surveys & Tutorials*, v. 20, p. 1418–1442, 01 2018.

ANAND, A. et al. XIA: An Architecture for an Evolvable and Trustworthy Internet. 11 2011.

ANDERSON, T. et al. The NEBULA Future Internet Architecture. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 7858 LNCS, p. 16–26, 2013. ISSN 16113349.

BALASUBRAMANIAM, S.; LEIBNITZ, K.; LIO, P.; BOTVICH, D.; MURATA, M. Biological principles for future Internet architecture design. *IEEE Communications Magazine*, v. 49, n. 7, p. 44–52, 2011.

BARRETT, D. *XIA Prototype Overview*. 2017. Disponível em: https://github.com/XIA-Project/xia-core/wiki/XIA-Prototype-Overview#x-stream-protocol-xsp. Acesso em: mar. 2020.

BARRETT, M. *An Introduction to Directed Acyclic Graphs*. 2020. Disponível em: https://cran.r-project.org/web/packages/ggdag/vignettes/intro-to-dags.html. Acesso em: fev. 2020.

BAS, A. *Performance of BMv2*. 2019. Disponível em: https://github.com/p4lang/behavioral-model/blob/master/docs/performance.md. Acesso em: mar. 2020.

BAS, A.; MOHSIN, W. *Announcing the P4Runtime v1.0 release*. 2019. Disponível em: https://p4.org/p4/p4runtime-v1-release. Acesso em: mar. 2020.

BIFULCO, R.; RETVARI, G. A survey on the programmable data plane: Abstractions, architectures, and open problems. *IEEE International Conference on High Performance Switching and Routing, HPSR*, v. 2018-June, 2018. ISSN 23255609.

BIONDI, P. *Scapy Documentation - Release 9165*. 2020. Disponível em: https://scapy.readthedocs.io/_/downloads/en/latest/pdf/. Acesso em: 2020.

BOSSHART, P. et al. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Computer Communication Review*, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, jul. 2014. ISSN 19435819.

BOSSHART, P. et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *Computer Communication Review*, v. 43, n. 4, p. 99–110, 2013. ISSN 01464833.

BRUNNER, M.; ABRAMOWICZ, H.; NIEBERT, N.; CORREIA, L. M. 4ward: A European perspective towards the future internet. *IEICE Transactions on Communications*, E93-B, n. 3, p. 442–445, 2010. ISSN 17451345.

BYERS, J. W.; MACHADO, M.; DOUCETTE, C.; BREEN, A. *Linux XIA: Enabling a crowd-sourced future Internet*. 2014. Presentation. Disponível em: https://docs.google.com/presentation/d/1OCiw2ahEuLcTohyKJ0UegiBLOeoH0qxJx9U7RnMo4bk/edit#slide=id.p. Acesso em: jan. 2020.

CHEN, M. et al. Towards incremental deployment of diverse network architectures on the Internet. *China Communications*, China Institute of Communications, v. 15, n. 5, p. 149–161, 2018. ISSN 16735447.

CHEN, Z.; LUO, H.; CUI, J.; JIN, M. Security analysis of a future internet architecture. *Proceedings - International Conference on Network Protocols, ICNP*, IEEE, 2013. ISSN 10921648.

CISCO. *Cisco Annual Internet Report (2018–2023) White Paper. Updated: February 17, 2020*. 2020. Disponível em: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. Acesso em: fev. 2020.

DAY, J. *Patterns in Network Architecture: A Return to Fundamentals*. [S.l.]: Prentice Hall, 2008. 429 p. ISBN 978-0-13-225242-3.

DING, W.; YAN, Z.; DENG, R. A Survey on Future Internet Security Architectures. *IEEE Access*, v. 4, p. 4374–4393, 01 2016.

DONATO, R. *TUN, TAP and Veth - Virtual Networking Devices Explained*. 2017. Disponível em: https://www.fir3net.com/Networking/Terms-and-Concepts/virtual-networking-devices-tun-tap-and-veth-pairs-explained.html. Acesso em: jan. 2020.

ELLIOTT, C. Geni - global environment for network innovations. In: *2008 33rd IEEE Conference on Local Computer Networks (LCN)*. [S.l.: s.n.], 2008. p. 8–8.

FINGERHUT, A. *Brief overview of P4*. 2018. Disponível em: https://github.com/jafingerhut/p4-guide. Acesso em: nov. 2019.

FINGERHUT, A. *P4 table behaviors*. 2019. Disponível em: https://github.com/jafingerhut/p4-guide/blob/master/docs/p4-table-behaviors.md. Acesso em: jan. 2020.

FISCHER, A.; BOTERO, J. F.; BECK, M. T.; MEER, H. de; HESSELBACH, X. Virtual Network Embedding: A Survey. *IEEE Communications Surveys Tutorials*, v. 15, n. 4, p. 1888–1906, 2013.

FISHER, D. A look behind the future internet architectures efforts. *ACM SIGCOMM Computer Communication Review*, v. 44, p. 45–49, 07 2014.

