Universidade Federal do Espírito Santo

Diego Rossi Mafioletti

Crops – Uma Proposta de Comutador Programável de Código Aberto para Prototipação de Redes

Vitória-ES
2015

Diego Rossi Mafioletti

Crops – Uma Proposta de Comutador Programável de Código Aberto para Prototipação de Redes

Dissertação para obtenção do grau de Mestre apresentada à Universidade Federal do Espírito Santo

Área de concentração: Ciência da Computação

Orientador: Magnos Martinello

Vitória-ES
2015

Mafioletti, Diego R.
Crops – Uma Proposta de Comutador Programável de Código Aberto para Prototipação de Redes
88 páginas
Dissertação (Mestrado) - Universidade Federal do Espírito Santo.

1. SDN
2. OpenFlow
3. NOS
4. Firmware
5. Tabelas de Grupos

Diego Rossi Mafioletti

Crops – Uma Proposta de Comutador Programável de Código Aberto para Prototipação de Redes

Dissertação para obtenção do grau de Mestre apresentada à Universidade Federal do Espírito Santo. Área de concentração: Ciência da Computação.

Aprovada em 01/09/2015
Pela comissão organizadora

Prof. Dr. Magnos Martinello
Orientador

Prof. Dr. Rodolfo da Silva Villaça
Co-Orientador

Prof. Dr. José Gonçalves Pereira Filho
Examinador

Prof. Dr. Sidney Cunha de Lucena
Examinador

Dedico este trabalho à minha família e amigos.

Agradecimentos

Agradeço ao meu orientador, Magnos, por ter me aceitado como seu orientando, mesmo sabendo de minhas limitações de horário, compromissos com terceiros e deslocamento complicado. Muito obrigado pelas tantas revisões de texto, motivações e ensinamentos passados durante meus estudos; ao meu co-orientador, Rodolfo, por suas dicas tão pontuais e revisões em artigos.

Aos meus amigos do LabNerds, em especial Alextian e Eros, por estarem sempre dispostos à colaborar com seu tempo, que também era curto, e por isso precioso para todos nós; muito obrigado pelas ideias, revisões, e tantas outras coisas que passamos neste período de estudos. Agradeço à meus pais, pelo apoio e incentivo para minha carreira de estudos, desde o início.

Finalizo agradecendo minha família, esposa e companheira, Schaira, por todos os momentos em que necessitei de ajuda, por estar sempre comigo, dando-me incentivo, encorajamento, tempo, e acima de tudo, pela paciência e amor que recebo todos os dias; e minha filha, Isabela, que mesmo sendo criança, prestava auxílio só de estar ao meu lado durante minhas tarefas; e, às duas, por serem os maiores incentivos para estar aqui hoje! Amo vocês!

Obrigado senhor, por tudo de bom que tenho em minha vida.

Resumo

Redes Definidas por *Software* (SDN) prometem um caminho tecnológico para fortalecer os usuários com habilidade de inovar em suas redes. Porém, a diversidade de *switches* de rede que suportam SDN ainda é um obstáculo para os engenheiros de rede dispostos a desenvolver aplicações inovadoras devido à implementação do *hardware* ser fechada e proprietária. Essa diversidade implica em desafios significantes no controle dos *switches* SDN e o desenvolvimento de aplicações SDN de alta performance, o que não contempla um dos pilares de SDN: permitir a pesquisa e inovação em redes de computadores. A proposta deste trabalho é explorar o limite de alguns equipamentos *commodities* de rede, tal como o *switch* Mikrotik RouterBoard, no qual seu *firmware* original é substituído pelo OpenWRT, uma distribuição baseada no GNU Linux, juntamente com o Open vSwitch (OvS), um *switch* virtual de código aberto independente de *hardware*, para a criação de um novo ambiente para experimentação em redes de computadores. Como prova de conceito, o protótipo foi implementado em *switches* comerciais de baixo custo, conduzindo experimentos com o objetivo de analisar certas características do protocolo OpenFlow portadas nestes equipamentos, como quantidade máxima de entradas na tabela de fluxo, a vazão de dados possível utilizando tamanhos variados de pacotes de rede, comparando seus resultados com a implementação original disponibilizada pelo fabricante e a influência da utilização da CPU do *switch* no resultado. Por fim, foi construído um *switch* com balanceamento de carga estocástico utilizando tabelas de grupos, um recurso disponível no OpenFlow a partir da versão 1.2, que somente foi possível graças à natureza de código fonte aberto das ferramentas escolhidas. Esta combinação de plataformas abertas representa um passo natural no desenvolvimento, implementação, e avaliação de aplicações SDN.

Palavras-chave: SDN, OpenFlow, NOS, *Firmware*, Tabelas de Grupos

Abstract

Software Defined Network (SDN) promises a technological path to empower users with the ability to innovate in their networks. However, the diversity of network switches supporting SDN are still roadblocks for network engineers willing to develop innovative applications due to the closed and proprietary hardware implementation. This diversity leads to significant challenges in the control of SDN switches and the development of high performance SDN applications, hampering the core proposal of SDN: to enable fast innovation in real networks. The proposal of this work is to explore the limits of some commodity network hardware, such as Mikrotik RouterBoard switch, in which its proprietary firmware was replaced by the OpenWRT, a distribution based on GNU Linux, together with the Open vSwitch (OvS), a hardware-agnostic open source virtual switch, to create a new environment for experimentation in computer networking. As proof of concept, the prototype was implemented in low cost commercial switches, conducting experiments in order to analyze certain features of the OpenFlow protocol ported in these equipment, such as maximum number of entries in the flow table, the data plane performance using different sizes of network packets, comparing their results with the original implementation provided by the manufacturer and the influence of switch CPU utilization in the result. This combination of open platforms represent a natural step in the development, deployment, and evaluation of SDN applications.

Keywords: SDN, OpenFlow, NOS, Firmware, Group Tables

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ARM Advanced RISC Machine

CPqD Centro de Pesquisa e Desenvolvimento em Telecomunicações

IPK Itsy Package

MIPS Microprocessor without Interlocked Pipeline Stages

NAT Network Address Translation

NDN Named Data Networking

NERDS Núcleo de Estudos em Redes Definidas por Software

NOS Network Operating System

OF OpenFlow

OS Operating System

OVS Open vSwitch

PowerPC Performance Optimization With Enhanced RISC – Performance Computing

RISC Reduced Instruction Set Computing

SDN Software Defined Networking

SoC System on Chip

TCAM Ternary Content Access Memory

Sumário

Capítulo 1

Introdução

Presenciamos um cenário crescente de popularização e de diversificação dos usos das tecnologias de informação e comunicação em todos os aspectos da vida humana. Não há atividade da sociedade moderna que não tenha sido afetada pela revolução digital. Da educação ao sistema de saúde; da mídia aos negócios; dos procedimentos burocráticos do Estado às formas de contatos sociais entre indivíduos, tudo de alguma forma evoluiu ou foi totalmente modificado pela tecnologia.

No pilar desta evolução encontramos as Redes de Computadores que permitem a conexão física e lógica nas quais trafegam todos os dados necessários para a comunicação. É possível afirmar que a maioria das atividades da sociedade moderna de alguma forma atravessa uma ou mais Redes o que a torna essencial para a manutenção deste crescimento.

Com a constante preocupação em oferecer suporte que atenda às demandas da sociedade moderna, as Redes receberam um massivo investimento, seja da comunidade acadêmica ou da indústria em uma nova abordagem. Essa nova abordagem permite a implantação de Redes com maior grau de programabilidade, facilitando a adoção de novas tecnologias de maneira gradual e controlada. A tal conceito deu-se o nome de Redes Definidas por *Software*, ou *Software Defined Networking* (SDN).

Nesta seção, aspectos introdutórios são apresentados, de forma a delimitar o assunto e simplificar o entendimento do trabalho proposto. Os objetivos (geral e específico), assim como as principais contribuições e estrutura da dissertação também são destacados.

1.1 Contexto

As Redes Definidas por *Software* surgiram como um promissor conceito para o projeto de novas arquiteturas para a Internet. SDN é uma proposta baseada na separação dos planos de dados e de controle em uma interface uniforme independente de fornecedor para o mecanismo de encaminhamento (ex: OpenFlow [McKeown et al. 2008]) e em um plano de controle logicamente centralizado. O plano de dados consiste nos elementos ativos da rede, que implementam o encaminhamento de pacotes baseado em informações contidas

em suas tabelas de fluxos. Ao plano de controle é delegado o controle global da rede, que inclui a lógica de encaminhamento, as funções de coordenação e sinalização, e a gestão de todos os equipamentos do plano de dados. Este novo paradigma torna mais simples a construção de novas arquiteturas de rede de computadores, já que o controle passa a ser centralizado.

O OpenFlow é um padrão aberto que permite a programação dos elementos ativos da rede, tais como roteadores, *switches* ou pontos de acesso sem fio. Trata-se de uma proposta pragmática com grande potencial para suportar o grau de programabilidade que as SDN necessitam [Kotronis et al. 2012].

A construção e a validação de protótipos em SDN têm acontecido essencialmente por meio de ferramentas de emulação, como o Mininet [Lantz et al. 2010]. Essas ferramentas viabilizam uma rápida prototipação utilizando apenas um computador através de primitivas de virtualização do sistema operacional, tais como processos e *namespaces* de rede. Com essas primitivas, as ferramentas de emulação permitem, rapidamente, criar, interagir e customizar protótipos de redes utilizando *switches* OpenFlow, independentemente do *hardware* a ser utilizado.

Contudo, o ambiente emulado não garante um alto grau de fidelidade, quando comparado a um ambiente real, devido, principalmente, à restrições impostas pelo *hardware* presente em um *switch* de rede. Além disso, pequenas alterações nestes ambientes afetam drasticamente a latência e a taxa de transferência nas redes, tornando o uso do ambiente emulado um grande limitador para os projetistas em SDN validarem as aplicações e terem garantias de como elas irão se comportar com *switches* OpenFlow [Huang et al. 2013].

Os equipamentos habilitados com Openflow têm sido projetados pelos fabricantes como um sistema fechado, no qual o projetista de redes torna-se dependente da implementação do fabricante (ex: Mikrotik, Pica8). Se a inovação do projetista requer modificação no plano de dados, por exemplo, encaminhar pacotes para uma porta com uma certa probabilidade, então o equipamento não permite tal operação e impede sua validação experimental. Esses fatores limitam significativamente o potencial de inovação na arquitetura SDN, tanto pela restrição imposta pelo modelo de encaminhamento, quanto pelo sistema fechado pelo fabricante.

A ideia de criar plataformas abertas para experimentação em SDN/OpenFlow tem sido um grande motivador para permitir aos projetistas de redes inovar livremente e, consequentemente, testar com confiança seus projetos em *hardware* de prateleira não especializado. E neste contexto, é desejável que essa plataforma seja programável pelo projetista, tenha um desempenho aceitável e custo relativamente baixo.

Já a heterogeneidade de diferentes dispositivos SDN no que diz respeito aos planos de dados e de controle pode nos levar a um comportamento inesperado de aplicações de rede, bem como problemas de performance. O conhecimento dos limites de performance dos dispositivos SDN pode aprimorar a habilidade dos engenheiros de rede de computadores

no projeto e desenvolvimento de aplicações SDN em um ambiente de rede heterogêneo [Casado et al. 2012, Lazaris et al. 2014]. Como exemplo podemos citar uma aplicação SDN de roteamento, que pode selecionar caminhos que evitam um certo equipamento de baixa capacidade em uma rede para provisionar serviços de alta vazão e baixa latência.

1.2 Objetivos

1.2.1 Objetivo Geral

Propor um comutador de código aberto e programável para prototipação de Redes utilizando equipamentos de baixo custo (Crops).

1.2.2 Objetivos Específicos

- Estudar os comutadores implementados em *software* para SDN;
- Descrever os sistemas embarcados disponíveis para comutação de pacotes;
- Identificar os trabalhos relacionados com o foco da pesquisa;
- Explicar a proposta do Crops – comutador de código aberto programável para prototipação em redes;
- Descrever o desenvolvimento e a implementação da proposta do comutador de código aberto programável para prototipação em redes;
- Avaliar a proposta com diferentes recursos e arranjos.

1.3 Contribuição

Esse trabalho apresenta uma proposta de comutador de código aberto programável para prototipação em SDN. Essa proposta utiliza equipamentos não especializados de prateleira para demonstrar a viabilidade de habilitar o protocolo OpenFlow em equipamentos reais, permitindo assim que novas características possam ser incorporadas pelo projetista de rede em um cenário mais realista.

Um estudo usando comutadores Mikrotik RouterBoard foi realizado, através da implementação de um *switch* estocástico de prateleira, capaz de fazer um balanceamento de carga através das portas do equipamento de modo randômico. Métricas de desempenho nos planos de dados e controle também são apresentados por meio de um comparativo entre a proposta de código aberto e uma plataforma fechada existente usando *hosts* físicos.

No ambiente desta dissertação destacam-se as recentes publicações do autor:

1. SPALLA, E. ; MAFIOLETTI, D. ; LIBERATO, A. ; Rothenberg C. ; CAMARGOS, L. ; VILLACA, R. ; MARTINELLO, MAGNOS . Estrategias para Resiliencia em SDN : Uma Abordagem Centrada em Multi-Controladores Ativamente Replicados. In: XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2015, Vitoria-ES. Anais do SBRC, 2015.

2. LIBERATO, A. ; MAFIOLETTI, D. ; SPALLA, E. ; VILLACA, R. ; MARTINELLO, M. . Avaliação de Desempenho de Plataformas para Validação de Redes Definidas por Software. In: Wperfomance, 2014, Brasília. WPerformance - XIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação, 2014.

3. LIBERATO, A. ; SPALLA, E. ; MAFIOLETTI, D. ; VILLACA, R. ; MARTINELLO, M. . Balanceamento de Carga em SDN Provido por Comutadores Estocásticos de Prateleira. In: IV Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF), 2014, Florianópolis. Simpósio Brasileiro de Redes de Computadores - SBRC, 2014.

4. MARTINELLO, M. ; VILLACA, R. ; LIBERATO, A. ; MAFIOLETTI, D. ; SPALLA, E. ; CABELINO, R. . Plataformas Abertas para Infraestruturas Definidas por Software : Projeto, Implementação e Experimentos. In: Infraestruturas Definidas por Software (IDS) no WRNP, 2014, Florianópolis. Workshop da Rede Nacional de Ensino e Pesquisa (WRNP), 2014.

5. SPALLA, E. ; MAFIOLETTI, D. ; LIBERATO, A. ; VILLACA, R. ; MARTINELLO, M. . Resiliência no Plano de Controle para Redes Definidas por Software. In: IV Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF), 2014, Florianópolis. Simpósio Brasileiro de Redes de Computadores - SBRC, 2014.

Cada publicação contribuiu para a construção de um currículo teórico e prático, permitindo incorporar o conhecimento adquirido na proposta.

1.4 Estrutura

Este trabalho está estruturado como segue:

- O Capítulo 2 apresenta os conceitos básicos, características e recursos referentes às Redes Definidas por *Software*. Também nesta seção são debatidos os principais encaminhadores virtuais baseados em *software*. Por último, são descritos os sistemas embarcados para comutação de pacotes.

- O Capítulo 3 discute os trabalhos relacionados, analisando-os a fim de reforçar a justificativa e a relevância do trabalho que está sendo proposto, além de apresentar características e métodos que serviram de fonte de inspiração para o presente trabalho.
- O Capítulo 4 evidencia a proposta de um comutador de código aberto programável para prototipação em redes. Também são descritos os blocos funcionais necessários para a construção da proposta.
- O Capítulo 5 descreve os procedimentos e detalhes da implementação para o desenvolvimento da proposta.
- O Capítulo 6 expõe os resultados de um Testbed experimental utilizando a proposta.
- Finalmente, o Capítulo 7 traz as conclusões e as perspectivas dos trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo tem o objetivo de apresentar aspectos teóricos importantes para a compreensão do trabalho. São abordados os principais conceitos sobre as tecnologias de Redes Definidas por *Software* (SDN) e OpenFlow, itens sobre os quais o trabalho é fundamentado. Além disso, o capítulo pretende fornecer a base necessária sobre sistemas embarcados para comutação de pacotes e a portabilidade do sistema operacional Linux sobre esses sistemas, considerados indispensáveis para a contextualização do trabalho apresentado.

O Capítulo está organizado da seguinte forma: a Seção 2.1 discute os conceitos de Redes Definidas por *Software*, sua divisão em plano de controle e plano de dados, e sua realização utilizando OpenFlow através de comutadores virtuais, tecnologia avaliada pela proposta implementada; a Seção 2.2 introduz o conceito de *switches* virtuais, citando exemplos e opções mais utilizadas na atualidade; a Seção 2.3 explica a modificação de equipamentos não especializados para funcionamento com SDN/OpenFlow; e, por fim, a Seção 2.4 apresenta algumas considerações sobre o capítulo.

2.1 Redes Definidas por *Software*

Comparando as redes de computadores de 20 anos atrás com as atuais, fica claro que um longo caminho foi percorrido e muito se evoluiu em termos de capacidade de transmissão, confiabilidade, segurança e área de abrangência. As tecnologias de camada física evoluíram proporcionando comunicação de alta capacidade. Os dispositivos de rede melhoraram seu poder computacional, proporcionando assim o surgimento de uma grande quantidade de aplicações de rede inovadoras. No entanto, a rede em sua estrutura não passou por muitas mudanças.

Na infraestrutura existente, tarefas que compõem a funcionalidade geral da rede, como roteamento ou decisões de acesso são delegadas a equipamentos fechados de vários fabricantes diferentes, cada um executando seus próprios *firmwares*.

Esta base bem estabelecida de equipamentos com *softwares* proprietários que compõem toda a infraestrutura de rede não abre muito espaço para novas idéias, como pesquisa de

novos protocolos de roteamento, que precisam ser testados em larga escala em redes reais [Sherwood et al. 2010].

A ausência de flexibilidade no controle do funcionamento interno dos equipamentos, assim como o alto custo da infraestrutura vigente, são barreiras para a evolução das arquiteturas e para a inovação decorrente da oferta de novos serviços e aplicações de rede.

A partir das diversas propostas que visavam a centralização da inteligência de controle da rede, tais como redes ativas [Tennenhouse and Wetherall 1996], o modelo 4D [Greenberg et al. 2005] e o Ethane [Casado et al. 2007], surgiu a arquitetura de Redes Definidas por *Software* (SDN – *Software Defined Networks*).

Um dos maiores benefícios da arquitetura SDN é a divisão do plano de controle e plano de dados. Quando um pacote é recebido por um *switch* SDN, algumas informações do cabeçalho do pacote são extraídas e uma operação de busca (*lookup*) é realizada na **Tabela de Fluxo**.

A Tabela de Fluxo é responsável por armazenar regras, ações e contadores referentes a cada fluxo. Essas entradas podem ser interpretadas como decisões em cache (*hardware*) do plano de controle (*software*), sendo, portanto, a mínima unidade de informação no plano de dados da rede [Rothenberg et al. 2010].

Se houver uma entrada correspondente (regras) nessa Tabela, a ação associada para essa entrada é executada. Para descartar ou encaminhar o pacote para uma porta de saída (ou para um grupo de portas de saída), existem algumas ações que podem ser tomadas pelo *switch*. Se não houver nenhuma entrada correspondente, o pacote é copiado e enviado para o controlador no plano de controle. Esse evento é chamado de *packet-in*.

Um *switch* SDN puro não implementa nenhum protocolo de roteamento ou descoberta para popular sua Tabela de Fluxo; esse trabalho é feito por um controlador que opera o plano de controle de uma rede SDN. As entradas na Tabela de Fluxo somente podem ser instaladas, removidas ou modificadas por um controlador. Uma interface padrão de comunicação entre o controlador e o *switch* é necessária para habilitar o encaminhamento de pacotes para o plano de controle, assim como instalação, remoção ou modificação de entradas na Tabela de Fluxo. O protocolo OpenFlow é a primeira tecnologia difundida para implementação de SDN, permitindo a programabilidade do plano de encaminhamento de dispositivos de rede (plano de dados).

Como resultado dessa arquitetura, há algumas penalidades na performance do OpenFlow quando eventos *packet-in* são gerados. A latência da rede entre o *switch* e o controlador é um aspecto preocupante, e é desejado que seja reduzida a quantidade de eventos *packet-in*, adicionando ou removendo entradas proativamente na Tabela de Fluxo. A outra opção seria aumentar o tamanho da Tabela de Fluxo, a fim de acomodar um grande número de entradas; mas nem sempre é possível aumentar o tamanho desta tabela, devido a restrições e limitações de *hardware* na arquitetura de um *switch* de rede.

Ainda sobre latência no OpenFlow, normalmente um pacote sofre influência do tempo

de propagação e do tempo de encaminhamento quando precisa percorrer um caminho na rede. Tempo de propagação depende somente da propriedade física do meio de transmissão, e está fora do escopo desse trabalho. O tempo de encaminhamento é o tempo que o pacote gasta dentro de um *switch* e depende de várias funções aplicadas ao pacote. No geral, o tempo de comutação em um *switch* OpenFlow possui três componentes que são detalhados na sequência: Busca, Encaminhamento e Controle [Dürr and Kohler 2014].

- Busca (*Lookup*): quando um *switch* recebe um pacote em uma de suas portas de entrada, ele faz a busca por uma correspondência em sua Tabela de Fluxo, a fim de determinar o que fazer com o pacote. Essa função é normalmente efetuada por um ASIC (*Application Specific Integrated Circuit* ou Circuito Integrado de Aplicação Específica), analisando partes do cabeçalho do pacote.

- Encaminhamento (*Forwarding*): um pacote que foi encontrado sua correspondência na Tabela de Fluxo é transferido através do sistema interno de encaminhamento, desde a porta de entrada até a porta de saída. Se a porta de saída está transmitindo um outro pacote, o novo pacote é colocado na fila de saída. O tempo que o pacote vai ficar na fila depende do tempo do outro pacote para atravessar a mesma porta de saída, e da prioridade atribuída a esse tráfego. Em geral, o tempo de encaminhamento é maior que o tempo de busca.

- Controle (*Control*): se não há uma correspondência para o pacote na Tabela de Fluxo, ou se a ação configurada para o mesmo é "enviar ao controlador", o mecanismo do *switch* de rede encapsula parte ou o pacote inteiro em uma mensagem de controle *packet-in*, e o envia para o controlador.

O padrão do protocolo OpenFlow especifica somente as mensagens e interfaces para a comunicação entre o controlador e o *switch* [ONF 2012]. A implementação e especificação dos dispositivos são de responsabilidade dos fabricantes, baseados em seu custo e limitações de tecnologia. A diversidade de implementações conduz a diferenças significantes na performance e capacidades de vários *switches*, que acabam impactando na inovação a nível de aplicação SDN.

Uma das características principais de um *switch* SDN é o *design* da Tabela de Fluxo. O tamanho dessa tabela, a política de *cache* e os algoritmos para conferência de fluxos têm influência direta na taxa de transferência e latência alcançados por um *switch* SDN. A Figura 2.1 mostra uma arquitetura típica de um *switch* SDN. O controlador é externo ao *switch*. Os componentes **Comunicação**, **Sistema Operacional** e **Plano de Dados** são parte do *hardware* do *switch*, e são os principais componentes que interessam nesse trabalho. O componente **Comunicação** é responsável pela comunicação entre o *switch* e o controlador, e implementa o padrão OpenFlow, ou qualquer outra solução equivalente.

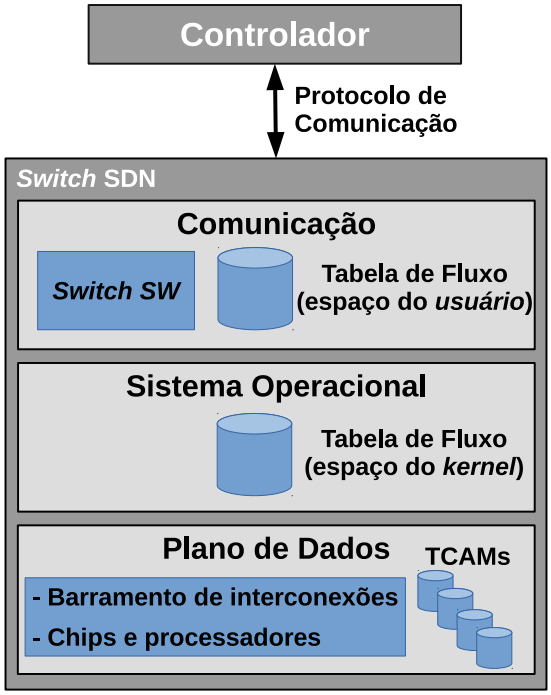


Figura 2.1: Uma arquitetura típica de um *switch* SDN.

Tabelas de Fluxo em *software* também podem ser implementadas no componente **Comunicação**. O componente **Sistema Operacional** é responsável pela conversão de alto nível das interfaces (como as mensagens OpenFlow) para a linguagem de baixo nível de máquina do *hardware* do dispositivo de rede. Ele também é responsável pela implementação do HAL (*Hardware Abstraction Layer*) para facilitar o desenvolvimento de *drivers* em nível de comunicação. Frequentemente, em *switches* de *software*, a Tabela de Fluxo em espaço do *kernel* é também implementada como um componente do Sistema Operacional, para conseguir melhor performance quando comparada à Tabela de Fluxo no espaço de usuário. A Tabela de Fluxo como um componente do Sistema Operacional geralmente possui um espaço de memória muito limitado, restringindo assim a quantidade máxima de entradas que podem ser instaladas nessa Tabela de Fluxo.

