Um novo método de acesso ao meio baseado em *Workspaces* para suporte a arquiteturas de Internet do Futuro implementado em FPGA

Romerson Deiny Oliveira

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2019

Romerson Deiny Oliveira

Um novo método de acesso ao meio baseado em *Workspaces* para suporte a arquiteturas de Internet do Futuro implementado em FPGA

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Pedro Frosi Rosa
Coorientador: Daniel Gomes Mesquita

Uberlândia
2019

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

O48n
2019

Oliveira, Romerson Deiny, 1987-
Um novo método de acesso ao meio baseado em Workspaces para suporte a arquiteturas de Internet do Futuro implementado em FPGA [recurso eletrônico] / Romerson Deiny Oliveira. - 2019.

Orientador: Pedro Frosi Rosa.
Coorientador: Daniel Gomes Mesquita.
Tese (Doutorado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Disponível em: http://doi.org/10.14393/ufu.te.2019.32
Inclui bibliografia.
Inclui ilustrações.

1. Computação. 2. Arquitetura de computador. 3. Redes de computadores. 4. Internet. 5. FPGAs. I. Rosa, Pedro Frosi, 1959-, (Orient.). II. Mesquita, Daniel Gomes, 1975-, (Coorient.). III. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

CDU: 681.3

Rejâne Maria da Silva – CRB6/1925

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

ATA DE DEFESA

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Tese, 02/2019, PPGCO				
Data:	25 de junho de 2019	Hora de início:	14:00	Hora de encerramento:	17:07
Matrícula do Discente:	11413CCP007				
Nome do Discente:	Rômerson Deiny Oliveira				
Título do Trabalho:	Um Novo Método de Acesso ao Meio Baseado em \textit{Workspaces} para Suporte a Arquiteturas de Internet do Futuro Implementado em FPGA				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Sistemas de Computação				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se na sala 1B132, Bloco 1B, Campus Santa Mônica, da Universidade Federal de Uberlândia, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Rodrigo Sanches Miani - FACOM/UFU, João Henrique de Souza Pereira - FACOM/UFU, Sérgio Takeo Kofuji - LSI/USP, Vanderlei Bonato - ICMC/USP, Daniel Gomes Mesquita - UNIPAMPA (Coorientador) e Pedro Frosi Rosa - FACOM/UFU orientador do candidato.

Ressalta-se que o Prof. Dr. Sérgio Takeo Kofuji participou da defesa por meio de videoconferência desde a cidade de São Paulo-SP, o Prof. Dr. Vanderlei Bonato da cidade de São Carlos-SP e o Prof. Dr. Daniel Gomes Mesquita da cidade de Sant'Ana do Livramento-RS. Os outros membros da banca e o aluno participaram *in loco*.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Pedro Frosi Rosa, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado.

Esta defesa faz parte dos requisitos necessários à obtenção do título de Doutor.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Rodrigo Sanches Miani**, **Professor(a) do Magistério Superior**, em 26/06/2019, às 18:34, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Pedro Frosi Rosa**, **Professor(a) do Magistério Superior**, em 26/06/2019, às 21:47, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **João Henrique de Souza Pereira**, **Professor(a) do Magistério Superior**, em 26/06/2019, às 23:03, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Vanderlei Bonato**, **Usuário Externo**, em 09/07/2019, às 09:36, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **SERGIO TAKEO KOFUJI**, **Usuário Externo**, em 09/08/2019, às 17:17, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Daniel Gomes Mesquita**, **Usuário Externo**, em 14/08/2019, às 22:52, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1349724** e o código CRC **3C350BF7**.

Referência: Processo nº 23117.056538/2019-85 SEI nº 1349724

À minha mãe. Pelos sonhos que renunciou em favor dos meus.

Agradecimentos

A Deus, em primeiro lugar, pelas oportunidades que me permitiu encontrar.

Aos meus pais, irmãos e sobrinhos, por incondicionalmente terem apoiado cada passo da minha vida sem duvidar da minha realização.

Aos colegas de laboratório que, além de verdadeiros amigos, são companheiros de batalha. A eles, pela incessante ajuda e por todos os momentos de cooperação, um tanto exaustivas, que não nos deixaram desistir. Em especial ao Diego, ao Marcelo, ao Alex, ao Acrísio, ao Giovanni, ao Damaso, ao Natal e ao Caio.

À CAPES pelo apoio financeiro.

Ao Prof. Rui Aguiar, do Instituto de Telecomunicações da Universidade de Aveiro, pela ajuda efetiva na pesquisa e pela constante solicitude em orientar-me.

Ao Prof. Daniel Gomes Mesquita, por todas as exigências que me prepararam para seguir meu caminho acadêmico. De forma especial, por acreditar em mim e pelo exemplo de integridade.

Ao Prof. Pedro Frosi Rosa, grande amigo, pela oportunidade, confiança e conselhos que tanto ajudaram no desenvolvimento da pesquisa e da minha vida pessoal. Ainda, pela sabedoria com que conduziu nossos cafés. "Se cheguei até aqui foi porque me apoiei no ombro dos gigantes".

"As for the future, your task is not to foresee, but to enable it."
(Antoine de Saint-Exupéry)

Resumo

No processo evolutivo da Internet, o provimento de serviços pelas camadas intermediárias não acompanhou a evolução da complexidade das aplicações nem a capacidade dos equipamentos de rede. Tais camadas não compreendem, por exemplo, algumas necessidades de QoS, mobilidade, *multihoming*, transmissão *multicast* e web semântica. As Redes Definidas por Software (SDN) introduzem determinado grau de programabilidade à rede para diminuir esta lacuna e a Entity Title Architecture (ETArch) ataca os problemas de multicast e mobilidade propondo uma nova forma de endereçamento que torna a comunicação mais inteligente e a rede mais flexível entre as camadas de Rede e Aplicação. Ainda assim, o atendimento aos serviços nesta arquitetura pode ser otimizado caso a flexibilização atinja o nível do Enlace, que é onde o hardware encontra o software em se tratando da implementação das funções da rede. Assim, este trabalho desenvolve um mecanismo programável de controle de acesso ao meio, no nível imediatamente acima da camada física em um *Switch* SDN orientado a *Workspace*, que atende aos requisitos de flexibilidade e desempenho no estabelecimento e manutenção de conexões multicast, também orientadas a *Workspace*, do plano de dados da arquitetura ETArch. O desenvolvimento da pesquisa envolve especificar os requisitos de um Switch ETArch, projetar e prototipar em FPGA mecanismos que formam um tecido de granularidade suficiente para a recombinação dos blocos elementares, provendo um controle parametrizável de acesso ao meio para as camadas superiores da arquitetura.

Palavras-chave: SDN. ETArch. Switch Reconfigurável. Redes Programáveis. FPGA.

Abstract

The Internet infrastructure has not evolved uniformly over the years. The middle layers did not get the same evolution if compared to the Application requirements and hardware equipment capabilities. Some aspects such as Quality of Service, Mobility, Multihoming, and Multicast transmissions are not quite understood by the Transport and Network layers. Trying to fill in the gap between layers, the Software Defined Networks introduce programmability to the infrastructure. In this context, the Entity Title Architecture (ETArch) proposes an addressing scheme to overcome multicast and mobility issues. Nevertheless, service delivery in this architecture can still be optimized by adding some kind of flexibility to the Link level, right where the hardware and software interface each other in network functions implementations. In this context, this research builds a medium access control programmable mechanism, in a Workspace-oriented SDN Switch, that meets the parameters of flexibility and performance to create, modify and delete Workspace-based multicast connections in the ETArch data plane. The developing process involves specifying the Switch requirements, designing and prototyping in FPGA a set of hardware modules and mechanisms to be combined and provide programmability to the forwarding plane.

Keywords: SDN. ETArch. Reconfigurable Switch. Programmable Networks. FPGA..

Lista de ilustrações

Lista de tabelas

Lista de siglas

ASIC Application Specific Integrated Circuit

BIR B Intermediate Representation

CAM Content Addressable Memory

DNS Domain Name System

DTS Domain Title Service

DTSA Domain Title Service Agent

EAP Estrutura Analítica do Projeto

ETArch Entity Title Architecture

ETSCP ETArch Switch Configuration Protocol

FHRP First Hop Router Protocol

FIB Forwarding Information Base

FIoT Future Internet of Things

FPGA Field-Programmable Gate Array

HARP High Availability Router Protocol

HIP Host Identify Protocol

HLS High Level Synthesis

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IP Internet Protocol

IPG Interpacket Gap

LAN Local Area Network

LLC Logical Link Control

LISP Locator/ID Separation Protocol

MAC Media Access Control

NAT Network Address Translation

NE Network Element

NFV Network Function Virtualization

NIC Network Interface Card

NRE Non Recurring Engineering

OF Open FLow

ONF Open Networking Foundation

OSI Open System Interconnection

OVS Open vSwitch

OWL Web Ontology Language

PMI Project Management Institute

PIF Protocol Independent Forwarding

PoA Point of Attachment

P4 Programming Protocol-Independent Packet Processors

RTL Register Tranfer Level

SDR Software Defined Radio

SDN Software Defined Network

TCAM Ternary Content-Addressable Memory

TCP Transmission Control Protocol

UDP User Datagram Protocol

VHDL Very High Speed Integrated Circuits Hardware Description Language

VLAN Virtual Local Area Network

VLAN ID VLAN Identifier

VoIP Voice over IP

Sumário

Capítulo 1

Introdução

O crescente sucesso da Internet é devido, em parte, à sua simplicidade arquitetural. Desde que surgiu, a Internet se consolidou como um padrão *de facto* das redes de computadores. Enquanto em meados de 1970 havia apenas 9 servidores Internet, hoje, suas interconexões remontam a uma estrutura milhões de vezes mais complexa em números de enlaces e de nós de interconexão. A Figura 1 ilustra este crescimento mostrando a rede de interconexão entre servidores.

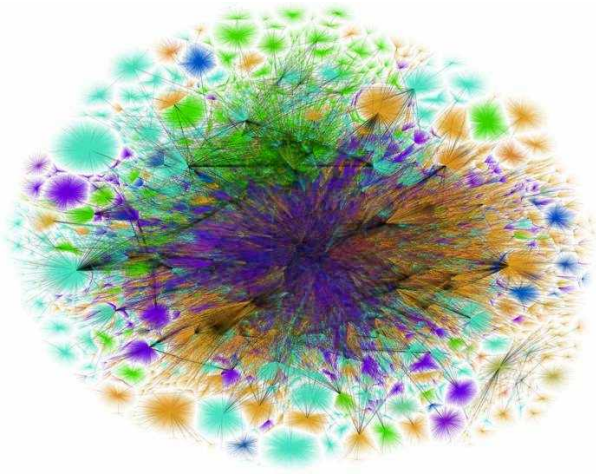


Figura 1: O Mapa da Internet em 2015[1]

Já é muito grande, e ainda crescente, a quantidade de dispositivos conectados e de aplicações que requerem uma infraestrutura de rede mais inteligente que a atual. Segundo o *International Telecommunications Union* (ITU, 2018), o número de usuários conectados cresceu cerca de 380% entre 2005 e 2018, o que reflete, em números absolutos, em uma ascensão de 1,024 bilhão para 3,896 bilhões de usuários.

Cada usuário hoje se conecta, comumente, através de mais de um ponto de conexão (simultaneamente ou não), por diferentes tecnologias físicas e, para além disso, cada

[1] Fonte: Adaptado de <http://time.com/3952373/internet-opte-project/>

máquina conectada, seja ela uma estação de trabalho, um servidor, um *smartphone* ou uma coisa[2], pode hospedar diversas aplicações que interagem por meio da Internet.

Por esta complexidade, as aplicações exigem cada vez mais da infraestrutura de rede e alguns requisitos como mobilidade, segurança, disponibilidade, sustentabilidade, largura de banda e qualidade de serviço passaram a ser cada vez mais recorrentes para as aplicações. O que ocorre é que as aplicações, embora exijam estes requisitos por natureza, deixam a cargo da rede o melhor cenário para satisfazê-los.

Os limites da abordagem não estruturada da Internet estão sendo atingidos: a pilha de protocolos da Internet se tornou muito complexa (Einsiedler et al., 2013). E, em decorrência disso, a Internet está a enfrentar problemas estruturais e de escalabilidade que dificilmente serão resolvidos por mudanças incrementais em sua estrutura de camadas e protocolos (PAN; PAUL; JAIN, 2011) (ALBERTI, 2013).

Segundo Pereira (2012), as evoluções significativas mais recentes nas estruturas dos protocolos *Transmission Control Protocol (TCP)* (IETF, 1981b), *User Datagram Protocol (UDP)* (IETF, 1980), *Internet Protocol (IP)* (IETF, 1981a) ocorreram nos primeiros anos da década de 1980 e não acompanharam o aumento da capacidade computacional dos elementos de rede e das necessidades da computação distribuída.

Vale lembrar que a camada de Aplicação evoluiu de simples compartilhamento de arquivos para sistemas complexos como *e-commerce*, redes sociais e serviços elásticos na rede, e os protocolos de Transporte e Rede da Internet não compreendem, por exemplo, algumas necessidades de QoS, segurança, mobilidade, *multihoming*, transmissão *multicast* e Web semântica.