GARCIA, L. F. U. et al. Introdução à Linguagem P4: Teoria e Prática. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, p. 1–43, 2018.

GAVAZZA, J. A. et al. Future Internet Exchange Point (FIXP): Enabling Future Internet Architectures Interconnection. In: *Advanced Information Networking and Applications*. [S.l.]: Springer International Publishing, 2020. p. 703–714. ISBN 978-3-030-44040-4.

GAVAZZA, J. A.; SANTOS, M.; JR, P. D. M.; VERDI, F.; MONTEIRO, J. A. Implementação de um switch em P4 com suporte simultâneo a múltiplas arquiteturas de Internet do Futuro. In: . [S.l.: s.n.], 2019. p. 13–18.

GRANDL, R. et al. Supporting network evolution and incremental deployment with XIA. In: *SIGCOMM'12 - Proceedings of the ACM SIGCOMM 2012 Conference Applications, Technologies, Architectures, and Protocols for Computer Communication*. [S.l.: s.n.], 2012. ISBN 9781450314190.

GRASA, E. et al. Design principles of the Recursive InterNetwork Architecture (RINA). p. 1–10, 2014.

GUIMARÃES, C.; QUEVEDO, J.; FERREIRA, R.; CORUJO, D.; AGUIAR, R. L. Exploring interoperability assessment for Future Internet Architectures roll out. *Journal of Network and Computer Applications*, Elsevier Ltd, v. 136, n. March 2018, p. 38–56, 2019. ISSN 10958592.

GUPTA, A. et al. SDX: A Software Defined Internet Exchange. In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. New York, NY, USA: Association for Computing Machinery, 2014. (SIGCOMM '14), p. 551–562. ISBN 9781450328364.

HAN, B.; GOPALAKRISHNAN, V.; JI, L.; LEE, S. Network Function Virtualization: Challenges and Opportunities for Innovations. *IEEE Communications Magazine*, v. 53, n. 2, p. 90–97, 2015.

HAN, D. et al. XIA: Efficient Support for Evolvable Internetworking. In: *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012. p. 309–322.

HANCOCK, D.; MERWE, J. V. D. Hyper4: Using p4 to virtualize the programmable data plane. *CoNEXT 2016 - Proceedings of the 12th International Conference on Emerging Networking EXperiments and Technologies*, Association for Computing Machinery, p. 35–49, 2016.

HARAI, H. AKARI architecture design for new generation network. *LEOS Summer Topical Meeting*, IEEE, v. 2, p. 155–156, 2009. ISSN 10994742.

HILL, J.; ALOSERIJ, M.; GROSSO, P. Tracking Network Flows with P4. *Proceedings of INDIS 2018: Innovating the Network for Data-Intensive Science, Held in conjunction with SC 2018: The International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, p. 23–32, 2019.

HILL, J.; GROSSO, P. Securing Networks with P4. *University of Amsterdam System, System and Network Engineering*, p. 1–13, 2018.

HU, H. et al. A Survey on New Architecture Design of Internet. In: *2011 International Conference on Computational and Information Sciences*. [S.l.: s.n.], 2011. p. 729–732.

JACOBSON, V. et al. Networking Named Content. In: *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*. New York, NY, USA: Association for Computing Machinery, 2009. (CoNEXT '09), p. 1–12. ISBN 9781605586366.

JAIN, R. Internet 3.0: Ten problems with current internet architecture and solutions for the next generation. *Proceedings - IEEE Military Communications Conference MILCOM*, 2006.

JIANPING, W.; LILI, L.; DAN, L. The road towards future Internet. *Journal of Communications and Information Networks*, v. 1, n. 1, p. 86–97, 2016. ISSN 2096-1081.

JOKELA, P.; ZAHEMSZKY, A.; ROTHENBERG, C. E.; ARIANFAR, S.; NIKANDER, P. LIPSIN: Line Speed Publish/Subscribe Inter-Networking. *SIGCOMM Comput. Commun. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 39, n. 4, p. 195–206, ago. 2009.

KALJIC, E.; MARIC, A.; NJEMCEVIC, P.; HADZIALIC, M. A Survey on Data Plane Flexibility and Programmability in Software-Defined Networking. *IEEE Access*, v. 7, p. 47804–47840, 2019. ISSN 21693536.

KESHAV, S. Paradoxes of Internet Architecture. *IEEE Internet Computing*, v. 22, p. 96–102, 01 2018.

KHONDOKER, R.; NUGRAHA, B.; MARX, R.; BAYAROU, K. Security of selected future internet architectures: A survey. *Proceedings - 2014 8th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2014*, IEEE, p. 433–440, 2014.

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*. New York, NY, USA: Association for Computing Machinery, 2007. (SIGCOMM '07, 4), p. 181–192. ISBN 9781595937131. ISSN 0146-4833.

KREUTZ, D. et al. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, v. 103, n. 1, p. 14–76, 2015.