O componente **Plano de Dados** implementa o barramento de interconectividade entre as portas. No caso dos *switches* de *hardware*, a Tabela de Fluxo pode também ser implementada nesse componente, usando TCAMs (*Ternary Content-Addressable Memory*) [Yun 2012]. Uma TCAM possui um mecanismo de busca de alta velocidade, capaz de verificar a correspondência de todas as entradas na Tabela de Fluxo em um único ciclo de *clock* do processador. Porém, TCAMs não são usadas em todos os *switches* de *hardware*, por conta de seu alto custo e consumo de energia.

Casado [Casado et al. 2012] apresenta características desejáveis para SDN, separando-as em características de *hardware* e de *software*. O *hardware* deve ser simples, de custo acessível, fácil de operar e independente de fabricante, evitando que os usuários sejam obrigados a usar equipamentos de um único fornecedor. Também deve suportar inovações

futuras, evitando atualizações desnecessárias. Com relação ao *software*, Casado entende que ele deve ser flexível, estruturado e capaz de suportar uma ampla variedade de requisitos (isolação, virtualização, engenharia de tráfego, controle de acesso, entre outros). Também deve ser modular e expansível, permitindo inclusão e alteração de módulos.

2.1.1 Plano de Controle

O plano de controle de uma rede SDN é a abstração que encapsula os conceitos relacionados ao controle de uma rede de computadores convencional. Com essa nova arquitetura, em que o controle ou a inteligência da rede é desacoplada da camada física dos equipamentos, isto é, do plano de dados, surge o sistema operacional de redes. Basicamente, os sistemas operacionais fornecem um mecanismo para simplificar a interação do plano de controle com o plano de dados, e uma interface para que aplicações possam ser desenvolvidas. Por exemplo, um projetista de redes ao escolher uma plataforma *network OS (Operating System)*, passa a ter uma visão de alto do nível do plano de controle. Desta forma, programar uma rede SDN torna-se uma tarefa muito parecida como desenvolver uma aplicação em uma linguagem de programação qualquer.

Do mesmo modo que em outras arquiteturas, nas quais existem diversos sistemas operacionais, em SDN não é diferente. Controladores são desenvolvidos cada um com características e objetivos bastante variados, cobrindo desde o controle de ambientes para experimentações acadêmicas, até redes de grande porte. A seguir, listamos alguns controladores e suas principais características.

NOX [Gude et al. 2008] é um controlador OpenFlow desenvolvido inicialmente pela Nicira® e então, a partir de 2008, disponibilizado para a comunidade. A interface de programação disponível para criar aplicações, assim como a linguagem utilizada no desenvolvimento do controlador em questão, foi o C++. Uma das principais características deste controlador é o alto desempenho. Atualmente, possui suporte à versão 1.0 do OpenFlow, entretanto existe uma versão modificada pelo CPqD (Centro de Pesquisa e Desenvolvimento em Telecomunicações) que suporta parcialmente a versão 1.3 [Fernandes 2013]. A Figura 2.2 apresenta a arquitetura do controlador NOX.

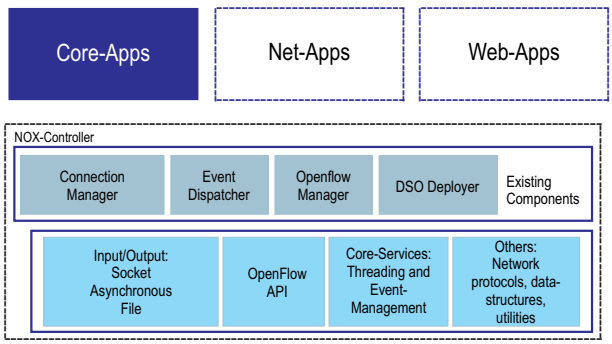


Figura 2.2: Organização do controlador NOX [Rao 2015a].

A partir do controlador NOX surgiu um novo projeto, cujo objetivo é prover uma interface mais simples para controladores SDN. Desenvolvido em Python, o controlador POX [POX 2013] é uma versão do controlador NOX tradicional. Este controlador normalmente é utilizado como uma alternativa ao NOX em experimentos e prototipação, já que possui uma interface mais amigável.

OpenDayLight [OpenDaylight 2013] é um projeto *open source* cujo desenvolvimento tem participação de grandes empresas como Cisco®, Citrix®, Microsoft®, IBM®, dentre outras. O principal objetivo da comunidade é acelerar o processo de popularização do uso de SDN, e como parte do projeto, disponibilizam a plataforma para o desenvolvimento de aplicações em SDN.

O OpenFlow é um dos padrões suportados pelo OpenDaylight, e atualmente tem suporte às versões 1.0 e 1.3 do OpenFlow. A Figura 2.3 ilustra a organização do controlador OpenDayLight.

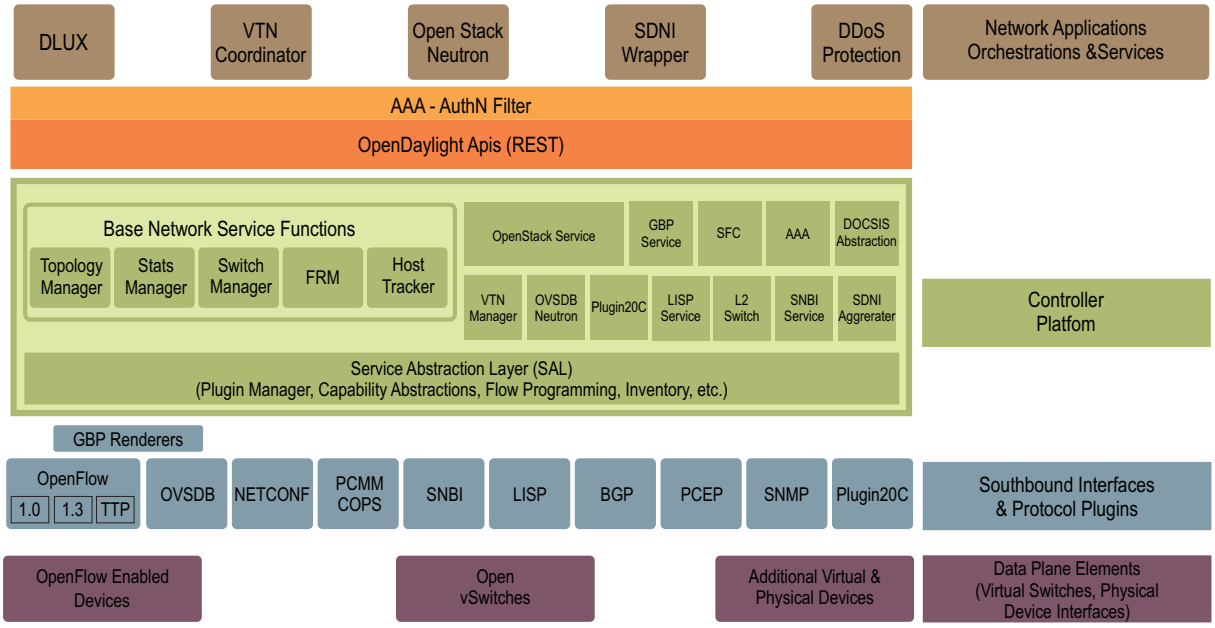


Figura 2.3: Organização do controlador OpenDayLight [OpenDayLight 2015].

Ryu [RYU 2014] é um controlador *open source* desenvolvido por um grupo japonês da NTT Lab's. O projeto é totalmente implementado em Python, e possui boa integração com outras ferramentas de rede, como por exemplo o OpenStack [Corradi et al. 2012]. Um dos principais pontos do projeto é o suporte a vários protocolos de *southbound*, como OpenFlow, NetConf e OF-Config.

O desenvolvimento constante por parte da comunidade permite que o controlador esteja atualizado com a versões mais recentes do Openflow. Atualmente, possui suporte total até a versão 1.4, e algumas implementações parciais da versão 1.5. A arquitetura do controlador japonês Ryu é apresentada na Figura 2.4.

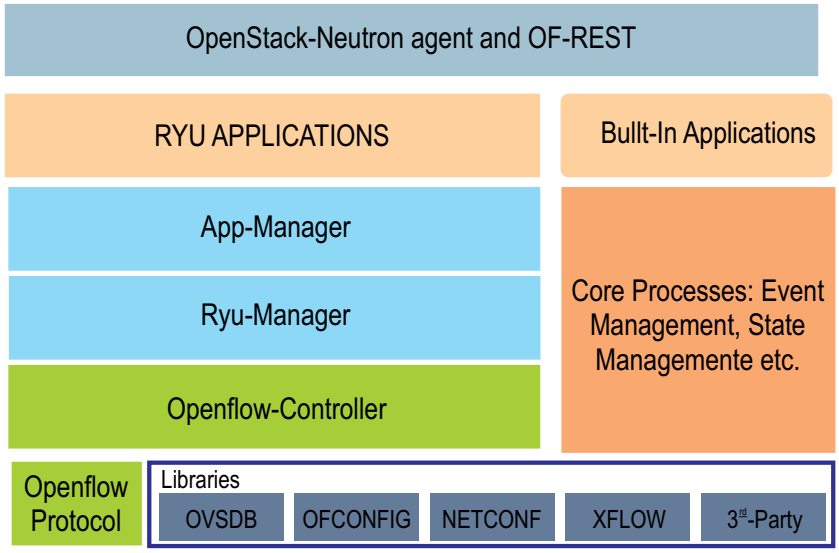


Figura 2.4: Organização do controlador Ryu [Rao 2015b].

A Tabela 2.1 sintetiza o resumo dos principais controladores, juntamente com informações adicionais, como a versão do OpenFlow suportado e a linguagem da API.

Tabela 2.1: Tabela com o resumo das características dos principais Controladores.

Controlador	**Suporte OpenFlow**	**Licença**	**Linguagem API**
Nox	1.0	GPLv3	C++
Pox	1.0	GPLv3	Python
OpenDayLight	1.0 e 1.3	EPL v1.0	Java
Ryu	1.0, 1.1, 1.2, 1.3 e 1.4	Apache 2.0	Python

2.1.2 Plano de Dados

O plano de dados ou plano de encaminhamento em uma Rede Definida por *Software* tem como função central o encaminhamento de pacotes, isto é, escoar o tráfego da rede. Logo, o plano de dados passa a ser representado pelos equipamentos de rede, sejam eles físicos ou virtuais, que possuam a capacidade de encaminhar pacotes.

De modo geral, os equipamentos de encaminhamento tradicionais possuem uma ou mais Tabelas de Fluxos, normalmente em porções de memória TCAM (*Ternary Content Access Memory*), onde são implementados serviços de *firewall*, NAT (*Network Address Translation*), *traffic shaping*, roteamento, entre outros. Vale lembrar que, no paradigma de rede tradicional, como a arquitetura é fechada, cada fabricante faz essas implementações de modo distinto. Em equipamentos OpenFlow, o conjunto de instruções do protocolo pode ser implementado em diferentes equipamentos, já que a tabela de fluxos é definida pelo protocolo.

Um *switch* OF (OpenFlow), também chamado *datapath*, pode ser representado em três partes:

1. Tabela de Fluxos: nessa tabela, cada fluxo é associado a uma ação, e assim, os pacotes que chegam ao equipamento são encaminhados de acordo com as regras instaladas na Tabela de Fluxos;

2. Canal Seguro: esse canal conecta o equipamento ao controlador remoto, utilizando uma conexão criptografada, e permite que mensagens e pacotes sejam enviadas entre o controlador e o *switch*;

3. O protocolo OpenFlow: o protocolo define um padrão aberto de comunicação entre o controlador e o *switch*, que permite a programação da tabela de fluxos do equipamento por meio de uma interface de alto nível. Assim, o projetista de redes não precisa se preocupar com as características do equipamento.

Entre os diversos *switches* OpenFlow, podemos destacar os *switches* físicos, que são equipamentos embarcados com a capacidade de encaminhar pacotes de rede, interligando *hosts* diretamente por suas portas físicas. Alguns *switches* comerciais já suportam o padrão OpenFlow, como é o caso do Pica8 [SDN 2015], Mikrotik RouterBoard RB750GL, RB2011iLS-IN, RB2011UAS-IN, RB450G, DATACOM DM4100 POE e Brocade ICX7450-24P.

Há também os *switches* virtuais, que são *softwares* capazes de encaminhar pacotes utilizando a pilha de rede do sistema operacional do *host* onde são executados. Os mais utilizados na atualidade, seja para prototipação ou até mesmo em ambientes de produção, são: CPqD Softswitch13 [CPqD 2014], Click Modular Router [Kohler et al. 2000], Open vSwitch (OvS) [Pfaff et al. 2009] e eXtensible openFlow DataPath Daemon (xDPd) [xDPd 2015].

A Tabela 2.2 apresenta uma visão sistemática dos comutadores citados.

Tabela 2.2: Tabela de especificações de alguns *switches* OF

	Switch	**Suporte OpenFlow**	**Licença**	**Encaminhamento**
Físico	Pica8	1.0, 1.1, 1.2, 1.3, 1.4	Copyright	Kernel+*hardware*
	RouterBoard	1.0	Copyright	Espaço usuário[1]
	Datacom	1.0	Copyright	*Hardware*
	Brocade	1.0, 1.1, 1.2, 1.3	Copyright	*Hardware*
Virtual	CPqD Softswitch13	1.0 e 1.3	BSD	Espaço usuário
	Click Modular Router	1.0[2]	BSD	Kernel
	xDPd	1.0, 1.1, 1.2, 1.3[3]	MPL 2.0	Espaço usuário+kernel
	Open Vswitch	1.0, 1.1, 1.2, 1.3	Apache 2.0	Espaço usuário+kernel

2.2 *Switches* Virtuais

Switches virtuais, ou *switches* em *software*, são aplicações construídas com a finalidade de encaminhar pacotes utilizando a pilha de rede do sistema operacional do *host* onde são executados, capazes de transformar um computador, ou uma porção dele (ex. uma máquina virtual) em um encaminhador de pacotes.

Os *switches* virtuais anexam uma interface interna presente na aplicação à uma interface de rede física (Ethernet) ou lógica (ex. veth, tap, entre outras) do sistema, criando assim uma ponte entre conexão interna/externa à aplicação. Com isso, os dados que seriam direcionados à uma porta física ou lógica são encaminhados para essa interface interna, permitindo assim o tratamento desses dados de acordo com as premissas do *switch* virtual, que pode variar desde ao simples encaminhamento de pacotes de rede como em uma *bridge*, à classificação dos pacotes através da análise de cabeçalhos dos protocolos de rede, elevando o grau de abstração no tratamento e encaminhamento de pacotes de dados na rede.

Dentre os atuais *switches* virtuais, algumas implementações destacam-se por apresentarem características distintas, que vão desde o uso restrito em ambiente acadêmico, provendo uma interface amigável ao desenvolvedor, e em alguns casos com elementos que simbolizam funções e atribuições de rede, até a utilização em ambiente de produção, entregando desempenho e flexibilidade ao plano de dados.

As principais implementações de *switches* virtuais serão discutidas abaixo, seguindo a sequência definida pela Tabela 2.2 exposta na Subseção 2.1.2: CPqD Softswitch13, Click Modular Router, xDPd e por fim, Open vSwitch.

2.2.1 CPqD SoftSwitch13

O SoftSwitch CPqD é voltado para a prototipagem de novas funcionalidades e novas versões do protocolo OF pelo seu código simples implementado em uma linguagem tradicional C. Essas características ajudaram a criar uma comunidade ativa e fizeram da ferramenta uma boa escolha tanto para o primeiro contato com OpenFlow como para experimentações mais avançadas [Fernandes and Rothenberg 2014].

Utilizando como base a implementação do *switch* virtual Ericsson TrafficLab Software Switch que suporta OpenFlow 1.1, foi aperfeiçoado para utilizar o protocolo OpenFlow versão 1.3. O encaminhador foi implementado somente em modo usuário no sistema

[1]Conforme resultados de experimentos realizados nesse equipamento, o desempenho sugere que o mesmo seja implementado em espaço de usuário, sendo impossível sua total constatação por se tratar de um software fechado

[2]Suporte a OpenFlow introduzido através do elemento OpenFlowClick (http://archive.openflow.org/wk/index.php/OpenFlowClick); versão alpha, com certas restrições de funcionalidade

[3]OpenFlow versão 1.3 é experimental no xDPd versão 0.4.x

operacional Linux, proporcionando uma experiência singular e trazendo funcionalidades que são opcionais nas especificações do protocolo OpenFlow 1.3, como tabelas de grupos **Select** e **Fast FailOver**, e a tabela **Meter**, recursos estes que não estão presentes em todos os demais *switches* virtuais disponíveis atualmente.

A Figura 2.5 ilustra os componentes e características do SoftSwitch13. O componente *Portas* são as portas do *switch* virtual, nas quais os pacotes trafegam. O componente *Netbee* é responsável pela análise de pacotes de rede e conversão dos campos de protocolo dos pacotes em estruturas de *match*, que serão enviadas posteriormente para as Tabelas de Fluxo. O componente *Oflib* converte as mensagens OF para comunicação interna com o *switch* virtual. Os componentes *Flow*, *Group* e *Meter Table* são as tabelas de fluxo, grupos e controle de fluxo, referentes ao protocolo OF.

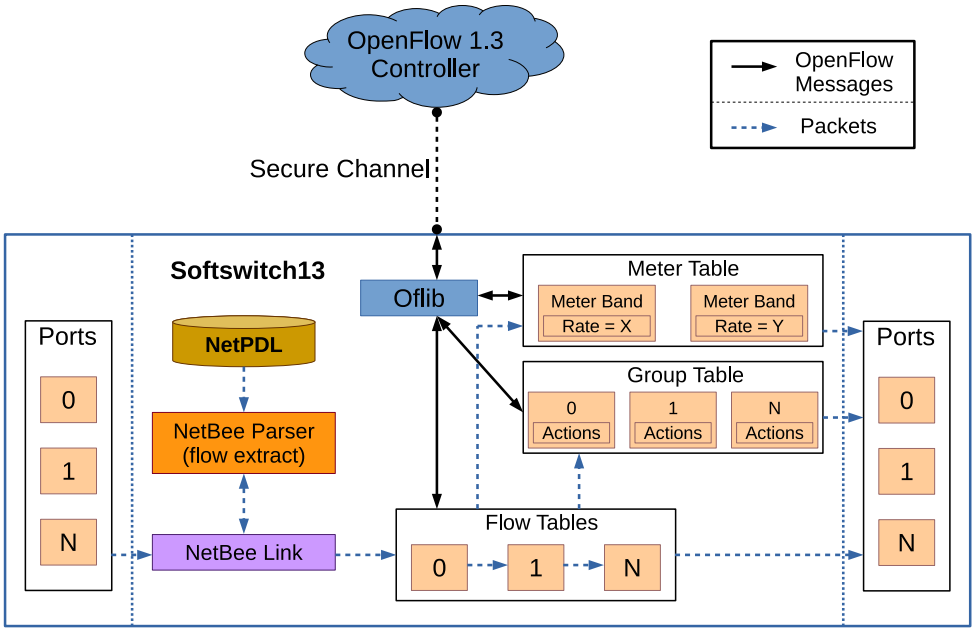


Figura 2.5: Arquitetura do SoftSwitch13 [Fernandes and Rothenberg 2014].

Por ser um projeto concebido para uso acadêmico, seu foco não é o desempenho, sendo praticamente inviável sua utilização em um ambiente de produção ou em equipamentos embarcados para prototipação.

2.2.2 Click Modular Router

O Click Modular Router [Kohler et al. 2000] é uma arquitetura de *software* para a construção de roteadores flexíveis e configuráveis. Um roteador Click é montado a partir de módulos de processamento de pacotes chamados elementos. Elementos individuais implementam funções de roteamento simples, como classificação de pacotes, filas, agendamento e interface com os dispositivos de rede. A configuração do roteador é um grafo direcionado com elementos nos vértices, e os pacotes fluem ao longo das extremidades do grafo. As

definições são escritas em uma linguagem declarativa que suporta abstrações ditadas pelo usuário.

A Figura 2.6 exibe como os elementos do Click são interconectados para a formação de um switch Ethernet funcional com suporte ao protocolo *spanning tree*. Apesar da funcionalidade de um *switch* ser responsabilidade do módulo **EtherSwitch**, os módulos **EtherSpanTree** e **Suppressor** são utilizados para evitar ciclos quando múltiplos *switches* são utilizados na mesma rede.

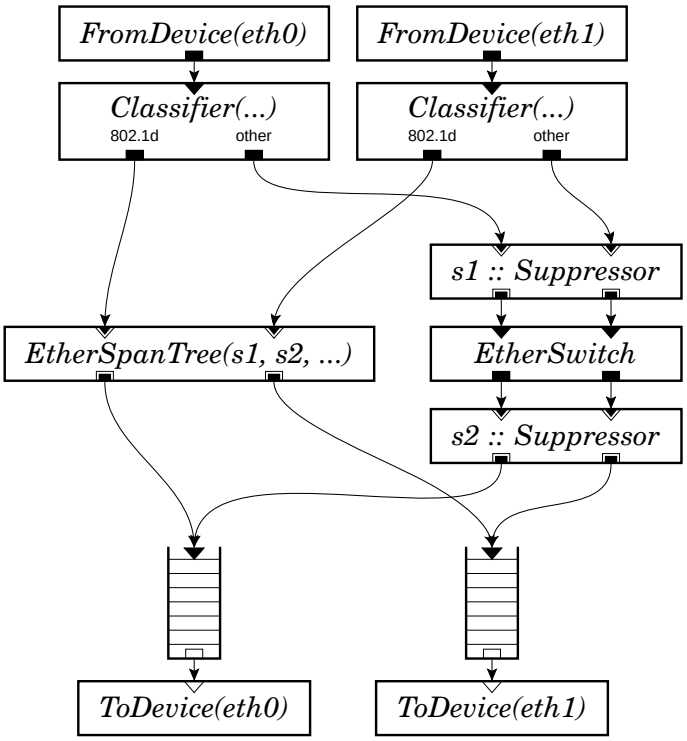


Figura 2.6: Composição de um *switch* Ethernet utilizando o Click [Kohler et al. 2000].

Contudo, o Click não especifica nenhuma interface de plano de controle. A interface de plano de controle está oculta por trás dos elementos individuais do Click [Kempf et al. 2011], o que torna difícil sua implementação em conjunto com o protocolo OpenFlow.

Apesar de existir uma proposta de um elemento com funcionalidades do OpenFlow para o Click [Mundada et al. 2009], o projeto atualmente encontra-se em versão *alpha*, e seu desenvolvimento inativo, congelando seu funcionamento apenas na versão 1.0 do protocolo. Há também restrições no que diz respeito ao seu funcionamento, dentre elas a mais impactante é o suporte à apenas um único plano de dados (*datapath*) definido estaticamente, sem a possibilidade de adição e remoção de portas durante sua execução, bem como desativação e ativação das mesmas sob demanda.

2.2.3 eXtensible openFlow DataPath Daemon (xDPd)

O eXtensible openFlow DataPath Daemon (xDPd) consiste em um *switch* virtual multi-plataforma, de código fonte aberto e suporte às versões 1.0, 1.1, 1.2 e 1.3[4] do protocolo OpenFlow. Foi construído com foco na performance e extensibilidade. Desse modo, o xDPd também pode ser considerado como um *framework* para construir elementos de encaminhamento de pacotes utilizando OpenFlow [xDPd 2015].

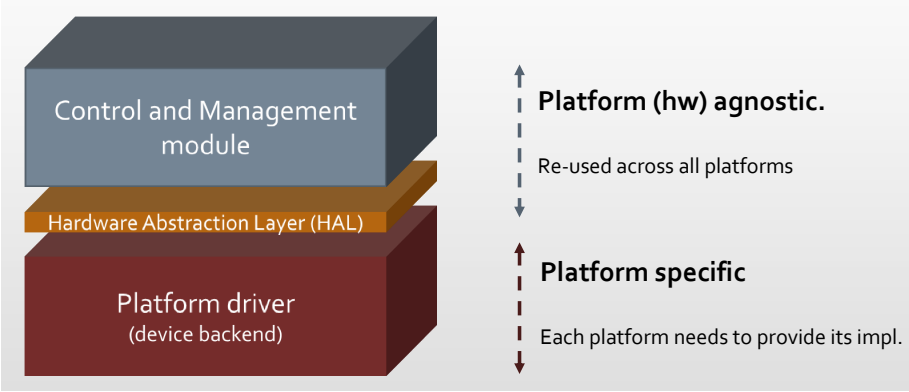


Figura 2.7: Arquitetura do xDPd [xDPd 2015].

Como descrito na Figura 2.7, a arquitetura do xDPd divide-se basicamente em três módulos:

1. Módulo de controle e gerenciamento, responsável pela lógica de encaminhamento em software (ou *software switch*)

2. Módulo HAL, ou camada de abstração de *hardware*, responsável pela API de integração do módulo de controle e gerenciamento com o módulo de *drivers* da plataforma

3. Módulo de drivers da plataforma, que pode ser puramente em *software*, como a pilha de encaminhamento do Linux ou Intel DPDK, híbrido, com algum tipo de aceleração em hardware e outros componentes em *software*, ou totalmente em *hardware*, como um ASIC.

