Implementação de um Comutador KeyFlow 10 Gbps para Redes Definidas por Software

Matheus de Souza[1], **Natan de Oliveira**[1], **Antônio M. Alberti**[1] **e Moisés R. N. Ribeiro**[2]

[1]ICT Lab., Instituto Nacional de Telecomunicações (INATEL),
Caixa Postal 05, Santa Rita do Sapucaí, Brasil.

[2]Laboratório de Telecomunicações (LABTEL), Departamento de Eng. Elétrica,
Universidade Federal do Espírito Santo, Av. Fernando Ferrari, 514,
Goiabeiras, Vitória, Brasil.

{matheus.souza,alberti}@inatel.br, natanelias@get.inatel.br

moises@ele.ufes.br

Abstract. *This article aims at presenting the design and implementation, using the NetFPGA-10G platform, of a stateless switch based on global labels (KeyFlow) for Software-Defined Networking (SDN) capable of operating at 10 Gbps. The validation of the prototype is also discussed, from the qualitative point of view, for verifying the correct forwarding of packages generated and received externally by a network acceleration card (NAC).*

Resumo. *Este artigo tem como objetivo apresentar a concepção e a implementação, na plataforma NetFPGA-10G, de um comutador baseado em rótulos globais e sem estados (KeyFlow) para Redes Definidas por Software (SDN) capaz de operar a 10 Gbps. A validação do protótipo também é discutida, do ponto de vista qualitativo, verificando o correto encaminhamento de pacotes gerados e recebidos externamente por uma placa aceleradora de rede.*

1. Introdução

Quando se fala em redes de computadores no contexto atual, é possível observar que uma boa parte das aplicações é limitada pela infraestrutura da rede. Essa infraestrutura é constituída por equipamentos proprietários e de alto custo, cujas arquiteturas possuem circuitos integrados dedicados ao processamento dos pacotes, dificultando, assim, a inovação originada da oferta de novos serviços que dependem de inovações na infraestrutura das redes subjacentes.

A falta de abertura e flexibilidade nos equipamentos de rede como *switches* e roteadores, os quais possuem uma camada de software fechada, traz como consequência um ambiente de rede que não consegue dar o suporte ideal para determinadas aplicações devido ao fato dos protocolos atuais terem sido desenvolvidos para atender principalmente demandas convencionais das redes locais, tornando assim bastante problemática a evolução das arquiteturas e a inovação originada da oferta de novos serviços [Rothenberg et al. 2010].

Nesse contexto, surgiu o conceito de Redes Definidas por Software (SDN - *Software Defined Networking*), que propõe a separação do plano de dados do plano de controle. O plano de controle é responsável por definir as rotas dos fluxos de pacotes dentro da

rede. O plano de dados tem a função de realizar o encaminhamento do fluxo de pacotes de acordo com regras definidas pelo plano de controle. Ao realizar essa separação, se torna necessário alterar somente a camada de controle para modificar as regras de definição do fluxo de pacotes, pois os equipamentos de rede podem compartilhar o mesmo plano de controle. Com isso é possível programar uma determinada rede de acordo com uma aplicação desejada. Além disso, com a introdução da SDN torna-se possível realizar testes e medições em redes em operação, fazendo com que os recursos disponíveis sejam melhor aproveitados [McKeown et al.].

O padrão mais difundido para SDN é o OpenFlow [Stallings 2013][Lara et al. 2014][Braun and Menth 2014]. Esse protocolo estabelece a interface de comunicação entre o controlador e os comutadores controlados, bem como o formato da tabela de encaminhamento. Apesar de bem difundido, o OpenFlow pode enfrentar problemas de escalabilidade devido à necessidade de manutenção de estado dos fluxos ativos nas tabelas de encaminhamento dos comutadores e a necessidade de comunicação com o controlador a cada novo fluxo. Uma proposta de complemento ao OpenFlow é o KeyFlow, cuja contribuição é o encaminhamento de pacotes sem a necessidade de consulta à tabelas de roteamento. O uso do KeyFlow apresenta uma melhora no desempenho da SDN em termos de latência, variabilidade na entrega de quadros e tempo de reconfiguração da rede [Martinello et al. 2014], além de redução do consumo de energia [Cercós et al. 2014]

Este artigo apresenta pela primeira vez a implementação de um comutador de rede utilizando o protocolo KeyFlow para a taxa de 10 Gbps, bem como testes de validação do hardware desenvolvido. A fim de contextualização, na Seção 2 é apresentada uma revisão de SDN e OpenFlow. Na Seção 3 é apresentado o modelo KeyFlow de SDN. Na Seção 4 são apresentados detalhes técnicos da implementação do comutador. Na Seção 5 é descrito o cenário de testes de validação. E, finalmente, na Seção 6 são apresentadas as conclusões do trabalho.