LAN, L.; JIANYA, C.; HONGYAN, C.; TAO, H.; YUNJIE, L. Resource scheduling virtualization in service-oriented future Internet architecture. *Journal of China Universities of Posts and Telecommunications*, The Journal of China Universities of Posts and Telecommunications, v. 22, n. 4, p. 92–100, 2015. ISSN 10058885.

LUTZ, R. Security and Privacy in Future Internet Architectures - Benefits and Challenges of Content Centric Networks. *ArXiv*, abs/1601.0, 2016.

MACHADO, M. *XIA-for-Linux*. 2015. Disponível em: https://github.com/AltraMayor/XIA-for-Linux/wiki. Acesso em: jan. 2020.

MACHADO, M.; DOUCETTE, C.; BYERS, J. W. Linux XIA: An interoperable meta network architecture to crowdsource the future internet. *ANCS 2015 - 11th 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, p. 147–158, 2015.

MACHADO, M. S. *Linux XIA: an interoperable meta network architecture*. OpenBU, 2014. 123 p. Doctoral Dissertation. Boston University. Disponível em: https://open.bu.edu/handle/2144/14389.

MAFFIONE, V.; SALVESTRINI, F.; GRASA, E.; BERGESIO, L.; TARZAN, M. A software development kit to exploit RINA programmability. *2016 IEEE International Conference on Communications, ICC 2016*, IEEE, 2016.

MARQUES, R. M. C. Named Data Networking with Programmable Switches. *FC-DI - Master Thesis (dissertation). Universidade de Lisboa*, p. 1–68, 2017.

MATTOS, D. M.; FERRAZ, L. H. G.; COSTA, L. H. M.; DUARTE, O. C. M. Virtual Network Performance Evaluation for Future Internet Architectures. *Journal of Emerging Technologies in Web Intelligence*, v. 4, n. 4, p. 304–314, 2012. ISSN 17980461.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833.

MCKEOWN, N.; SLOANE, T.; WANDERER, J. *P4 Runtime - Putting the Control Plane in Charge of the Forwarding Plane*. 2017. Disponível em: https://p4.org/api/p4-runtime-putting-the-control-plane-in-charge-of-the-forwarding-plane.html. Acesso em: feb. 2020.

MENG, Z.; CHEN, Z.; GUAN, Z. Seamless service migration for human-centered computing in future Internet architecture. In: *2017 IEEE SmartWorld Ubiquitous Intelligence and Computing, Advanced and Trusted Computed, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart City Innovation, SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI 2017 - Conference Proceedings*. [S.l.]: IEEE, 2017. p. 1–7.

MONTEIRO, J. A. S.; ALBERTI, A. M.; VERDI, F. L.; SILVA, F. O.; ROSA, P. F. *Alavancando a internet do futuro no Brasil através da coexistência e interconexão de múltiplas arquiteturas*. 2018. Disponível em: https://bv.fapesp.br/pt/auxilios/99492. Acesso em: jan. 2020.

MOREIRA, M. D. D.; FERNANDES, N. C.; COSTA, L. H. M. K.; DUARTE, O. C. M. Internet do Futuro: Um Novo Horizonte. In: *27º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.: s.n.], 2009. p. 59.

NAYLOR, D. et al. XIA: Architecting a More Trustworthy and Evolvable Internet. *SIGCOMM Comput. Commun. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 50–57, jul. 2014. ISSN 0146-4833.

NORDSTRÖM, E. et al. Serval: An End-Host Stack for Service-Centric Networking. In: *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012. p. 85–98. ISBN 978-931971-92-8.

OCONNOR, B. et al. Using P4 on Fixed-Pipeline and Programmable Stratum Switches. *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2019*, p. 2019–2020, 2019.

OSINSKI, T. *Network Prototyping Made Easy with P4 and Python*. 2020. Disponível em: https://p4.org/p4/prototyping-in-p4.html. Acesso em: mar. 2020.

P4-CONSORTIUM. *P4-16 Language Specification - version 1.2.0*. The P4 Language Consortium, 2019. Disponível em: https://p4.org/p4-spec/docs/P4-16-v1.2.0.html. Acesso em: oct. 2019.

PAN, J.; PAUL, S.; JAIN, R. A Survey of the Research on Future Internet Architectures. *IEEE Communications Magazine*, v. 49, n. 7, p. 26–36, 2011.

PAUL, S.; PAN, J.; JAIN, R. Architectures for the Future Networks and the next Generation Internet: A Survey. *Computer Communications*, Elsevier Science Publishers B. V., NLD, v. 34, n. 1, p. 2–42, jan. 2011. ISSN 0140-3664.

PEREIRA, J.; SILVA, F.; FILHO, E.; KOFUJI, S.; FROSI, P. Title model ontology for future internet networks. In: *The Future Internet. Future Internet Assembly 2011: achievements and technological promises*. [S.l.]: Springer Berlin Heidelberg, 2011. p. 103–116. ISBN 978-3-642-20898-0.