Porém, até o momento da escrita deste documento, poucas plataformas de *hardware* dão suporte ao xDPd, limitando seu uso a um número limitado de equipamentos embarcados. Além disso, a disponibilidade do código fonte de cada plataforma está sujeita à licenças dos fornecedores dos *hardwares*, e nem todos os *drivers* das plataformas podem ser entregues como código fonte aberto.

2.2.4 Open vSwitch (OvS)

O Open vSwitch (OvS) é um *software switch* multi-camada registrado sob a licença Apache 2.0. Ele é projetado para viabilizar uma automação de rede massiva através de extensões

[4] O suporte a OpenFlow 1.3 é dado como experimental na atual versão 0.4.X do xDPd

programáveis, e ainda dar suporte as interfaces de gerenciamento padrões e protocolos como OpenFlow, NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, entre outros. O OvS pode operar tanto como um *software switch*, sendo executado dentro de um *hipervisor* ou um equipamento embarcado para comutação de pacotes de rede, quanto no controle da pilha de encaminhamento de um ASIC de um *switch* especializado [vSwitch 2014].

O ponto chave do Open vSwitch está em sua divisão em dois *datapaths*: um em modo usuário e outro em modo kernel. O *datapath* em modo usuário é responsável pela classificação dos pacotes de acordo com a entrada da Tabela de Fluxo, pela comunicação entre o plano de dados e o plano de controle, e pela criação de uma entrada de *cache* no *datapath* em modo kernel. O *datapath* em modo kernel do Linux é responsável por encaminhar os pacotes de um fluxo de dados diretamente no modo kernel, sem a necessidade de mudar para o *datapath* de modo usuário a cada novo pacote.

A Figura 2.8 descreve a integração entre os dois *datapaths*. O primeiro pacote de um fluxo resulta em uma falha (*miss*), e o módulo do kernel (*datapath* em modo kernel) direciona o pacote para o componente de espaço de usuário (*daemon* ovs-vswitchd), que cria uma entrada no *cache* da decisão de encaminhamento para os pacotes subsequentes (*hit*), diretamente via módulo do kernel.

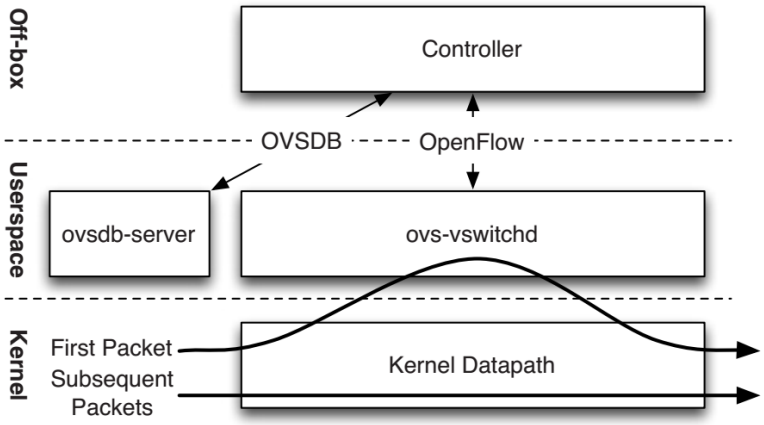


Figura 2.8: Os componentes e interfaces do Open vSwitch [Pfaff et al. 2015].

Este processo de *cache*, baseado em algoritmos avançados de classificação de pacotes, auxilia na redução do consumo de CPU, um recurso muito valioso em ambientes embarcados, resultando em um aumento na velocidade de encaminhamento de pacotes e realçando sua escolha para utilização em tais arquiteturas.

O OvS é extremamente portável, pois a maior parte do código é escrito em C, sendo assim independente de plataforma, e facilmente utilizado nos mais variados ambientes, como Linux, FreeBSD, NetBSD, Microsoft Windows, e igualmente adaptado para outras arquiteturas diferentes da tradicional x86, como as presentes em ambientes embarcados (MIPS, ARM, entre outros).

2.3 Sistemas Embarcados para Comutação de Pacotes

Um sistema embarcado consiste em um sistema baseado em microprocessador ou SoC (*System on Chip*), que é construído para controlar uma função ou um conjunto de funções [Heath 2002].

Esse tipo de sistema possui propriedades singulares quando comparados com os sistemas computacionais tradicionais, como baixo consumo de energia, tamanho reduzido, longos ciclos operacionais sem interrupção e baixo custo por unidade adquirida. Em contrapartida, os sistemas embarcados possuem limitações de recursos de processamento, e uma certa dificuldade para programação e interface com usuário.

Por sua versatilidade, são utilizados em diversas áreas, como automação residencial e industrial, no setor automobilístico, aviação, telecomunicações, redes de computadores, entre outros. Redes de computadores utilizam equipamentos embarcados conhecidos como roteadores (*routers*) e comutadores (*switches*), que também são conhecidos como **ativos de rede**.

Destacam-se nessa linha os equipamentos com arquitetura baseada em instruções RISC (Reduced Instruction Set Computing), como PowerPC (Performance Optimization With Enhanced RISC – Performance Computing), MIPS (Microprocessor without Interlocked Pipeline Stages) e ARM (Advanced RISC Machine).

Esses ativos de rede são gerenciados por um *firmware* que, basicamente, possui a função de rotear e encaminhar pacotes de rede através do uso de diferentes padrões de protocolos de comunicação de redes.

Um *firmware* é geralmente considerado como parte do *hardware*, pois reside dentro de um componente de *hardware*, tipicamente um dispositivo de armazenamento *flash* ou uma *ROM* [Heath 2002]. Ele é constituído por um micro sistema operacional, contendo ferramentas e funções que permitem o controle dos dispositivos presentes no equipamento embarcado de rede (portas físicas de conexão de rede, porta de gerenciamento, *chip* de comutação, memória, etc), e com característica de ser fortemente imutável, sendo que atualizações e novas funcionalidades somente são implementadas pelo fabricante do equipamento.

2.3.1 Linux em equipamentos embarcados para comutação de pacotes

Embora o termo "embarcado" seja comumente utilizado, não existe um *kernel* do Linux específico para aplicações embarcadas. Normalmente o que encontramos é o mesmo código-fonte do *Kernel* que foi construído para suportar diversos dispositivos, incluindo estações de trabalho e servidores, por exemplo. [Yaghmour et al. 2008] destaca que tal flexibilidade permite aos operadores e administradores a facilidade de configurar uma va-

riedade de recursos opcionais de acordo com sua utilização e necessidade. Isso torna o Linux o sistema operacional mais utilizados em sistemas embarcados.

Há um número grande de razões para escolha do sistema operacional Linux ao invés dos tradicionais sistemas embarcados. Entre elas, destacam-se:

- Qualidade e confiabilidade do código fonte: muitos programadores concordam que o *kernel* e outros projetos usados no sistema operacional Linux obedecem um certo critério de qualidade e confiabilidade. A razão para isso é o modelo de desenvolvimento de código aberto, que convida todos a contribuir para projetos, identificar problemas existentes, debater possíveis soluções e corrigir problemas efetivamente.

- Disponibilidade do código fonte: refere-se ao fato de que o código fonte do Linux, e de todas as ferramentas de compilação, estarem disponíveis sem quaisquer restrições de acesso ou custo, o que diretamente contrasta com os sistemas embarcados tradicionais.

- Suporte amplo a vários tipos de *hardware*: significa que o Linux suporta diferentes tipos de plataformas de *hardware* e dispositivos, e como a maioria desses *drivers* são desenvolvidos diretamente pela própria comunidade, não há risco do suporte dos mesmos serem descontinuados pelo fabricante.

- Padrões de protocolo de comunicação e *software*: que facilita a integração com ambientes de trabalho existentes (como uma rede Windows, por exemplo) e a portabilidade de *software* de outros sistemas "Unix-like" para o Linux.

- Ferramentas disponíveis: existe uma imensa variedade de ferramentas disponíveis para o Linux na atualidade, para praticamente todos os tipos de aplicação, e a grande maioria está disponível livremente na Internet.

- Suporte da comunidade: o amplo suporte da comunidade Linux é uma das maiores virtudes do sistema, sendo ele disponível via fóruns livres na Internet, ou listas de *e-mail*, de forma gratuita.

- Licenciamento: uma aplicação pode ser usada, modificada e redistribuída somente com a restrição de prover os mesmos direitos para seus destinatários, porém não implica na perda de controle sobre patentes e *copyrights* encorporados na aplicação que foi gerada.

- Independência do fornecedor: significa que você não precisa contar com nenhum fornecedor exclusivo para obter Linux ou para usá-lo.

- Custo: essencialmente, existem três componentes de custo embutidos na construção de um sistema embarcado tradicional, que são a configuração inicial de desenvolvimento (compra de licenças de desenvolvimento), ferramentas adicionais (compra de

ferramentas adicionais não adquiridas no componente anterior) e *royalties* de tempo de execução (pagos por unidade de dispositivos a serem usados). Com Linux, esse modelo de custo não existe: a maioria das ferramentas de desenvolvimento e componentes do sistema operacional são gratuitas, e suas licenças não permitem que sejam coletados nenhum *royalty* sobre seus componentes.

Conclui-se, baseado nos comentários anteriores, que o Linux é uma excelente opção de sistema operacional para equipamentos embarcados. No entanto, destacamos que tais vantagens também são exploradas por diversos fabricantes em seus dispositivos, como o que acontece com o Pica8 e o Mikrotik Routerboard.

Na próxima subseção descrevemos mais detalhes, em especial sobre o sistema de código fechado da MikroTik. Logo em seguida, descrevemos um sistema de código aberto comumente utilizado para sistemas embarcados.

2.3.2 *Firmware* de código fechado: MikroTik RouterOS

MikroTik RouterOS é o sistema operacional nativo dos equipamentos RouterBoard. É um sistema operacional autônomo baseado no *kernel* do Linux versão 3.3.5[5], criado para prover todos os recursos presentes no *hardware* da RouterBoard, como roteamento, *firewall*, gerenciamento de banda, ponto de acesso sem fio, servidor e/ou cliente VPN, entre outros.

Este sistema operacional possui várias interfaces de configuração e operação, que podem ser feitas desde via acesso ssh (*secure shell*), ou através da utilização de sua interface *web*, ou ainda por meio de um aplicativo de configuração chamado **Winbox**, que conta com uma interface gráfica intuitiva que auxilia nos procedimentos efetuados sobre o RouterOS.

Com o Winbox, também é possível instalar e remover funções do SO através da instalação e remoção de pacotes de funcionalidades pré-criados pelo fabricante (ex. funções de *wireless*, roteamento avançado, monitoramento, entre outros), atualizar o *firmware* e o gerenciador de inicialização (*RouterBoot*).

Apesar de suas facilidades, o RouterOS continua sendo um *firmware* de código fechado, permitindo apenas a configuração e ativação de funções de rede já existentes (ex. MPLS, OSPF, BGP, entre outros protocolos), ou instalações de pacotes estáticos, recusando novas funcionalidades além das criadas pelo desenvolvedor do sistema, sendo também limitadas pelo nível da licença adquirida no momento da compra do *firmware* ou do equipamento[6].

A partir de sua versão 6.0, o RouterOS possibilita a utilização do protocolo OpenFlow versão 1.0, através da instalação de seu respectivo pacote de funcionalidade. Por ser um *software* proprietário de plataforma fechada, o RouterOS não permite modificações e atualizações além do OpenFlow 1.0 padrão, disponibilizado pelo fabricante

[5] A partir do RouterOS versão 6.x

[6] Uma lista das licenças disponíveis e suas funcionalidades está disponível em http://wiki.mikrotik.com/index.php?title=Manual:License

[Liberato et al. 2014]. Por consequência de ser um *firmware* fechado, há uma forte restrição no que diz respeito a implementações de novas versões do protocolo OpenFlow e suas aplicabilidades, bem como modificações mais profundas no mecanismo de encaminhamento (ex. roteamento sem tabelas), restringindo o processo de inovação nas redes de computadores.

2.3.3 *Firmware* de código aberto: OpenWRT

O OpenWRT é uma distribuição GNU/Linux altamente extensível para dispositivos embarcados, dentre eles, o RouterBoard utilizado neste trabalho. Diferentemente dos demais *softwares* originais desses equipamentos, o OpenWRT foi construído a partir do zero para ser um sistema operacional completo e facilmente modificável, utilizando um *kernel* do Linux mais recente[7] que a maioria das outras distribuições [OpenWRT 2014].

O sistema operacional OpenWRT possui diversas características que o tornam propenso a ser empregado em sistemas embarcados, porém a que mais se destaca é o seu mínimo requisito de espaço de armazenamento e de memória RAM, sendo possível de ser portado em equipamentos com espaço de armazenamento reduzido (ex. equipamentos com apenas 4MB de memória flash), e com recursos de memória limitados (aproximadamente 8MB de memória RAM).

Outra característica importante é sua modularidade, permitindo a criação de aplicações que podem ser encapsuladas em um formato de arquivo gerenciável baseado no **opkg** (*Open Package Management*)[8], que por sua vez derivou-se do gerenciador IPK (*Itsy Package*), que em suma são variantes do gerenciador de pacotes do Debian Linux. Através desse gerenciador de pacotes, as aplicações e até módulos do *kernel* podem ser removidos, atualizados ou instalados em equipamentos embarcados sem a necessidade de substituição do *firmware* por completo.

Por ser baseado em um sistema operacional de código aberto, contribui para o desenvolvimento de novas tecnologias e permite que sejam alterados aspectos básicos de um equipamento embarcado, como seu modo de encaminhamento de pacotes, tornando-se uma solução flexível para programabilidade no plano de dados de uma rede de computadores. A programabilidade no plano de dados é um dos objetivos deste trabalho, trazendo consigo novas possibilidades na utilização de equipamentos de prateleira não especializados e transformando-os em encaminhadores com suporte a OpenFlow 1.X.

[7] OpenWRT 14.07 Barrier Braker utiliza o kernel do Linux versão 3.10.x; versões mais recentes do kernel (3.18.x) estão sendo utilizadas nas *releases* em desenvolvimento do OpenWRT

[8] https://code.google.com/p/opkg/

2.4 Considerações do Capítulo

Na primeira seção deste capítulo foram apresentados os conceitos básicos relacionados à área Redes Definidas por *software*. Observações relacionadas ao plano de dados e ao plano de controle, assim como o funcionamento da arquitetura foram objetos desta seção.

A segunda seção deste capítulo relatou um resumo sobre os principais comutadores de pacotes virtuais. Os comutadores comumente utilizados como: CPqD SoftSwitch13 que suporta as versões do OF 1.0 e 1.3; O Click Modular Router que permite o encaminhamento de pacotes em modo kernel no Linux; O xDPd que é uma proposta multi-plataforma e o OvS que suporta encaminhamento em modo usuário e *kernel* foram descritos.

Por último, na terceira seção é apresentado um resumo sobre sistemas embarcados para comutadores de pacotes e suas principais características. Em especial, esta seção descreve os conceitos iniciais e as características desejáveis que atenda a proposta.

Todos esses conceitos são necessários para o entendimento deste trabalho, visto que a proposta baseia-se em criar um comutador de código aberto e programável para prototipação de redes de computadores, que será melhor descrito no Capítulo 4.

O próximo capítulo traz uma breve descrição de alguns trabalhos que tratam de prototipação de Redes Definidas por *Software*.

Capítulo 3

Trabalhos Relacionados

Parte do crescimento das Redes Definidas por *Software* advém da grande variedade de ferramentas e ambientes de emulação e simulação disponíveis para prototipação. Tais ambientes permitem a rápida representação de um conjunto de nós de computadores, *switches*, roteadores e Pontos de Acesso sem fio de maneira relativamente simples.

O capítulo está organizado da seguinte maneira: na Seção 3.1 são apresentados os conceitos básicos sobre um ambiente de emulação comumente utilizado em SDN chamado "Mininet". Em seguida, na Seção 3.2 é apresentada uma evolução desta ferramenta.

Na Seção 3.3 é apresentada uma iniciativa de código fonte aberto para definição de ambientes de instalação para comutadores *fabric*. Logo em seguida, nas Seções 3.4 e 3.5 são descritos dois sistemas operacionais para comutadores baseado em distribuições Linux abertas. Finalmente, conclui-se com a Seção 3.6 que descreve também o uso de uma distribuição Linux, entretanto de código fechado.

3.1 Mininet

Simulação e emulação tem uma particular importância para a rápida prototipação e testes sem a necessidade da aquisição de custosos equipamentos reais. Mininet [Lantz et al. 2010] foi um dos primeiros sistemas desenvolvidos para prover uma maneira rápida e fácil para prototipação e avaliação de novos protocolos e aplicações SDN, através de um ambiente emulado, capaz de simular milhares de nós de rede usando o recurso de *containers namespaces* de rede presentes no sistema operacional Linux, capaz de envolver um recurso global do sistema em uma abstração com seu próprio aspecto isolado, neste caso, recursos de rede, como interfaces de rede, endereços de IP, entre outros.

Conforme vemos na Figura 3.1, o Mininet cria uma rede virtual combinando os processos dos *hosts* em *namespaces* de rede, e conectando-os através de pares de interface *ethernet* virtuais (veth – *virtual Ethernet*). Nesse exemplo, o Mininet conecta a rede à um *switch* OpenFlow no espaço de usuário [Lantz et al. 2010].

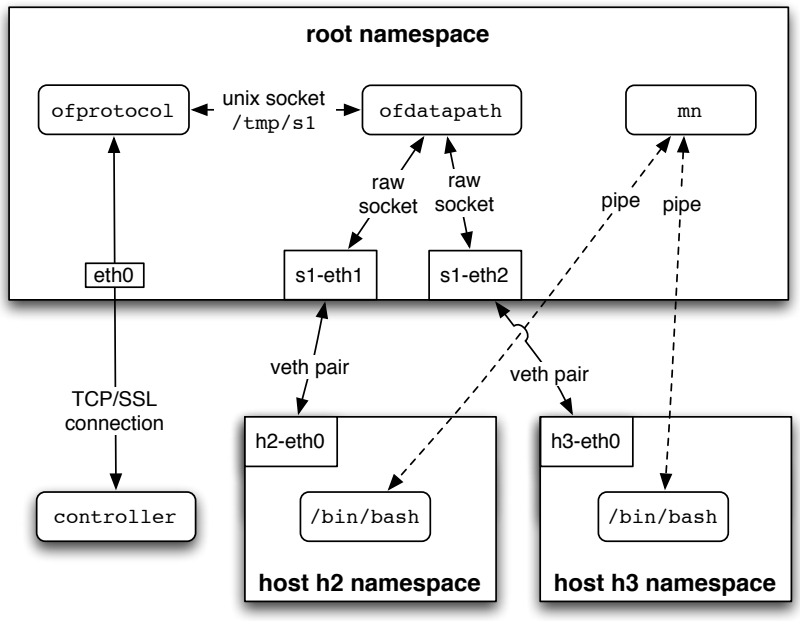


Figura 3.1: Mininet - um ambiente de fácil prototipação [Lantz et al. 2010]

Contudo, como exposto por [Huang et al. 2013], as escolhas de arquitetura de *hardware* e do *firmware* feitas pelo fabricante do *switch* podem impactar negativamente, tanto em funcionalidade quanto performance, as aplicações SDN. Tal impacto é difícil de se reproduzir em um ambiente emulado como o Mininet, o que pode acarretar em diversas divergências no resultado da aplicação, quando aplicada em um ambiente real.

3.2 Mininet-HiFi

O Mininet-HiFi [Handigol et al. 2012] é uma evolução do Mininet, que também emprega virtualização baseada em *containers* presentes no sistema operacional Linux, porém utilizando mecanismos para garantir isolamento de performance, provisionamento de recursos e monitoramento preciso em prol da fidelidade dos resultados.

Conforme Figura 3.2, onde vemos um exemplo de topologia de rede simulada utilizando Mininet-HiFi, os quadros pontilhados simbolizam como o sistema provê isolamento de performance e o monitoramento de recursos, utilizando *hosts* virtuais baseados em *containers*, interconectando-os através de um *switch* virtual e pares de interface *ethernet* virtuais.

Porém, como todo ambiente virtualizado, o resultado do experimento pode ser afetado pelo comportamento do *host* hospedeiro, como uso de disco rígido, CPU e memória RAM, demandados pelo próprio sistema operacional e outras aplicações. No caso do Mininet-HiFi, um resultado com alta fidelidade também depende da capacidade e desempenho do *host* hospedeiro, que é o principal limitador dos recursos a serem aprovisionados, como

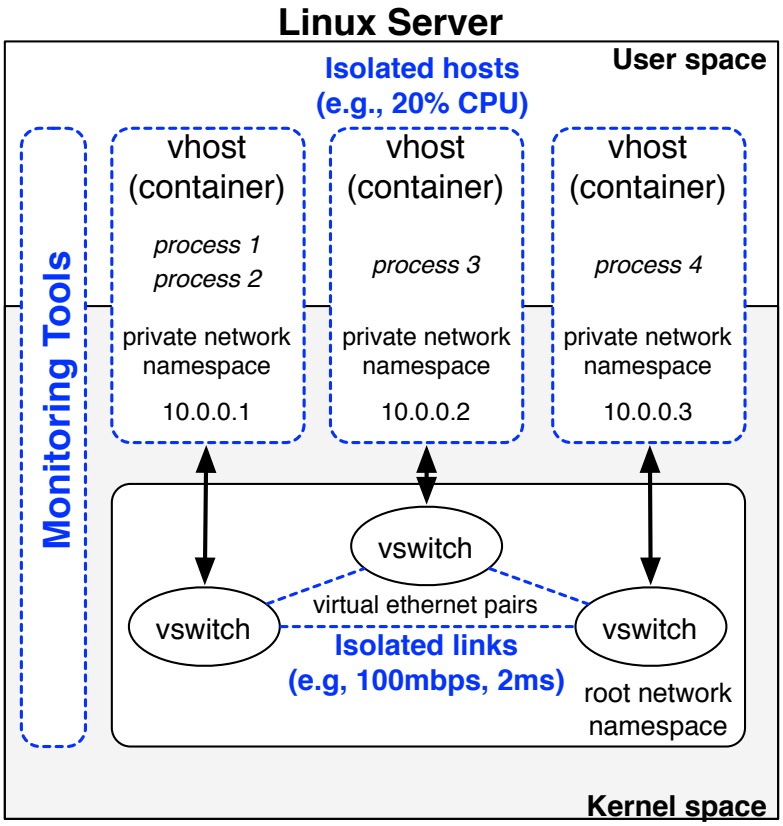


Figura 3.2: Mininet-HiFi: isolamento utilizando *containers* [Handigol et al. 2012]

quantidade de *hosts*, *switches* e ligações virtuais, bem como a velocidade total de cada ligação virtual. Nesse caso, um mal dimensionamento do ambiente a ser emulado pode acarretar em resultados com pouca ou nenhuma precisão.

3.3 ONIE

O projeto ONIE (*Open Network Install Environment*) é uma iniciativa de código fonte aberto que define um "ambiente de instalação" aberto para *switches* sem sistema operacional embarcado pelo fabricante (*bare-metal switches*). ONIE permite a criação de um ecossistema para esta modalidade de *switches*, onde o usuário final pode escolher qual sistema operacional de redes irá instalar no equipamento [Project 2015].

O ONIE é uma combinação de um gerenciador de inicialização (*boot loader*) e um pequeno sistema operacional baseado em Linux que provê um ambiente para provisionamento automatizado. Na Figura 3.3, vemos que o ONIE utiliza somente o conjunto de CPU do *switch* e suas portas de gerenciamento, representado aqui pela parte mais opaca da figura, deixando o plano de encaminhamento de dados para o sistema operacional de rede a ser embarcado no equipamento, que corresponde à parte mais transparente da imagem.

Por ser uma iniciativa recente na presente data deste trabalho, as arquiteturas supor-

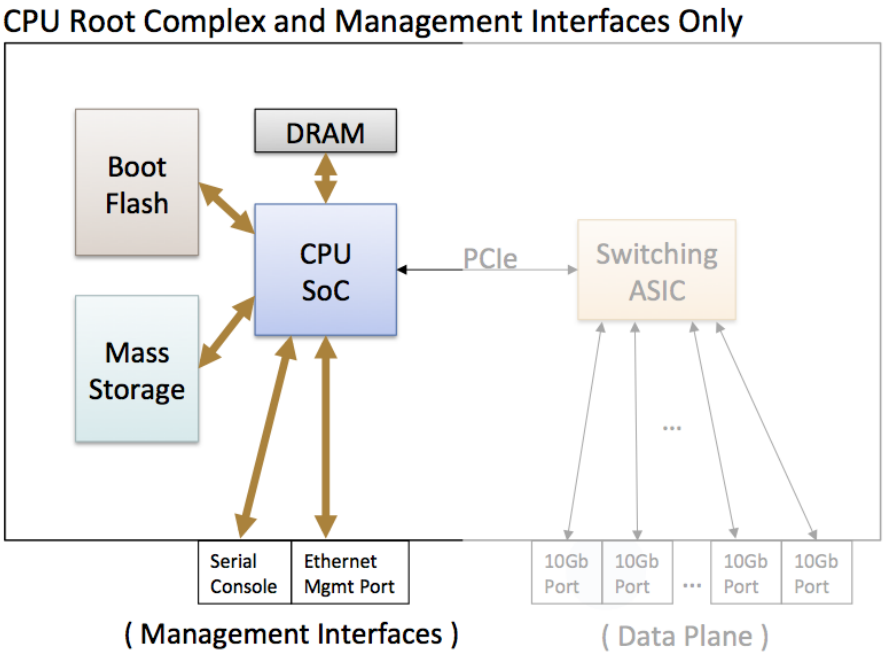


Figura 3.3: ONIE utiliza o conjunto de CPU do *switch* [Project 2015]

tadas pelo projeto são limitadas à alguns modelos de equipamentos presentes no mercado atual, como PowerPC e x86.