2. SDN - *Software Defined Networking*

O principal objetivo da arquitetura de uma SDN é realizar a separação física do plano de controle e do plano de encaminhamento de dados. O modelo atual trabalha com esses dois planos acoplados no próprio equipamento de rede. Ao realizar essa separação, os equipamentos terão como função única o encaminhamento de pacotes e a rede pode ser programada de acordo com uma aplicação específica, eliminando qualquer tipo de restrição relacionada com os protocolos de equipamentos proprietários [McKeown et al.].

A Figura 1 mostra uma visão geral da arquitetura de uma Rede Definida por Software. Nesse caso, os equipamentos de rede são vistos como uma camada única pelo controlador, que tem como função a tomada de decisão de encaminhamento de acordo com as informações contidas no cabeçalho de cada pacote. A camada de Aplicação se comunica com a camada de controle através de API's (*Application Programming Interfaces*) e enxerga a rede como um único *switch* lógico. No caso da comunicação entre o plano de controle e o plano de dados se torna necessário existir um protocolo padrão que possa levar controles entre os planos. Atualmente o protocolo mais utilizado para essa comunicação é o OpenFlow.

A infraestrutura de uma SDN tem como foco principal o encaminhamento dos

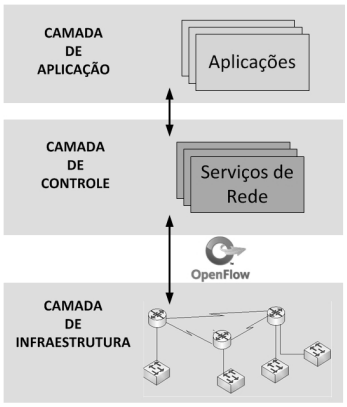


Figura 1. Arquitetura de uma Rede Definida por Software.

pacotes pelo plano de dados e o recebimento desses de acordo com as rotas dos fluxos de pacotes definidas pelo plano de controle. Sendo assim, o custo dos equipamentos utilizados no desenvolvimento pode ser reduzido se comparado aos equipamentos com software proprietários convencionais, tendo em vista que a estrutura do hardware empregado é bem mais simples.

2.1. OpenFlow

O protocolo OpenFlow foi desenvolvido na universidade de Stanford no ano de 2008 com o principal objetivo de atender a demanda de novas arquiteturas e protocolos de rede, que entram no contexto da necessidade de abertura e flexibilidade dos equipamentos comerciais já existentes [Open Networking Foundation 2012].

O OpenFlow estabelece a interface de comunicação entre o controlador e os comutadores controlados, bem como o formato da tabela de fluxos. Ao receber um pacote, o comutador realiza uma consulta em sua tabela que contém as regras de encaminhamento. Caso não haja nenhuma regra estabelecida para um determinado pacote, seu cabeçalho é enviado ao controlador para processamento e definição de uma nova regra. Uma vez que os controladores são responsáveis por definir as regras de fluxo, a complexidade dos comutadores é reduzida. O OpenFlow é o padrão mais difundido de SDN, no entanto, uma de suas principais limitações está relacionada com a escalabilidade da rede, tendo em vista que a tabela de fluxos tende a aumentar de acordo com o crescimento da rede e do número de fluxos, além da necessidade de manutenção de estado dos fluxos ativos nas tabelas de encaminhamento dos comutadores e da necessidade de comunicação com o controlador a cada nova regra desconhecida. Com isso, se torna necessário buscar alternativas que evitem a abordagem de regras em tabelas de fluxos [Martinello et al. 2014].

3. KeyFlow

O KeyFlow pode ser definido como um método de encaminhamento de pacotes dentro de uma SDN que substitui a tabela de regras pela operação de módulo (resto da divisão entre dois números inteiros) para determinar a porta de encaminhamento de um pacote. Este método traz como benefícios a melhora no desempenho da rede em termos de latência, variabilidade na entrega de quadros e tempo de reconfiguração [Martinello et al. 2014].