RAYCHAUDHURI, D.; NAGARAJA, K.; VENKATARAMANI, A. MobilityFirst: a robust and trustworthy mobility-centric architecture for the future Internet. *ACM SIGMOBILE Mobile Computing and Communications Review*, v. 16, p. 2–13, 12 2012.

REXFORD, J.; DOVROLIS, C. Point/Counterpoint Future Internet Architecture: Clean-Slate Versus Evolutionary Research. *Communications of the ACM*, v. 53, n. 9, p. 36–40, 09 2010. ISSN 00010782.

RIJSMAN, B. *Getting Started with P4*. 2019. Disponível em: https://p4.org/p4/getting-started-with-p4.html. Acesso em: 2020.

SILVA, F. D. O.; PEREIRA, J. H. D. S.; ROSA, P. F.; KOFUJI, S. T. Enabling Future Internet Architecture Research and Experimentation by Using Software Defined Networking. *Proceedings - European Workshop on Software Defined Networks, EWSDN 2012*, p. 73–78, 2012.

TARAN, A.; LAVROV, D. Future Internet Architecture Cleanslate vs Evolutionary Design. *Mathematical Structures and Modeling*, v. 3, p. 142–151, 2016.

TENNENHOUSE, D. L.; SMITH, J. M.; SINCOSKIE, W. D.; WETHERALL, D. J.; MINDEN, G. J. A Survey of Active Network Research. *IEEE Communications Magazine*, v. 35, n. 1, p. 80–86, 1997. ISSN 01636804.

THEODORO, L. C. et al. Entity title architecture pilot: Deploying a clean slate SDN based network at a telecom operator. *AICT 2015 - 11th Advanced International Conference on Telecommunications*, n. c, p. 144–149, 2015.

TUNCER, H.; MISHRA, S.; SHENOY, N. A survey of identity and handoff management approaches for the future Internet. *Computer Communications*, Elsevier B.V., v. 36, n. 1, p. 63–79, 2012. ISSN 01403664.

WANG, J.; WANG, L.; REN, J.; HOVHANNISYAN, H.; WANG, J. A generic and programmable name resolution system in future internet architectures. *IEEE Wireless Communications*, v. 24, n. 1, p. 10–17, 2017.

XIA, W.; WEN, Y.; FOH, C. H.; NIYATO, D.; XIE, H. A Survey on Software-Defined Networking. *IEEE Communications Surveys and Tutorials*, IEEE, v. 17, n. 1, p. 27–51, 2015. ISSN 1553877X.

XIA_OVERVIEW. *The eXpressive Internet Architecture: Architecture and Research Overview*. 2016. Disponível em: http://www.cs.cmu.edu/~xia/xia-overview/xia-overview.pdf. Acesso em: dez. 2019.

ZAHARIADIS, T. et al. Towards a Future Internet Architecture. *Future Internet Assembly LNCS*, v. 6656, p. 7–18, 2011. ISSN 03029743.

ZHANG, L. et al. Named Data Networking. *SIGCOMM Comput. Commun. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 66–73, jul. 2014. ISSN 0146-4833.