Já há alguns anos, uma série de limitações da arquitetura Internet são investigadas pela comunidade. De forma especial, Pereira (2012) aponta o problema do endereçamento de estações, aplicações e usuários, em que para algumas aplicações distribuídas, o importante é localizar o usuário e não sua estação ou aplicação, como é o caso de uma comunicação de voz sobre IP. Além disso, existe uma lacuna no suporte oferecido pelas camadas inferiores ao endereçamento de aplicações distribuídas com necessidade de comunicação de 1 para muitos ou de 1 para todos, i.e., *multicast* e *broadcast*.

Propostas para a Internet da próxima geração são facilmente encontradas na literatura, tais como XIA (ANAND et al., 2011), RINA (DAY; MATTA; MATTAR, 2008), SONATE (MÜLLER; REUTHER, 2009), NENA (MARTIN; VöLKER; ZITTERBART, 2011), Mobility First (SESKAR et al., 2011), NEBULA (ANDERSON et al., 2013) e NDN (ZHANG et al., 2010). Estudos sobre a escalabilidade do sistema de roteamento e endereçamento foram realizados pelo *Internet Engineering Task Force (IETF)*, como o *Locator/ID Separation Protocol (LISP)* (IETF, 2013). Ainda o *Host Identify Protocol (HIP)* (IETF, 2008a) e o NodeID (AHLGREN et al., 2006) trabalham separando semanticamente as funções de localização e identificação do IP.

[2] *Thing* - Definido no contexto de Internet das Coisas

De acordo com Pan, Paul e Jain (2011), a Internet deve ser pensada como uma arquitetura completa que trata os itens atualmente problemáticos como questões intrínsecas. Portanto, pensar roteamento, mobilidade, segurança e disponibilidade, por exemplo, é uma preocupação que deve pertencer ao projeto da rede e não visto como melhoramentos incrementais isolados, enquanto o restante da rede se mantém como está.

A flexibilização no uso de recursos da rede é uma tendência já em ascensão. Bem neste contexto, Casado et al. (2007) e McKeown et al. (2008) aplicaram o conceito de redes programáveis apresentado por Campbell et al. (1999) - que diz que um espectro infinito de serviços pode ser obtido a partir da programação de um conjunto de APIs (*Application Program Interface*) - e as Redes Definidas por Software (SDN[3]) (IETF, 2015) tomaram uma forma mais próxima da realidade e se destacaram nas pesquisas em Internet do Futuro.

Para Sezer et al. (2013), as SDN são uma filosofia de rede idealmente capaz de suportar a natureza dinâmica das funções de redes futuras ao reduzir o custo operacional através do hardware, software e gerenciamento simplificados. A *Open Networking Foundation (ONF)* define SDN por:

> *In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications* (ONF, 2012).

A ideia básica é a separação dos chamados Plano de Controle e Plano de Dados da rede, o que toma forma quando um controlador é executado por um software em servidores convencionais controlando fluxos em comutadores implementados em hardware para o encaminhamento dos dados (ONF, 2017c).

A premissa inicial destas redes era que o plano de dados deveria ser simples, restrito a funções de encaminhamento, com qualquer complexidade adicional sendo encaminhada e tratada por software, como reafirma Brebner (2015). Do ponto de vista da implementação, as funções do plano de controle são implementadas por uma camada de controle em software, enquanto o hardware cuida da camada de infraestrutura ou plano de dados e encaminhamento (ONF, 2014). Isso ocorre em virtude de o plano de controle ser consideravelmente mais complexo e sujeito a reprogramações do que o plano de dados.

A especificação do protocolo OpenFlow (MCKEOWN et al., 2008), bem como a construção de comutadores de rede (*switches*[4]) que o implementam, é um exemplo claro tanto da tendência de separação entre planos quanto da viabilidade do uso de um software controlador para os fluxos que implementam o plano de encaminhamento.

Acompanhando as iniciativas de Internet do Futuro e seguindo as premissas das SDN, cabe destacar a *Entity Title Architecture (ETArch)* (SILVA et al., 2012b) (SILVA et al., 2012a), uma iniciativa brasileira desenvolvida pelo grupo de pesquisa do qual este trabalho

[3] *Software Defined Network (SDN)*

[4] Doravante, o termo *switch* é usado de forma intercambiável com a expressão *comutadores de rede*.

é parte. A ETArch é uma arquitetura para a Internet do Futuro de abordagem *clean slate* que abre mão do conceito de *host* de origem e de destino. Nela, entidades que desejem se comunicar passam a fazer parte de um *Workspace*, que é um barramento lógico independente da topologia da rede e através do qual primitivas são encaminhadas.

Na ETArch, as aplicações são vistas como entidades. De uma forma mais generalista, qualquer máquina ou aplicação que deseje se comunicar por um *Workspace* é chamada Entidade e seus requisitos para as comunicações são considerados ao se estabelecer cada *Workspace*. *Workspaces*, que abstraem para as entidades a topologia e elementos de rede fisicamente conectados, fornecem um barramento lógico de comunicação flexível o suficiente para atender requisitos das aplicações.

Quem controla o ambiente distribuído de entidades interconectadas (e respectivos Títulos que as identificam) é o *Domain Title Service (DTS)*. O DTS é um sistema distribuído composto por agentes denominados *Domain Title Service Agent (DTSA)*, que são, funcionalmente, superconjuntos de controladores SDN. DTSAs são entidades responsáveis por comunicações de controle e monitoramento de um ou mais elementos de rede (*switches*) e, também, outros DTSAs.

Um fato que merece atenção neste contexto é que a programabilidade foi trazida para o plano de controle, eliminando do plano de dados complexidades ou ações que não estejam relacionadas ao encaminhamento de dados.

Um ponto de interseção entre as diversas arquiteturas que implementam SDNs ou outras propostas com a abordagem *clean slate* é a premissa de que haverá alguma conectividade no nível do Enlace. Então, propõem suas intervenções a partir da camada de Rede, deixando o controle do lógico do enlace e o controle do acesso ao meio a cargo de tecnologias legadas, notadamente a tecnologia Ethernet.

Em se tratando das implementações, desde o surgimento da tecnologia OpenFlow (ONF, 2015), ela se tornou a alternativa para implementar a separação entre controladores (*Controllers*) e elementos de redes (*switches* controlados), permitindo a configuração de fluxos a partir do casamento de ações e, portanto, a separação entre os planos. Por isso, os *switches* OpenFlow são a infraestrutura de implementação mais comum de redes SDN, inclusive a ETArch.

Switches conectados à Internet implementam, majoritariamente, a tecnologia Ethernet, que é o principal padrão para a camada de enlace, especificado pelos esforços das empresas Digital, Intel e Xerox (1980) e que se tornou o Padrão IEEE 802.3, cuja última revisão é de 2018 (IEEE, 2018). De acordo com Kerling (2018 apud RECH, 2014), mais de 95% de todas as redes locais cabeadas em todo o mundo usam o padrão Ethernet. *Switches* OpenFlow também operam quase que predominantemente na forma *Ethernet-based*, i.e., embora tenham fluxos configuráveis, o controle de acesso ao meio continua sendo padronizado pelo IEEE Std 802.3.

Entende-se, então, que a ETArch continua operando sobre Ethernet e sem flexibili-

zação do controle de acesso ao meio. Dessa forma, surge a possibilidade de adicionar programabilidade no nível do enlace de forma que a aplicação e o controlador tenham influência direta sobre o método de acesso ao meio.

O controle de acesso ao meio dos quadros pertencentes aos fluxos de dados é desempenhado pelo *switch*. É ele quem implementa o nível do enlace no centro da rede. É oportuno lembrar que a interface entre software e hardware ocorre no nível de enlace, tanto no núcleo da rede quanto nos *end-points*. Do ponto de vista de implementações SDN, o plano de controle é implementado em software e o plano de dados e encaminhamento é implementado em hardware, nos *switches*.

Mesmo com estas perspectivas sendo apontadas, a ETArch não possui um equipamento de comutação que implemente especificamente a filosofia de Orientação a *Workspaces*, conceito fundamental da ETArch criado pelo Modelo de Domínios e Títulos proposto por Pereira et al. (2011). Esta arquitetura carece, atualmente, de um *switch* capaz de suportar as comunicações entre entidades clientes e o controlador DTSA.

1.1 Hipótese

Considerando que as aplicações são projetadas para atender a necessidades de usuários e que o futuro aponta para usuários cada vez mais conectados, móveis e interessados em conteúdos multimídia, a hipótese é que os elementos de redes devam tomar parte no suporte aos requisitos com uma granularidade por usuário. Note-se que a experiência dos usuários influenciam decisivamente na avaliação de produtos de software, então, além dos requisitos das aplicações (banda, tempo real, elasticidade, usabilidade) há que se considerar a expectativa dos usuários.

Neste ponto, é intuitivo pensar que incluir em um *switch*, mecanismo de controle de acesso ao meio flexível, em sintonia com o paradigma de Redes Definidas por Software, atende a exigências de programabilidade e desempenho no estabelecimento de fluxos multicast, orientados a *Workspace*, do plano de dados da arquitetura ETArch.

1.2 Objetivos

Este trabalho tem o objetivo de projetar um elemento de rede (*switch*) cuja subcamada de controle de acesso ao meio seja definida pelos requisitos de comunicação de *Workspaces*. Este elemento servirá para demonstrar experimentalmente que a inclusão de um mecanismo programável de controle de acesso à camada física em um *switch* SDN atende aos requisitos de flexibilidade e desempenho das comunicações multicast, orientadas a *Workspace*.

Objetiva-se desenvolver um protótipo de um *switch* implementado em hardware reconfigurável que suporte elasticidade e agilidade na manutenção de *Workspaces* em redes

SDN. Isso deixa mais flexível e transparente a interface entre aplicações e o controle de acesso ao meio, aproximando o nível da aplicação e o controle da qualidade de serviço em cada *Workspace*. São objetivos específicos:

1. Especificação de um mecanismo de comunicação para o plano de controle, que defina a interface entre DTSA e *switch*;
2. Desenvolvimento de escalonador para permitir que quadros sejam tratados individualmente no mecanismo de comutação do *switch*;
3. Validar a operação do mecanismo de encaminhamento orientado a *Workspace*; e
4. Demonstrar que o comutador é capaz de estabelecer conexões; criar, parametrizar e excluir *Workspaces* sob demanda do controlador.

1.3 Contribuições

As principais contribuições deste trabalho são:

1. Especificação dos mecanismos de encaminhamento de dados da arquitetura ETArch, o que inclui as técnicas de aprendizado, construção e manutenção das tabelas de encaminhamento baseadas em *Workspaces*.
2. Projeto de um equipamento dedicado e reconfigurável para o plano de encaminhamento da ETArch, o que engloba:
   a) Protocolo e interface de comunicação entre controlador DTSA e *switch* ETArch;
   b) Mecanismos de envio e recebimento implementados para a filosofia de Orientação a *Workspace*; e
   c) Comutador reconfigurável escalável à inserção de demais blocos de hardware para atender futuros requisitos ETArch.

1.4 Organização do Documento

Este documento está estruturado da seguinte forma:

1. O Capítulo 1 contextualiza a proposta em relação à sua área de aplicação, apresenta os objetivos, motivação e contribuições originais, além da estrutura do texto.
2. O Capítulo 2 provê a base conceitual requerida e faz uma revisão da literatura correlata a tese. Primeiramente, são descritos o modelo e arquitetura de rede que introduzem a filosofia de orientação a *Workspace*, além de um detalhamento sobre

a camada de Enlace da Internet. Em seguida, uma análise dos trabalhos correlatos traz soluções para flexibilização da camada de enlace em Redes Definidas por Software, indo desde o controle de fluxos por descrição em linguagens de alto nível até a aplicação do hardware programável na organização dos equipamentos.

3. O Capítulo 3 detalha o método, i.e., explica de forma detalhada as tarefas realizadas, bem como apresenta uma visão geral do sistema construído para demonstrar a operação do *switch*.

4. O Capítulo 4 explica os requisitos do *switch* e detalha o subconjunto dos requisitos já incluídos no protótipo construído. Apresenta também a especificação do protocolo ETSCP para a interface de configuração entre DTSA e *switch*.

5. O Capítulo 5 traz com detalhes o projeto do *switch*, abordando os aspectos de arquitetura e de organização dos blocos do hardware digital.

6. O Capítulo 6 apresenta os dados relativos à prototipagem e à operação do *switch* em um ambiente de rede local e faz análises sobre os valores de tempo e frequência da operação do hardware.

7. O Capítulo 7, por fim, traz as considerações finais do trabalho, enumerando as principais contribuições, dificuldades e as propostas de trabalhos futuros.

Capítulo 2

Revisão de Literatura

2.1 Fundamentação Teórica

Este capítulo trata do aparato conceitual em que esta tese se baseia. É preciso entender tanto o modelo quanto a arquitetura de Internet do Futuro que apresentam a filosofia de orientação a *Workspace*, pois são a referência para os mecanismos de encaminhamento do *switch* ETArch. Também é oportuno discutir a tecnologia Ethernet para uso em redes locais e obter visão geral dos *switches* da arquitetura Internet, pois eles são o ponto de partida para uma implementação ainda que *clean slate*.