O método de encaminhamento no qual se baseia o KeyFlow consiste na criação de topologias *overlay* em redes OpenFlow através da utilização do Esquema de Informação

de Chave (KIS - *Key Information System*), proposto originalmente para redes ópticas [Wessing et al. 2002]. As chaves locais funcionam como identificadores de cada comutador na rede. Para que as chaves criadas sejam válidas, é necessário que, em cada caminho da rede, elas sejam definidas por números primos entre si.

Cada pacote que será encaminhado dentro da rede possui uma informação própria contida no cabeçalho chamada de rótulo. A relação entre as chaves locais e os rótulos se dá pela operação de módulo. Nesse caso, é realizada a divisão do rótulo do pacote pela chave local de cada comutador. Essa operação tem como principal objetivo a determinação da porta de saída do fluxo de pacotes que chegam a cada comutador da rede.

Os rótulos são gerados a partir do Teorema Chinês do Resto (TCR) [Kak 1985]. Para cada caminho ou circuito virtual, são definidos dois vetores. O primeiro vetor é determinado a partir das chaves que determinam o caminho desejado. O segundo vetor é determinado através do número da porta de saída de cada um dos comutadores da rede.

3.1. Teorema Chinês do Resto

A função do TCR é encontrar um rótulo para o pacote dada a informação do caminho que ele deve seguir na rede, que é definido pelo próprio controlador. Após o rótulo ser calculado, ele é enviado como resposta ao comutador de borda, que irá incorporá-lo ao cabeçalho do pacote. Com este rótulo, é possível fazer o processo inverso, ou seja, obter o caminho a ser seguido pelo pacote em cada comutador da rede.

Suponha que o caminho que o pacote deva percorrer contenha N nós. Com isso, definimos o vetor $J = (J_1, J_2, J_3, ..., J_n)$ e $K = (K_1, K_2, K_3, ..., K_n)$ com os inteiros que representam a chave de cada comutador no caminho desejado e a identificação das portas de saída dos comutadores correspondentes, respectivamente.

Queremos obter um inteiro R tal que permita recuperar o vetor K. O sistema admite solução única somente se as chaves dos comutadores forem primas entre si, ou seja, $mdc(J_i, J_j) = 1$, para $i \neq j$. Nesse contexto, pode-se definir o seguinte sistema de congruência:

$$\begin{cases} R \equiv K_1 \bmod J_1 \\ R \equiv K_2 \bmod J_2 \\ \dots \\ R \equiv K_n \bmod J_n. \end{cases}$$

Um algoritmo para obter o rótulo R através do TCR pode ser encontrado em [Cercós et al. 2014].

3.2. Rede KeyFlow

A Figura 2 mostra um exemplo de uma arquitetura de rede KeyFlow integrada a uma estrutura SDN. Neste exemplo, um pacote que sai do nó 43 com destino ao nó 31 atravessando os nós intermediários definidos por $J_i = (7, 10, 13)$ tem as portas de saída definidas por $K_i = (1, 4, 4)$. O caminho em estudo está definido pelas setas pontilhadas.

Para que o valor do rótulo do pacote seja definido, uma solicitação é realizada ao controlador, que o calcula com base no caminho definido até o destino. Logo após,

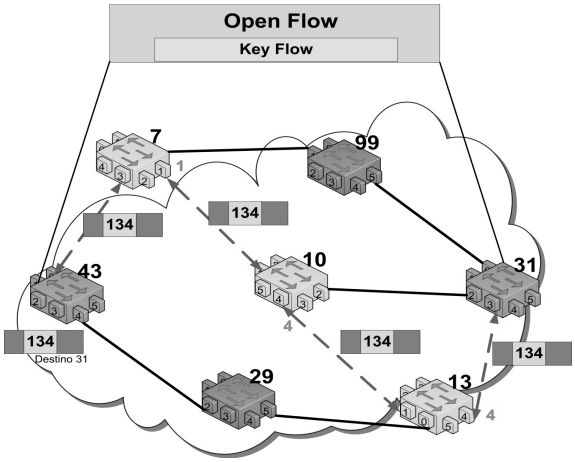


Figura 2. Arquitetura de uma rede KeyFlow integrada a uma SDN.