APÊNDICE A – XIA.P4

Código-fonte 14 – Algoritmo em P4 do plano de dados da arquitetura XIA

```
1  //xia.p4
2  //Desenvolvido por Jackson Nunes da Silva
3
4  #include <core.p4>
5  #include <v1model.p4>
6
7  //========================== H E A D E R S ============================
8
9  const   bit<16> ETHERTYPE_XIA = 0xc0de; //Ethernet Type
10 typedef bit<48> EthernetAddr;
11
12 header Ethernet_t {
13     EthernetAddr    dst_addr;
14     EthernetAddr    src_addr;
15     bit<16>         ethertype;
16 }
17
18 header XIP_fixedfields_t {
19     bit<8>      version;
20     bit<8>      next_hdr;
21     bit<16>     payload_len;
22     bit<8>      hop_limit;
23     bit<8>      num_dst;
24     bit<8>      num_src;
25     bit<8>      last_node;
26 }
27
28 struct   XIP_Node_t {
29     bit<32>     xid_type;
30     bit<160>    id;
31     bit<8>      edge0;
32     bit<8>      edge1;
33     bit<8>      edge2;
34     bit<8>      edge3;
35 }
36
37 header One_Node_t {
38     XIP_Node_t  node1;
39 }
40
41 header Two_Nodes_t {
42     XIP_Node_t  node1;
43     XIP_Node_t  node2;
44 }
45
46 header Three_Nodes_t {
47     XIP_Node_t  node1;
48     XIP_Node_t  node2;
49     XIP_Node_t  node3;
50 }
51
52 header Four_Nodes_t {
53     XIP_Node_t  node1;
54     XIP_Node_t  node2;
55     XIP_Node_t  node3;
56     XIP_Node_t  node4;
57 }
```

```
58
59 struct headers_t {
60     Ethernet_t          ethernet;
61     XIP_fixedfields_t   fixedfields;
62     One_Node_t          dst_one_node;
63     Two_Nodes_t         dst_two_nodes;
64     Three_Nodes_t       dst_three_nodes;
65     Four_Nodes_t        dst_four_nodes;
66     One_Node_t          src_one_node;
67     Two_Nodes_t         src_two_nodes;
68     Three_Nodes_t       src_three_nodes;
69     Four_Nodes_t        src_four_nodes;
70 }
71
72 struct metadata_t {
73     bit<8>      last_node;
74     bit<32>     dst_xid_type;
75     bit<32>     dst_node1_xid_type;
76     bit<32>     dst_node2_xid_type;
77     bit<32>     dst_node3_xid_type;
78     bit<32>     dst_node4_xid_type;
79     bit<160>    dst_id;
80     bit<160>    dst_node1_id;
81     bit<160>    dst_node2_id;
82     bit<160>    dst_node3_id;
83     bit<160>    dst_node4_id;
84     bit<8>      dst_edge0;
85     bit<8>      dst_edge1;
86     bit<8>      dst_edge2;
87     bit<8>      dst_edge3;
88     bit<32>     tmp_xid_type;
89     bit<160>    tmp_id;
90     bit<8>      edge;
91 }
92
93
94 //============================ P A R S E R ==============================
95
96 parser XIA_Parser(packet_in packet,
97                   out headers_t hdr,
98                   inout metadata_t meta,
99                   inout standard_metadata_t standard_meta) {
100
101     state start {
102         transition parse_ethernet;
103     }
104
105     state parse_ethernet {
106         packet.extract(hdr.ethernet);
107         transition select (hdr.ethernet.ethertype) {
108             ETHERTYPE_XIA: parse_xip_fixedfields;
109             default: accept;
110         }
111     }
112
113     state parse_xip_fixedfields {
114         packet.extract(hdr.fixedfields);
115         transition parse_select_xip_dst_nodes;
116     }
117
118     state parse_select_xip_dst_nodes {
119         transition select (hdr.fixedfields.num_dst) {
120             1: parse_dst_one_node;
121             2: parse_dst_two_nodes;
122             3: parse_dst_three_nodes;
```

```
123             4: parse_dst_four_nodes;
124             default: accept;
125         }
126     }
127
128     state parse_dst_one_node {
129         packet.extract(hdr.dst_one_node);
130         transition parse_select_xip_src_nodes;
131     }
132     state parse_dst_two_nodes {
133         packet.extract(hdr.dst_two_nodes);
134         transition parse_select_xip_src_nodes;
135     }
136     state parse_dst_three_nodes {
137         packet.extract(hdr.dst_three_nodes);
138         transition parse_select_xip_src_nodes;
139     }
140     state parse_dst_four_nodes {
141         packet.extract(hdr.dst_four_nodes);
142         transition parse_select_xip_src_nodes;
143     }
144
145     state parse_select_xip_src_nodes {
146         transition select (hdr.fixedfields.num_src) {
147             1: parse_src_one_node;
148             2: parse_src_two_nodes;
149             3: parse_src_three_nodes;
150             4: parse_src_four_nodes;
151             default: accept;
152         }
153     }
154
155     state parse_src_one_node {
156         packet.extract(hdr.src_one_node);
157         transition accept;
158     }
159     state parse_src_two_nodes {
160         packet.extract(hdr.src_two_nodes);
161         transition accept;
162     }
163     state parse_src_three_nodes {
164         packet.extract(hdr.src_three_nodes);
165         transition accept;
166     }
167     state parse_src_four_nodes {
168         packet.extract(hdr.src_four_nodes);
169         transition accept;
170     }
171 }
172
173
174 //================== V E R I F Y   C H E C K S U M ==================
175
176 control XIA_VerifyChecksum(inout headers_t hdr, inout metadata_t meta) {
177     apply { }
178 }
179
180
181 //========================== I N G R E S S ===========================
182
183 control XIA_Ingress(inout headers_t hdr,
184                     inout metadata_t meta,
185                     inout standard_metadata_t standard_metadata) {
186
187     bool dropped = false;
```

```
188
189     action drop_action() {
190         mark_to_drop(standard_metadata);
191         dropped = true;
192     }
193
194     action to_port_action (bit<9> port) {
195         hdr.fixedfields.last_node = meta.last_node;
196         hdr.fixedfields.hop_limit = hdr.fixedfields.hop_limit - 1;
197         standard_metadata.egress_spec = port;
198     }
199
200     table xia1 {
201         key = {
202             meta.dst_xid_type : exact;
203             meta.dst_id : exact;
204         }
205         actions = {