Este trabalho foi baseado na arquitetura MIPS presente em equipamentos de baixo custo disponíveis, que até então não possuem suporte junto ao projeto ONIE. A proposta a ser apresentada baseia-se na composição de um sistema operacional de redes completo, não limitando-se apenas ao provisionamento de outros sistemas.

3.4 Cumulus Linux

Cumulus Linux é um sistema operacional de redes baseado na distribuição Debian[1] desenvolvido pela empresa Cumulus Networks. Devido a essa característica, o sistema permite uma melhor integração e desenvolvimento de aplicações baseadas na distribuição Debian do Linux para esse ambiente [Networks 2015]. Sua instalação em *switches* é possível através do projeto ONIE, responsável por criar um ecossistema de *hardware* aberto, que facilita a instalação de qualquer sistema operacional de rede compatível com a plataforma de *hardware*[2].

Cumulus Linux é construído sobre um sistema de código fonte aberto, incluindo diversas ferramentas disponíveis no sistema operacional Linux para gerenciamento de encaminhamento de pacotes, roteamento, *firewall*, entre outros. No entanto, contém um *driver* de dispositivo proprietário (fechado) para fornecer aceleração de *hardware*, responsável por sincronizar o *kernel* com os componentes de baixo nível de rede do *switch*, o que

[1] https://www.debian.org/index.pt.html
[2] https://github.com/opencomputeproject/onie

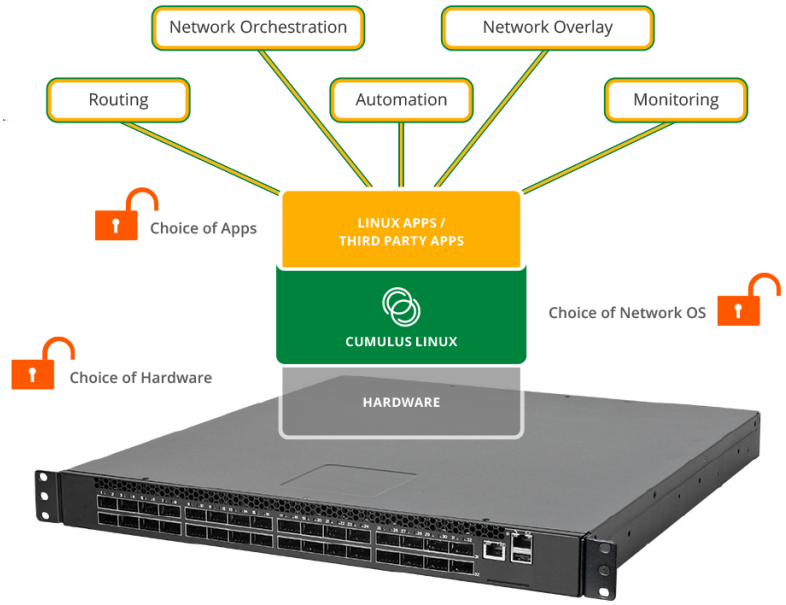


Figura 3.4: Cumulus Linux: Um novo paradigma de ecossistema aberto para switches [Networks 2015]

limita a capacidade de inovação no que tange a este elemento fechado, tornando o projetista dependente do fabricante e limitando a quantidade de equipamentos disponíveis compatíveis com esta solução.

3.5 Open Network Linux

Open Network Linux (ONL) é uma distribuição baseada no sistema operacional Linux específica para *switches* sem sistema operacional embarcado (*bare-metal*) construídos a partir de componentes de *commodities*. O ONL utiliza o sistema ONIE para ser instalado como um *firmware* no equipamento embarcado [Linux 2015].

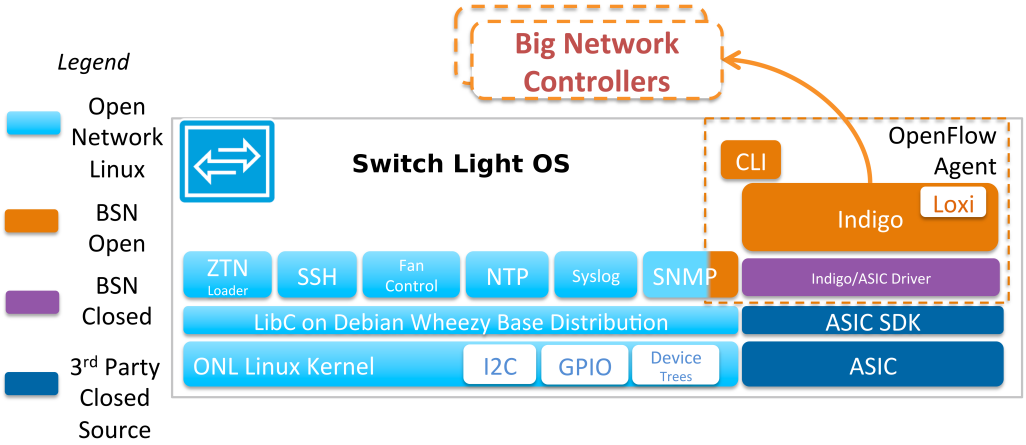


Figura 3.5: Open Network Linux: Layout da arquitetura proposta utilizando um sistema operacional de código aberto [Linux 2015]

Como ilustrado na Figura 3.5, os componentes (em azul) presentes na arquitetura do ONL incluem: kernel e bibliotecas do Linux baseados na distribuição Debian, uma coleção de *drivers* de dispositivos, aplicativos de gerenciamento e acesso remoto, *scripts* de instalação e inicialização e um *bootloader*.

Ainda na Figura 3.5, vemos os componentes dos *drivers* especializados do ASIC em conjunto com o agente OpenFlow Indigo (em lilás). Este agente OpenFlow (em laranja) é responsável por fazer a integração do sistema operacional, *drivers* do ASIC e o plano de controle.

Apesar da arquitetura ser baseada em código aberto, o *driver* e o SDK do ASIC são proprietários, limitando o suporte a poucos dispositivos compatíveis, nos quais os fabricantes disponibilizam seus SDKs e *drivers* de forma fechada.

3.6 PicOS

O PicOS é mais uma proposta de distribuição baseada no Debian Linux, portado para equipamentos embarcados de rede sem sistema operacional embarcado. A natureza agnóstica de *hardware* do PicOS é impulsionada pelo fato do sistema operacional não ser firmemente acoplado ao ASIC do *switch*, à CPU ou memória [SDN 2015].

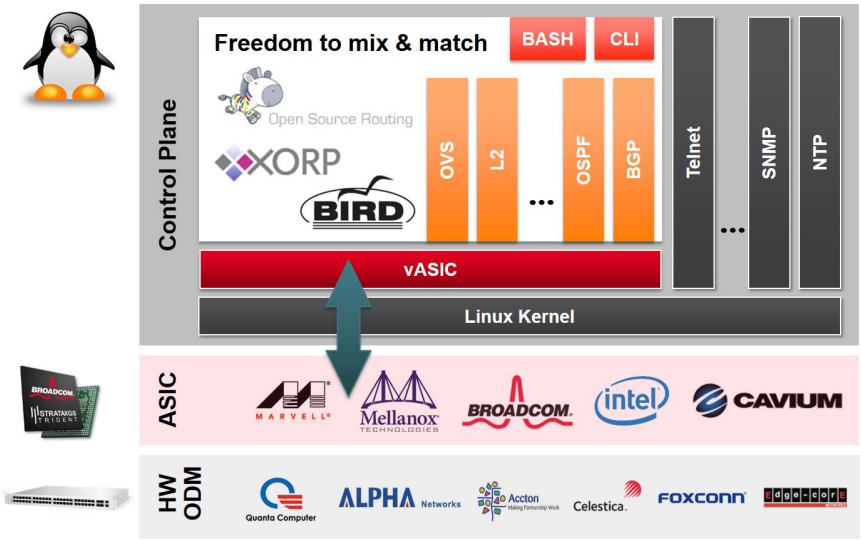


Figura 3.6: Arquitetura proposta pelo PicOS [SDN 2015].

Conforme ilustrado na Figura 3.6, o PicOS é baseado em: um *kernel* do Linux padrão, capaz de evoluir em conjunto com a comunidade; uma camada de abstração de *hardware* (vASIC® – Virtual ASIC), que provê suporte a múltiplos modelos de *hardware*; protocolos tradicionais de roteamento e encaminhamento (ex. BGP, OSPF, camada 2, entre outros); e uma versão modificada e proprietária do Open vSwitch, que provê suporte a aceleração via *hardware* e ao protocolo OpenFlow na versão 1.4, além de integração com tecnologias como CloudStack e OpenStack.

Devido à essa natureza fechada da máquina de encaminhamento responsável pela funcionalidade do protocolo OpenFlow, há uma certa limitação em relação à modificações no plano de dados do equipamento, restringindo a inovação quando dependente de alterações neste componente, ou impedindo sua substituição por outro *switch* virtual que poderia empregar métodos de encaminhamento de pacotes diferentes do proposto na arquitetura.

3.7 Considerações do Capítulo

O ambiente emulado Mininet é uma ferramenta poderosa comumente utilizada em Redes Definidas por *Software* que permite a criação de uma grande infraestrutura de rede virtual em um único computador. É um ambiente escalável que utiliza as primitivas do sistema operacional para prover isolamento, além de ser uma ferramenta simples de usar. No entanto, até o momento da escrita deste trabalho, o Mininet ainda não oferece o desempenho e a fidelidade dos resultados de um ambiente real, embora seus resultados sirvam como parâmetro inicial para um projetista.

Para obter resultados fidedignos algumas propostas que utilizam equipamentos surgiram. Em [SDN 2015] e [Linux 2015] é proposto utilizar uma abordagem chamada de "equipamento aberto", porém, esses projetos ou não são de código fonte totalmente aberto, possuindo partes proprietárias, ou encontram-se em estágio inicial, dando suporte a um número limitado de equipamentos. Por esse motivo, implementações proprietárias de *software* e *hardware* ainda são adotadas pelos fabricantes de dispositivos de rede embarcados.

Outra abordagem mais recente sugere que equipamentos ativos de rede sejam comercializados sem um sistema operacional embarcado de fábrica (*bare-metal switch*), deixando a decisão de qual sistema operacional de rede será utilizado no equipamento para o usuário final. Tal proposta é vantajosa do ponto de vista do fabricante, que pode desempenhar seu papel principal com maior afinidade, que é desenvolver um *hardware* de alto desempenho, enquanto que o desenvolvimento do *software* deve ficar a cargo de uma outra empresa ou uma comunidade de código aberto.

Para tornar possível a implementação do *bare-metal switch* é necessário superar algumas restrições. Um relevante desafio é que o *driver* do controlador ASIC é disponibilizado como um *framework* de código fonte fechado, suportando um número limitado de tipos de arquiteturas disponíveis no mercado atual. Esta abordagem acaba impactando no aumento de custo do projeto a ser executado, por se tratar de equipamentos especializados em *hardware*.

Este capítulo apresentou uma breve descrição dos trabalhos relacionados com a proposta. Estes conceitos e discussão se fezem necessários para o entendimento do trabalho, tendo em vista que a proposta baseia-se em criar um comutador aberto programável para redes. No próximo capítulo é apresentado a proposta do comutador aberto de código programável e suas respectivas premissas.

Capítulo 4

Crops – Uma Proposta de Comutador Programável de Código Aberto para Prototipação de Redes

O OpenFlow é um padrão aberto que permite a programação dos elementos ativos da rede, tais como roteadores, *switches* ou pontos de acesso sem fio. Trata-se de uma proposta pragmática com grande potencial para suportar o grau de programabilidade que as SDN necessitam.

Os equipamentos habilitados com OpenFlow têm sido projetados pelos fabricantes como um sistema fechado, no qual o projetista de redes torna-se dependente da implementação de *firmware* do fabricante.

Se a inovação do projetista requer modificação no plano de dados, como por exemplo encaminhar para uma porta com uma certa probabilidade sem utilizar um controlador e de forma pró-ativa, ou definir um roteamento baseado em origem [Martinello et al. 2014, Ramos et al. 2013], e até mesmo substituir a lógica de encaminhamento tradicional da rede, utilizando o conceito de Named Data Networking (NDN)[Smetters and Jacobson 2009], então o equipamento não permite tal operação, por possuir um *firmware* proprietário, impedindo sua validação experimental em um ambiente real.

Esses fatos limitam significativamente o potencial de inovação na arquitetura SDN, tanto pela restrição imposta pelo modelo de encaminhamento, quanto pelo sistema fechado do fabricante.

Neste capítulo é apresentada a proposta do Crops, um comutador de código aberto programável para prototipação de redes. Objetivo é ilustrar os requisitos para a construção da proposta, permitindo sua validação e reprodução em equipamentos de prateleira.

A hipótese desta dissertação é que *se for utilizado um hardware de baixo custo que permita a instalação de um sistema operacional e de uma máquina de encaminhamento de código aberto, então será possível obter um comutador de código aberto programável.*

A Figura 4.1 apresenta a contextualização da hipótese.

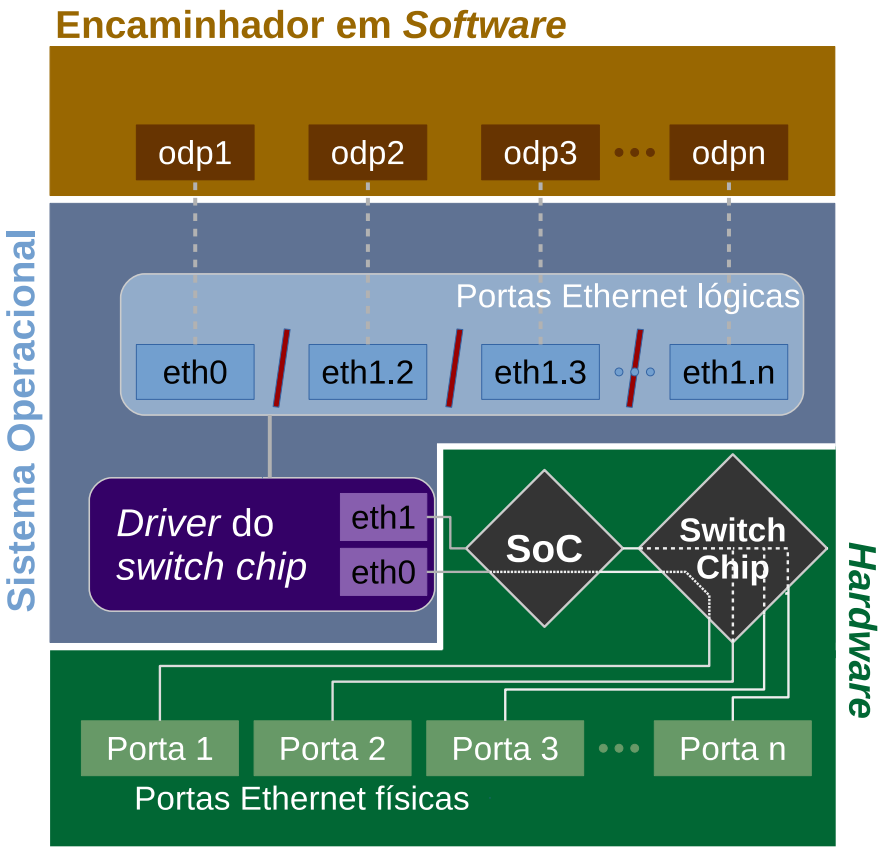


Figura 4.1: Arquitetura proposta: uma sistemática de operação em camadas.

Ainda na Figura 4.1, observe que o *hardware* deverá fornecer suporte à comunicação com o *switch chip*, sendo assim possível transmitir e receber pacotes utilizando as portas físicas do equipamento.

Já o sistema operacional deve ser *open source*, possuindo suporte à criação de portas lógicas virtuais para a comunicação com a máquina de encaminhamento. Outro importante recurso do SO é fornecer suporte ao *driver* do *switch chip* do equipamento.

O requisito para a máquina de encaminhamento é que ela deverá ser baseada em *software*, e ter disponível seu código fonte também aberto, fazendo interface com o SO que é executado no equipamento. Com essa abordagem pode-se utilizar a flexibilidade das aplicações (*software*) de forma a atender a hipótese levantada.

Destaca-se que o foco deste trabalho não é a criação de um *switch* comercial de alto desempenho, com acesso às instruções de *hardware* do equipamento. A proposta representa uma evolução natural desejável na prototipação de redes de computadores. O objetivo é transformar um ativo de rede de baixo custo em um comutador programável totalmente aberto, com suporte à prototipação em SDN, acrescentando um ambiente de prototipação mais rico, conforme ilustração da Figura 4.2.

Para melhor entendimento, esta seção foi dividida em três partes, organizadas em uma abordagem "*bottom-up*".

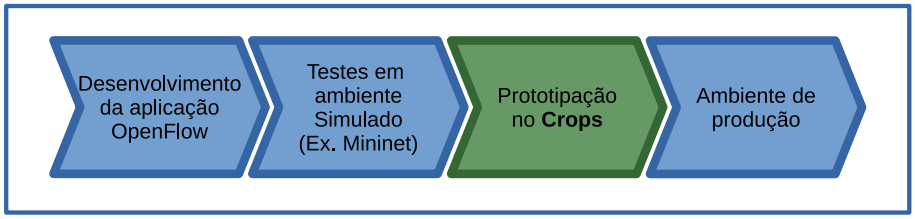


Figura 4.2: Evolução do desenvolvimento de uma aplicação SDN.

Na subseção 4.1 são descritas as observações relacionadas ao uso de equipamentos não especializados de prateleira. O intuito é demonstrar a viabilidade de habilitar o protocolo OpenFlow nestes equipamentos, permitindo incorporar novas características pelo projetista de rede. Como por exemplo, customizar funcionalidades inovadoras antes não disponíveis nas soluções proprietárias, habilitar recursos opcionais disponível no protocolo, como alguns tipos de tabelas de grupos entre outros.

Acredita-se que a validação em um *Testbed* real – com *hosts*, interfaces de rede e ativos de rede fisicamente distribuídos – abre novas perspectivas que não seriam possíveis por meio do ambiente emulado tradicional.

Por exemplo, este novo ambiente permitirá avaliar a carga de tráfego de rede que um *switch* físico consegue suportar, ou até mesmo a capacidade de transferência de um *host* físico, sem a influência direta de um *host* hospedeiro (CPU, memória, disco rígido) que ocorre quando utilizam-se vários *switches* e *hosts* virtuais em um ambiente emulado tradicional.

Na Subseção 4.2 serão apresentadas as observações referentes ao NOS, peça fundamental na construção do comutador proposto.

Em seguida, na Subseção 4.3, são descritos os requisitos necessários para a máquina de encaminhamento baseada em *software*, responsável pela comutação dos pacotes no *switch*.

4.1 Equipamentos não especializados de prateleira

Conforme mencionado anteriormente, para a seleção de um *hardware* compatível com a proposta, buscou-se avaliar alguns requisitos. O custo do equipamento foi um dos itens analisados, devendo ser objeto incentivador à prototipação sobre ativos de rede, uma vez que equipamentos com maior desempenho são comercializados a valores muito altos, além de não possuírem suporte a substituição de seu *firmware* proprietário por um sistema de código aberto. O *hardware* escolhido foi o Mikrotik RouterBoard. Seu custo é relativamente baixo (aproximadamente US$ 12,00 por porta Gigabit[1]).

Estes equipamentos são comumente utilizados no mercado, principalmente em provedores de Internet de pequeno/médio porte. Tal características reforça a possibilidade de transformá-lo em uma *white box*, permitindo a portabilidade de outros sistemas operacio-

[1]Custo referente ao modelo RB750GL, disponível no site http://routerboard.com/RB750GL

nais sobre sua arquitetura. Devido à programabilidade disponível em equipamentos SDN e sua facilidade de permitir inovações através de aplicações de rede e modificações no plano de dados, provedores de Internet podem ser potenciais consumidores desta tecnologia.

Portanto, nossa proposta de *white-box switch*[2] baseia-se em transformar *hardware* comercial, que possui um *firmware* proprietário fechado, em um *switch* SDN, através da substituição de seu sistema operacional original por um novo sistema operacional de rede, com suporte a OpenFlow através do uso de um *switch* virtual em *software*.

4.2 Sistema Operacional de Rede

O termo Sistema Operacional de Rede (ou NOS – Network Operating System) pode assumir significados distintos:

1. um sistema operacional capaz de gerenciar e compartilhar recursos locais ou remotos (ex. impressoras, dispositivos de armazenamento, aplicações, entre outros) entre vários computadores em uma rede local ou através da utilização de uma VPN;

2. um sistema operacional especializado embarcado em dispositivos de rede, atuando como um gerenciador de funcionalidades mais básicas, como encaminhamento, roteamento e classificação de pacotes, provendo também uma interface entre o usuário e as funções de rede disponíveis;

3. um controlador SDN [Kreutz and Ramos 2014], que consiste em um sistema executado fora do sistema embarcado (ex. em um servidor), capaz de gerenciar vários ativos de rede de maneira centralizada, e ao mesmo tempo prover integração entre suas aplicações e prover uma visão global dos recursos disponíveis em uma rede de computadores.

Nesta proposta, NOS referencia-se ao item **2**, ou seja, criar um ambiente harmonizado entre sistema operacional e aplicações embarcadas, especializadas em redes de computadores, como um *switch* virtual com suporte a OpenFlow, permitindo ao projetista de redes inovar no sentido da programabilidade do plano de dados, graças aos recursos disponíveis neste conjunto de sistema operacional Linux e encaminhador em *software* de código aberto. A programabilidade proposta pelo plano de dados neste trabalho será melhor detalhada a seguir, na Seção 4.3.1.

Para o desenvolvimento deste trabalho, foi escolhida a distribuição Linux OpenWRT como protótipo para um NOS, pelos seguintes motivos:

[2]Um *switch* sem sistema operacional embarcado, que permite a instalação de um *firmware* de código aberto

- É livre e de código fonte aberto, permitindo adaptações ou modificações em seu código de forma espontânea, ou seja, sem que haja a necessidade de solicitar permissão ao seu proprietário para modificá-lo.
- É compatível com uma gama de equipamentos embarcados, incluindo a arquitetura disponível para este trabalho (MIPS).
- É modular e baseado em sistema de gerenciamento de pacotes, permitindo a customização de partes do sistema ou instalação de novos módulos, sem a total substituição do *firmware* do equipamento.
- Uma vez que é baseado no sistema operacional Linux, subentende-se que quase todos os aplicativos portados para Linux possam ser também portados para funcionar no OpenWRT, e consequentemente, em um equipamento embarcado de outra plataforma.

A construção deste sistema operacional de rede, incluindo modificações da estrutura de inicialização, sistema de configuração de rede, adaptações entre interfaces físicas e lógicas, dentre outras inferências, requer um processo de compilação cruzada a partir do código fonte modificado[3], possibilitando sua construção com instruções referentes à arquitetura do *host* de destino (neste caso um *switch* RouterBoard, baseado na arquitetura MIPS).

As modificações necessárias e adaptações no código fonte do sistema operacional e aplicativo de encaminhamento serão discutidas no decorrer do Capítulo 5, incluindo as ferramentas e processos que englobam a compilação cruzada, que são descritos na Subseção 5.2.3 do Capítulo referenciado.

4.3 Open vSwitch como motor de encaminhamento programável

Atualmente, existem diversas propostas de *switches* virtuais, e a maioria deles já nasce com a concepção estrutural e funcional de SDN, consequentemente dando suporte ao protocolo OpenFlow.

De todas as alternativas disponíveis, o Open vSwitch (OvS) destaca-se por sua versatilidade e portabilidade, estando disponível em diversos sistemas operacionais, como Microsoft Windows, FreeBSD, NetBSD, e sobretudo, o sistema operacional Linux, estando disponível neste último como parte integrada de seu *kernel*.

[3]Distribuição modificada do SO OpenWRT está disponível em http://git.lprm.inf.ufes.br/diego/openwrt, somente para uso acadêmico, mediante solicitação de acesso via email do desenvolvedor: loxxxa@gmail.com

Essa integração nativa entre OvS e *kernel* do Linux facilita a implementação de novas características sobre o encaminhador, mantendo a compatibilidade com o restante do sistema operacional.

O Open vSwitch está basicamente dividido em três componentes principais:

1. *ovsdb-server*: um gerenciador do banco de dados do Open vSwitch; o banco de dados é responsável por armazenar toda a configuração presente no OvS, como *bridges*, interfaces, regras na Tabela de Fluxo.