2.1.1 Modelo de Títulos para a próxima geração da Internet

Pereira et al. (2011) define um Modelo de Títulos que aproxima semanticamente as camadas superiores e inferiores nas próximas gerações de Internet e tem como objetivo contornar alguns problemas da Arquitetura TCP/IP: dificuldade de compreensão semântica das necessidades de comunicação pelas camadas intermediárias; limitações pela baixa evolução também das camadas intermediárias e aumento da complexidade crescente da arquitetura atual; e quantidade de endereços insuficiente para a demanda atual e futura.

O Modelo de Títulos, que permite o endereçamento unificado de entidades de forma horizontal, baseia-se em diversos elementos conceituais e alguns deles merecem especial atenção por trazerem um significado consideravelmente diferente da Arquitetura Internet. Pereira (2012) define:

> *Entidade:* Elemento comunicante cujas necessidades de comunicação podem ser compreendidas semanticamente e suportadas pela camada de Serviço e as subsequentes camadas inferiores de Enlace e Física.
>
> *Título:* É a designação única para garantir uma identificação não ambígua. Uma identidade única de entidade.
>
> *Workspace:* É um barramento lógico que tem um Título e possui elementos de rede para suportar a comunicação das Entidades. É criada por uma Entidade que deseja comunicar com propósito específico e define seus requerimentos e funcionalidades.

> *Serviço de Domínio de Título DTS*[1]*:* É um domínio capaz de compreender e registrar as instâncias das entidades, suas propriedades e necessidades, facilitando a comunicação entre elas. Este domínio possui abrangência mundial na Internet e escalabilidade horizontal e hierárquica, formada por elementos de comunicação locais, mestres e escravos, semelhante ao *Domain Name System (DNS).*

A comunicação de voz sobre IP (*Voice over IP (VoIP)*) é um caso de uso que ilustra adequadamente a motivação do Modelo de Títulos, pois demonstra que as aplicações apresentam elevada dificuldade em atender a necessidade de comunicação 1 para muitos e 1 para todos (*multicast* e *broadcast*, respectivamente). O objetivo da comunicação é a troca de informações entre dois ou mais seres humanos e para que esta comunicação seja possível na arquitetura TCP/IP são necessários vários endereços: endereço MAC (*Medium Access Control*), endereço IP (*Internet Protocol*), endereço da aplicação (Porta do TCP ou UDP), endereço do AS (Número do *Autonomous System*) e endereço do usuário (*login*).

O Modelo de Título define a unificação dos endereços em Título e permite que seja utilizado um endereço único para a comunicação entre as entidades, com suporte semântico das necessidades de comunicação por uma camada de serviço. Assim, as necessidades de comunicação das entidades são suportadas por esta camada de serviço em cada contexto. A Figura 2 mostra a relação entre as camadas TCP/IP e do Modelo de Título.

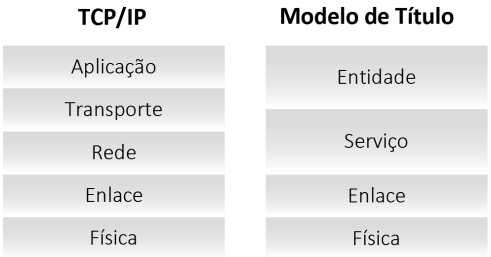


Figura 2: Camadas TCP/IP e Modelo de Títulos.
Fonte: (PEREIRA et al., 2011)

Para que a camada de Serviço ofereça suporte às necessidades das entidades, que podem alterar conforme o contexto, o Modelo de Título define o uso de ontologia para que a camada de Entidade possa se comunicar semanticamente com a camada de Serviço. Esta, por sua vez, traduzirá a comunicação em funcionalidades por meio das camadas de Enlace e Física. No trabalho original, o autor utiliza a *Web Ontology Language (OWL)* (LACY, 2005).

O modelo independe da forma de roteamento utilizada. O DTS é definido para orquestrar a comunicação entre entidades e dar suporte às suas necessidades, pelas redes de computadores. A comunicação entidade-entidade substitui o paradigma cliente-servidor e toda entidade nasce apta para a comunicação em rede. O atendimento e resposta de uma

[1] DTS: Do inglês *Domain Title Service*

requisição de serviço são realizados, ou não, a partir da decisão de um controle superior. Portanto, deixa de haver a necessidade de uma aplicação em uma estação ser iniciada e aguardar conexões, pois as entidades naturalmente são aptas para a comunicação distribuída.

A ontologia do Modelo de Títulos define classes hierárquicas para a construção do Modelo de Título de Entidade. A classe *Thing* é a superclasse da taxonomia e possui, no próximo nível, as subclasses: *Camada*, *DTS*, *Título*, *Entidade* e *Necessidade*. Cada classe se desdobra em subclasses que tem propriedades definidas através do uso de *Slots*. Os *slots*, por sua vez, tem seus valores definidos por *facets*. A partir disso, instâncias podem ser criadas para a identificação de entidades comunicantes na rede.

2.1.2 *Entity Title Architecture*

Silva et al. (2012b) e Silva et al. (2012a) apresentam a *Entity Title Architecture (ETArch)*, uma arquitetura que tem como referência o Modelo de Títulos e cujo objetivo principal é aproximar semanticamente a camada de aplicação e as camadas inferiores em relação às camadas TCP/IP. Esta aproximação semântica visa facilitar a agregação de tráfego *Multicast* e expressar os requisitos de aplicações de tal forma que eles permeiem a arquitetura em todas as suas camadas. A arquitetura parte de uma filosofia *clean slate* (REXFORD; DOVROLIS, 2010) e se diferencia das arquiteturas tradicionais por diminuir a distância entre as aplicações e as camadas inferiores, facilitando a passagem de requisitos entre os diferentes níveis abstratos da rede.

Ela retoma o conceito de *entidade* do Modelo de títulos. Uma Entidade possui ao menos um título, por meio do qual pode ser unicamente identificada, e uma localização, denominada Ponto de Conexão (*Point of Attachment (PoA)*). Um *Namespace* é o espaço de identificação discreto responsável por adicionar um significado na relação entre Título e Entidade.

Workspace é um barramento lógico de comunicação *multi-end* que independe da topologia, ao qual Entidades podem se ligar para participar de um domínio de comunicação e é naturalmente *multicast*, permitindo que as Entidades se liguem ou desliguem facilmente do domínio de comunicação. Ele é reconhecido pelas seguintes propriedades: Título, Lista de *Network Element (NE)*, Lista de Capacidades, Requisitos, Visibilidade (público ou privado) e Nível.

A iniciação, manutenção e encerramento de um *Workspace* é controlado pelo DTS, que é constituído por agentes denominados *Domain Title Service Agent DTSA*. A Figura 3 ilustra o cenário característico de um ambiente DTS/ETArch e apresenta os principais elementos da comunicação.

Os DTSAs criam os *Workspaces* a partir da solicitação de Entidades, que informam suas necessidadeas e suas capacidades de comunicação. O DTSA então, reconfigura o NE

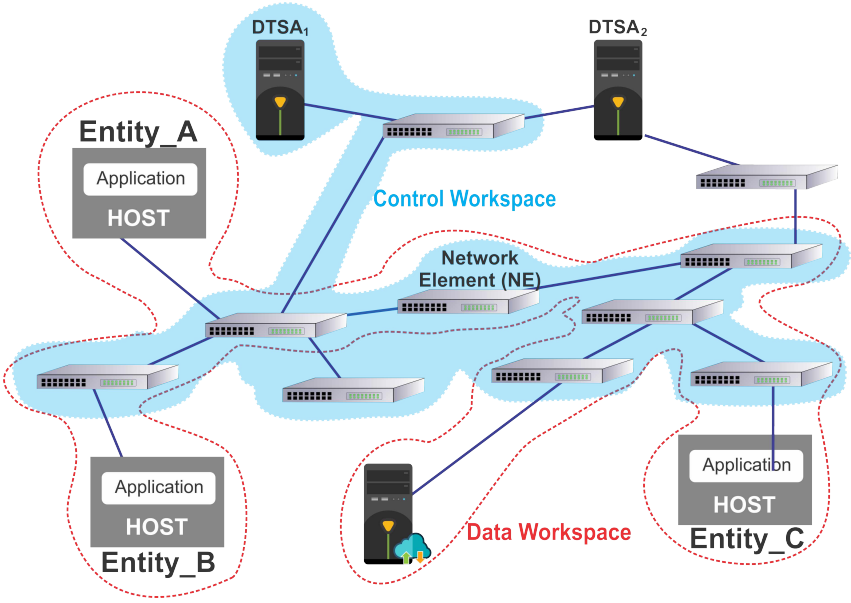


Figura 3: DTS, DTSA, Entidade, NE e Workspace.
Fonte: Adaptado de (SILVA et al., 2012b)

para atender àquela Entidade em relação às suas necessidades, criando novos *Workspaces* ou anexando-a a algum já existente.

Workspaces podem ser de controle ou de dados, diferenciando-se pelos três aspectos principais: natureza (se transferem primitivas de dados de aplicação ou apenas de controle), criação e as Entidades que se ligam a ele (apenas DTSAs podem se comunicar por workspaces de controle). Os comportamentos dos NEs são controlados externamente pelos agentes DTSA numa relação unívoca, sendo que um DTSA pode controlar uma lista de um ou mais NEs e um NE é controlado por apenas um DTSA (SILVA, 2013).

2.1.3 Camadas ETArch

O padrão arquitetural em camadas é adotado como ponto de partida para a especificação da Arquitetura ETArch, que define de cima para baixo três camadas denominadas: *Application*, *Communication* e *Link*, representadas na Figura 4. Elas tomam como base a proposta da Figura 2.

A camada *Application* se aproxima da camada de Aplicação da Internet, enquanto a camada *Link* se aproxima das camadas Física e de Enlace. O foco da proposta ETArch é a camada *Communication*. Até as pesquisas realizadas neste momento, esta é a camada que trata essencialmente da capacidade da ETArch se adaptar a requisitos de Entidades. A representação não usual desta camada significa exatamente isso. A camada *Communication* substitui as funções de Rede e Transporte, enquanto a *Application* cuida dos serviços de Sessão, Apresentação e Seção.

O roteamento ETArch adota a filosofia de roteamento *out-of-band* (NETO et al., 2015),

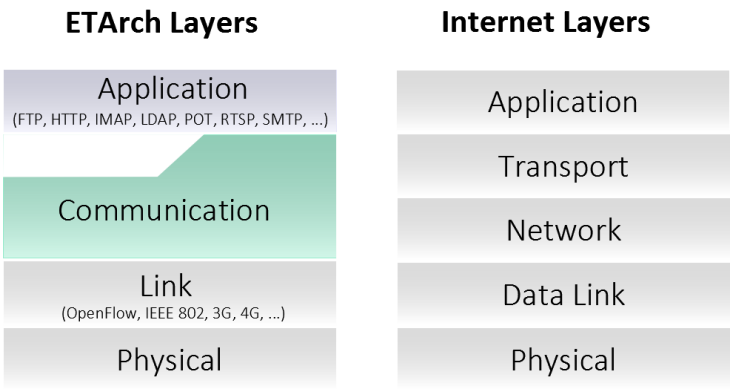


Figura 4: Camadas ETArch *vs.* Camadas Internet
Fonte: (SILVA et al., 2012a)

em que a descoberta do caminho é feita anteriormente à troca de dados e este é mantido durante todo o ciclo de vida do *Workspace*.

2.1.4 Ethernet IEEE Std 802.3

A família de tecnologias Ethernet foi padronizada pela *Institute of Electrical and Electronics Engineers (IEEE)* como IEEE Std 802.3 (IEEE, 2018) pela primeira vez em 1983, baseando-se na especificação das empresas Digital, Intel e Xerox (1980), descrevendo a comunicação *half-duplex* em fio de cobre a uma taxa máxima de 10 Mb/s. Desde então, muitos recursos adicionais foram adicionados, incluindo operação *full-duplex* e taxas de dados mais altas dos primeiros 100 Mb/s (referidos como Fast Ethernet), depois 1000 Mb/s (referidos como Gigabit Ethernet) e mais recentemente até 100 Gb/s, um aumento de quatro ordens de grandeza. Isso permitiu que a Ethernet se tornasse a tecnologia dominante usada em LANs de computadores em todo o mundo.

A última revisão do padrão é a IEEE Std 802.3-2018. Ele descreve uma arquitetura em camadas que, em um alto nível de abstração, difere entre a camada física (PHY) e a subcamada de controle de acesso ao meio (MAC). A principal responsabilidade da camada física é transmitir dados opacos sobre um meio físico, enquanto o controle de acesso ao meio deve cuidar do encapsulamento da carga útil em quadros Ethernet válidos, endereçamento de nós na rede e detecção de erros. Isso é intencionalmente muito semelhante às camadas inferiores do modelo *OSI* IEC 7498-1 (ISO, 1994), que descreve a comunicação como interação de camadas abstratas com responsabilidades bem definidas, como mostrado na Figura 5.