206             drop_action;
207             to_port_action;
208         }
209         size = 1024;
210         default_action = drop_action;
211     }
212     table xia2 {
213         key = {
214             meta.dst_xid_type : exact;
215             meta.dst_id : exact;
216         }
217         actions = {
218             drop_action;
219             to_port_action;
220         }
221         size = 1024;
222         default_action = drop_action;
223     }
224     table xia3 {
225         key = {
226             meta.dst_xid_type : exact;
227             meta.dst_id : exact;
228         }
229         actions = {
230             drop_action;
231             to_port_action;
232         }
233         size = 1024;
234         default_action = drop_action;
235     }
236     table xia4 {
237         key = {
238             meta.dst_xid_type : exact;
239             meta.dst_id : exact;
240         }
241         actions = {
242             drop_action;
243             to_port_action;
244         }
245         size = 1024;
246         default_action = drop_action;
247     }
248
249     apply {
250
251         meta.dst_edge0 = 0;
252         meta.dst_edge1 = 0;
```

```
253         meta.dst_edge2 = 0;
254         meta.dst_edge3 = 0;
255
256         if (hdr.fixedfields.num_dst == 1) {
257             if (hdr.fixedfields.last_node == 1) {
258                 meta.dst_edge0 = hdr.dst_one_node.node1.edge0;
259             }
260             meta.dst_node1_xid_type = hdr.dst_one_node.node1.xid_type;
261             meta.dst_node1_id = hdr.dst_one_node.node1.id;
262         }
263
264         else if (hdr.fixedfields.num_dst == 2) {
265             if (hdr.fixedfields.last_node == 1) {
266                 meta.dst_edge0 = hdr.dst_two_nodes.node1.edge0;
267                 meta.dst_edge1 = hdr.dst_two_nodes.node1.edge1;
268             }
269             else if (hdr.fixedfields.last_node == 2) {
270                 meta.dst_edge0 = hdr.dst_two_nodes.node2.edge0;
271                 meta.dst_edge1 = hdr.dst_two_nodes.node2.edge1;
272             }
273             meta.dst_node1_xid_type = hdr.dst_two_nodes.node1.xid_type;
274             meta.dst_node1_id = hdr.dst_two_nodes.node1.id;
275             meta.dst_node2_xid_type = hdr.dst_two_nodes.node2.xid_type;
276             meta.dst_node2_id = hdr.dst_two_nodes.node2.id;
277         }
278
279         else if (hdr.fixedfields.num_dst == 3) {
280             if (hdr.fixedfields.last_node == 1) {
281                 meta.dst_edge0 = hdr.dst_three_nodes.node1.edge0;
282                 meta.dst_edge1 = hdr.dst_three_nodes.node1.edge1;
283                 meta.dst_edge2 = hdr.dst_three_nodes.node1.edge2;
284             }
285             else if (hdr.fixedfields.last_node == 2) {
286                 meta.dst_edge0 = hdr.dst_three_nodes.node2.edge0;
287                 meta.dst_edge1 = hdr.dst_three_nodes.node2.edge1;
288                 meta.dst_edge2 = hdr.dst_three_nodes.node2.edge2;
289             }
290             else if (hdr.fixedfields.last_node == 3) {
291                 meta.dst_edge0 = hdr.dst_three_nodes.node3.edge0;
292                 meta.dst_edge1 = hdr.dst_three_nodes.node3.edge1;
293                 meta.dst_edge2 = hdr.dst_three_nodes.node3.edge2;
294             }
295             meta.dst_node1_xid_type = hdr.dst_three_nodes.node1.xid_type;
296             meta.dst_node1_id = hdr.dst_three_nodes.node1.id;
297             meta.dst_node2_xid_type = hdr.dst_three_nodes.node2.xid_type;
298             meta.dst_node2_id = hdr.dst_three_nodes.node2.id;
299             meta.dst_node3_xid_type = hdr.dst_three_nodes.node3.xid_type;
300             meta.dst_node3_id = hdr.dst_three_nodes.node3.id;
301         }
302
303         else if (hdr.fixedfields.num_dst == 4) {
304             if (hdr.fixedfields.last_node == 1) {
305                 meta.dst_edge0 = hdr.dst_four_nodes.node1.edge0;
306                 meta.dst_edge1 = hdr.dst_four_nodes.node1.edge1;
307                 meta.dst_edge2 = hdr.dst_four_nodes.node1.edge2;
308                 meta.dst_edge3 = hdr.dst_four_nodes.node1.edge3;
309             }
310             else if (hdr.fixedfields.last_node == 2) {
311                 meta.dst_edge0 = hdr.dst_four_nodes.node2.edge0;
312                 meta.dst_edge1 = hdr.dst_four_nodes.node2.edge1;
313                 meta.dst_edge2 = hdr.dst_four_nodes.node2.edge2;
314                 meta.dst_edge3 = hdr.dst_four_nodes.node2.edge3;
315             }
316             else if (hdr.fixedfields.last_node == 3) {
317                 meta.dst_edge0 = hdr.dst_four_nodes.node3.edge0;
```

```
318                 meta.dst_edge1 = hdr.dst_four_nodes.node3.edge1;
319                 meta.dst_edge2 = hdr.dst_four_nodes.node3.edge2;
320                 meta.dst_edge3 = hdr.dst_four_nodes.node3.edge3;
321             }
322             else if (hdr.fixedfields.last_node == 4) {
323                 meta.dst_edge0 = hdr.dst_four_nodes.node4.edge0;
324                 meta.dst_edge1 = hdr.dst_four_nodes.node4.edge1;
325                 meta.dst_edge2 = hdr.dst_four_nodes.node4.edge2;
326                 meta.dst_edge3 = hdr.dst_four_nodes.node4.edge3;
327             }
328             meta.dst_node1_xid_type = hdr.dst_four_nodes.node1.xid_type;
329             meta.dst_node1_id = hdr.dst_four_nodes.node1.id;
330             meta.dst_node2_xid_type = hdr.dst_four_nodes.node2.xid_type;
331             meta.dst_node2_id = hdr.dst_four_nodes.node2.id;
332             meta.dst_node3_xid_type = hdr.dst_four_nodes.node3.xid_type;
333             meta.dst_node3_id = hdr.dst_four_nodes.node3.id;
334             meta.dst_node4_xid_type = hdr.dst_four_nodes.node4.xid_type;
335             meta.dst_node4_id = hdr.dst_four_nodes.node4.id;
336         }
337 
338         meta.edge = meta.dst_edge0;
339 
340         if (meta.edge == meta.dst_edge0) {