2. *ovs-vswitchd*: um *daemon* executado em espaço de usuário, responsável por gerenciar as instâncias de *switches* virtuais no sistema local, acessando o banco de dados do OvS para atualizar as configurações quando alteradas.

3. *openvswitch.ko*: um módulo disponível apenas no sistema operacional Linux, responsável pelo encaminhamento de pacotes em espaço de *kernel*.

Nossa proposta contempla modificações efetuadas no módulo do *kernel* do Linux (item 3), acrescentando funções para o funcionamento da semântica *Select*, recurso que não está disponível na versão do OvS utilizada neste trabalho[4].

Contudo, como nosso foco não almeja a performance e sim funcionalidade, a solução implementada não utiliza o encaminhamento em modo *kernel*, não fazendo uso de recursos como o cache de encaminhamento presente no módulo do OvS. Ao invés disso, o cache é esvaziado a cada novo pacote que combina com a regra inserida na Tabela de Fluxo referente à tabela de grupos *Select*, forçando que a decisão de escolha de encaminhamento seja feita de maneira estocástica, ou seja, de modo aleatório, através do espaço de usuário. Na Subseção 4.3.1 são descritas as observações relacionadas ao plano de dados programável.

4.3.1 Plano de Dados Programável

Uma plataforma que suporta programabilidade deve oferecer ao projetista um ambiente que permita inovar livremente sem a dependência dos fabricantes. É desejável que tal ambiente permita a validação de seus experimentos com confiança nos resultados, além de desempenho similar aos produtos encontrados no mercado.

Neste trabalho, a programabilidade proposta baseia-se na mesma premissa, trazendo autonomia ao plano de dados através das tabelas de grupos opcionais, em conjunto com um *switch* virtual de código aberto e um sistema operacional Linux, habilitados para uso em equipamentos embarcados.

Com este conjunto, espera-se abrir também novas perspectivas quanto à programabilidade em relação ao método de encaminhamento que um *switch* embarcado pode assumir,

[4]Open vSwitch versão 2.3.0

como roteamento na origem [Ramos et al. 2013], cujo o encaminhamento é realizado através da informações contidas no cabeçalho dos pacotes Ethernet.

Em [Martinello et al. 2014] é proposto o encaminhamento baseado em chaves (global e local), de maneira estática ou dinâmica [Vencioneck et al. 2014], utilizando um plano de dados de código aberto baseado em um motor de encaminhamento programável: o Open vSwitch. Todas essas proposta são passíveis de implementação utilizando a abordagem proposta.

Como prova de conceito, projetamos um comutador estocástico que permite avaliar a programabilidade do comutador proposto. Apesar da proposta de programabilidade de SDN, onde o plano de controle dita as regras ao plano de dados, novas características sugeridas a partir da versão 1.2 do protocolo OpenFlow trazem uma certa autonomia ao plano de dados.

Este fato pode ser constatado revisando a descrição das tabelas de grupos, mais especificamente nos tipos *Select* e *Fast Failover*:

- *Select*: "Os pacotes são processados por um único *bucket*[5] no grupo, baseado em um algoritmo de seleção **computado no *switch***" [ONF 2011]
- *Fast Failover*: "Esse tipo de grupo permite ao ***switch* mudar o encaminhamento** sem a necessidade de consultar um controlador" [ONF 2011]

Subentende-se então que, mesmo havendo um plano de controle centralizado, o plano de dados necessita de uma certa autonomia quanto ao método de encaminhamento, para que o ativo de rede possa encaminhar pacotes com o mínimo de atraso e da maneira mais adequada ao cenário empregado.

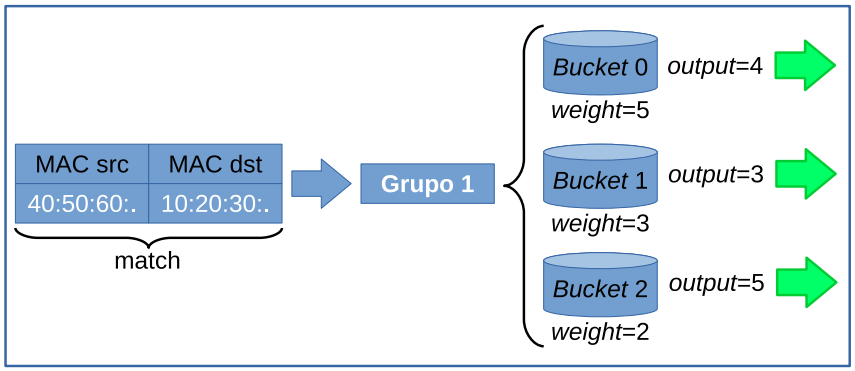


Figura 4.3: Método **Select**

O método **Select** seleciona e encaminha um *bucket* de um respectivo grupo de *buckets*. Os pacotes são processados por um único *bucket* no grupo, baseado em um algoritmo de escolha calculado pelo switch [ONF 2011] (ex. um *hash* em uma tupla, um *round-robin*

[5] *Bucket* pode ser entendido como um tipo de *buffer*, ou seja, uma área de armazenamento temporário de dados

simples ou de maneira estocástica). Na Figura 4.3 é ilustrado o método *Select*, como um tipo de grupo com sua própria semântica. Note que o valor do peso (*weight*) que é atribuído ao fluxo determina a probabilidade de execução da respectiva ação.

O método **Fast Failover** encaminha um pacote para o primeiro *bucket* ativo. Cada *bucket* de ação é associado com uma porta específica e/ou grupo que controla seu estado de atividade (*watch*). Os *buckets* são verificados na ordem definida pelo grupo, e o primeiro *bucket* que é associado a um grupo ou porta ativa é selecionado. Esse tipo de grupo permite que o *switch* modifique o encaminhamento sem consultar o controlador. Se nenhum *bucket* estiver ativo, os pacotes são descartados [ONF 2011], conforme ilustrado na Figura 4.4.

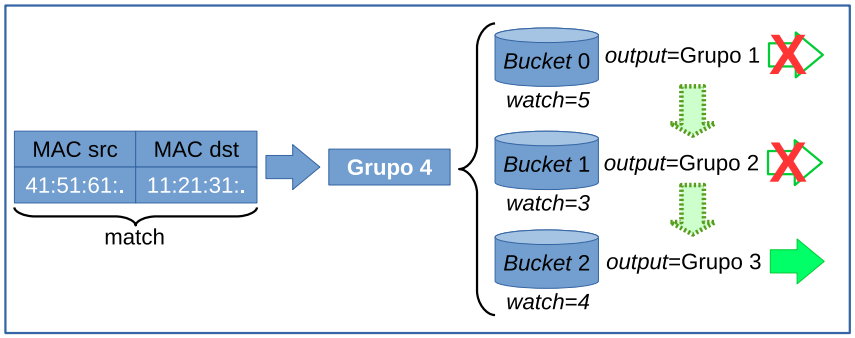


Figura 4.4: Método **Fast Failover**

4.4 Considerações do Capítulo

A primeira seção deste capítulo apresentou a descrição dos requisitos para a seleção do *hardware* de baixo custo utilizado na proposta. Optou-se pelo uso dos equipamentos Mikrotik RouterBoard. Estes equipamentos são largamente comercializados e possuem desempenho aceitável. Outro ponto relevante é que tais equipamentos estavam disponíveis no Laboratório NERDS – Núcleo de Estudos em Redes Definidas por *Software*.

Na segunda seção deste capítulo foram relatados os requisitos necessários para a escolha do SO. Para atender tais requisitos da proposta, como compatibilidade e suporte ao *hardware* Mikrotik, foi escolhido como SO o OpenWRT. Além disso é um SO *open source* e extremamente flexível e com um bom desempenho.

A última seção deste capítulo apresentou a descrição do máquina de encaminhamento. Optou-se em utilizar o OvS que permite a integração nativa com o Linux, facilitando a implementação de novas características sobre o encaminhador, mantendo a compatibilidade com o restante do sistema operacional.

Ainda na Seção 4.3, como prova de conceito relatou-se a concepção de um comutador estocástico que utiliza as novas *features* do OpenFlow 1.3. O objetivo foi ilustrar a implementação da programabilidade utilizando o comutador proposto.

A próxima seção demonstra a implementação da proposta, juntamente com as observações e os detalhes necessários para sua reprodução.

Capítulo 5

Implementação do Protótipo

A proposta de comutador de código fonte aberto foi projetada sobre o sistema operacional OpenWRT [OpenWRT 2014], utilizando como motor de encaminhamento o Open vSwitch (OvS)[Subramanian et al. 2009], modificados e compilados para a arquitetura MIPS presente nos equipamentos RouterBOARD empregados no trabalho.

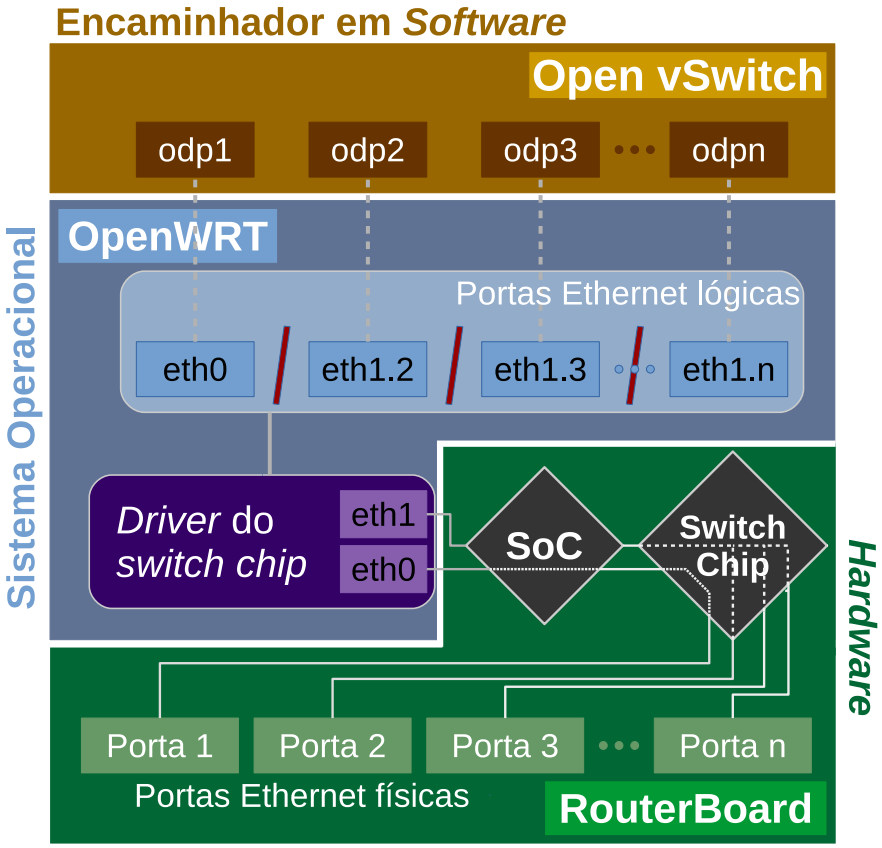


Figura 5.1: Proposta do protótipo

Na Figura 5.1 é ilustrada a arquitetura da proposta, onde um sistema operacional, de código fonte aberto e capaz de ser portado para diversas plataformas, apoia-se em um *hardware* embarcado que possui a flexibilidade de trabalhar com outros sistemas além

do original (*default*) disponibilizado pelo fabricante, e sobre esse sistema operacional é implementado um encaminhador em *software*, também de código fonte aberto, habilitado para utilizar o protocolo OpenFlow na versão 1.X[1].

Com esse arranjo sincrônico, obtemos uma poderosa ferramenta para prototipação de novas aplicações, ou até mesmo adaptação de aplicações já criadas para um uso futuro em ambiente de produção com outros equipamentos similares, além de abrir um novo horizonte para criação de novos métodos de encaminhamento e implantação de novas funcionalidades presentes no protocolo OpenFlow a partir da versão 1.2, que podem não estar disponíveis em equipamentos embarcados, por restrição dos fabricantes.

Assim como no Capítulo 4, as subseções seguintes descrevem cada componente da proposta em uma análise "*bottom-up*". Também serão comentadas as modificações necessárias nesses componentes para o perfeito funcionamento do comutador de código fonte aberto.

5.1 *Hardware*: Mikrotik RouterBOARD

O *hardware* presente nas RouterBoards, apesar de ser construído para um propósito único simplificado, que é encaminhar e rotear pacotes de rede, é heterogêneo no sentido da empregabilidade dos componentes embutidos no seu circuito. Com isso, podemos encontrar uma variedade de SoCs diferentes, que controlam uma quantidade definida de portas Ethernet físicas, dependendo do modelo a ser utilizado.

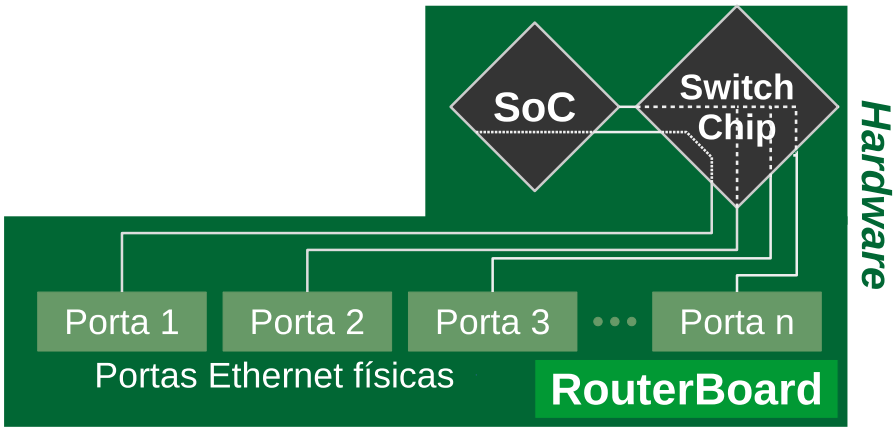


Figura 5.2: *Hardware* Mikrotik RouterBoard

Porém, todos os equipamentos utilizados nesse trabalho apresentaram uma configuração similar à descrita na Figura 5.2, que basicamente adota um *chip* responsável pelo encaminhamento em *hardware* (*switch chip*), e um outro *chip* que é, simplificadamente, o

[1] Algumas caracterísicas do OpenFlow 1.2 precisaram ser implementadas no Open vSwitch durante o trabalho, por serem opcionais de acordo com a especificação do protocolo e por não estarem disponíveis na versão utilizada atualmente.

CPU do equipamento, encarregado do processamento de todas as instruções que não estejam ligadas ao encaminhamento em *hardware*. A quantidade de portas Ethernet físicas varia de apenas 3 a 10, sendo que algumas são apenas *fast Ethernet*, enquanto outras são *gigabit Ethernet*.

Em alguns modelos, o *switch chip* concatena um determinado conjunto de portas Ethernet físicas e entrega ao *CPU* apenas uma porta Ethernet lógica; já o *CPU* é ligado quase que diretamente à uma outra porta Ethernet física, disponibilizando um total de duas portas Ethernet lógicas para o sistema operacional.

Nas seção seguinte, serão descritos os procedimentos metodológicos adotados para que o sistema operacional consiga subdividir essas portas Ethernet lógicas em portas VLANs, associando-as a sua respectiva porta Ethernet física.

5.2 Sistema Operacional: OpenWRT

O sistema operacional OpenWRT possui, dentre outras virtudes, a capacidade de ser adaptado a diversas plataformas de *hardware*, graças a sua natureza de código fonte aberto. Outro ponto forte, do ponto de vista de programabilidade, é ser baseado no *kernel* do Linux, que é extremamente portável; outra vantagem é seu tamanho reduzido, sendo possível instalar uma distribuição completa, com todas as opções de roteamento e encaminhamento, sobre um sistema com apenas 4 megabytes de armazenamento.

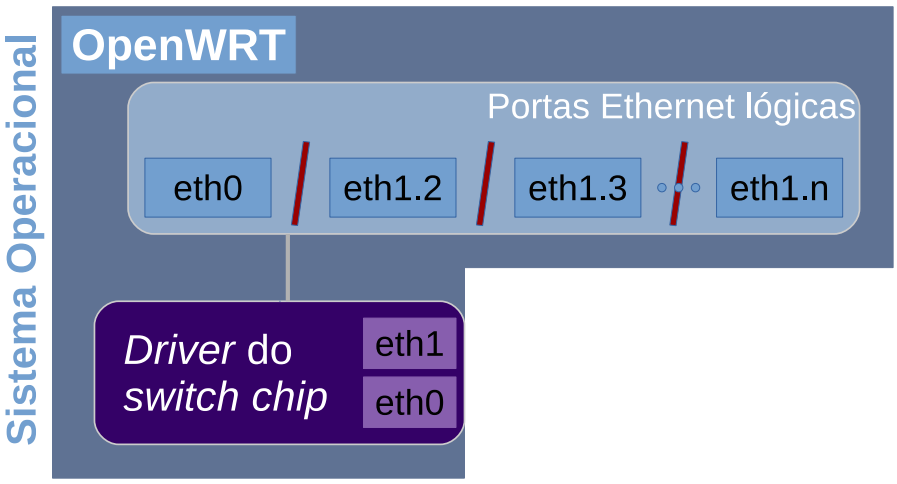


Figura 5.3: Sistema operacional OpenWRT

Conforme Figura 5.3, o módulo do *driver* do *switch chip* recebe do ASIC apenas duas interfaces lógicas (representadas por eth0 e eth1), que são responsáveis pela conexão com as portas Ethernet físicas do equipamento. O sistema operacional, de acordo com o *layout* de rede estipulado em seu arquivo de configurações, é capaz de subdividir uma única interface lógica (eth1, por ex.) em diversas interfaces Ethernet lógicas (eth1.1, eth1.2, eth1.3, por ex.), sendo que cada uma será a representação lógica de cada interface

Ethernet física para o sistema operacional. Cabe ao *driver* do *switch chip* fazer essa associação entre as portas Ethernet lógicas e as portas Ethernet físicas, através de uma interface entre o sistema operacional Linux e o ASIC.

Após essa associação e divisão das portas Ethernet, o equipamento perde a capacidade de encaminhar pacotes, ou seja, cessa-se a comunicação entre todas as portas Ethernet físicas. A partir daí, podemos acoplar um novo mecanismo de encaminhamento de pacotes, baseado em *software*, com capacidade de ser programável, suportando o protocolo OpenFlow em sua versão 1.3.

Para facilitar o entendimento e a reprodução da proposta, as próximas subseções detalham conceitos básicos. Informações complementares, *scripts* e procedimentos técnicos são relatadas no Apêndice A.

5.2.1 RouterBOOT e o sistema de inicialização do OpenWRT

Toda RouterBoard possui um gerenciador de *boot*, responsável pela inicialização do sistema operacional embarcado no equipamento, seja o RouterOS original, ou até mesmo o OpenWRT. O RouterBOOT também é responsável pela configuração básica da RouterBoard, permitindo o gerenciamento e modificação de certos parâmetros do sistema, como *clock* do processador, método de inicialização (via rede, via *flash*), atualização do *firmware*, entre outros.

O RouterBOOT é atualizado frequentemente pelo fabricante, para suprir uma possível modificação de especificação presente em novos equipamentos RouterBoard, como novos modelos de CPU e memória empregados pelo fabricante, ou até mesmo novos modelos de equipamento. Com essa atualização, porém, alguns parâmetros de inicialização que são repassados ao sistema operacional OpenWRT são alterados, como a identificação do equipamento.

Portanto, uma modificação do sistema de inicialização do OpenWRT se fez necessária para que o mesmo conseguisse identificar qual modelo de equipamento seria gerenciado, definindo assim quais interfaces Ethernet lógicas seriam criadas e associadas às portas Ethernet físicas da RouterBoard, de acordo com o *layout* do arquivo de configuração de rede citado na Seção 5.2.

5.2.2 *Driver* do *switch chip*

O OpenWRT possui um módulo de *kernel* do Linux, de código fonte aberto, responsável pelo gerenciamento do *switch chip* da RouterBOARD. Esse *driver*, em conjunto com o aplicativo *swconfig*, são responsáveis pela associação das portas Ethernet físicas do equipamento às portas Ethernet lógicas do sistema operacional, usando a interface Ethernet entregue pelo *hardware*.

O mecanismo de pesquisa ou lógica de resolução de endereço (ARL - *Address Resolution Logic*) presente no *switch chip* extrai o endereço MAC de origem e de destino de cada *frame* recebido a partir de cada porta Ethernet física. O ARL realiza todas as buscas de endereços MAC, executa as funções de aprendizagem (*learning*) e envelhecimento (*aging*) na mesma velocidade da conexão (*wire speed*). A base de dados dos endereços MAC é armazenada na memória SRAM embarcada no equipamento, com um tamanho de entradas pré-definidas, de acordo com o modelo do *switch chip* (normalmente entre 1024 e 2048 entradas) [AR8 2011].

Para o correto funcionamento do mecanismo de encaminhamento modificado, torna-se necessária a desativação da função de *learning*; esse processo normalmente é executado pelo driver do *switch chip* no momento da divisão das portas em VLANs. Porém, alguns modelos de *switch chip* presentes nas RouterBOARDS (modelo RB450G, por exemplo), possuem uma falha na desativação da função de *learning*, ocasionando em um não encaminhamento dos pacotes entre as portas VLANs.

Para contornar esse *bug*, foram efetuadas modificações no *driver* do *switch chip*, adicionando uma opção para controlar a função de *learning* em tempo de execução, através do comando *swconfig* acrescido do parâmetro *enable_learning* com a opção 0 (desativar) ou 1 (ativar), ou diretamente no arquivo de configuração de rede do sistema operacional OpenWRT.

Uma vez divididas as interfaces entregues pelo *driver* em interfaces Ethernet lógicas, cada uma fica encarregada de representar uma porta Ethernet física, previamente definida pelo arquivo de configuração de rede. Portanto, todos os pacotes com origem e destino em uma determinada interface Ethernet lógica são movidos para a porta Ethernet física correspondente, e vice-versa. Contudo, com a divisão das portas físicas em portas lógicas, aspectos como a detecção do estado da porta em relação ao cabo de rede não são herdados, impedindo o correto funcionamento de algumas funções do OpenFlow, como a semântica *fast failover*, que depende do estado da porta Ethernet para encaminhar os pacotes para uma porta funcional.

Houve a necessidade, então, de implementar um método de detecção de estado de conexão da porta física diretamente no *driver* do *switch chip*, e assim associar o estado da conexão diretamente à interface lógica correspondente, usando a mesma sistemática presente no *kernel* do Linux e obedecendo a sintaxe atual do código fonte do *driver*, para não interferir no funcionamento e na performance do sistema.

Esta detecção parte do princípio da utilização de um monitoramento de estado da porta Ethernet física, através de uma função inserida na fila global do *kernel* do Linux (*schedule_delayed_work*[2]), que funciona como um *loop* de eventos globais no sistema operacional. Quando detectada a falha ou a restauração da conexão em uma determinada porta física, a função repassa essa informação para a porta lógica previamente

[2]https://www.kernel.org/doc/htmldocs/kernel-api/

associada[3], através de uma primitiva do *kernel* (*netif_carrier_on* para conectada, ou *netif_carrier_off* para desconectada), definindo seu estado para o sistema operacional e toda a pilha de rede do Linux.

A opção de detecção do estado da porta Ethernet foi adicionada ao *driver* do *switch chip*, sendo possível de ser ativada ou não, através do comando *swconfig* acrescido do parâmetro *port_status* com a opção 0 (desativar) ou 1 (ativar).

5.2.3 Compilação Cruzada

Compilação cruzada consiste em compilar um código fonte em um sistema hospedeiro de uma arquitetura computacional, gerando um arquivo binário com instruções de outra arquitetura, utilizando um *toolchain*.

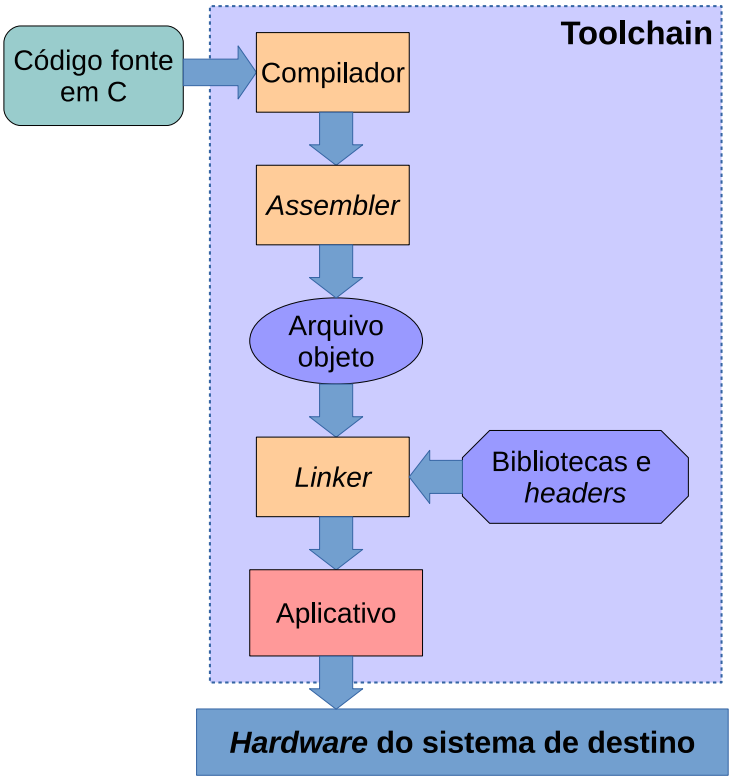


Figura 5.4: *Toolchain*: uma coleção de ferramentas e bibliotecas para compilação cruzada.