As camadas comunicam somente com as camadas diretamente acima e abaixo delas. Desde que as interfaces sejam compatíveis, a implementação de uma camada pode ser trocada livremente por outra. A camada Ethernet PHY corresponde diretamente à camada física do modelo OSI, enquanto a subcamada MAC junto com a subcamada de controle

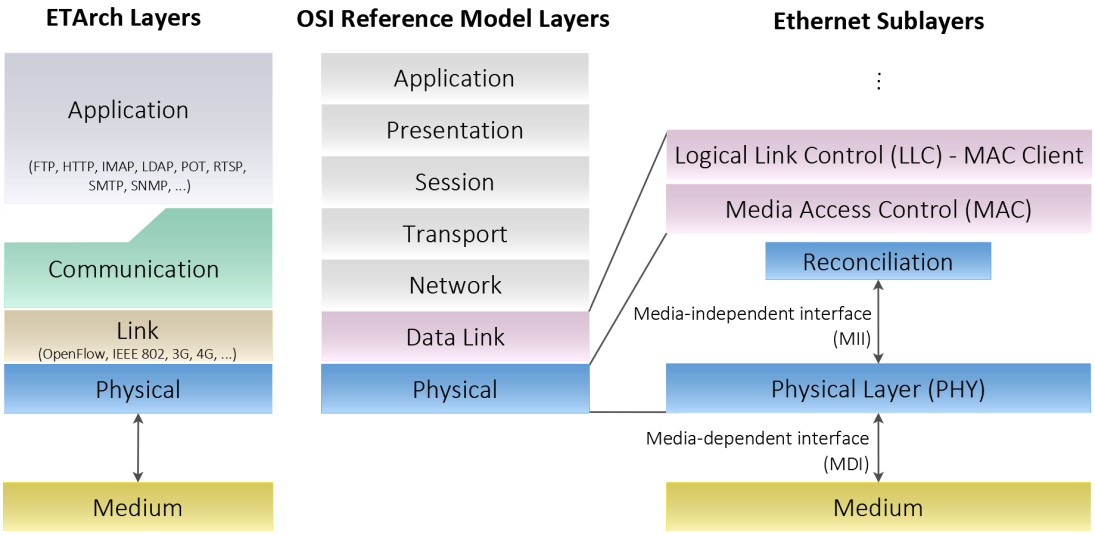


Figura 5: Relação entre modelo de referência OSI, ETArch e Ethernet.
Fonte: Adaptado de (SILVA et al., 2012a) e (IEEE, 2018)

de link lógico LLC formam a camada de Enlace (*Data Link*) de dados OSI. A subcamada LLC não é obrigatória e não é especificada no padrão IEEE 802.3.

A subcamada Ethernet LLC lida com a comunicação entre as camadas superiores e as camadas inferiores. Isso é tipicamente entre o software de rede e o hardware do dispositivo. A subcamada LLC obtém os dados do protocolo de Rede, que normalmente é um pacote IPv4, e adiciona informações de controle para ajudar a entregar o pacote ao nó de destino. A LLC é usada para se comunicar com as camadas superiores do aplicativo e fazer a transição do pacote para as camadas inferiores para entrega.

A LLC é implementada em software e sua implementação é independente do hardware. Em um computador, a LLC pode ser considerada o software de *driver* da *Network Interface Card (NIC)*, um programa que interage diretamente com o hardware da NIC para passar os dados entre a subcamada MAC e a mídia física. O MAC constitui a subcamada inferior da camada de Enlace. O MAC é implementado por hardware, normalmente na NIC do computador. As especificidades são especificadas nos padrões IEEE 802.3.

Entre as tecnologias de camada física estabelecidas no IEEE Std 802.3, as mais usadas atualmente são 100BASE-TX com uma taxa de dados de 100 Mb/s e 1000BASE-T com uma taxa de dados de 1000 Mb/s em fios de cobre. A transmissão de 10 Mb/s que gerou inicialmente a padronização Ethernet é preservada sob o nome 10BASE-T, mas não é mais usada em larga escala. Como 10BASE-T, 100BASE-TX e 1000BASE-T compartilham um fio de cobre de par trançado blindado, conforme definido no padrão TIA/EIA-568-A como seu meio físico, é comum que dispositivos suportem múltiplos padrões. Existe um procedimento de negociação automática compatível com versões anteriores que garante que os parceiros de enlace estabeleçam uma conexão no modo mais rápido possível (KERLING, 2018).

2.1.5 Switches L2

A troca de informações na Internet começa a ter valor semântico na camada de Enlace. O dispositivo mais inteligente que opera tais operações é o comutador de rede ou *switch* L2 (IEEE, 2001). A função de um *switch* é receber quadros e encaminhá-los para enlaces de saída de acordo com as regras de encaminhamento.

Para tal, os dispositivos L2 constroem tabelas de endereços de hardware (tabela de comutação/encaminhamento - *switching table*) com pelo menos três itens: endereço de hardware para os dispositivos conectados (tal como endereço *MAC*), a porta a que cada dispositivo se conecta e um *timestamp* para controlar a atividade do enlace. Para *switch* Ethernet, esta tabela é chamada de *MAC Address Table* ou *L2 Forwarding Table* (KUROSE; ROSS, 2006).

Utilizando estas informações, os dispositivos tomarão decisões inteligentes de encaminhamento baseadas nos cabeçalhos dos quadros. Um quadro pode ser então encaminhado por apenas uma porta ao invés de todas as portas como era feito em *Hubs*. Este encaminhamento no nível da camada 2 já foi chamado de *bridging*, que caiu em desuso e agora é comumente referido por *switching*.

Os comutadors L2 têm a capacidade de construir suas tabelas de comutação de forma automática, seguindo o esquema:

1. A tabela está inicialmente vazia;
2. Para cada quadro recebido por uma porta, uma entrada na tabela é inserida (*SrcAdd,Port,Time*);
3. Endereços são excluídos depois de certo tempo de inatividade.

A velocidade com que quadros chegam às portas de saída pode temporariamente exceder à capacidade daquele enlace. Assim, estes *switches* utilizam filas para armazenar quadros que chegam e saem. Para uma configuração típica, conforme a Figura 6, o quadro chega pela porta 1 do *switch*, do HostA ao HostB. O quadro então é colocado na fila de entrada daquela porta. O *switch* então examina o cabeçalho L2, realiza uma busca pelo endereço de destino na tabela de endereços de hardware e determina que a porta 2 alcança o Host B; o quadro é então colocado na fila de saída.

Caso não houvesse registro do endereço de hardware para o Host B, o *switch* encaminharia o quadro por todas as outras portas, exceto a de origem, associada a um endereço de *brodacast* (*flooding*), para ir construindo a tabela.

Os endereços são armazenados em memórias voláteis, muito rápidas e caras, as memórias *Content Addressable Memory (CAM)* (CISCO, 2017a), permitindo buscas rápidas em suas tabelas. Em contrapartida, os dados são perdidos em caso de reinicialização do equipamento. A maioria dos *switches* suportam também configuração estática, i.e., adicionar manualmente entradas à sua tabela.

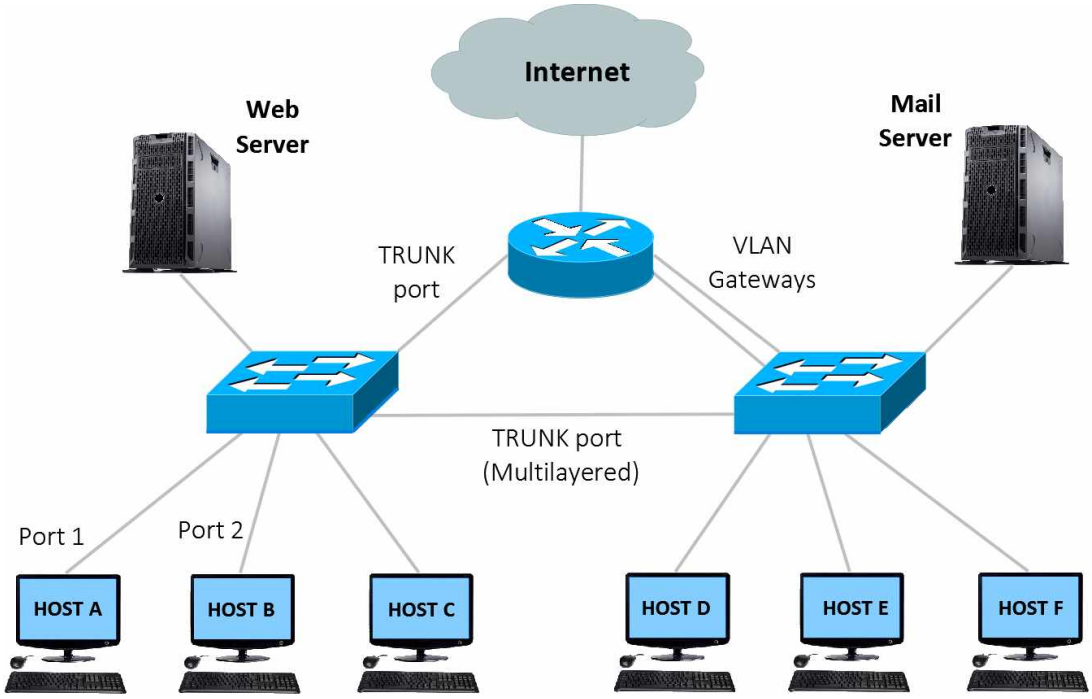


Figura 6: Virtual LANs em *switch*

Uma porta pode conter múltiplas filas de entrada e saída, isso permite que tráfego crítico seja priorizado sobre quadros com menos restrições. Em geral, estes requisitos de QoS são implementados depois das decisões de encaminhamento baseados em prioridade, elegibilidade de descarte de pacotes e tempo (IEEE, 2001).

Internamente, cada fabricante adiciona suas técnicas de encaminhamento e otimizações na organização do dispositivos, embora obedeçam aos padrões dos protocolos L2 para a interoperabilidade. Cisco, Datacom, Juniper, 3Com, Alcatel, Ericsson, Broadcom, IBM e DELL são grandes fabricantes do mercado.

Por padrão, um *switch* encaminhará ambos os tráfegos *broadcast* e *multicast* por todas as portas exceto àquela de origem. Ainda assim, eles podem ser logicamente segmentados em diferentes domínios de broadcast, utilizando Virtual LANs (VLANs).

2.1.5.1 VLAN

As *Virtual Local Area Network (VLAN)* (IEEE, 2011) são redes locais virtuais operando simultaneamente sobre uma mesma infraestrutura física. Segundo Kurose e Ross (2006), as VLANs aparecem como uma solução para os problemas de isolamento de tráfego, uso ineficiente dos comutadores e gerenciamento de usuários dos comutadores. Em um *switch* com n portas, elas são separadas em grupos e a cada grupo é atribuído um VLAN ID para isolar o tráfego dos outros grupos. Assim, diferentes redes locais podem comutar pacotes entre si utilizando um mesmo equipamento físico. As portas pertecentes a determinada VLAN são associadas em uma tabela que contém, pelo menos, o VLAN

ID e as portas que fazem parte daquela VLAN.

Cada VLAN representa um único domínio de *broadcast*. O tráfego entre os dispositivos de uma mesma VLAN é comutado por definição, enquanto a comutação entre diferentes VLANs requer controle L3 quer seja no mesmo dispositivo ou em um outro dispositivo conectado. Para o inter VLAN, pode haver ou (i) um roteador conectado a múltiplas portas do *switch* sendo uma porta *gateway* para cada VLAN; ou (ii) um roteador conectado a uma única porta do *switch* (*trunk link*) ou, por fim, (iii) o uso de um *switch* multicamadas (*Multilayered Switch*) que suporta encaminhamento L2 e L3. Estas situações são ilustradas pela Figura 6.

VLANs podem ser combinadas por portas ou por endereços MAC. Neste último caso, os endereços MAC são relacionados às portas pelo *switch* em suas tabelas de encaminhamento. Para que uma porta possa receber tráfego de várias VLANs, ela deve ser configurada como *trunk port*. Estas *trunk ports* geralmente se aplicam à conexão inter *switches* para a extensão de VLANs, através dos padrões IEEE 802.1Q (IEEE, 2011).

Switches multicamadas (CISCO, 2007) utilizam as tabelas chamadas *Forwarding Information Base (FIB)* para decisões L3. Encaminhamentos L2 e L3 são normalmente diferenciados por encaminhamento e roteamento e são baseados, respectivamente, em tabelas de hardware/encaminhamento e de roteamento.

Os *switches* multicamadas configuram o tráfego inter VLAN através da configuração de interfaces lógicas de roteamento em suas portas. Então, para cada interface deste tipo é atribuído um endereço IP e estas portas passam então a ser o *gateway* para os hosts em cada VLAN. Daí, o *switch* insere entradas na tabela de roteamento com conexão direta entre estes *gateways*. Configurações adicionais aparecem dependendo do modelo do dispositivo o controle requer endereçamento IP.

2.2 Trabalhos Relacionados

Com a disseminação do conceito de SDN, inserir programabilidade nos diferentes planos e níveis da rede ganhou muita evidência. Essa programabilidade pode permear todo o intervalo entre o nível físico e o das aplicações e com diversas tecnologias de implementação. Este capítulo traz uma visão do que se tem feito no nível de enlace de dados, que é o domínio desta pesquisa, com implementações em hardware para provimento de alguma flexibilidade da infraestrutura da rede. Remonta-se também o interesse por plataformas e linguagens capazes de descrever, independente de protocolos, o comportamento de um equipamento de rede. Ademais, interessam-nos ainda a forma como estas informações se tornam descrições de hardware e os mecanismos que implementam a sua reconfiguração.