341             if (meta.dst_edge0 == 1) {
342                 meta.dst_xid_type = meta.dst_node1_xid_type;
343                 meta.dst_id = meta.dst_node1_id;
344                 meta.last_node = 1;
345                 xia1.apply();
346             }
347             else if (meta.dst_edge0 == 2) {
348                 meta.dst_xid_type = meta.dst_node2_xid_type;
349                 meta.dst_id = meta.dst_node2_id;
350                 meta.last_node = 2;
351                 xia1.apply();
352             }
353             else if (meta.dst_edge0 == 3) {
354                 meta.dst_xid_type = meta.dst_node3_xid_type;
355                 meta.dst_id = meta.dst_node3_id;
356                 meta.last_node = 3;
357                 xia1.apply();
358             }
359             else if (meta.dst_edge0 == 4) {
360                 meta.dst_xid_type = meta.dst_node4_xid_type;
361                 meta.dst_id = meta.dst_node4_id;
362                 meta.last_node = 4;
363                 xia1.apply();
364             }
365             meta.edge = meta.dst_edge1;
366         }
367 
368         if (meta.edge == meta.dst_edge1) {
369             if (meta.dst_edge1 == 1) {
370                 meta.dst_xid_type = meta.dst_node1_xid_type;
371                 meta.dst_id = meta.dst_node1_id;
372                 meta.last_node = 1;
373                 xia2.apply();
374             }
375             else if (meta.dst_edge1 == 2) {
376                 meta.dst_xid_type = meta.dst_node2_xid_type;
377                 meta.dst_id = meta.dst_node2_id;
378                 meta.last_node = 2;
379                 xia2.apply();
380             }
381             else if (meta.dst_edge1 == 3) {
382                 meta.dst_xid_type = meta.dst_node3_xid_type;
```

```
383                 meta.dst_id = meta.dst_node3_id;
384                 meta.last_node = 3;
385                 xia2.apply();
386             }
387             else if (meta.dst_edge1 == 4) {
388                 meta.dst_xid_type = meta.dst_node4_xid_type;
389                 meta.dst_id = meta.dst_node4_id;
390                 meta.last_node = 4;
391                 xia2.apply();
392             }
393             meta.edge = meta.dst_edge2;
394         }
395 
396         if (meta.edge == meta.dst_edge2) {
397             if (meta.dst_edge2 == 1) {
398                 meta.dst_xid_type = meta.dst_node1_xid_type;
399                 meta.dst_id = meta.dst_node1_id;
400                 meta.last_node = 1;
401                 xia3.apply();
402             }
403             else if (meta.dst_edge2 == 2) {
404                 meta.dst_xid_type = meta.dst_node2_xid_type;
405                 meta.dst_id = meta.dst_node2_id;
406                 meta.last_node = 2;
407                 xia3.apply();
408             }
409             else if (meta.dst_edge2 == 3) {
410                 meta.dst_xid_type = meta.dst_node3_xid_type;
411                 meta.dst_id = meta.dst_node3_id;
412                 meta.last_node = 3;
413                 xia3.apply();
414             }
415             else if (meta.dst_edge2 == 4) {
416                 meta.dst_xid_type = meta.dst_node4_xid_type;
417                 meta.dst_id = meta.dst_node4_id;
418                 meta.last_node = 4;
419                 xia3.apply();
420             }
421             meta.edge = meta.dst_edge3;
422         }
423 
424         if (meta.edge == meta.dst_edge3) {
425             if (meta.dst_edge3 == 1) {
426                 meta.dst_xid_type = meta.dst_node1_xid_type;
427                 meta.dst_id = meta.dst_node1_id;
428                 meta.last_node = 1;
429                 xia4.apply();
430             }
431             else if (meta.dst_edge3 == 2) {
432                 meta.dst_xid_type = meta.dst_node2_xid_type;
433                 meta.dst_id = meta.dst_node2_id;
434                 meta.last_node = 2;
435                 xia4.apply();
436             }
437             else if (meta.dst_edge3 == 3) {
438                 meta.dst_xid_type = meta.dst_node3_xid_type;
439                 meta.dst_id = meta.dst_node3_id;
440                 meta.last_node = 3;
441                 xia4.apply();
442             }
443             else if (meta.dst_edge3 == 4) {
444                 meta.dst_xid_type = meta.dst_node4_xid_type;
445                 meta.dst_id = meta.dst_node4_id;
446                 meta.last_node = 4;
447                 xia4.apply();
```

```
448             }
449         }
450     }
451 }
452
453
454 //============================ E G R E S S ============================
455
456 control XIA_Egress(inout headers_t hdr,
457                    inout metadata_t meta,
458                    inout standard_metadata_t standard_metadata)
459 {
460     apply { }
461 }
462
463
464 //================= C O M P U T E   C H E C K S U M ==================
465
466 control XIA_ComputeChecksum(inout headers_t hdr,
467                             inout metadata_t meta)
468 {
469     apply { }
470 }
471
472
473 //========================= D E P A R S E R ==========================
474
475 control XIA_Deparser(packet_out packet,
476                      in headers_t hdr)
477 {
478     apply {
479         packet.emit(hdr.ethernet);
480         packet.emit(hdr.fixedfields);
481         packet.emit(hdr.dst_one_node);
482         packet.emit(hdr.dst_two_nodes);
483         packet.emit(hdr.dst_three_nodes);
484         packet.emit(hdr.dst_four_nodes);
485         packet.emit(hdr.src_one_node);
486         packet.emit(hdr.src_two_nodes);
487         packet.emit(hdr.src_three_nodes);
488         packet.emit(hdr.src_four_nodes);
489     }
490 }
491
492
493 //============================ S W I T C H ============================
494
495 V1Switch(XIA_Parser(),
496 XIA_VerifyChecksum(),
497 XIA_Ingress(),
498 XIA_Egress(),
499 XIA_ComputeChecksum(),
500 XIA_Deparser()) main;
```