Um *toolchain* consiste em uma coleção de ferramentas de *software* necessárias para compilar um aplicativo. Tradicionalmente, isto inclui um compilador C (e outras linguagens), um construtor (*assembler*), um *linker*, bibliotecas e *headers* da linguagem C, que são organizados conforme ilustrado na Figura 5.4.

Estes últimos componentes (bibliotecas e *headers*) são bibliotecas de código compartilhado que atuam como uma "embalagem" em volta da API do kernel do Linux, e são compartilhados por qualquer aplicação que esteja sendo executada no sistema. Alguns

[3] A associação entre porta física e porta lógica é feita de maneira automática, na inicialização da rede no sistema operacional OpenWRT

componentes adicionais podem ser incluídos em um *toolchain*, como bibliotecas extras (ex. zlib para compressão), um depurador, um gerador de perfil (*profile*) e um verificador de memória [Heath 2002].

Um *toolchain* multi-plataforma é construído para ser executado em uma plataforma de desenvolvimento (ex. arquitetura x86), porém é capaz de construir programas que podem ser executados em outras plataformas de destino (ex. arquitetura MIPS).

Com essa ferramenta podemos executar uma compilação cruzada, modificar um sistema operacional e desenvolver aplicações especializadas em redes sobre uma arquitetura, com instruções nativas de uma arquitetura de destino específica, permitindo que sejam portados para equipamentos embarcados.

O SO OpenWRT disponibiliza em sua distribuição de desenvolvimento uma ferramenta chamada **Buildroot**: um conjunto de arquivos com definições de construção (*Makefiles*) e *patches* que automatizam a geração do *toolchain* de compilação cruzada para sistemas embarcados. Graças a esta ferramenta foi possível a implementação das modificações e a construção do *firmware* específico para a arquitetura requisitada.

A instalação do *firmware* construído para o *hardware* da RouterBoard é efetuada utilizando aplicações e ferramentas nativas do OpenWRT e da plataforma de desenvolvimento (neste caso, Linux em um ambiente x86), que possibilitam um processo de *boot* remoto e posterior instalação do SO no equipamento. As particularidades técnicas sobre como portar o Crops em equipamentos embarcados são apresentadas no Apêndice B, descrito em um passo-a-passo sucinto e objetivo.

5.3 Plano de Dados: Open vSwitch

O *software switch* Open vSwitch (OvS) é uma implementação de encaminhador em *software* muito versátil, capaz de utilizar a velocidade do encaminhamento em modo *kernel* do Linux, juntamente com a versatilidade do modo usuário para comunicação com protocolos de alto nível, como OpenFlow.

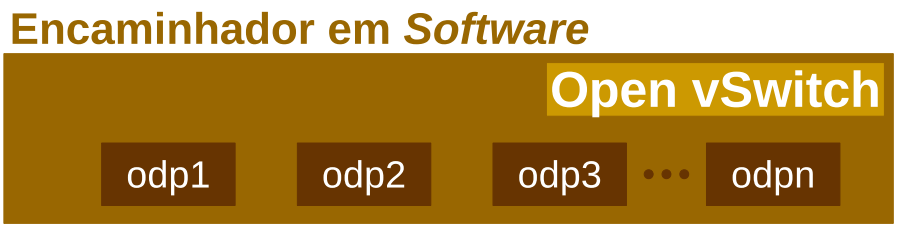


Figura 5.5: Plano de dados com OvS

A Figura 5.5 descreve a estrutura interna do OvS em relação às portas do *switch* virtual, descritas aqui como **odp0**, **odp1**, **odp2** e **odpn**. Cada porta virtual é associada a uma interface Ethernet lógica, que por sua vez é associada a uma porta Ethernet física,

conforme detalhado na Figura 5.1, tornando assim possível o tratamento de pacotes de rede usando SDN, mais especificamente o protocolo OpenFlow.

Para realizar o balanceamento de carga, foi necessária uma modificação no código fonte do OvS, a fim de permitir o encaminhamento dos pacotes de forma estocástica, alterando a função *xlate_select_group* no arquivo *ofproto-dpif-xlate.c*, de modo a gerar um número aleatório que representa a probabilidade de escolha de um *bucket*. Essa probabilidade é usada para selecionar uma porta de saída, ou direcionar o tráfego para outro tipo de grupo, como o *Fast Failover*.

Após modificações no código fonte, foi necessária a portabilidade do OvS para a plataforma MIPS, ou seja, compilação cruzada de bibliotecas e código fonte, para a implantação do encaminhador no sistema operacional OpenWRT. Aqui podemos destacar a modularidade do OpenWRT, que uma vez instalado no equipamento, não se faz necessário sua reinstalação por completo a cada nova versão modificada do Open vSwitch, sendo requerido apenas a reinstalação do componente modificado, através do sistema de gerenciamento de pacotes presente no OpenWRT.

Detalhes técnicos, como *scripts* para inicialização das regras e ações OpenFlow referentes às tabelas de grupos estão disponíveis no Apêndice C. Demais modificações no código fonte do Open vSwitch, para possibilitar a implementação do Select, estão descritos no Apêndice D.

5.4 Considerações do Capítulo

Algumas considerações podem ser feitas sobre a implementação do protótipo do Crops, desde a escolha do *hardware* até o mecanismo de encaminhamento em *software*. O *hardware* escolhido tem excelente relação custo/benefício, além de ser extremamente flexível no que diz respeito à substituição de seu *firmware* original. Apesar de certas modificações serem obrigatórias para pleno funcionamento da proposta no ambiente escolhido, o sistema operacional OpenWRT apresentou-se como uma solução viável, com característica modular e código fonte adaptável, desde o *driver* dos componentes do *hardware* até o sistema de configuração de interfaces de rede.

O procedimento de compilação cruzada também foi um ponto chave para a proposta, mostrando-se dinâmica no que diz respeito à construção do ambiente e, posteriormente, do *firmware*, graças às ferramentas presentes no *framework* de desenvolvimento do SO OpenWRT. O encaminhador OvS também mostrou-se uma opção viável, pois apesar de não possuir todas as premissas definidas no protocolo OpenFlow versão 1.2 ou posterior, a organização de seu código fonte tornou sua implementação factível.

O próximo Capítulo contextualiza os experimentos realizados sobre a proposta do Crops, bem como seus resultados e comentários pertinentes à conclusão do projeto.

Capítulo 6

Testbed para Avaliação Experimental

O principal objetivo desta seção é investigar e analisar a proposta do Crops. Para tal foram construídos cenários que permitem explorar a programabilidade do comutador de código aberto programável.

Destaca-se que, conforme mencionado anteriormente, o objetivo da proposta não é obter melhor desempenho quando comparado com plataformas fechadas. No entanto, apenas para fins de aferição, esta seção também apresenta e discute os resultados desta análise.

Para melhor compreensão, este capítulo ficou estruturado da seguinte forma: na primeira seção são detalhadas as configurações dos equipamentos e os procedimentos metodológicos e ferramentas utilizadas na coleta dos resultados.

Na segunda seção são apresentados e discutidos os resultados obtidos no plano de controle, em especial a capacidade suportada de entradas na tabela de fluxo.

Na terceira seção são expostos os resultados relacionados ao plano de dados, avaliando o desempenho e a estabilidade de vazão da plataforma quando comparada à plataforma fechada utilizada nos equipamentos. Também nesta mesma seção, como prova de conceito, são apresentados e discutidos os resultados de um comutador estocástico de prateleira com características de balanceamento de carga e resiliência no plano de dados.

Por fim, são tecidos alguns comentários sobre os resultados obtidos.

6.1 Ambiente de Avaliação

A abordagem de avaliação da proposta de comutador de código fonte aberto foi efetuada sistematicamente, utilizando uma série de modelos de equipamentos de prateleira de baixo custo, mais detalhados na Tabela 6.1.

Foram utilizados dois *hosts* físicos, um denominado *host* de origem, responsável pela geração de pacotes de rede, e outro *host* de destino, encarregado pela recepção dos dados, conforme descrito abaixo:

Tabela 6.1: Comutadores utilizados e suas especificações.

RB	Modelo	CPU	Mem	Interfaces
1	RB750GL	MIPS 24Kc V7.4 @400MHz	64 MB	5 Gigabit
2	RB2011iLS-IN	MIPS 74KcV4.12 @600MHz	64 MB	5 Fast, 5 Gigabit e 1 SFP
3	RB2011UAS-IN	MIPS 74KcV4.12 @600MHz	128 MB	5 Fast, 5 Gigabit e 1 SFP
4	RB450G	MIPS 24Kc V7.4 @680MHz	256 MB	5 Gigabit

- *Host* de origem: um PC Dell OptiPlex 7010 com um processador Intel Core i5-3570 3.40GHz, 8GB de RAM, equipado com duas interfaces de rede Gigabit, sendo uma Intel Gigabit PCI Express Ethernet I350-T4, com quatro portas Gigabit (utilizada no plano de dados), e uma Broadcom Gigabit PCI Express Ethernet Onboard, disco rígido de 500GB SATA 7200RPM, sistema operacional Linux Fedora Core 21 64bits.

- *Host* de destino: um Laptop Sony Vaio VPCSB25FB com um processador Intel Core i5-2410M 2.9GHz, 8GB de RAM, equipado com uma interface de rede Gigabit Realtek Onboard, disco rígido de 500GB SATA 5400RPM, sistema operacional Linux Fedora Core 21 64bits.

6.1.1 Medições

Para cada parâmetro a ser medido coletaram-se 30 amostras de 120 segundos de duração cada, nos diferentes cenários. Foi calculado o intervalo de confiança de 95% para essas amostras e este mostrou-se aceitável para garantir a confiabilidade do resultado as experimentações.

De modo a gerar o tráfego necessário de acordo com as demandas de cenários específicos nos ambientes citados, foi utilizada a ferramenta iperf3 [Jon Dugan et al. 2014], capaz de gerar tráfego com pacotes TCP ou UDP, este último a uma taxa específica. No ambiente de experimentação, os *hosts* não foram o gargalo dos testes, fato comprovado através de testes efetuados fazendo-se uma ligação direta entre as interfaces de rede de cada um dos *host*.

Na avaliação do comutador estocástico, o tráfego foi capturado por meio da ferramenta *tshark*[1] a partir do *host* de destino.

Finalmente, para medir o percentual de uso de CPU das RouterBoards, a ferramenta de monitoramento *mpstat*[2] foi utilizada através da conexão ssh (*shell* seguro) disponível no *switch*.

[1]https://www.wireshark.org/docs/man-pages/tshark.html
[2]http://www.linuxcommand.org/man_pages/mpstat1.html

Todos os testes e coletas de dados foram sistematizados por meio de *script*, e posteriormente armazenados em arquivos texto, convertidos para o formato CSV (*comma-separated values* ou valores separados por vírgula) para análise e geração dos resultados.

A cada rodada do experimento com preenchimento da Tabela de Fluxo, o *switch* pára de responder quando sua tabela está totalmente cheia. Isso se deve ao modelo de implementação do OvS que, por padrão, não define um limite para as entradas na Tabela de Fluxo do protocolo OF, ocasionando em uma total utilização da memória RAM do equipamento, local onde a tabela é acomodada. Portanto, a cada rodada de testes, foi necessário a reinicialização da RouterBoard, e com isso a Tabela de Fluxo do equipamento foi totalmente esvaziada antes do início do novo teste.

A ferramenta *iperf* não possui um método para limitar a vazão de dados utilizando o protocolo TCP; portanto, quando requisitada tal limitação, o experimento foi conduzido através do uso do protocolo UDP.

6.2 Plano de controle: Tabela de Fluxo

A quantidade de entradas na Tabela de Fluxo de um *software switch* é limitada apenas pelos recursos de *hardware* (ex. quantidade de SDRAM disponível nos modelos de RouterBoard utilizados) disponíveis na plataforma onde são executados [Pfaff 2013].

A topologia responsável pela análise da Tabela de Fluxo é mostrada na Figura 6.1, onde apenas um *host* é responsável por inserir as regras em um *switch* (SW) através do plano de controle (conexão representada pela linha pontilhada preta).

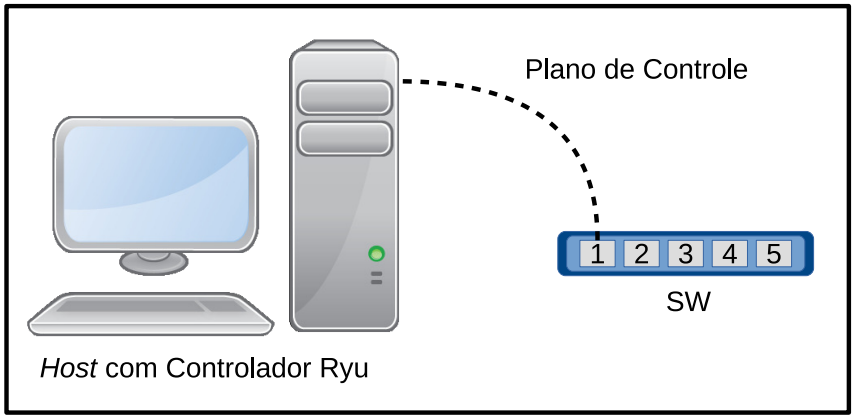


Figura 6.1: Topologia utilizada para avaliar a Tabela de Fluxo

Para avaliar a quantidade de entradas possíveis na Tabela de Fluxo do Open vSwitch embarcado, foi desenvolvida uma aplicação em um *host* utilizando o controlador OpenFlow Ryu, chamada **fill-table.py**, brevemente descrita no Algoritmo 1.

A aplicação cria campos de regras randômicos através da função *randomizedFields* (linha 3), gerando campos de *match* do OpenFlow de acordo com a Tabela 6.2, sem sobreposição dos mesmos (linha 4). Após a geração das regras, a aplicação as instala usando

Algorithm 1 Aplicação no controlador Ryu: fill-table.py

```
1: rulesArray ← [ ]
2: while true do
3:    rules ← randomizedFields(srcPort, dstPort, ...)
4:    if not foundInArray(rules, rulesArray) then
5:       addFlowtoSwitch(rules)
6:       rulesArray.append(rules)
7:    end if
8: end while
```

a função *addFlowtoSwitch* (linha 5), que envia uma mensagem de *FlowMod* (mensagem que modifica o estado do *switch* e de sua Tabela de Fluxo), e adiciona a regra em um vetor (linha 6) para que possa ser comparado com a próxima regra gerada, e verificar se houve alguma sobreposição na mesma.

A aplicação é encerrada manualmente, logo após preencher a Tabela de Fluxo do Open vSwitch por completo, e consequentemente, quando toda a memória do *switch* estiver preenchida e o mesmo pára de responder à aplicação. No final, o tamanho do vetor representa a capacidade máxima de entradas na Tabela de Fluxo do equipamento.

A Tabela 6.2 apresenta o tamanho das regras de fluxo em bits, correspondente a um conjunto de aplicações. Por exemplo, uma aplicação de comutação em camada 2 (bridge) ou um *firewall*/roteador em camada 3 usando os protocolos IPv4 ou IPv6.

Tabela 6.2: *Matches* de aplicação na Tabela de Fluxo

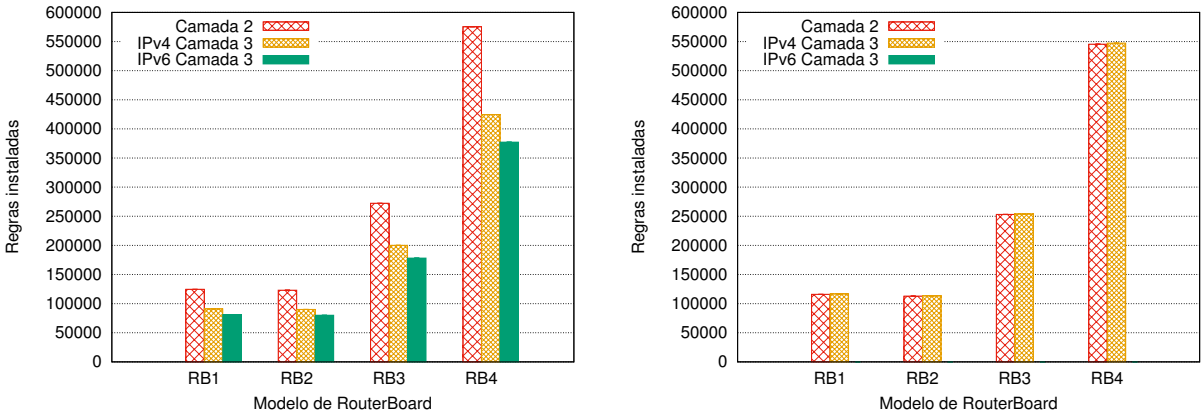
Aplicação	**Campos de *match* do OF**	**Tamanho em Bits**
Camada 2	OF_IN_PORT, OF_ETH_DST, OF_ETH_SRC	128
IPv4 Camada 3	OF_IN_PORT, OF_ETH_DST, OF_ETH_SRC, OF_ETH_TYPE, OF_VLAN_VID, OF_IP_PROTO, OF_TCP_SRC, OF_TCP_DST, OF_IPV4_SRC, OF_IPV4_DST	261
IPv6 Camada 3	OF_IN_PORT, OF_ETH_DST, OF_ETH_SRC, OF_ETH_TYPE, OF_VLAN_VID, OF_IP_PROTO, OF_TCP_SRC, OF_TCP_DST, OF_IPV6_SRC, OF_IPV6_DST	453

6.2.1 Capacidade de entradas na Tabela de Fluxo

Durante todas as rodadas dos experimentos com Tabela de Fluxo, os modelos de RouterBoard comportaram-se de maneira similar: todas as regras foram inseridas em um espaço de tempo relativamente parecido, e quando sua tabela estava completamente preenchida, o *switch* parava de responder ao controlador. Em seguida, o sistema operacional embar-

cado encerrava o processo responsável pelo controle do OpenFlow, reação que também era esperada, por conta dos recursos exauridos do sistema. Esse foi o momento considerado como Tabela de Fluxo cheia, ou seja, completamente preenchida de regras e ações.

Como os *switches* possuem características físicas diferentes, mais especificamente a quantidade de memória RAM, houve uma certa variação da quantidade de regras inseridas em cada equipamento. O mais importante a observar foi que, conforme Figura 6.2, o crescimento do número de regras nas Tabelas de Fluxo dos equipamentos não foi linear em relação a quantidade total de memória RAM.



(6.2.1) *Tabela de Fluxo: OpenWRT* (6.2.2) *Tabela de Fluxo: RouterOS*

Figura 6.2: Quantidade máxima de entradas suportadas por tipo de aplicação.

Como ilustrado na Figura 6.2.1, comparando os modelo RB3 (128MB RAM) e RB2 (64MB RAM) usando o Crops, nota-se que a RB3 possui o dobro da quantidade de memória em relação a RB2. Porém, de acordo com o resultado, uma aplicação que faz *match* em camada 2 suportaria 270.000 entradas na RB3, enquanto na RB2 conseguiria 124.500 entradas, que é menos da metade. Tal diferença de quantidade de regras suportadas pela RB3 foi maior em aproximadamente 8% (21000 regras) do que o dobro (249.000 regras) das regras suportadas na RB2. E esta diferença é ainda maior, quando comparamos a RB4 (256MB RAM) com a RB3 (128MB RAM), chegando a 13% maior que o dobro.

Esse fato pode ser explicado pelo motivo do sistema operacional OpenWRT, juntamente com o *switch* virtual Open vSwitch, consumirem aproximadamente 17MB do total da memória RAM do dispositivo embarcado, quando não há nenhuma regra instalada. O conjunto OpenWRT + OvS ocupa uma quantidade proporcionalmente diferente em cada equipamento; nas RB1 e RB2, o conjunto corresponde a 26,5% da RAM disponível, na RB3 13,2% e na RB4 6,6% do total do dispositivo.

Portanto, quanto menor for o tamanho da memória RAM do dispositivo, maior é o espaço proporcional utilizado pelo conjunto OpenWRT + OvS, e consequentemente, menor será a capacidade líquida total de entradas na Tabela de Fluxo.

Já na Figura 6.2.2, que mostra o experimento utilizando o RouterOS, notamos uma certa constância nos resultados entre a Camada 2 e a Camada 3 com Ipv4, em todos os modelos de RouterBoard. Como a implementação do OpenFlow do RouterOS é de código fonte fechado (proprietária), apenas podemos supor que a mesma utiliza uma Tabela de Fluxo com a quantidade e tamanho de campos estáticos.

Outra característica que se repete neste resultado é a relação entre quantidade de entradas na Tabela de Fluxo e total de memória RAM do dispositivo, sendo que a quantidade de entradas é sempre maior em relação a proporção de memória disponível no dispositivo, devido ao mesmo motivo exposto anteriormente: quantidade de memória RAM utilizada pelo sistema operacional quando não há entradas na Tabela de Fluxo.

Vale lembrar que, como a implementação do OpenFlow no RouterOS suporta apenas instruções da versão 1.0, sem suporte a entradas na Tabela de Fluxo utilizando o protocolo IPv6, não foi possível obter resultados com Camada 3 usando o protocolo citado.

Comparando as duas implementações dispostas nas Figuras 6.2.1 e 6.2.2, podemos ver que o Crops sobressai-se nos resultados com Camada 2, conseguindo acomodar mais regras em cerca de 7,4% na RB1, 9% com a RB2, 7,5% na RB3 e 5,5% na RB4.

Já na Camada 3 com IPv4, o RouterOS conseguiu melhores resultados, conseguindo superar a quantidade de entradas na Tabela de Fluxo na RB1, RB2, RB3 e RB4 em 28%, 25,7%, 26,8% e 28,9%, respectivamente. Porém é importante enfatizar que o Crops suporta OpenFlow versão 1.3, e sua Tabela de Fluxo foi projetada para acomodar regras e ações referentes à essa versão do protocolo, o que pode explicar uma maior ocupação de espaço pelas regras utilizando mais campos, como no caso do experimento com Camada 3 e IPv4.

Isto também reforça a vantagem do Crops sobre o RouterOS de possuir a capacidade de trabalhar com o OF versão 1.3, e consequentemente, dando suporte a utilização de regras com o protocolo IPv6.

6.3 Plano de Dados

Na avaliação do plano de dados, foram utilizadas duas abordagens: i) análise de **máxima performance** e **estabilidade de vazão**, e ii) validação das características das tabelas de grupos presentes no OpenFlow a partir da versão 1.2. Para isso, foram utilizadas duas topologias distintas, sendo que cada topologia está associada a uma abordagem, conforme descrito no decorrer do Capítulo.

Máxima performance refere-se a quantidade máxima de tráfego possível, por um curto período de tempo, suportada pelo *switch*. O experimento avaliou o comportamento do comutador de acordo com o tamanho dos pacotes, seguindo especificações contidas na RFC2544 [Bradner and McQuaid 1999], que sugere os seguintes tamanhos para pacotes Ethernet: 64, 128, 256, 512, 1024, 1280, 1518 bytes.

Estabilidade de vazão corresponde ao comportamento do comutador em relação a um determinado período de tempo, medindo a capacidade de vazão do equipamento em relação ao consumo de CPU, baseando-se no melhor resultado no teste de **máxima performance** proposto anteriormente. Neste caso, foi utilizado o tamanho máximo do pacote Ethernet segundo a RFC2544 citada anteriormente, que corresponde a 1518 bytes.

A topologia referente à análise da máxima performance e estabilidade de vazão é descrita na Figura 6.3, onde um *host* origem é responsável pela geração de pacotes, e um *host* destino é responsável pela recepção e análise do tráfego. Apenas um *switch* com interfaces Gigabit foi utilizado a cada rodada para interconectar os dois *hosts*, tornando a topologia mais simples e evitando uma possível contaminação do resultado por parte da latência adicional de *switches* cascateados.

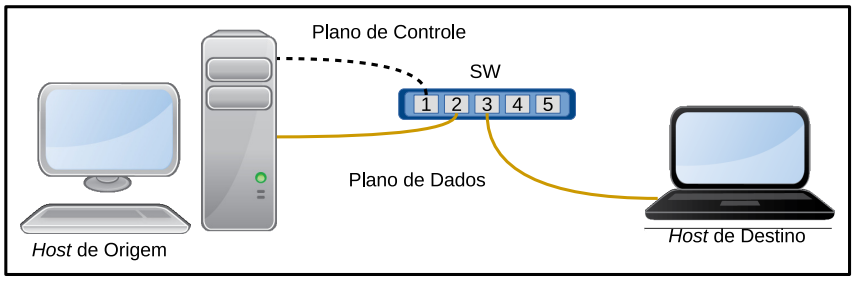


Figura 6.3: Topologia utilizada para avaliar a performance dos equipamentos

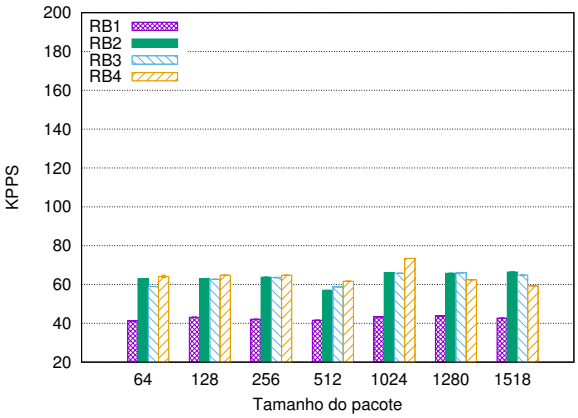
Neste cenário, as regras de encaminhamento foram instaladas pró-ativamente em cada *switch*, com intuito de apenas executar o encaminhamento de pacotes entre os dois *hosts*. Para isso, foi utilizado o controlador Ryu com uma aplicação OF capaz de inserir as regras e ações necessárias para o encaminhamento de pacotes de uma porta do *switch* diretamente para outra porta.