2.2.1 *OpenFlow Switch*

McKeown et al. (2008) apresentam os *switches* OpenFlow, que separam o plano de dados do plano de controle. Sharma et al. (2013) explicam que um pacote estranho que chega ao *switch* é enviado ao controlador para a tomada de decisão sobre o seu encaminhamento. Há um modelo de programação centralizada, onde um ou mais controladores gerenciam os *switches*. Há uma tabela de fluxos e uma ação associada a cada entrada. Não há tratamento individual de pacote, mas um mais limitado conjunto de ações por fluxo. Os principais elementos são: a tabela de fluxos, o protocolo e um canal seguro com o controlador.

Há dois tipos de *switches* OpenFlow: *i) Dedicated OpenFlow switches*: é o elemento idealizado, cujo de caminho de dados não tem inteligência, i.e., apenas encaminha pacotes entre portas, conforme definido por um processo de controle remoto. Fluxos são definidos de acordo com o *match* com as entradas da tabela que são, por sua vez, parâmetros de cabeçalhos TCP/IP; e *ii) OpenFlow-enabled switches*: são os *switches*/roteadores/*access points* comerciais que adicionaram os elementos de um OpenFlow. O canal seguro e o protocolo OpenFlow serão portados para o sistema operacional e a tabela de fluxos fica em hardware já existente, tipo memórias TCAM.

Os *switches* OpenFlow se tornaram o meio mais difundido para implementar as SDNs. Analisar cabeçalhos Ethernet, IP ou TCP usando OpenFLow não é um problema. Mas se for preciso analisar um novo protocolo, também será necessário esperar o OpenFLow adicionar suporte a ele (ou poderá utilizar software ou hardware específicos e coprocessamento para uma análise por pacotes). Comercialmente, são implementados sobre *switches* Ethernet, utilizam parâmetros de campos de mensagens da pilha TCP/IP tanto para configuração de fluxos quanto para tomada de decisão nas ações das tabelas e não permitem alteração no nível do Enlace.

2.2.2 Redes Definidas por Software e implementadas em Hardware

Algumas iniciativas para incluir hardware reconfigurável no plano de encaminhamento são apresentadas nesta Seção.

FPGA-based Software Defined Radios

Bhatnagar et al. (2013) apresentam um método de desenvolvimento de hardware flexível para rádios definidos por software (*Software Defined Radio (SDR)*) aplicados a Internet do Futuro das Coisas (FIoT). Eles buscam uma solução para a aumentar o grau de flexibilidade no desenvolvimento de sistemas sem fio e permitir a prototipagem remota de protocolos sem fio em um hardware via Internet. Para tal, os autores apresentam um

esquema com SDRs contendo um FPGA Virtex 6, da Xilinx, combinado com um fluxo de desenvolvimento com ferramentas de síntese de alto nível (*High Level Synthesis (HLS)*). Uma camada física ZigBee foi implementada a partir de linguagem C para a geração de formas de onda.

A partir de especificações em alto nível para SDR, os autores testam a geração de Linguagem de Descrição de Hardware (HDL) com a ferramenta de síntese Catapult C e comparam o resultado com HDL codificado diretamente. Em termos de complexidade, o código VHDL gerado pela Catapult C é difícil de ser entendido em comparação com o codificado diretamente. Os autores concluíram que a maior vantagem do uso de síntese de alto nível é o tempo de desenvolvimento que foi reduzido drasticamente. Bhatnagar et al. (2013) indicam ainda que no futuro pode haver protótipo de redes sem fio flexíveis.

FPGA-based approach for organization of SDN switch

O trabalho de Kalyaev e Melnik (2015) se baseia na ideia de construir *switches* reconfiguráveis que têm o FPGA como hardware. Afirmam que o OpenFlow ainda é a melhor opção para implementação de SDN mas apontam os seguintes problemas com estes *switches*: *i)* a camada mais baixa de o *switch Open FLow (OF)* é a mesma que um *switch* tradicional, então às vezes fica muito caro realizar as atualizações e manter a compatibilidade dos dois tipos de rede; *ii)* nenhum dos *switches* tem compatibilidade 100% com todos os recursos do OpenFlow, mesmo por causa da adaptação que tem de ser feita aos tradicionais, ficando alguns recursos a não serem implementados; e *iii)* a guerra comercial faz com que nem sempre os fabricantes publiquem que os swithces suportam OpenFLow.

Os autores apontam os mecanismos de busca nas tabelas e análise de cabeçalhos como sendo os gargalos de processamento, afirmam que grande parte dos *switches* utiliza memórias *Ternary Content-Addressable Memory (TCAM)* para as tabelas de fluxos e isso pode ficar muito caro, além de enfrentar problemas nas atualizações de funções de *match*. Então eles propõem a construção de tabelas em FPGA que dão suporte ao OpenFlow. Apresentam cálculos que estimam o tempo de processamento para busca (*parsing/matching*) e simulam valores para um FPGA Virtex 6. A base de comparação é um *switch* de 48 portas 1 Gbps e um de 4 porta 10 Gbps, que os autores apontam serem a média do estado da arte. Não apresentam dados de simulação nem prototipagem.

Programmable hardware for SDN

O trabalho de Brebner (2015) se aplica ao contexto de SDNs e *Network Function Virtualization (NFV)*. Apresenta uma proposta de construção de equipamentos de rede programáveis que têm o FPGA como hardware. É uma proposta que usa um compilador para deixar a configuração do hardware ser ditada por linguagens mais abstratas e de alto nível e um mecanismo de atualização de *firmwares* quando necessário. O resultado é a

plataforma Xilinx SDNet (XILINX, 2017) para o projeto de plano de dados programável. A proposta considera a implementação de um mecanismo de configuração independente de protocolos que torna o equipamento ainda mais flexível que o OpenFlow para o plano de dados. É um desenvolvimento dos laboratórios da Xilinx.

Reconfigurable Technologies for Next Generation Internet and Cluster Computing

O trabalho de Unnikrishnan (2013) apresenta técnicas que integram FPGAs para habilitar plataformas de virtualização de redes. O sistema é focado em resolver problemas de escalabilidade a partir da tecnologia de virtualização por *containers*. Segundo os autores, a técnica melhora de uma a duas vezes o *throughput* e latência em relação às tecnologias atuais. Os autores mostram que o uso de reconfiguração parcial pode ser explorado para reconfigurar dinamicamente os parâmetros das redes virtuais sem afetar outras redes compartilhadas no hardware no caso de NFVs.

Propõem o Reclick como um novo modelo de programação para prototipar protocolos de rede em FPGA. Ele tem uma aplicação de entrada baseada na arquitetura de software Click (KOHLER et al., 2000) que é abstratamente mais alta que a maioria das linguagens de descrição de hardware e provê uma infraestrutura para o reuso de hardware que otimiza a utilização do FPGA para planos de dados virtuais. Os componentes individuais podem ser descritos em ReClick ou usando descrições *Register Tranfer Level (RTL)*. Como exemplo utilizam um roteador IPv4 que alcança 1 Gbps de operação numa NetFPGA.

Roteador SDN em hardware independente de protocolo com análise, casamento e ações dinâmicas

No trabalho desenvolvido por Pacífico et al. (2018), foi projetado e implementado em hardware um roteador SDN que tem em seu núcleo uma máquina virtual eBPF (uma máquina virtual com um conjunto de instruções bem definido usado para filtragem de pacotes). É um sistema independente de protocolo que permite fazer a análise, casamento e ações de forma dinâmica através de instruções eBPF. As instruções eBPF são geradas a partir de programas escritos em linguagem C ou P4. O roteador foi implementado em uma NetFPGA 1G e processa cadeias de 64 bits por ciclo de clock, fragmentando os pacotes para processamento. Um pacote de 1500 Bytes, por exemplo, leva $2us$ para ser processado. O roteador é capaz de armazenar e processar apenas um pacote por vez. O trabalho utiliza o padrão Ethernet e não faz considerações sobre modificações no nível de enlace da rede.

2.2.3 (Re)configuração de equipamentos de rede

Switches são majoritariamente construídos sobre *ASICs* devido ao desempenho. Assim, as configurações pós *bootstrapping* são relativas à configurações realizadas em software, geralmente por uma das interfaces LAN do *switch* e através de software proprietário, como acontece na linha Catalyst da Cisco (CISCO, 2017b) e se estende aos outros fabricantes. Com a chegada dos *switches* OpenFlow e o advento das SDN, essa estrutura se modifica de tal forma que o suporte para a configuração e reconfiguração pós *bootstrapping* pode existir tanto apenas no nível de descrição com linguagens de alto nível quanto com reflexo no hardware do encaminhamento, conforme alguns trabalhos listados a seguir.

Protocol Independent Forwarding e P4

De acordo com Brebner (2015), há uma pressão para evoluir o OpenFlow de forma a ampliar as tecnologias a que ele se destina, os protocolos que suporta, os cabeçalhos reconhecidos e a forma com que pacotes são processados, já que eles se baseiam nos chips de *switches* existentes. Neste contexto, em 2014/15 a *ONF* avançou no projeto *Protocol Independent Forwarding (PIF)* (ONF, 2017b).

A essência do PIF é que o plano de dados seja baseado em *switches* programáveis. Esta abordagem se apóia em dois extremos: a configuração pode definir a arquitetura do *switch* e do *firmware* customizado que pode então adicionar características que variam ao longo do tempo. O projeto foi concluído em 2017 e um dos resultados foi o *B Intermediate Representation (BIR)* (ONF, 2017a), cuja documentação está no *Github*. Esta direção do trabalho está agora em progresso pelo P4.org, o consórcio que trata da linguagem *Programming Protocol-Independent Packet Processors (P4)* para processamento de pacotes (P4.ORG, 2017).

P4 (BOSSHART et al., 2014) é uma nova linguagem de programação de alto nível criada para descrever o comportamento do plano de dados de qualquer sistema que encaminha, modifica ou inspeciona tráfego de rede. A linguagem P4 descreve o comportamento do plano de dados a partir de um paradigma baseado em casamento de ações (mas não especifica como ele é implementado). A linguagem não define os mecanismos pelos quais os programas são compilados, carregados e executados nos sistemas de processamento de pacotes, nem os mecanismos de encaminhamento de dados e nem os mecanismos utilizados pelo plano de controle para ações de *match* nas tabelas. O trabalho de interpretar o código P4 e gerar um código em mais baixo nível (por exemplo em HDL) relativo a um dispositivo em específico fica a cargo do fabricante.

P4FPGA

P4FPGA (WANG et al., 2017) é uma plataforma que realiza conversão de programa em linguagem P4 para System Verilog. O P4FPGA foi prototipado em um FPGA Xilinx da

família Virtex 7. Há três componentes: *i*) o gerador de código, que tem como funcionalidade produzir um pipeline de processamento de pacotes; *ii*) o sistema *runtime*, que fornece uma abstração independente do hardware para funcionalidades básicas incluindo gerenciamento de memória, gerenciamento de *transceiver* e comunicação controle/hospedeiro; e *iii*) o otimizador, que é utilizado para otimizar o paralelismo no hardware. P4FPGA embora não seja um comutador, permite que funções do nível do enlace sejam compiladas e o hardware correspondente é gerado em FPGA, entretanto, não faz considerações sobre enlace diferente do Ethernet e, por possuir uma biblioteca predefinida de funções de hardware, não permite parametrização das funções na subcamada de controle de acesso ao meio sem HDL previamente implementado.

PISCES

O PISCES, *a Programmable Protocol-Independent Software Switch*, na verdade é um comutador em software que suporta a linguagem P4 baseado no *Open vSwitch (OVS)*. É uma versão do OVS que permite *parse*, *match* e *actions* gerados pelo compilador P4 para descrever o plano de dados do comutador. Embora não seja projetado para ser executado em hardware, é interessante notar a tendência de virtualização dos *switches* nos *data centers*. Os autores, Shahbaz et al. (2016), afirmam que hoje os *data centers* possuem mais *switches* em software do que físicos o que justifica os trabalhos desta natureza. Cabe salientar toda a operação virtual acontece sobre uma infraestrutura física que, hoje, não oferece opções de flexibilização no nível mais baixo da camada de enlace.

Design, Implementation, and Test of a Tri-Mode Ethernet MAC on an FPGA

O trabalho de Kerling (2018), embora não esteja no domínio de SDN, apresenta um projeto para a subcamada MAC Ethernet e sua implementação em VHDL com prototipagem em FPGA. O objetivo principal é disponibilizar a subcamada de controle de acesso ao meio implementada em hardware de forma simplificada e apenas com funções consideradas essenciais pelos autores, tanto do ponto de vista interno como externo. O MAC suporta comunicação com a interface de transmissão da camada física via padrão MII em 10,100 e 1000 Mbps com detecção automática de velocidade. O trabalho também se dedica à implementação do nível de enlace da rede, mas se mantém no padrão TCP/IP.