APÊNDICE B – XIP.PY

Código-fonte 15 – Implementação do cabeçalho do pacote XIP com *Scapy*

```
#xip.py - Implementado por Jackson Nunes da Silva

# scapy.contrib.description = XIP
# scapy.contrib.status = loads

"""
XIP (eXpressive Internet Protocol).
"""

from scapy.all import *
import sys, os
from scapy.packet import *
from scapy.fields import *
from scapy.layers.l2 import Ether
from scapy.layers.l2 import *

class XIP(Packet):
    name="XIP"
    fields_desc = [ XByteField('version', 1),
                    XByteField('next_hdr', 0x00),
                    XShortField('payload_len', 0),
                    XByteField('hop_limit', 0),
                    XByteField('num_dst', 0),
                    XByteField('num_src', 0),
                    XByteField('last_node', 0),
                    XBitField('dst_n1_xid_type', 0x00, 32),
                    XBitField('dst_n1_id', 0, 160),
                    XByteField('dst_n1_edge0', 0),
                    XByteField('dst_n1_edge1', 0),
                    XByteField('dst_n1_edge2', 0),
                    XByteField('dst_n1_edge3', 0),
                    XBitField('dst_n2_xid_type', 0x00, 32),
                    XBitField('dst_n2_id', 0, 160),
                    XByteField('dst_n2_edge0', 0),
                    XByteField('dst_n2_edge1', 0),
                    XByteField('dst_n2_edge2', 0),
                    XByteField('dst_n2_edge3', 0),
                    XBitField('dst_n3_xid_type', 0x00, 32),
                    XBitField('dst_n3_id', 0, 160),
                    XByteField('dst_n3_edge0', 0),
                    XByteField('dst_n3_edge1', 0),
                    XByteField('dst_n3_edge2', 0),
                    XByteField('dst_n3_edge3', 0),
                    XBitField('dst_n4_xid_type', 0x00, 32),
                    XBitField('dst_n4_id', 0, 160),
                    XByteField('dst_n4_edge0', 0),
                    XByteField('dst_n4_edge1', 0),
                    XByteField('dst_n4_edge2', 0),
                    XByteField('dst_n4_edge3', 0),
                    XBitField('src_n1_xid_type', 0x00, 32),
                    XBitField('src_n1_id', 0, 160),
                    XByteField('src_n1_edge0', 0),
                    XByteField('src_n1_edge1', 0),
                    XByteField('src_n1_edge2', 0),
                    XByteField('src_n1_edge3', 0) ]

bind_layers(Ether, XIP, type=0XC0DE)
```