6.3.1 Desempenho do Plano de Dados

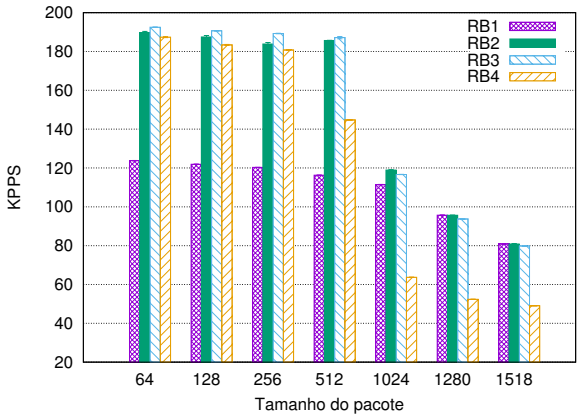
Cada equipamento utilizado nos experimentos possui uma capacidade específica para processamento de pacotes de rede. Essa medida é definida em PPS (pacotes por segundo), e é determinante para obtenção de velocidades de transmissão de pacotes próximas da máxima possível (*wire-speed*).

Ao transformar um comutador de pacotes, que antes utilizava seu ASIC para encaminhar o tráfego de rede, em um comutador de pacotes SDN por meio de um *switch* em *software*, perde-se parte de sua performance em detrimento da programabilidade do plano de dados. Analisar o desempenho do Crops torna-se então inevitável, ao mesmo tempo que compará-lo à implementação original provida pelo fabricante no RouterOS.

Com isso, a capacidade nominal de processamento de pacotes do encaminhador tende a diminuir, uma vez que a CPU agora é encarregada desse processo, e não mais o *switch chip*.

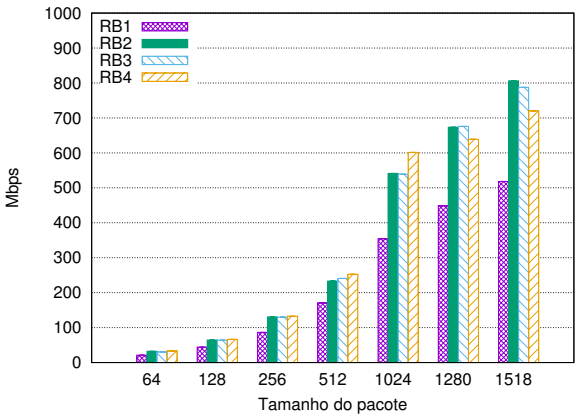


(6.4.1) *Vazão em Kpps: OpenWRT*

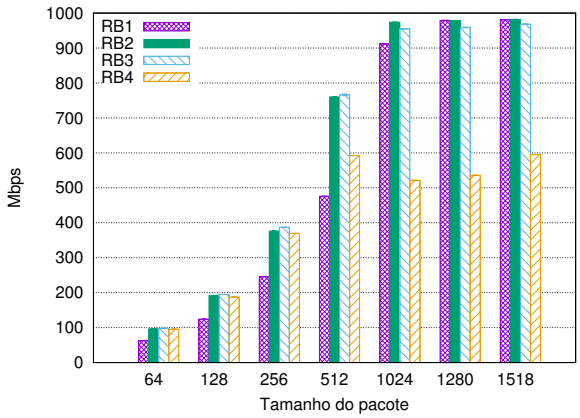


(6.4.2) *Vazão em Kpps: RouterOS*

Figura 6.4: Vazão máxima obtida em Kpps utilizando protocolo UDP.



(6.5.1) *Vazão em Mbps: OpenWRT*



(6.5.2) *Vazão em Mbps: RouterOS*

Figura 6.5: Vazão máxima obtida em Mbps utilizando protocolo UDP.

Considerando tamanho de pacotes variados, conforme especificado no Capítulo 5, cada *switch* exibiu um comportamento singular no experimento com o Crops, como podemos ver nas Figuras 6.4.1 e 6.5.1:

- A RB1, equipamento com menor poder de processamento, obteve taxas de KPPS (KiloPPS) e Mbps condizentes à sua capacidade, oscilando entre os diversos tamanhos de pacotes e conseguindo melhor resultado com pacotes de 1280 bytes.
- RB2 e RB3, apesar de compartilharem configurações parecidas (família do CPU e velocidade em MHz), diferenciam-se na versão do RouterBOOT, responsável pelo gerenciamento do *hardware* do *switch* (endereçamento de memória, portas Ethernet, entre outros); nesse caso, a RB2 possui a versão mais recente do *firmware*, o que pode explicar o melhor resultado para a RB2 em quase todos os tamanhos de pacotes, perdendo apenas com pacotes de 512 e 1280 bytes.

- A RB4 teve comportamento atípico, conseguindo melhores resultados com pacotes menores que 1280 bytes, e degradando o desempenho a partir desse limite; aqui podemos destacar que, além do RouterBOOT ser o mais antigo das versões avaliadas, seu *switch chip* também é diferente dos demais modelos.

Comparando Crops e RouterOS, percebe-se que o desempenho do SO original do equipamento possui uma certa vantagem sobre o novo SO *open source* em quase todos os equipamentos, com exceção da RB4, que obteve melhor desempenho com pacotes acima de 1024 bytes, conforme exibido nas Figuras 6.4 e 6.5. Este comportamento da RB4 mostra que a solução de código aberto pode atingir resultados superiores ao sistema original RouterOS, em determinadas circunstâncias.

Uma hipótese provável seria que um amadurecimento nas modificações no *driver* do *switch chip*, possa trazer mais desempenho à proposta de comutador *open source*, assim como novas funcionalidades. Como a proposta original do trabalho não almeja o alto desempenho, possíveis otimizações e implementações de novas funcionalidades serão discutidas com mais detalhes no Capítulo 6.4.

Outro aspecto investigado, ainda sobre performance, foi a estabilidade de vazão por um determinado período de tempo, ou seja, como seria o comportamento do Crops com pacotes de maior tamanho possível especificado em [Bradner and McQuaid 1999], ou seja, 1518 bytes. Neste experimento, foi escolhida a RB2, por ser o *switch* com melhor resultado para o tamanho de pacote selecionado, conforme denotado no experimento anterior.

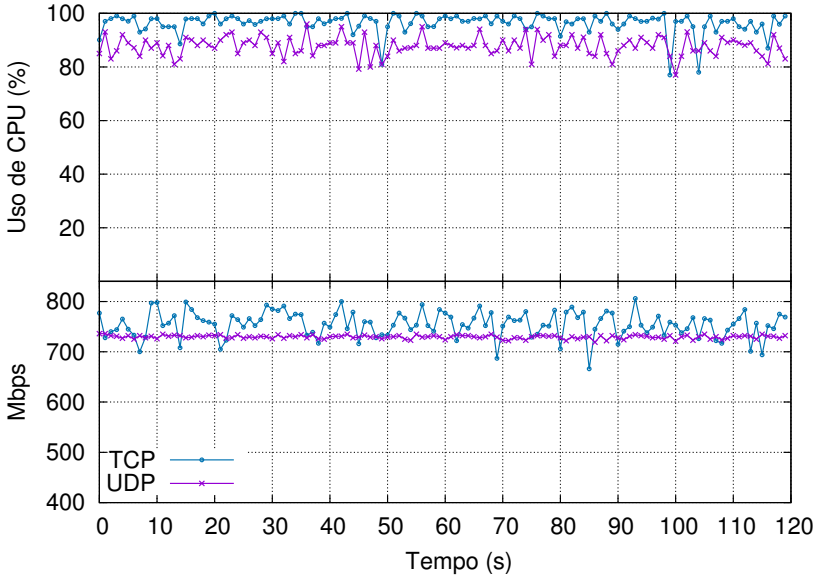


Figura 6.6: Estabilidade de vazão em um período de tempo específico.

A Figura 6.6 ilustra o consumo de CPU em razão da vazão, em um período de 120 segundos, utilizando tráfego de dados nos protocolos TCP e UDP em um único sentido, do *Host* de Origem para o *Host de Destino*. O protocolo TCP obteve maior vazão média, ficando com desempenho de 3,5% superior quando comparado ao protocolo UDP, porém

o consumo de CPU em TCP ficou 10% acima quando comparado com o protocolo UDP, mostrando-se menos eficiente em relação à utilização de recursos versus vazão.

Este comportamento em relação aos dois protocolos já era esperado, uma vez que o protocolo TCP utiliza pacotes de confirmação (ACK) enviados de volta ao transmissor, ocupando parte da banda total disponível (banda total = *upload* + *download*, e neste caso, o ACK ocupa a parte de *upload*), causando as oscilações que podem ser vistas no gráfico deste experimento, sendo menos eficiente que o protocolo UDP.

6.3.2 Comutador estocástico

A segunda topologia, conforme Figura 6.7, foi projetada para examinar os diferentes caminhos, variando o número de saltos, a fim de observar o comportamento de um comutador estocástico sobre essas condições. Cada caminho possui um peso diferente, estaticamente atribuído através das regras inseridas previamente em cada *switch*.

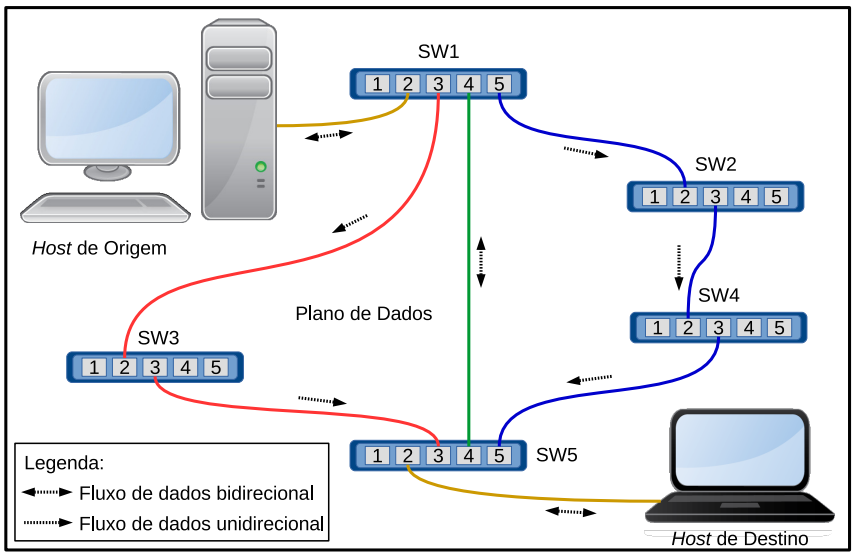


Figura 6.7: Topologia utilizada para avaliar a funcionalidade da semântica Select + Fast Failover

O *switch* **SW1** é responsável por executar as semânticas *Select* e *Fast Failover*, enquanto os demais apenas possuem regras de encaminhamento simples, direcionando o tráfego de uma porta a outra, conforme setas direcionais indicativas da Figura 6.7. No *switch* **SW5** foram instaladas regras do protocolo OpenFlow que fazem os pacotes de rede voltarem exclusivamente pela porta 4, que está ligada diretamente ao *switch* **SW1**. Por ser o caminho mais curto, apenas esse foi escolhido para o tráfego bidirecional entre os dois *hosts*.

Ainda de acordo com a Figura 6.7, foram definidos os parâmetros dos grupos referente a semântica *Select*, definindo o **Grupo 1** com 50% do tráfego de dados saindo pela Porta 4 (caminho com menos saltos), 30% pela Porta 3 (caminho intermediário), e 20% do tráfego

remanescente saindo pela Porta 5 (caminho mais longo). Já o **Grupo 2** foi definido com os seguintes parâmetros: 60% do tráfego direcionado para a Porta 4, e 40% para a Porta 3.

Para explorar a característica de resiliência do *Select*, definimos regras que direcionavam o tráfego de dados para o primeiro *bucket* ativo, ou seja, **Grupo 1** definido anteriormente. Em caso de falha da Porta 5, todo o tráfego seria direcionado ao próximo *bucket* ativo, nesse caso o **Grupo 2**. Por fim, se a Porta 3 falhasse, todos os dados seriam encaminhados para a Porta 4 diretamente.

No cenário de avaliação do comutador estocástico, foram instaladas regras pró-ativas diretamente nos *switches*, definindo os grupos e suas prioridades, além do encaminhamento simples entre portas, dispensando assim o uso de um controlador.

6.3.3 Avaliação do comutador estocástico

A avaliação do balanceamento de carga através do uso de um comutador estocástico foi aferida em duas etapas: primeiro com o protocolo TCP, sem limitação da vazão dos dados, e em seguida com o protocolo UDP, limitando a vazão, conforme detalhado abaixo.

Utilizando o protocolo TCP, obteve-se o desempenho máximo possível do *switch* usado no experimento. Uma vez que a solução foi implementada em modo usuário do OvS, durante o decorrer do experimento o consumo de CPU ficou em torno de 100%. Porém, como o objetivo não era atingir a máxima performance, e sim balanceamento de carga entre os *links* conforme estipulado na configuração das regras inseridas na Tabela de Fluxo, o resultado foi condizente com o experimento.

Conforme podemos ver na Figura 6.8, houve uma notável oscilação nas linhas, coloridas propositalmente em concordância com a Figura 6.7 mostrada anteriormente, para facilitar a compreensão dos caminhos associados às portas.

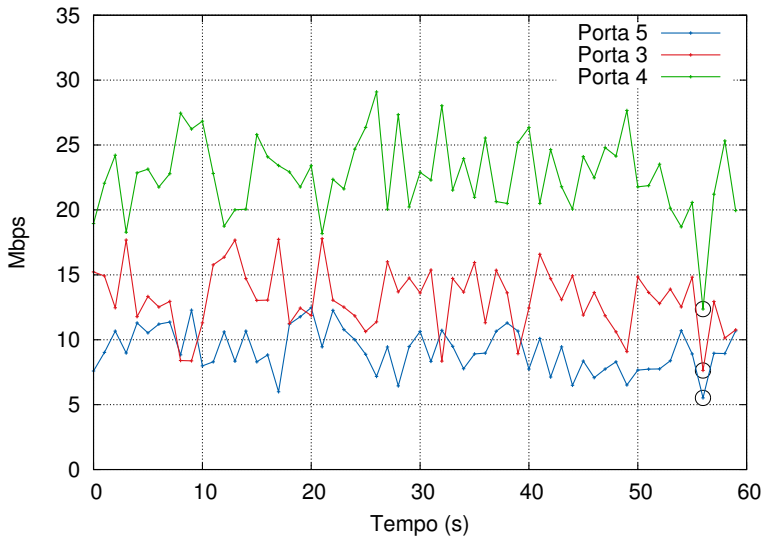


Figura 6.8: Balanceamento estocástico usando protocolo TCP

Porém, além das oscilações, podemos notar também certas quedas de tráfego simultaneamente nas três conexões, como pode ser visto nos pontos destacados no gráfico. Tal comportamento é associado ao elevado consumo de CPU, avaliado através da ferramenta *mpstat*, que priva o sistema operacional de recursos básicos como tempo de interrupção, ocasionando uma perda de até 43% da capacidade total de encaminhamento de pacotes, que ficou na média de 45Mbps somando as três conexões.

Com o protocolo UDP foi possível limitar da velocidade máxima de pacotes gerados pela ferramenta *iperf*, eliminando a influência do consumo de CPU do *switch* no resultado do experimento.

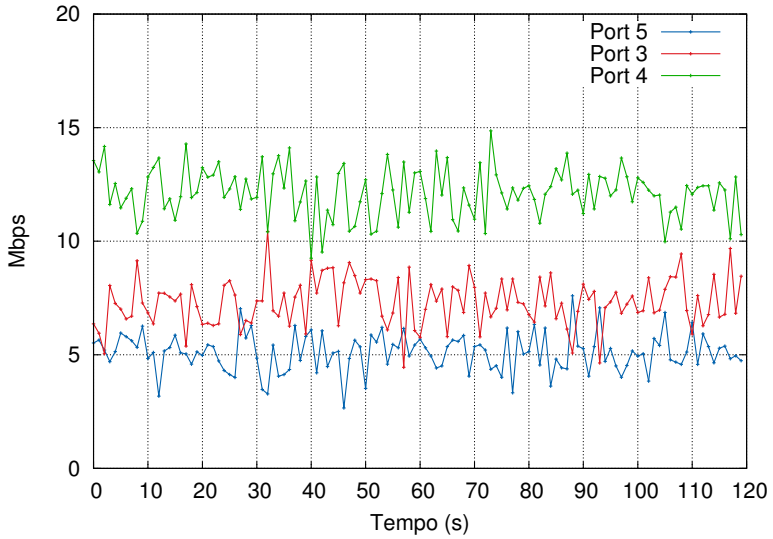


Figura 6.9: Balanceamento estocástico usando protocolo UDP e limitação na vazão

A Figura 6.9 mostra o resultado com banda total limitada em 25Mbps, que corresponde a um pouco acima da metade do desempenho total do equipamento selecionado, deixando o consumo de CPU em torno de 70%.

Nota-se, ainda na Figura 6.9, que mesmo sem a influência do consumo de CPU, houve uma grande variação na vazão de cada link atribuído à semântica *Select*, exibido através das oscilações das linhas do gráfico. Isso pode ser explicado pela natureza randômica da solução desenvolvida no código fonte do OvS. Contudo, na média, a escolha de cada *bucket* é respeitada de acordo com seu respectivo peso atribuído. No caso do experimento usando protocolo UDP, a distribuição entre as portas do *switch* **SW1** obedeceu os pesos configurados, que foi 49,3% na porta 4, 29,8% na porta 3 e 20,9% na porta 5.

6.3.4 Balanceamento de carga com resiliência

Após os resultados dos experimentos com balanceamento de carga, são apresentados os dados do experimento que acrescenta resiliência ao plano de dados sem a necessidade de consulta a um controlador, utilizando a semântica *Select* em conjunto com a semântica

Fast Failover.

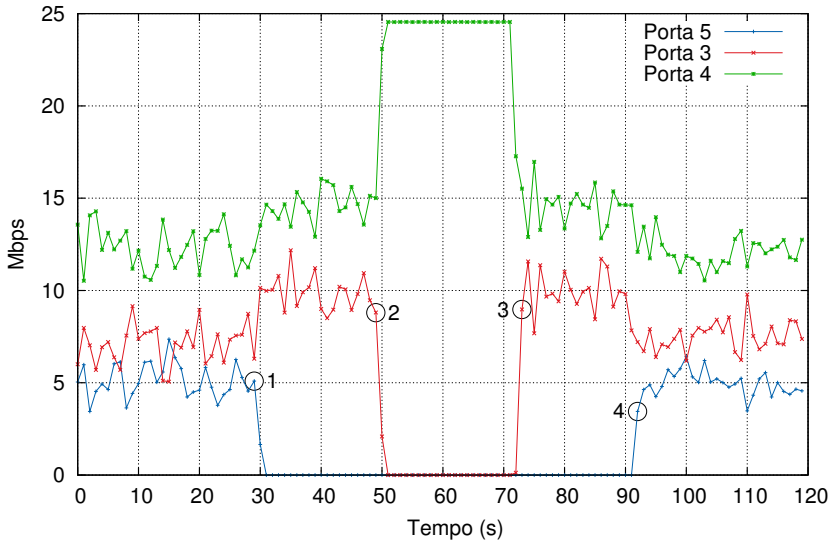


Figura 6.10: Balanceamento estocástico e resiliência.

A Figura 6.10 exibe o comportamento de um tráfego gerado do *host* de origem para o *host* de destino, utilizando o protocolo UDP e com a vazão limitada em 25Mbps, respeitando a topologia descrita na Figura 6.7. Assim como na Figura 6.9, a variação gerada no gráfico deve-se à natureza randômica do algoritmo de escolha do caminho implementado no código fonte do OvS.

Conforme exibido na Figura 6.10, o experimento inicia-se com os três caminhos disponíveis (Grupo 1). Para demonstrar a resiliência, no tempo 30 (círculo 1), uma falha foi introduzida, desconectando o cabo de rede da porta 5 do *switch* **SW1**, conforme ilustrado na Figura 6.11.

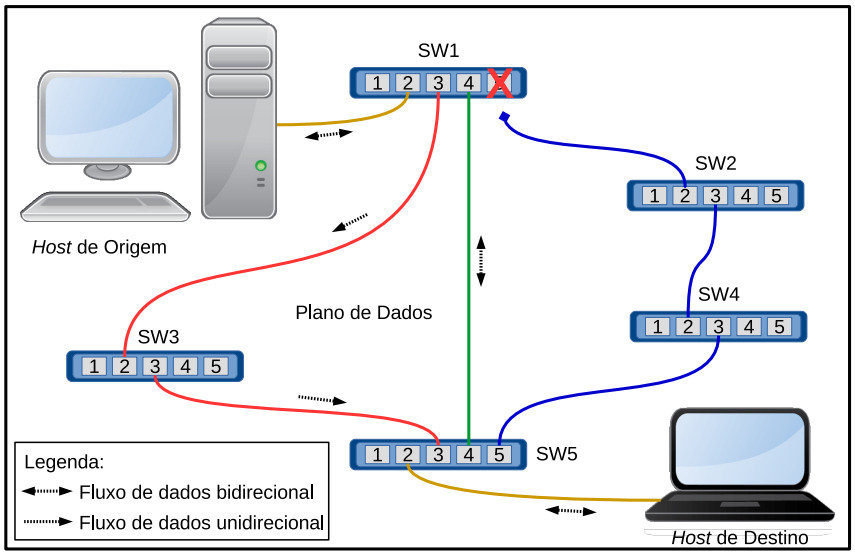


Figura 6.11: Resiliência – Fase 1: Removendo um dos caminhos da topologia.

Pode-se notar que o tráfego imediatamente sobe nas portas 3 e 4 (Grupo 2), enquanto o tráfego da porta 5 cai a zero (Figura 6.10), respeitando a primeira premissa definida em caso de falha em uma das portas.

Em seguida, no tempo 50 (círculo 2 da Figura 6.10), o cabo de rede foi desconectado da porta 3 do *switch* **SW1**, como podemos ver na Figura 6.12.

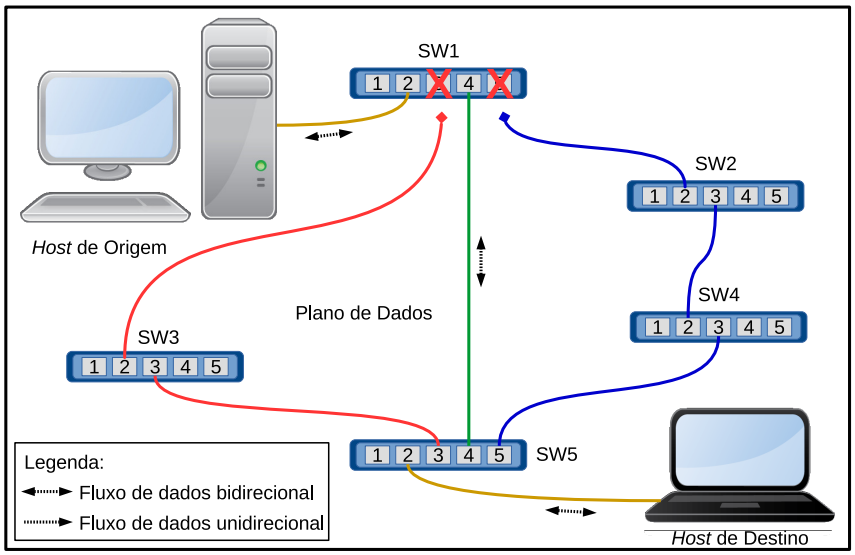


Figura 6.12: Resiliência – Fase 2: Removendo dois caminhos da topologia.

Nesse momento, todo tráfego é redirecionado para a porta 4 (Grupo 3), enquanto o tráfego na porta 3 também cai a zero, assim como ocorrido anteriormente com a porta 2, conforme visto na Figura 6.10. A partir de agora, o tráfego flui somente pela porta 4, porém a semântica Fast Failover continua a monitorar as demais portas inativas, mas não deixando que o tráfego seja encaminhado por estes caminhos, até que seja detectada a restauração da conexão, ou seja, o estado das portas.

Provada a capacidade de resiliência, resta agora demonstrar a capacidade de recuperação do estado anterior, referente ao comportamento estocástico definido na regra *Select*.