2.2.4 Interface Hardware/Software

Os mecanismos que tentam diminuir a lacuna entre o desenvolvimento de aplicações utilizando FPGA e o software refletem diretamente na forma com que o *switch* pode ser (re)programado. Além das iniciativas para descrição de processamento de pacotes, há diferentes abordagens para a geração do hardware customizado a partir destas aplicações.

Labrecque et al. (2009) implementam o NetThreads, um sistema multiprocessado com processadores *multithread* (*soft SoPC multiprocessor*) para tarefas de processamento de pacotes e apresentam um *framework* para simulação e compilação que torna o sistema mais fácil para um programador. Sincronização é o principal problema deste multiprocessador. Os processadores são embarcados dentro do pipeline em uma placa NetFPGA. As funções de processamento são descritas em linguagem C e executadas nos microprocessadores sem um sistema operacional. O melhor resultado foi conseguido em uma aplicação *Network Address Translation (NAT)*, cujo processamento chega a 5k pacotes por segundo, com uma media de 4 ciclos por instrução.

A Xilinx tem a ferramenta SDNet (XILINX, 2017) que agora é compatível com o P4. É o mecanismo descrito por Brebner (2015) que inclui uma descrição das funções de rede em uma linguagem de alto nível (P4), um compilador proprietário para geração do HDL e um mecanismo de atualização de *firmware*, quando necessário.

Há ainda a abordagem SwitchBlade, apresentada por Anwer et al. (2010), que permite que os blocos de hardware sejam inseridos ou retirados sem a necessidade de ressintetizar o hardware. Os blocos de hardware ficam pré-sintetizados e os usuários selecionam quais deles serão necessários para processar os pacotes. Cada módulo é escrito em Verilog. Esta seleção é inserida em cada pacote de entrada na forma de um cabeçalho *SwitchBlaze* (*bitmap*), o cabeçalho é inserido na chegada do pacote e retirado na saída. Cada módulo no caminho de dados examina este *bitmap* que é anexado ao pacote para processar ou não aquele pacote. Este sistema foi desenvolvido especialmente para a virtualização de planos em NFV.

A arquitetura de software Click (KOHLER et al., 2000), já citado anteriormente, é um *framework* para construir softwares modulares de roteadores. Click permite criar configurações a partir de módulos chamados elementos. Estas configurações são escritas em uma linguagem Click customizada. Entretanto, não há reuso dos blocos diretamente no hardware. Nikander et al. (2010) descrevem um experimento para investigar se ferramentas comerciais de HLS podem suportar síntese de configurações modulares Click para o hardware. Os autores propõem também um conjunto de ferramentas para compilar Click (*C++ based*) em Verilog utilizando uma representação intermediária (LLVM). Elas geram um *RTL* sintetizável a partir de códigos C/C++ para códigos que foram escritos com o hardware já em mente, mas existe lacuna entre os mecanismos conhecidos de reuso de software e reuso de hardware para redes devido a questões tais como chamadas de função, ponteiros de ponteiros e alocação dinâmica de memória.

2.2.5 Conclusão do Capítulo

Os trabalhos estão numa linha de pesquisa similar a esta, ora propondo hardware de rede reprogramável ou reconfigurável, ora mecanismos de programação e configuração destes equipamentos, ora linguagem para processamento de pacotes. Embora nenhuma

delas seja direcionada por uma arquitetura de Internet do Futuro especificamente, há uma série de semelhanças com a proposta aqui apresentada, que são resumidas na Tabela 1. Esta tese traz como contribuição principal, quando comparada aos trabalhos relacionados, a possibilidade de modificação da camada de acesso ao meio em um ambiente de orientação a *Workspaces*.

Tabela 1: Visão geral dos trabalhos relacionados

Trabalho	**Ano**	**Elemento de Rede**	**Hardware ou Software**	**Reprogramável**	**Orientado a *Workspace***	**Modificação da MAC**
(MCKEOWN et al., 2008) Openflow	2008	*switch*	Fwd Hardware Ctrl Software	Sim	Não	Não
(BHATNAGAR et al., 2013) FPGA-based Software Defined Radios	2013	Radio para FIoT	Fwd Hardware Ctrl Software	Sim	Não	Pode haver
(UNNIKRISHNAN, 2013) Reclick	2013	Flexível	Fwd Hardware *Toolchain* Software	Sim	Não[a]	Não
(KALYAEV; MELNIK, 2015) FPGA-based approach for SDN switch	2015	*switch*	Hardware (Proposta)	Sim	Não	Não
(XILINX, 2017) Programmable hardware to SDN	2015	Não especificado	Hardware (Proposta) *Toolchain* Software	Sim	Não[b]	Não
(SHAHBAZ et al., 2016) PISCES	2016	Flexível	Software	Sim	Não	Não
(WANG et al., 2017) P4FPGA	2017	Flexível	Fwd Hardware *Toolchain* Software	Sim	Não[c]	Não
(KERLING, 2018) Tri-Mode Ethernet MAC on an FPGA	2018	MAC Ethernet	Hardware	Não	Não	Não
(PACíFICO et al., 2018) Roteador SDN em Hw	2018	Roteador	Hardware	Sim	Não	Não

[a] Não tem na biblioteca. O HDL deve ser escrito para inclusão antes da compilação.
[b] Não especificado.
[c] Não tem na biblioteca. O HDL deve ser escrito para inclusão antes da compilação.

Capítulo 3

Método

Para direcionar o trabalho de forma lógica um método de pesquisa deve ser estabelecido. Para Wazlawick (2009) o método consiste na sequência de passos necessários para demonstrar que o objetivo proposto foi atingido, i.e., se os passos definidos no método forem executados, os resultados obtidos deverão ser convincentes. Neste contexto, este capítulo se atenta ao método adotado e tem como principais contribuições *i)* descrever a sequência de passos realizados na construção desta pesquisa e *ii)* delinear seu escopo.

Esta pesquisa tem caráter exploratório e experimental, segundo classificação de Wazlawick (2009), e desenvolve o que ele chama de "apresentação de algo diferente", onde o autor propõe uma forma diferente de resolver um problema, cria e realiza os testes que demonstram que sua proposta apresenta algum tipo de melhoria em relação a outras. Ainda quanto ao método de abordagem, a pesquisa pode ser classificada de acordo com o que Lakatos e Marconi (2003) classificam como Método Hipotético-Dedutivo, que reflete o processo de percepção de um problema, proposição de uma solução e realização de testes pela observação e experimentação.

Este trabalho tem o objetivo de projetar um elemento de rede (*switch*) cuja subcamada de controle de acesso ao meio seja definida pelos requisitos de comunicação de *Workspaces*.

É importante que o objetivo esteja bem definido para que se tome qualquer decisão relacionada ao método. Assim, uma vez que o objetivo deste trabalho é projetar um *switch* cuja subcamada de controle de acesso ao meio seja definida pelos requisitos de comunicação de *Workspaces* e pretende-se desenvolver o módulo de encaminhamento do *switch* ETArch, o método deve contemplar as etapas de especificação, implementação e testes, bem como um sistema capaz de demonstrá-lo em funcionamento. A Estrutura Analítica do Projeto (EAP) é apresentada na Figura 7 e criada com base nas práticas definidas pelo *Project Management Institute (PMI)* (PMI, 2013).

De acordo com Wazlawick (2009), a revisão bibliográfica não é parte do método propriamente dito, mas um prerrequisito. Ainda assim, aqui será incluída para uma visão completa da sequência lógica de ações. Na Figura 7, esta etapa é compreendida pelo ramo Revisão de Literatura.

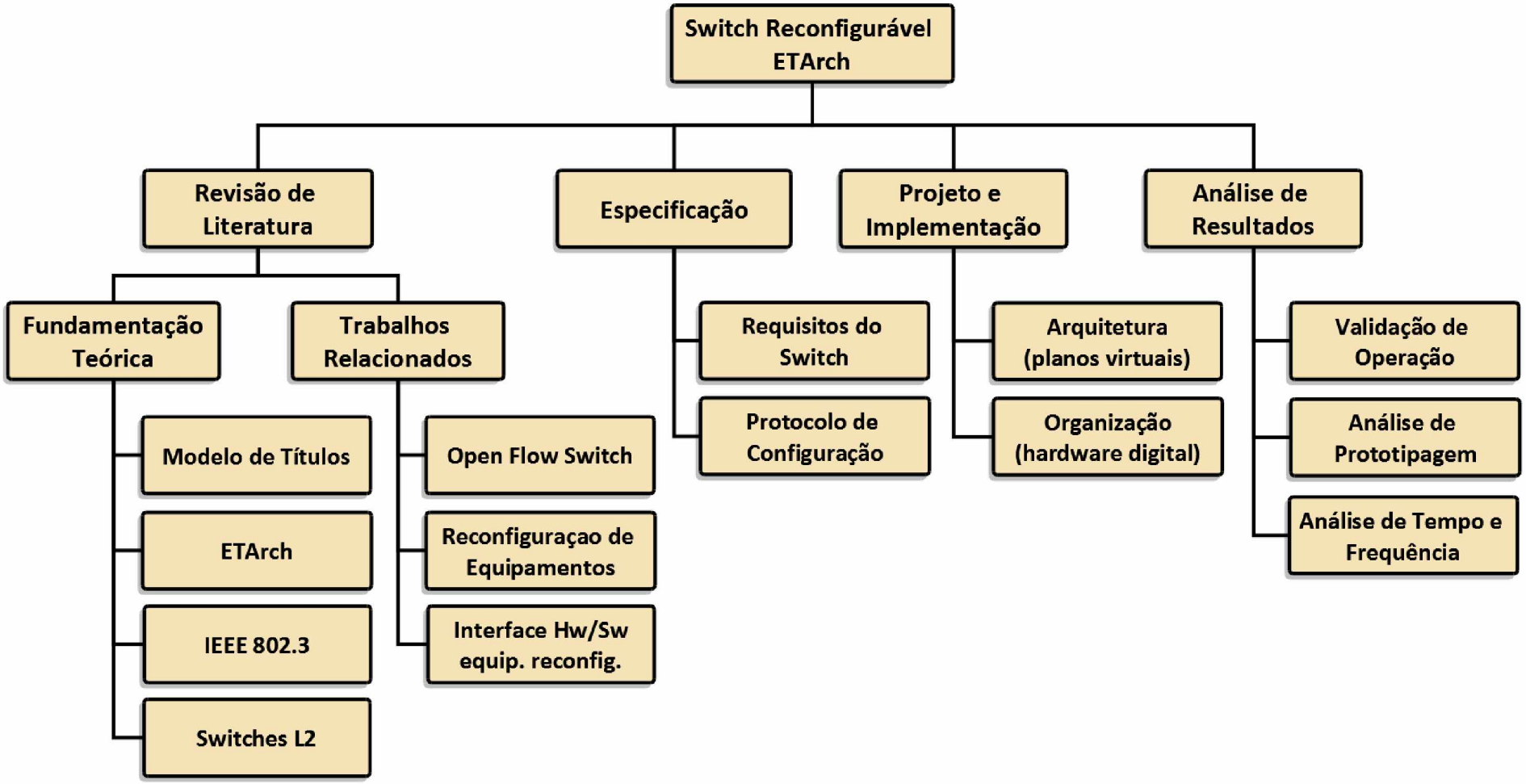


Figura 7: Método: Estrutura Analítica do Projeto

3.1 Revisão de Literatura

Esta etapa compreende basicamente duas linhas: conceitos para fundamentação teórica e soluções relacionadas ao encaminhamento de *switches* baseadas em Computação Reconfigurável. Pela decomposição nos níveis da EAP para este ramo, ainda na Figura 7, esta etapa envolve:

- ❏ O resgate do Modelo de Títulos e sua concepção conceitual da nova proposta de endereçamento horizontal não baseada em TCP/IP;
- ❏ A proposta arquitetural que aproxima o Modelo de Títulos de uma implementação, i.e., ETArch;
- ❏ Mecanismos atuais de encaminhamento de pacotes no nível da camada de enlace da arquitetura Internet;
- ❏ Soluções em hardware reconfigurável para flexibilização do hardware de SDNs e
- ❏ Interfaces de configuração e reconfiguração de dispositivos de rede.

3.2 Especificação de Requisitos

A arquitetura ETArch se divide em plano de dados e plano de controle. No plano de controle, os DTSAs administram o DTS e a criação, manutenção e exclusão de *Workspaces*. Para a comunicação entre entidades e entre DTSA, os *switches* são os elementos de rede que dão suporte à troca de mensagens no plano de dados.

Workspaces podem ter diferentes requisitos de comunicação de acordo com políticas da rede e é previsível que estes requisitos se tornem cada vez mais complexos, acompanhando as aplicações. Entretanto, é preciso definir neste momento quais são os requisitos considerados essenciais para que uma comunicação possa de fato acontecer integralmente utilizando a arquitetura ETArch e *Workspaces*. Em outras palavras, é necessário dar forma a um *Workspace* em se tratando, pelo menos, de mecanismos de encaminhamento de mensagens, visto pelo switch.

Ainda pela EAP da Figura 7, o primeiro item desdobramento do ramo Especificação é a especificação de requisitos do switch. É importante salientar que este trabalho trata apenas da comunicação que tem reflexo direto nos mecanismos do Switch. A completude da rede é verificada pela combinação dos serviços oferecidos pelo DTSA, pelo *switch* e pelas entidades. Neste caso, o *switch* é alvo de especificações e o restante dos entes participantes são objetos de pesquisa de trabalhos paralelos a este.