o controlador envia essa informação aos nós de borda, que operam como comutadores OpenFlow pois devem modificar o cabeçalho do pacote inserindo o rótulo recebido, por exemplo, no campo de Vlan ID. Os nós centrais são comutadores KeyFlow que realizam o encaminhamento dos pacotes sem consulta em tabela de fluxos como no exemplo da Figura 2 no qual o rótulo do pacote é definido com o valor 134. Após a chegada ao nó 7, é realizada a operação de módulo, sendo que $134 \bmod 7 = 1$ determina que a porta 1 seja a de saída. A operação se repete nos nós 10 e 13: $134 \bmod 10 = 4$, $134 \bmod 13 = 4$, determinando assim as respectivas portas de saída dos nós.

4. Implementação do Protótipo

A seguir serão abordados detalhes técnicos do desenvolvimento do protótipo do comutador KeyFlow sobre a plataforma NetFPGA-10G.

4.1. NetFPGA-10G

A NetFPGA-10G é uma das plataformas para prototipagem de hardware e software livre desenvolvidas pelo grupo NetFPGA das universidades de Stanford e Cambridge. O projeto foi criado para fornecer plataformas de pesquisa e ensino que possibilitam o desenvolvimento em hardwares de aplicações em rede com alto desempenho. As placas acompanham um material de apoio composto de projetos de referência documentados que servem como ponto de partida para novos projetos [Gibb et al. 2008].

Os projetos de referência que acompanham a placa NetFPGA-10G seguem a estrutura lógica padrão (*"standard pipeline"*) mostrada na Figura 3. Cada um dos blocos mostrados é composto por blocos lógicos funcionais (*IPcores*) agrupados em estruturas modulares com interfaces bem definidas. Ao desenvolver um novo projeto, é comum inserir um novo bloco lógico na estrutura padrão ou modificar um dos existentes. O presente projeto teve como referência o comutador OpenFlow desenvolvido por Tatsuya Yabe [Yabe]. Modificações foram realizadas para retirar os módulos de busca em tabela e modificações de cabeçalhos próprios da lógica OpenFlow, além de acrescentar o módulo de encaminhamento desenvolvido segundo a lógica KeyFlow, chegando na estrutura ilustrada na Figura 4.

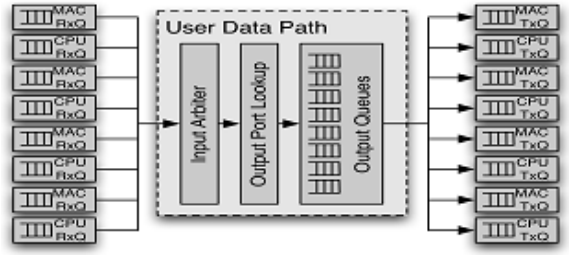


Figura 3. *Standard pipeline* para NetFPGA-10G.

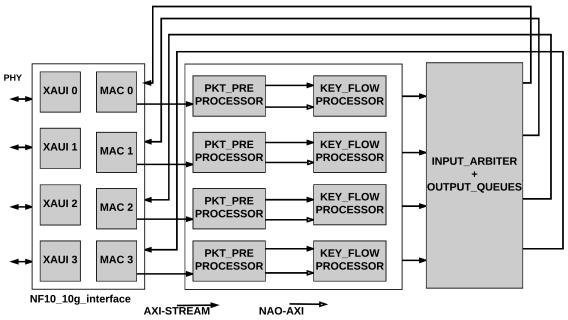


Figura 4. Diagrama em blocos da estrutura implementada.

4.2. Interface AXI

Para manter a interoperabilidade e o reuso de *IPcores*, os desenvolvedores que utilizam as placas NetFPGA são encorajados a adotarem as mesmas interfaces nos blocos criados. As interfaces adotadas para a comunicação entre os blocos da NetFPGA-10G são versões refinadas pela Xilinx dos protocolos AXI4-Lite e AXI4-Stream (AXI - *Advanced eXtensible Interface*), desenvolvidos pela ARM, como parte da família AMBA (*Advanced Microcontroller Bus Architecture*), para interconexão de blocos em *SoC's*. O protocolo AXI4-Lite é utilizado para comunicações com requisitos de memória mapeada, simples e de baixa vazão; e o protocolo AXI4-Stream, para fluxo de dados de alta vazão. A Tabela 1 mostra o comprimento e a função dos campos do barramento AXI4-Stream que é utilizado nos blocos descritos adiante.