Para isso, continuando na Figura 6.10, no tempo 70 (círculo 3) o cabo é reconectado à porta 3 do *switch* **SW1**. O tráfego sobe novamente na porta 3, enquanto que o mesmo diminui ligeiramente na porta 4. Finalmente, no tempo 90 (círculo 4), o cabo é reconectado a porta 2, a topologia e o tráfego da rede voltam ao seu estado inicial, respeitando a semântica *Select* previamente definida, conforme esperado.

Cabe aqui lembrar que este experimento só pode ser executado devido às modificações efetuadas no *driver* do sistema operacional OpenWRT, responsável pelo gerenciamento das portas Ethernet do *switch* e de seu estado de conexão. Sem esta implementação, o tráfego continuaria sendo enviado para todas as portas definidas na regra da tabela de grupos, mesmo que estivessem sem conexão, resultando em perdas de pacotes no *host* destino e, consequentemente, invalidando o resultado.

6.4 Comentários Finais

Na primeira seção deste capítulo foram descritas as configurações dos equipamentos empregados no Testbed, juntamente com as ferramentas e procedimentos metodológicos utilizados. O objetivo desta seção foi ilustrar o ambiente utilizado para aferir a proposta.

Na segunda seção foram apresentados os resultados no plano de controle. O objetivo foi avaliar o comportamento da proposta na interação com a lógica de controle utilizada nas Redes Definidas por Software.

Por último, a terceira seção relata os resultados no plano de dados. O principal objetivo desta seção foi apresentar os resultados da avaliação de encaminhamento dos pacotes, variando o tamanho e o protocolo. Também nesta seção demonstramos como prova de conceito a programabilidade do comutador. Para tal, um comutador estocástico e tolerante a falhas é demonstrado.

Capítulo 7

Conclusão e Trabalhos Futuros

Na atualidade, a utilização de dispositivos embarcados que possibilitam a programabilidade do plano de dados limita-se apenas à poucos *switches bare-metal*. Porém tal programabilidade restringe-se apenas em implementações de aplicativos sobre protocolos de rede já existentes, como OpenFlow em alguns casos, ou somente protocolos legados (OSPF, BGP, IGMP, entre outros).

No caso de *switches* de baixo custo e performance limitada [Liberato et al. 2014], o OvS mostra-se como uma escolha condizente para componente de comunicação, harmonizando com a Figura 2.1. Tal abordagem é interessante para uma proposta de comutador de código fonte aberto e programável para redes – Crops, e seu uso em conjunto com um sistema operacional Linux modular e compacto, como o OpenWRT, capaz de ser embarcado em dispositivos de rede de baixo custo como as RouterBoards, formam o componente da solução proposta neste trabalho.

Em relação aos resultados dos experimentos realizados, no que diz respeito à performance do Crops, constatamos uma certa diminuição da vazão, no geral, quando comparado à solução proprietária original, o RouterOS. Porém, podemos ver também que em certos casos, o resultado ficou favorável à proposta de código aberto, evidenciando seu potencial para substituição do modelo fechado imposto pelo fabricante.

Ainda nos resultados, a perda de performance foi suplantada pela flexibilidade no plano de dados, com ganhos substanciais no que diz respeito à programabilidade utilizando características descritas pelo protocolo OpenFlow a partir de sua versão 1.2. Características estas que foram desenvolvidas pela proposta do projeto, utilizando um encaminhador virtual de código aberto, o Open vSwitch, e implantadas em equipamentos de baixo custo, porém com alta capacidade de inovação por meio de sua programabilidade. Sua funcionalidade superou as expectativas, por se tratar de um equipamento simples, de baixo custo e fácil aquisição no mercado atual. Espera-se também conseguir um melhor desempenho com outros modelos da mesma linha, porém com muito mais recursos embarcados que os disponíveis no atual trabalho.

Para trabalhos futuros, pretendemos expandir a programabilidade em equipamentos

embarcados para além do OpenFlow, abrangendo uma série de novas propostas que atualmente só existem em ambientes emulados (Mininet), como alterações no método de encaminhamento tradicional para uma abordagem sem tabelas, como [Martinello et al. 2014, Ramos et al. 2013], ou com mudança dinâmica entre os métodos [Vencioneck et al. 2014], utilizando o encaminhador em software OvS. Outra proposta, atualmente em desenvolvimento pelo Laboratório NERDS, utiliza o Crops como base para modificação da lógica de encaminhamento tradicional de rede baseado em endereçamento via protocolo IP, através do uso de NDN [Smetters and Jacobson 2009] em equipamentos embarcados. Há também a possibilidade de explorarmos a capacidade de *hardware* do equipamento embarcado, através de modificações em seu *driver* de código fonte aberto e alterando, por exemplo, sua lógica de encaminhamento baseado em tabelas de endereços MAC para uma abordagem sem tabelas (*table-less*), como em [Martinello et al. 2014, Ramos et al. 2013], conseguindo assim maior performance com operações simples de escolha de caminhos em redes.

Revisitar estas propostas já implementadas no "mundo virtual" e trazê-las para o "mundo real" é, em nosso ponto de vista, uma evolução natural para o desenvolvimento de novas abordagens no ambiente de redes de computadores. Ambiente este construído principalmente sobre equipamentos ativos de rede, capazes de interligar vários *hosts* físicos ou virtuais, da maneira mais eficiente e racional possível.

Referências Bibliográficas

[AR8 2011] (2011). *AR8327/AR8327N Seven-port Gigabit Ethernet Switch Data Sheet*. Atheros Communications, Incorporated. Rev. 1.0.

[Bradner and McQuaid 1999] Bradner, S. and McQuaid, J. (1999). Benchmarking methodology for network interconnect devices. http://www.ietf.org/rfc/rfc2544.txt.

[Casado et al. 2007] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: Taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12.

[Casado et al. 2012] Casado, M., Koponen, T., Shenker, S., and Tootoonchian, A. (2012). Fabric: A retrospective on evolving sdn. In *HotSDN*.

[Corradi et al. 2012] Corradi, A., Fanelli, M., and Foschini, L. (2012). Vm consolidation: A real case based on openstack cloud. *Future Generation Computer Systems*, pages –.

[CPqD 2014] CPqD, F. (2014). Openflow 1.3 software switch. https://github.com/CPqD/ofsoftswitch13.

[Dürr and Kohler 2014] Dürr, F. and Kohler, T. (2014). Comparing the Forwarding Latency of OpenFlow Hardware and Software Switches. Technical Report Computer Science 2014/04, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany.

[Fernandes 2013] Fernandes, E. L. (2013). Nox 1.3 oflib. https://github.com/CPqD/nox13oflib.

[Fernandes and Rothenberg 2014] Fernandes, E. L. and Rothenberg, C. E. (2014). OpenFlow 1.3 Software Switch. *Anais do 32º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos- SBRC*, pages 1021–1028.

[Greenberg et al. 2005] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. (2005). A clean slate 4d approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, 35(5):41–54.

[Gude et al. 2008] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). NOX: towards an operating system for networks. *Computer Communication Review*, 38(3):105–110.

[Handigol et al. 2012] Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., and McKeown, N. (2012). Reproducible network experiments using container-based emulation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies - CoNEXT '12*, page 253, New York, New York, USA. ACM Press.

[Heath 2002] Heath, S. (2002). *Embedded Systems Design*. Newnes.

[Huang et al. 2013] Huang, D. Y., Yocum, K., and Snoeren, A. C. (2013). High-fidelity switch models for software-defined network emulation. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, page 43.

[Jon Dugan et al. 2014] Jon Dugan, E., Elliott, S., Prabhu, K., and Jef Poskanzer, E. (2014). Iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool. https://github.com/esnet/iperf.

[Kempf et al. 2011] Kempf, J., Whyte, S., Ellithorpe, J., Kazemian, P., Haitjema, M., Beheshti, N., Stuart, S., and Green, H. (2011). OpenFlow MPLS and the open source label switched router. *2011 23rd International Teletraffic Congress (ITC)*, pages 8–14.

[Kohler et al. 2000] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. (2000). The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297.

[Kotronis et al. 2012] Kotronis, V., Dimitropoulos, X., and Ager, B. (2012). Outsourcing the routing control logic. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks - HotNets-XI*, pages 55–60, New York, New York, USA. ACM Press.

[Kreutz and Ramos 2014] Kreutz, D. and Ramos, F. (2014). Software-Defined Networking: A Comprehensive Survey. *arXiv preprint arXiv: ...*, pages 1–61.

[Lantz et al. 2010] Lantz, B., Heller, B., and McKeown, N. (2010). A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*, pages 1–6, New York, New York, USA. ACM Press.

[Lazaris et al. 2014] Lazaris, A., Tahara, D., Huang, X., Li, L. E., Voellmy, A., Yang, Y. R., and Yu, M. (2014). Tango: Simplifying sdn control with automatic switch property inference, abstraction, and optimization. In *CoNEXT*.

[Liberato et al. 2014] Liberato, A., Mafioletti, D. R., Spalla, E., Martinello, M., and Villaca, R. (2014). Avaliação de Desempenho de Plataformas para Validação de Redes Definidas por Software. In *CSBC 2014 - WPerformance*.

[Linux 2015] Linux, O. N. (2015). Open network linux. http://opennetlinux.org.

[Martinello et al. 2014] Martinello, M., Ribeiro, M. R. N., De Oliveira, R. E. Z., and De Angelis Vitoi, R. (2014). Keyflow: A prototype for evolving SDN toward core network fabrics. *IEEE Network*, 28(2):12–19.

[McKeown et al. 2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69.

[Mundada et al. 2009] Mundada, Y., Sherwood, R., and Feamster, N. (2009). An openflow switch element for click. In *in Symposium on Click Modular Router*.

[Networks 2015] Networks, C. (2015). Cumulus networks - open approach. http://cumulusnetworks.com/cumulus-linux/open-approach.

[ONF 2011] ONF, O. N. F. (2011). Openflow switch specification version 1.2 (wire protocol 0x03). https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf.

[ONF 2012] ONF, O. N. F. (2012). Openflow switch specification version 1.3.0 (wire protocol 0x04). https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[OpenDaylight 2013] OpenDaylight (2013). Opendaylight sdn controller platform (oscp):overview. https://wiki.opendaylight.org/view/OpenDaylight_SDN_Controller_Platform_(OSCP):Overview.

[OpenDayLight 2015] OpenDayLight (2015). Opendaylight "helium" – the rise of open sdn. "http://www.opendaylight.org/software".

[OpenWRT 2014] OpenWRT (2014). Openwrt. https://openwrt.org.

[Pfaff 2013] Pfaff, B. (2013). The open vswitch database management protocol. https://tools.ietf.org/html/rfc7047.

[Pfaff et al. 2009] Pfaff, B., Pettit, J., Koponen, T., Amidon, K., Casado, M., and Shenker, S. (2009). Extending Networking into the Virtualization Layer. In *8th ACM Workshop on Hot Topics inNetworks*, volume VIII, page 6. ACM.

[Pfaff et al. 2015] Pfaff, B., Pettit, J., Koponen, T., Jackson, E. J., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., and Casado, M. (2015). The design and implementation of open vswitch. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 117–130, Berkeley, CA, USA. USENIX Association.

[POX 2013] POX (2013). About pox - noxrepo. http://www.noxrepo.org/pox/about-pox.

[Project 2015] Project, O. C. (2015). Open Network Install Environment. https://github.com/opencomputeproject/onie/wiki.

[Ramos et al. 2013] Ramos, R. M., Martinello, M., and Esteve Rothenberg, C. (2013). SlickFlow: Resilient source routing in Data Center Networks unlocked by OpenFlow. In *38th Annual IEEE Conference on Local Computer Networks*, pages 606–613. IEEE.

[Rao 2015a] Rao, S. (2015a). Nox,the original openflow controller. http://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller.

[Rao 2015b] Rao, S. (2015b). Ryu, a rich featured open source sdn controller supported by ntt labs. http://goo.gl/WMbhSU.

[Rothenberg et al. 2010] Rothenberg, C. E., Nascimento, M. R., Salvador, M. R., and Magalhães, M. F. (2010). Openflow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. *Cad. CPqD Tecnologia, Campinas*, 7(1):65–76.

[RYU 2014] RYU, P. T. (2014). Ryu sdn framework using openflow 1.3. http://osrg.github.io/ryu-book/en/Ryubook.pdf.

[SDN 2015] SDN, P. W. B. (2015). White box switch os - pica8. http://pica8.org/white-box-switches/white-box-switch-os.php.

[Sherwood et al. 2010] Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G. (2010). Can the production network be the testbed? In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA. USENIX Association.

[Smetters and Jacobson 2009] Smetters, D. and Jacobson, V. (2009). Securing network content. *Palo Alto Research Center (PARC)*, pages 1–7.

[Subramanian et al. 2009] Subramanian, L., Leland, W. E., and Mahajan, R., editors (2009). *Eight ACM Workshop on Hot Topics in Networks (HotNets-VIII), HOTNETS '09, New York City, NY, USA, October 22-23, 2009*. ACM SIGCOMM.

[Tennenhouse and Wetherall 1996] Tennenhouse, D. L. and Wetherall, D. J. (1996). Towards an active network architecture. *SIGCOMM Comput. Commun. Rev.*, 26(2):5–17.

[Vencioneck et al. 2014] Vencioneck, R. D., Vassoler, G., Martinello, M., Ribeiro, M. R., and Marcondes, C. (2014). FlexForward: Enabling an SDN manageable forwarding engine in Open vSwitch. In *10th International Conference on Network and Service Management (CNSM) and Workshop*, pages 296–299. IEEE.

[vSwitch 2014] vSwitch, O. (2014). An Open Virtual Switch. http://openvswitch.org/.

[xDPd 2015] xDPd (2015). The extensible openflow datapath daemon (xdpd) - bringing innovation into the fast path. http://www.xdpd.org.

[Yaghmour et al. 2008] Yaghmour, K., Masters, J., Ben-Yossef, G., and Gerum, P. (2008). *Building Embedded Linux Systems*. O'Reilly Media.

[Yun 2012] Yun, S. (2012). An efficient TCAM-based implementation of multipattern matching using covered state encoding. *IEEE Transactions on Computers*, 61(2):213–221.

Apêndice A

Configurações de Rede

A.1 Associação entre portas físicas e VLANs: modelo RB2011UAS-IN

```
config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

config globals 'globals'
        option ula_prefix 'fdd1:0a00:7397::/48'

config interface 'sfp'
        option ifname 'eth0.11'
        option proto 'none'

#Definicao das VLANs
config interface port1
        option ifname 'eth0.1'
        option proto 'static'
        option ipaddr '192.168.1.1'
        option netmask '255.255.255.0'

config interface port2
        option ifname 'eth0.2'
        option proto 'none'

config interface port3
        option ifname 'eth0.3'
        option proto 'none'

config interface port4
```

```
        option ifname 'eth0.4'
        option proto 'none'

config interface port5
        option ifname 'eth0.5'
        option proto 'none'

config interface port6
        option ifname 'eth1.1'
        option proto 'none'

config interface port7
        option ifname 'eth1.2'
        option proto 'none'

config interface port8
        option ifname 'eth1.3'
        option proto 'none'

config interface port9
        option ifname 'eth1.4'
        option proto 'none'

config interface port10
        option ifname 'eth1.5'
        option proto 'none'

#Associacao das portas X vlans em switch0 (5 portas giga + 1 sfp)
config switch
        option name 'switch0'
        option reset '1'
        option enable_vlan '1'

config switch_vlan
        option device 'switch0'
        option vlan '1'
        option ports '0t 1'

config switch_vlan
        option device 'switch0'
        option vlan '2'
        option ports '0t 2'

config switch_vlan
        option device 'switch0'
        option vlan '3'
        option ports '0t 3'
```

```
config switch_vlan
        option device 'switch0'
        option vlan '4'
        option ports '0t 4'

config switch_vlan
        option device 'switch0'
        option vlan '5'
        option ports '0t 5'

config switch_vlan
        option device 'switch0'
        option vlan '11'
        option ports '0t 6'


#Associacao das portas X vlans em switch1 (5 portas fast)
config switch
        option name 'switch1'
        option reset '1'
        option enable_vlan '1'

config switch_vlan
        option device 'switch1'
        option vlan '5'
        option ports '0t 1'

config switch_vlan
        option device 'switch1'
        option vlan '4'
        option ports '0t 2'

config switch_vlan
        option device 'switch1'
        option vlan '3'
        option ports '0t 3'

config switch_vlan
        option device 'switch1'
        option vlan '2'
        option ports '0t 4'

config switch_vlan
        option device 'switch1'
        option vlan '1'
        option ports '0t 5'
```

Apêndice B

Portando o Crops em Equipamentos Embarcados

Iniciamos o projeto criando um ambiente para compilação cruzada (*cross-compiling*), uma vez que a arquitetura da RouterBoard baseia-se em MIPS, o que torna imprescindível a geração desse ambiente específico para que seja possível a construção do OpenWRT e seus pacotes. Tal ambiente foi criado em uma distribuição Linux x86, utilizando-se as ferramentas disponíveis na distribuição do código fonte do OpenWRT, como compilador e bibliotecas necessárias.

A próxima etapa foi a substituição do sistema operacional RouterOS, embarcado por padrão nas RouterBoards, pelo sistema aberto OpenWRT. Os passos abaixo descrevem esse processo:

1. Instalação e configuração dos serviços de BOOTP, TFTP e HTTP; deste modo, quando os *switches* inicializarem, poderão se conectar a este servidor, afim de receber a gravação definitiva do novo *firmware* na RouterBoard;

2. acesso dos equipamentos através de comunicação serial padrão RS-232 durante sua inicialização, alterando a sequência de *boot*, para que eles possam inicializar via BOOTP, ao invés da inicialização tradicional via memória *flash*;

3. inicialização da RouterBoard no sistema OpenWRT, assistida via console serial, onde o sistema é carregado por meio da memória RAM do equipamento, utilizando a imagem criada anteriormente durante o processo de compilação cruzada;

4. efetuação da gravação definitiva do OpenWRT na RouterBoard, com uso do aplicativo *wget2nand*, disponível no próprio sistema, executando assim a gravação do *firmware* na memória *flash* do equipamento.

A partir desse momento, o OpenWRT está embarcado na RouterBoard, e sua configuração deve ser definida de acordo com cada modelo de equipamento, visto que cada

qual possui suas próprias especificações e características. O acesso ao equipamento e suas configurações pode ser feito via linha de comando, com utilização do console serial, ou conexão *shell* seguro (ssh). Algumas configurações são compartilhadas por todos os modelos, entre elas configurações de VLAN e habilitação do protocolo OpenFlow, cujo procedimento é descrito a seguir:

1. Configuração das interfaces de rede do equipamento para funcionamento independente através de VLANs: cada porta é associada a uma VLAN que se comunica com o *switch chip*, por meio do aplicativo *swconfig* que determina qual VLAN é responsável pela(s) porta(s);

2. ativação do *switch* virtual específico para o experimento, neste caso o Open vSwitch;

3. definição das interfaces VLANs a serem associadas ao encaminhador e suas configurações, através dos arquivos de configuração apresentados no Apêndice A.

Após esses passos, os equipamentos estão aptos para funcionamento com OpenFlow e poderão ser utilizados na implementação do protótipo.

Apêndice C

Regras e Ações OpenFlow

As definições das regras na Tabela de Fluxo foram inseridas por meio de *scripts* gravados em cada *switch*, que executavam a ferramenta *ovs-ofctl* com suas relativas opções. Cada *datapath* é representado no *script* como *sw1*, *sw2*, *sw3*, *sw4* e *sw5*, correspondentes aos seus respectivos *switches* RouterBoard, detalhados a seguir.

C.1 *Script* em SW1

```
#!/bin/sh
#Grupo 1 (Select): Porta 4 = 50%, Porta 3 = 30%, Porta 5 = 20%
ovs-ofctl -O OpenFlow13 add-group sw1 group_id=1,type=select, \
    bucket=output:4,weight=5, \
    bucket=output:3,weight=3, \
    bucket=output:5,weight=2
#Grupo 2 (Select): Porta 4 = 60%, Porta 3 = 40%
ovs-ofctl -O OpenFlow13 add-group sw1 group_id=2,type=select, \
    bucket=output:4,weight=6, \
    bucket=output:3,weight=4
#Grupo 3 (All): Porta 4 = 100%
ovs-ofctl -O OpenFlow13 add-group sw1 group_id=3,type=all,bucket=output:4
#Grupo 4 (Fast Failover):
# * monitore Porta 5, direcionando trafego para Grupo 1;
# * monitore Porta 3, direcionando trafego para Grupo 2;
# * monitore Porta 4, direcionando trafego para Grupo 3;
ovs-ofctl -O OpenFlow13 add-group sw1 group_id=4,type=fast_failover,
    bucket=watch_port:5,group:1, \
    bucket=watch_port:3,group:2, \
    bucket=watch_port:4,group:3
#Encaminha fluxo da Porta 2 (Host Origem) para o Grupo 4
ovs-ofctl -O OpenFlow13 add-flow sw1 table=0,in_port=2,actions=group:4
#Encaminha fluxo da Porta 4 (Host Destino) para a Porta 2 (Host Origem)
ovs-ofctl -O OpenFlow13 add-flow sw1 table=0,in_port=4,actions=output:2
```

C.2 *Script* em SW2

```
#!/bin/sh
#Encaminha fluxo da Porta 2 (SW1) para a Porta 3 (SW4)
ovs-ofctl -O OpenFlow13 add-flow sw2 in_port=2,actions=output:3
```

C.3 *Script* em SW3

```
#!/bin/sh
#Encaminha fluxo da Porta 2 (SW1) para a Porta 3 (SW5)
ovs-ofctl -O OpenFlow13 add-flow sw3 in_port=2,actions=output:3
```

C.4 *Script* em SW4

```
#!/bin/sh
#Encaminha fluxo da Porta 2 (SW2) para a Porta 3 (SW5)
ovs-ofctl -O OpenFlow13 add-flow sw4 in_port=2,actions=output:3
```

C.5 *Script* em SW5

```
#!/bin/sh
#Encaminha fluxo da Porta 3 (SW3) para Porta 2 (Host Destino),
#modificando o campo TOS (Type of Service)
ovs-ofctl -O OpenFlow13 add-flow sw5 in_port=3, \
    actions=mod_nw_tos:0x04,output=2

#Encaminha fluxo da Porta 4 (SW1) para Porta 2 (Host Destino),
#modificando o campo TOS (Type of Service)
ovs-ofctl -O OpenFlow13 add-flow sw5 in_port=4, \
    actions=mod_nw_tos:0x08,output=2

#Encaminha fluxo da Porta 5 (SW4) para Porta 2 (Host Destino),
#modificando o campo TOS (Type of Service)
ovs-ofctl -O OpenFlow13 add-flow sw5 in_port=5, \
    actions=mod_nw_tos:0x0c,output=2

#Encaminha fluxo da Porta 2 (Host Destino) para Porta 4 (SW1)
ovs-ofctl -O OpenFlow13 add-flow sw5 in_port=2,actions=output=4
```

Apêndice D

Modificações no Open vSwitch

O Open vSwitch, em sua versão 2.3.0, não disponibiliza a funcionalidade da tabela de grupos referente ao Select, uma vez que sua implementação é opcional de acordo com [ONF 2011]. Modificações em algumas funções no código fonte foram necessárias para que o *switch* virtual se comportasse como um encaminhador estocástico, utilizando a semântica Select em conjunto com Fast Failover. As funções modificadas são mostradas abaixo:

D.1 Alteração do arquivo *ofproto-dpif-xlate.c*

```
// inicializa variavel checagem randomica
static bool is_srand_initialized = false;

static const struct ofputil_bucket *
group_best_live_bucket(const struct xlate_ctx *ctx,
                       const struct group_dpif *group,
                       uint32_t basis)
{
    uint32_t rand_num = 0, sum = 0;
    const struct ofputil_bucket *bucket = NULL;
    const struct list *buckets;

    // inicializa a semente randomica uma unica vez
    if (!is_srand_initialized) {
        srandom(time(NULL));
        is_srand_initialized = true;
    }

    // gera um numero randomico entre 1 e 10
    rand_num = (random() % 10) + 1;

    group_dpif_get_buckets(group, &buckets);
```

```
    LIST_FOR_EACH (bucket, list_node, buckets) {
        if (bucket_is_alive(ctx, bucket, 0)) {
           sum += bucket->weight;
            if (rand_num <= sum) {
                return bucket; // retorna este bucket
            }
        }
    }

    return bucket; // retorna NULL
}


static void
xlate_select_group(struct xlate_ctx *ctx, struct group_dpif *group)
{
    struct flow_wildcards *wc = &ctx->xout->wc;
    const struct ofputil_bucket *bucket;
    uint32_t basis;

    // Esta linha diz a logica do cache que cada pacote no fluxo
    // em questao deve ir para o espaco de usuario (SLOW PATH)
    ctx->xout->slow |= SLOW_CONTROLLER;

    basis=hash_bytes(ctx->xin->flow.dl_dst,sizeof ctx->xin->flow.dl_dst,0);
    bucket=group_best_live_bucket(ctx, group, basis);
    if (bucket) {
        memset(&wc->masks.dl_dst, 0xff, sizeof wc->masks.dl_dst);
        xlate_group_bucket(ctx, bucket);
    }
}
```