Uma vez especificados os requisitos essenciais para estabelecimento, manutenção e exclusão de *Workspaces* - chamados requisitos básicos -, é preciso delinear quais são as atribuições do DTSA e quais são do Switch. O *switch* ETArch atua ativa ou passivamente

com funções equivalentes ao encaminhamento e roteamentos de pacotes da Internet, mas no paradigma de Orientação a *Workspace* o algoritmo de roteamento é executado pelo DTSA. Ainda não há um claro limite entre as operações do hardware e a do software no encaminhamento dos dados e esta etapa trata disso.

Com a proposta de um mecanismo de encaminhamento operando em um *switch* ETArch surge também a necessidade da especificação de um protocolo para operar no plano de controle. O protocolo deve definir o padrão de comunicação entre DTSA e *switch* para a configuração dos *Workspaces*. Então, isso também é feito nesta etapa. Os resultados das atividades desta etapa são apresentados no Capítulo 4.

3.3 Projeto e Implementação

Acabada a fase de especificação, parte-se então para o projeto e implementação. São duas vertentes a serem trabalhadas: *i)* a interface de controle entre controlador DTSA e *switch* e *ii)* os mecanismos de encaminhamento de dados. Para ambos são realizados tanto a definição dos parâmetros arquiteturais como o projeto organizacional no nível de transferência entre registradores (ainda de acordo com a Figura 7).

A interface de comunicação é a via pela qual o controlador envia mensagens de controle para (re)configuração do *switch* e a forma com que estas mensagens são entendidas nos blocos de hardware. Por mecanismos de encaminhamento entende-se literalmente a distribuição dos dados entre entes comunicantes conectadas ao Switch.

No âmbito da implementação, a plataforma de base do hardware é a placa DE2 com FPGA Altera da família Cyclone II (ALTERA, 2006). É um kit de desenvolvimento de baixo custo com um FPGA de baixa velocidade de processamento se comparado outras famílias empregadas em aplicações de rede. Apesar de a placa disponível não ter limitações consideráveis na quantidade e velocidade dos recursos disponíveis, a prova de conceito a ser realizada pode ser estendida às tecnologias mais recentes quando os requisitos de desempenho e *throughput* forem determinantes para a rede. Conforme explica a Subseção 3.4, placas DP83848 Ethernet, comercializadas pela Waveshare (2015), também foram utilizadas como interfaces de rede.

A codificação foi feita em *Very High Speed Integrated Circuits Hardware Description Language (VHDL)* e seguindo a sequência do fluxo de projeto tradicional em FPGA: projeto da arquitetura, simulação funcional e síntese (ASHENDEN, 2007). Os blocos de hardware foram implementados de forma modular, i.e., com fraco acoplamento e sendo simulados e prototipados individualmente. Os blocos foram integrados progressivamente. Várias adaptações foram feitas ao longo do processo de desenvolvimento, tais como a largura de barramento implementada, o tamanho das memórias e o comprimento dos registradores. Isso se deve à limitação de recursos lógicos do FPGA e os resultados deste processo estão apresentados no Capítulo 5.

As seguintes plataformas e ferramentas foram utilizadas para a prototipagem:

- ❏ 1 placa de prototipagem DE2 com FPGA Altera Cyclone II;
- ❏ 4 placas Waveshare DP83848 Ethernet;
- ❏ Quartus II Web Edition versão 13.1 para edição de códigos em linguagem de descrição de hardware, síntese e configuração do FPGA; e
- ❏ Modelsim Altera 6.6d Starter Edition para a simulação funcional dos blocos de hardware.

3.4 Prova de conceito e validação

A última etapa do método, definida pelo ramo Análise de Resultados na Figura 7, trata da demonstração de funcionamento e validação dos mecanismos implementados. O objetivo é demonstrar a criação e manutenção de diferentes *Workspaces* por um mesmo *switch* e a parametrização de requisitos de cada *Workspace*. Neste primeiro momento, não é objetivo do trabalho obedecer a parâmetros estritos de desempenho, embora eles ofereçam incentivos para a codificação.

Uma forma de demonstrar o funcionamento é montar um ambiente de rede local com entidades, *switch* e uma versão do controlador DTSA com funções reduzidas. O sistema de demonstração construído conta com estrutura geral mostrada pela Figura 8.

Para o *switch*, a plataforma de prototipagem é o kit de desenvolvimento DE2 com FPGA Altera Cyclone II. O emulador de DTSA é um computador pessoal com sistema operacional Ubuntu 16.04 executando software Ostinato para a injeção de quadros na rede (SRIVATS, 2019). A interface Ethernet *enps0* desta máquina está conectada à porta 1 do *switch*, por onde o DTSA e o *switch* comunicam entre si.

O *switch* conta com quatro *Datapaths* de dados, sendo um *Datapath* para cada PHY. Assim, dois outros *Datapaths* também estão conectados a placas Ethernet em computadores pessoais. No *Datapath3* está conectada a interface *enps1* do mesmo computador que simula o DTSA. Para esta interface, também está sendo executado o software Ostinato para a geração de quadros de dados. Todos os quadros enviados têm tamanho 512 bits, que é o tamanho mínimo transmitido no padrão IEEE 803.2. Ao *Datapath2*, conecta-se um outro computador pessoal com sistema operacional Windows 7 utilizado para para recebimento de quadros. Todas as interfaces dos PCs são monitoradas pelo Wireshark (2019) para garantir que os quadros gerados pelo Ostinato estejam sendo enviados, bem como para monitorar todo e qualquer quadro sendo recebido.

O controlador DTSA é objeto de outras pesquisas do mesmo grupo, de forma que para a demonstração neste trabalho uma ferramenta capaz de gerar quadros com formato e

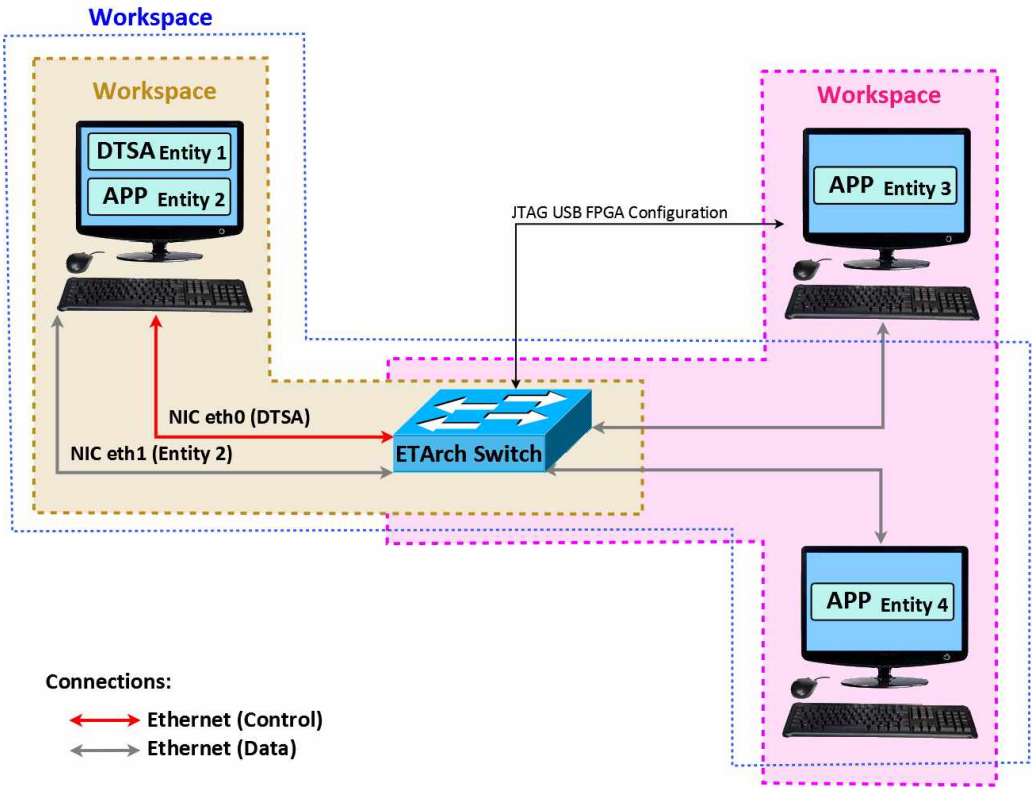


Figura 8: Sistema de demonstração construído

conteúdo compatíveis com o protocolo de configuração especificado se aplica integralmente ao papel de *Resource Adapter* do DTSA (OLIVEIRA. et al., 2019).

A Figura 9 ilustra as conexões na placa que hospeda o processamento principal:

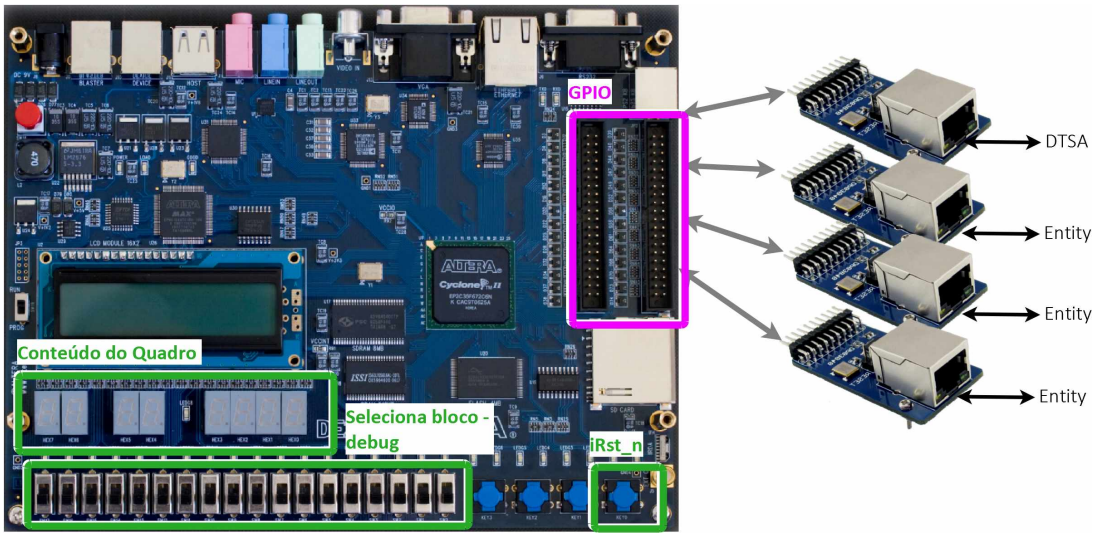


Figura 9: Configuração do kit DE2 e interfaces de rede

A placa DE2 possui um conector RJ45 para interface Ethernet de fábrica, mas não foi utilizado devido ao controlador PHY já implementar o MAC no mesmo chip, não

sendo possível alteração. Então, foram utilizadas quatro placas auxiliares contendo um *DP83848C Ethernet Physical Layer Transceiver* em cada e conectadas ao FPGA através dos pinos de expansão GPIO da placa. Os LEDS, *switches* e Displays da placa foram utilizados para *debug*, de tal forma que se pode monitorar as transições das máquinas de estado de cada componente e visualizar o conteúdo de cada quadro que passa pelo *switch*, inclusive os que chegam com erro.

Montado o sistema, a realização de testes envolve demonstrar troca de mensagens com significação semântica entre entes comunicantes. Os processos a serem monitorados no experimento são descritos na Tabela 2. A partir deles, infere-se variáveis experimentais (E) e variáveis medidas (M) para controlar a análise dos resultados. Quatro processos foram enumerados para ser observados

Tabela 2: Processos avaliados

Nome	**Variáveis**	**Descrição**
P1	*E1: Disparo de mensagem do DTSA* *M1: Configuração feita no FPGA*	Testar recebimento e reconhecimento de mensagens de configuração vindas do DTSA
P2	*E2: Solicitar criação e exclusão de Workspace pelo DTSA* *M2: Workspace operacional*	Testar criação de múltiplos *Workspaces*
P3	*E3: Envio de mensagens por diferentes entidades a diferentes Workspaces* *M3: Entrega de mensagens aos Workspaces*	Testar transmissão e entrega de dados entre diferentes *Workspaces*
P4	*E4: Modificar parâmetro de Workspace* *M4: Workspace operacional antes e depois da edição*	Avaliar edição de *Workspace*

3.5 Fluxograma de Atividades

Visto que o método compreende uma sequência lógica de passos, a Figura 10 ilustra o fluxo de desenvolvimento indo desde a especificação dos requisitos até o teste de troca de mensagens entre DTSA e entidades em um ambiente de rede local com comunicações via *Workspace*. Cabe chamar atenção neste ponto para as atividades *A4* e *A5* da fase de Projeto e Implementação. Elas realimentam um ciclo de desenvolvimento iterativo baseado na prototipagem rápida em FPGA.