Tabela 1. Campos do barramento AXI4-Stream.

Campo	Comprimento em bits	Função
ACLK	1	*Clock* global
ARESETN	1	*Reset* global
TVALID	1	Utilizado pelo *master* para sinalizar inicio de transferência
TDATA	64	Dados transferidos
TREADY	1	Utilizado pelo *slave* para sinalizar que aceita iniciar transferência.
TSTRB	8	Sinaliza dados inválidos.
TLAST	1	Sinaliza fim de transferência.
TUSER	128	Metadados da informação transferida.

4.3. Blocos Funcionais

A seguir são apresentados os blocos descritos em Verilog que constituem a implementação realizada.

4.3.1. NF10_10g_interface

A função desse bloco é fazer a interface com o *transceiver* e executar a lógica da subcamada MAC do padrão Ethernet 10Gbps. Esse bloco é parte do material de referência e é composto pelos *IPcores* da Xilinx, Xaui (Xilinx 10 Gigabit *Attachment Unit Interface*) e 10GMAC (10 Gigabit Ethernet *Media Access Controller*) e por um adaptador para o barramento AXI4-Stream. Os sinais provenientes do meio físico são capturados e decodificados pelo *transceiver*. O *transceiver* e o FPGA se comunicam através de um barramento XAUI. No interior do FPGA, os dados que chegam pela interface XAUI são convertidos para o padrão XGMII pelo *IPcore* XAUI e enviados para o *IPcore* 10GMAC no qual ocorre os tratamentos da sub camada de controle de acesso ao meio. Por fim, os dados são convertidos para o formato AXI-Stream padrão. Os pacotes enviados seguem o mesmo fluxo, mas da direção oposta. As interfaces XAUI e XGMII são definidas no padrão Ethernet 10Gbps (IEEE 802.3-2012).

4.3.2. Pkt_preprocessor

Esse bloco é originário do projeto de referência e foi desenvolvido para analisar os cabeçalhos dos pacotes que chegam e organizar os campos extraídos em um formato próprio para a busca na tabela OpenFlow. No presente projeto, é utilizado desse bloco somente a capacidade de analisar cabeçalhos IEEE 802.1Q. No contexto do KeyFlow, o espaço do cabeçalho destinado ao campo "Vlan ID" no padrão IEEE 802.1Q, é utilizado para armazenar o KeyFlow ID (chave que determina a rota do pacote na rede). No comutador KeyFlow, a função do pkt_preprocessor passou a ser o elemento que extrai o KeyFlow ID de cada pacote.

4.3.3. KeyFlow_processor

O KeyFlow_processor é o bloco principal do projeto e tem como funcionalidade executar a operação de resto da divisão inteira entre o valor do KeyFlow ID (rótulo do pacote) e a chave local do comutador e dai determinar a porta de encaminhamento para cada pacote. Na Figura 5 é possível observar a constituição interna do bloco. Durante o desenvolvimento, foi utilizado o simulador ISim fornecido pela Xilinx . O ISim permite analisar o comportamento dos sinais ao longo do tempo, como mostrado na Figura 6 que será utilizada na explicação do fluxo de operação do bloco.

Os pacotes que chegam são armazenados na fila "pkt_fifo" e os correspondentes metadados (campo TUSER do barramento AXI4-Stream) são armazenados na fila "mdata_fifo_in". Assim que o bloco pkt_preprocessor termina de extrair o KeyFLow ID, o sinal "valid" vai a nível lógico "1" (ponto "A" da Figura 6). Quando isto ocorre, um valor válido é colhido no registro "kf_id" para a divisão. O sinal "valid" é atrasado, em

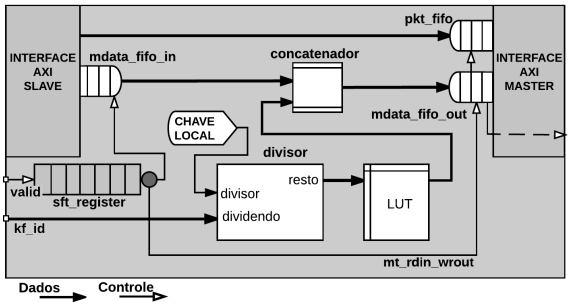


Figura 5. Constituição do bloco KeyFlow Processor.

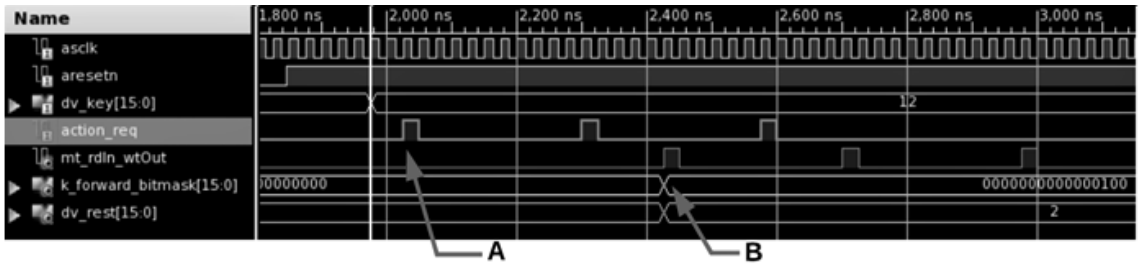


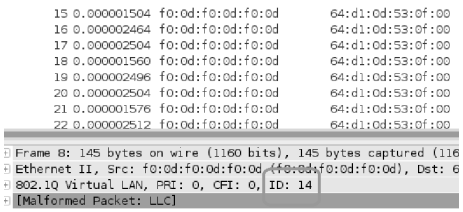
Figura 6. Curvas obtidas por simulação.

"sft_register", por dezesseis pulsos de *clock* devido à latência do divisor. A divisão é executada por um *IPcore* fornecido pela Xilinx com capacidade de iniciar uma divisão por ciclo de *clock* . O divisor opera com paralelismo (*pipeline*) e sua latência (tempo entre o início e o fim de uma divisão) é de M ciclos de *clock* para divisões de restos inteiros, sendo M o comprimento em bits do valor do dividendo. Neste caso, $M = 16$. Para manter a vazão, o divisor fica constantemente habilitado, no entanto, o valor em sua saída só é válido quando a saída do "sft_register" está em nível lógico "1". Assim que o resto da divisão é obtido, uma parte do metadado é alterada, nos blocos "LUT" e "concatenador", para a máscara de bits correspondente à porta de saída do pacote (Ponto "B" da Figura 6). A saída do "sft_register" está diretamente ligada ao habilitador de leitura da fila "mdata_fifo_in" e ao habilitador de escrita da fila "mdata_fifo_out", ou seja, depois de alterado, o metadado é transferido da fila de entrada para a fila de saída se tornando acessível ao bloco subsequente. O sinal que indica que a fila "mdata_fifo_out" não está vazia é ligado ao sinal "TVALID" da interface AXI4-Stream *Master*. Portanto, quando o processo descrito é concluído o próximo bloco é sinalizado e poderá ocorrer a transferência simultânea do pacote e seu metadado.

4.3.4. Input_arbiter + outport_queues

Encapsula dois blocos da estrutura padrão proposta para a NetFPGA-10G. Esse bloco contém quatro interfaces de entrada (AXI4-Stream *Slave*) e quatro interfaces de saída (AXI4-Stream *Master*). Sua primeira função é determinar qual das interfaces de entrada será atendida. A escolha da interface a ser atendida é feita por um simples algoritmo de varredura circular (*round-robin*), ou seja, uma interface com pacotes na fila é atendida a cada vez. Outra função desse bloco é direcionar o fluxo de pacotes de cada interface para uma porta pré-determinada. O direcionamento da porta de saída é feito pela verificação

(a) Conexão da aceleradora com comutador KeyFlow.



(b) Pacotes capturados na porta 4.

Figura 7. Testes e resultados.

dos bits 16 ao 23 do metadado de cada pacote. A porta, ou as portas, para as quais o pacote é direcionado segue a codificação *one-hot encoded* mostrada na Tabela 2. Cada bit em nível lógico "1" direciona o pacote para a porta correspondente. As portas MACx são portas físicas da placa e as portas CPUx são portas virtuais associadas à interface PCIE (não utilizada neste projeto).

Tabela 2. Máscara de bits para porta de saída.

Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
MAC0	CPU0	MAC1	CPU1	MAC2	CPU2	MAC3	CPU3

5. Validação

O protótipo foi submetido a testes de caráter qualitativo para verificar sua operabilidade. Durante os testes, foi utilizada uma aceleradora (Napatech modelo NT20E2-PTP) conectada ao comutador de modo a formar um circuito fechado, como mostrado na Figura 7(a). As funções da aceleradora foram de replicar um arquivo de captura injetando pacotes a 10 Gbps em uma das portas do comutador e capturar o fluxo de saída em outra porta. O arquivo replicado continha pacotes idênticos entre si exceto pelo valor do rótulo KeyFlow que podia conter os valores 11, 12, 13 ou 14. A Figura 7(b) mostra um dos registros obtidos nos testes. Nesse caso, ao conectar a porta de captura da aceleradora na porta 4 do comutador, constatou-se que somente os pacotes com o rótulo Keyflow igual a 14 eram encaminhados para aquela interface. Este era o resultado esperado tendo em vista que o valor da chave local do comutador no momento dos testes era 10 e que $14 \bmod 10 = 4$. Resultados equivalentes foram observados nas outras 3 portas do comutador, validando a implementação.

6. Conclusões

Este trabalho apresentou a primeira implementação de um comutador de rede utilizando o protocolo KeyFlow capaz de operar em 10 Gbps. Foram apresentados conceitos fundamentais envolvidos em uma rede definida por software, bem como sua expansão através da lógica de encaminhamento com rótulos globais e encaminhadores sem estados do KeyFlow. O hardware desenvolvido foi validada através de testes com fonte e destino externo à NetFPGA. A implementação aqui apresentada pode servir de referência para o desenvolvimento de comutadores com vazão maior e mais capacidade de comutação usando formas alternativas de encaminhamento.

7. Agradecimentos

Este trabalho foi parcialmente financiado pela Finep, com recursos do Funttel, contrato No 01.14.0231.00, sob o projeto Centro de Referência em Radiocomunicações (CRR) do Instituto Nacional de Telecomunicações – Inatel, Brasil. Este trabalho também conta com o suporte do projeto Horizon 2020 da União Européia para pesquisa, desenvolvimento tecnológico e demonstração sob no. 688941 (FUTEBOL), assim como do Ministério Brasileiro da Ciência, Tecnologia e Inovação (MCTI) por meio da RNP e do CTIC. Agradecemos também ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e Fundação de Amparo à Pesquisa e Inovação do Espírito Santo (FAPES).

Referências

Braun, W. and Menth, M. (2014). Software-defined networking using openflow: Protocols, applications and architectural design choices. *Future Internet: Open Access Journal*, pages 302–336.

Cercós, S. S., Oliveira, R., Vitoi, R., Martinello, M., Ribeiro, M., Fagertun, A. M., and Monroy, I. T. (2014). Tackling openflow power hog in core networks with keyflow. *Electronics Letters*, 50(24):1847–1849.

Gibb, G., Lockwood, J. W., Naous, J., Hartke, P., and McKeown, N. (2008). Netfpga: An open platform for teaching how to build gigabit-rate network switches and routers. *IEEE Transactions on Education*, 51:364–369.

Kak, S. (1985). Computational aspects of the Ãryabhata. *Indian Journal Of History Of Science*, pages 62–71.

Lara, A., Kolasani, A., and Ramamurthy, B. (2014). Network innovation using openflow: A survey. *IEEE Communications Surveys & Tutorials*, pages 493–512.

Martinello, M., Ribeiro, M. R., and de Oliveira, R. E. Z. (2014). Keyflow: a prototype for evolving sdn toward core network fabrics. ieee network. *IEEE Network*, 7:12–19.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38.

Open Networking Foundation (2012). Software-defined networking: The new norm for networks. https://www.opennetworking.org.

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*, 7:1–6.

Stallings, W. (2013). Software-defined networks and openflow. *The Internet Protocol Journal*, 38:2–14.

Wessing, H., Christiansen, H., and Fjelde, T. (2002). Novel scheme for packet forwarding without header modifications in optical networks. *Journal of Lightwave Technology*, 20:1277–1283.

Yabe, T. Openflow implementation on netfpga-10g, design document. https://docs.google.com/document/d/1ZwHXQZocKwQls6Ted8VZO8h9MjBtu9WxV2fAY44eOgE/edit.