A escolha por uma implementação em hardware reconfigurável foi motivada por dois fatores principais. O primeiro é que os mecanismos de encaminhamento do plano de dados de *Switches* lidam com o processamento de grandes quantidades de quadros encaminhados,

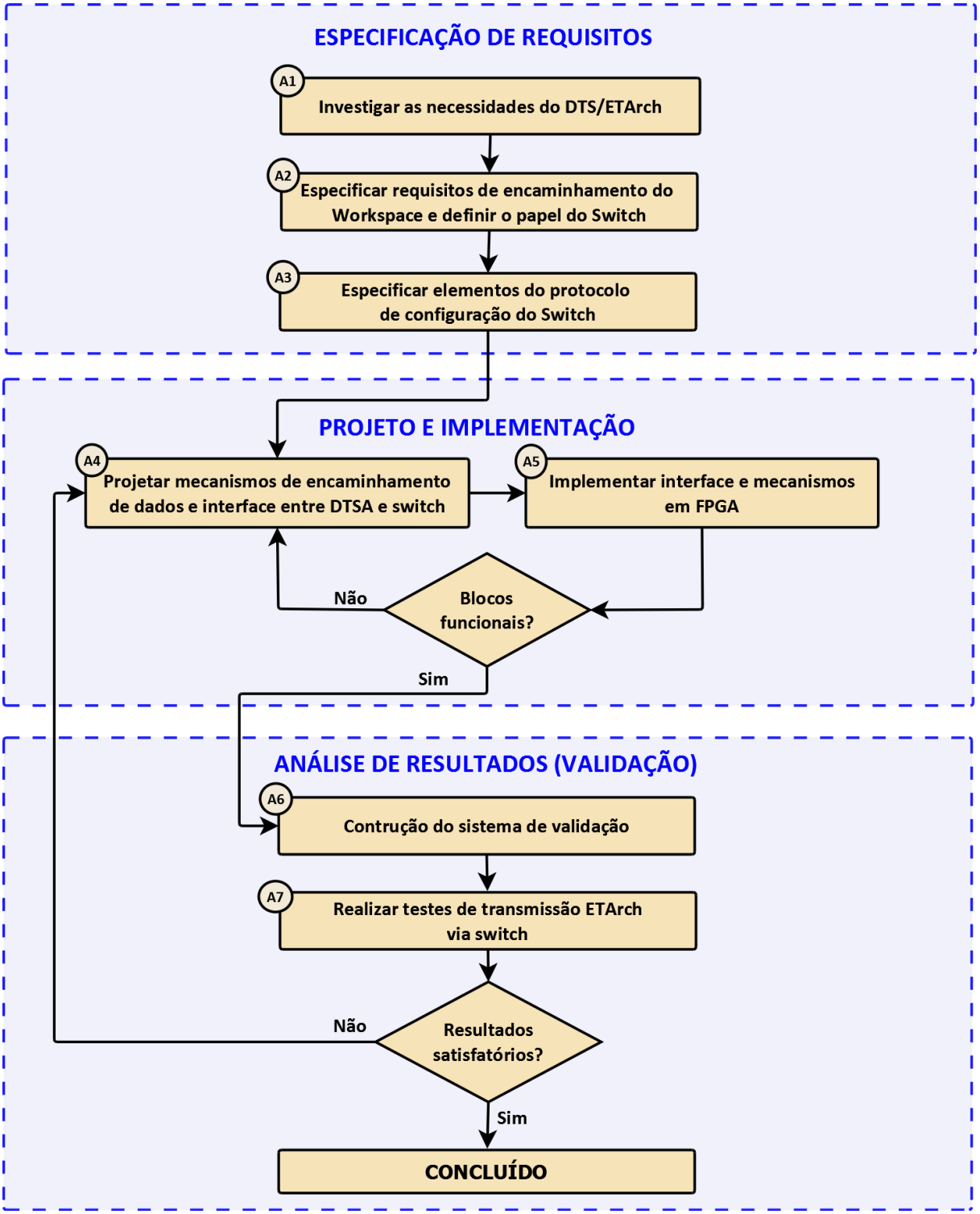


Figura 10: Método: Fluxograma de atividades

i.e., processamento rápido de muitas pequenas porções de dados, o que é feito com melhor desempenho em hardware. Ainda que desempenho não seja um dos requisitos nesta etapa do desenvolvimento, prototipá-lo em hardware reconfigurável facilita a continuidade do processo. Em segundo, a possibilidade de realizar melhoramentos nas especificações e

obter prototipagem rápida simultaneamente. Este é o ponto que define a simultaneidade entre as tarefas *A4* e *A5* do Projeto e Implementação.

Capítulo 4

Especificação de Requisitos do Switch e Protocolo de Configuração

Com a horizontalização do endereçamento e a não utilização de endereços desde a camada de *Communication* (Figura 4), a comunicação entre enlace e a camada de serviços passa a obedecer diferentes padrões em relação aos existentes no encapsulamento TCP/IP. Assim, este trabalho foca no *switch* para a arquitetura ETArch e pretende especificar e projetar a interface de controle entre a camada *Communication* e a camada *Link*. Além disso, este trabalho se aprofunda nos mecanismos da camada *Link*, sua interface entre hardware e software e projeta e implementa em hardware um protótipo do conjunto de serviços básicos da camada *Link* da ETArch.

Por enquanto, não precisamos da flexibilização do hardware de tal forma a atender diferentes arquiteturas de rede não compatíveis com TCP/IP. Precisamos de um switch que interconecte nós ETArch, que já ofereça suporte à criação e manutenção de *Workspaces* e, principalmente, que possa, na interface hardware/software do enlace, trazer mecanismos para o hardware que refletirão diretamente nos requisitos dos *Workspaces* definidos no nível da aplicação. Paralelamente, o *switch* deve ter uma interface configurável capaz de entender a solicitação dos requisitos solicitados e ativar/desativar/reconfigurar sua organização diretamente no hardware.

A aproximação semântica entre as camadas proposta pelo Modelo de Títulos pode obter resultados melhores e mais escaláveis se considerar também a camada ETArch *Link*, visto que ela também pode compartilhar do novo esquema de endereçamento. Isso se torna um forte motivador para a construção de um comutador reconfigurável, que pode até manter o enlace compatível com Ethernet, do ponto de vista físico, mas que seja passível de abstrair esta camada para futuro rearranjo.

O conjunto de serviços é definido neste trabalho com base na ontologia definida em (PEREIRA, 2012). É esperado do *switch* que ele reconheça uma solicitação de reconfiguração para inserção e exclusão de entidades quando solicitados pelo DTSA. Deve também ser capaz de criar *Workspaces* a partir de uma lista de capacidades, bem como modificar

as capacidades de um *Workspace* em operação.

4.1 Requisitos e Suporte do Hardware

O primeiro passo é definir, a partir da arquitetura ETArch, os requisitos que compõem o conjunto inicial de recursos do *switch*. Um *switch*, por si só, pode ser tão complexo quanto maior o número de funcionalidades que adicionamos. Do ponto de vista da reconfiguração, parametrizar capacidade e requisitos pode envolver um conjunto bastante complexo de funcionalidades.

Cabe lembrar que há uma pesquisa em andamento, paralela a esta, para definir como os requisitos de uma entidade são passados ao DTSA. Neste trabalho, o foco está na interface DTSA/*Switch*. Para tal, a ontologia representada inicialmente por Pereira (2012) serve de referência na representação dos requisitos. A Figura 11 recupera esta estrutura ontológica e adapta em função da aplicabilidade à ETArch. Para o domínio deste trabalho, os itens *Layer, DTS, Title e Entity* são bem definidos e finitos, enquanto que o item *Requirement* pode ser tão populado quanto for a complexidade adicionada à comunicação. Vale lembrar, novamente, que os requisitos aqui são a respeito única e exclusivamente da Entidade tipo *Switch*.

Na Figura 11, o lado direito apresenta as subclasses baseadas na ETArch. Para a *Layer*, por exemplo, uma instância pode ser marcada como *Application*, *Communication* e *Link*. Em relação ao DTS, pode se tratar de um domínio de dados ou de controle. O título por ora não obedece a um padrão determinístico, pois é função do gerenciador de títulos. Eles são selecionados obedecendo apenas ao formato estabelecido ainda neste capítulo. Já na classe *Entity*, limitamos às opções a DTSA, *switch*, *Workspace* e *Application*. Este conjunto de entidades é não exaustivo. Foram selecionados os principais elementos de comunicação com o objetivo de ter um conjunto restrito de opções na fase de prova de conceito.

Por fim, a classe *Requirements* apresenta seis subclasses: *Send/Receive*, *Mobility*, *Confirmed Delivery*, *Quality of Service (QoS)*, *Security* e *High Availability*. Embora haja modelos de comunicação para balizar a completude de novas arquiteturas de rede, conforme discutem Einsiedler et al. (2013), este conjunto compõe um plano inicial suficiente para a operacionalização de conexões aplicáveis a comunicações além de demonstração acadêmica. É uma união das principais funcionalidades dos serviços e protocolos de rede, atacando diretamente o problema do *Multicast* na Internet. Cabe, posteriormente, adicionar outros requisitos que vão atender aplicações mais específicas e aumentando a base de dados para parametrização.

O conjunto de requisitos é separado em dois subconjuntos, os básicos e os avançados. O critério adotado foi o seguinte: para qualquer comunicação acontecer, por mais elementar que seja, ela dependerá dos requisitos básicos. Os avançados são acrescentados à medida

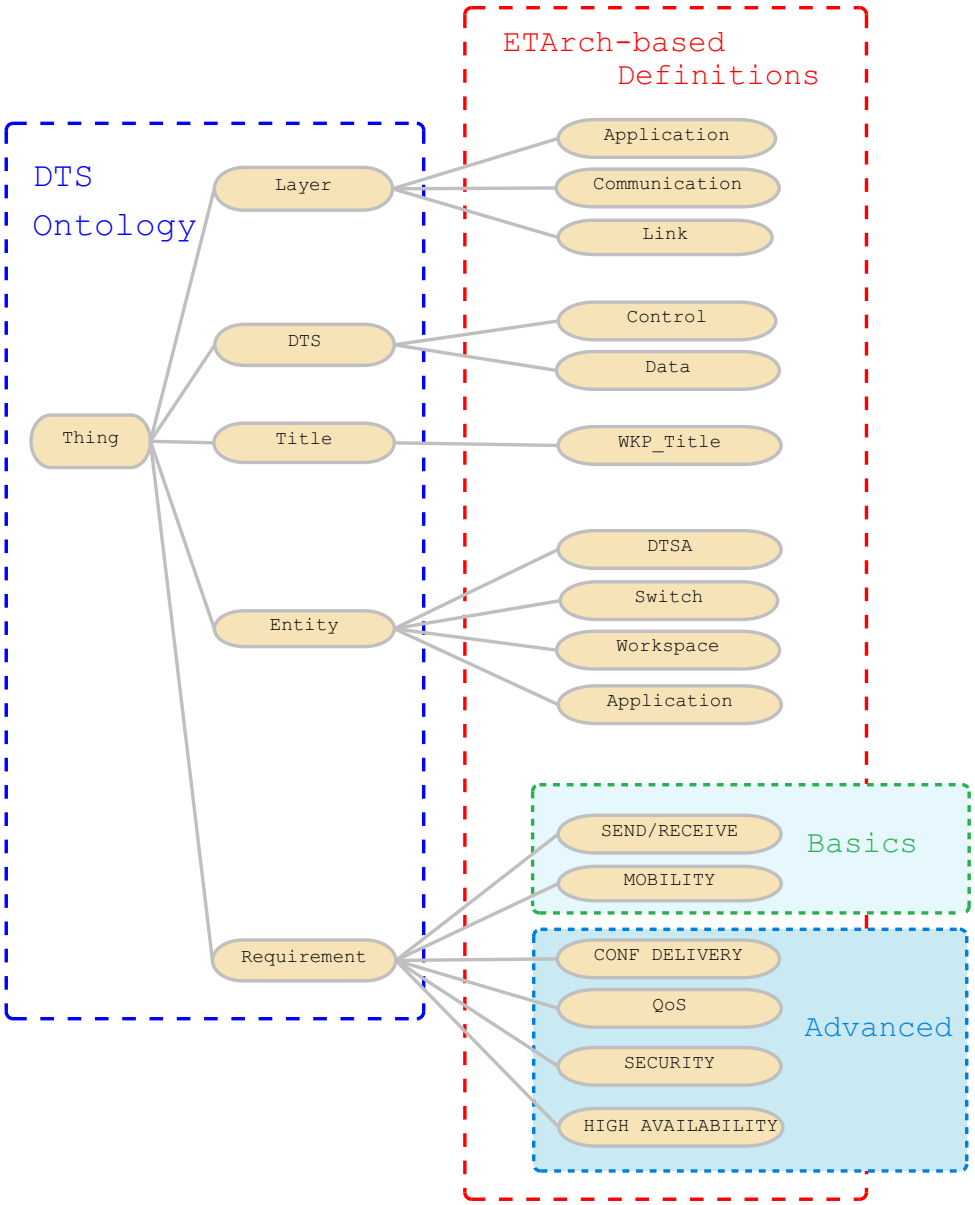


Figura 11: Conjunto de Requisitos

que os *Workspaces* ficam mais complexos.

A Figura 12 aprofunda na hierarquização da classe dos requisitos para caracterização dos *Workspaces*. O último nível apresenta os parâmetros de cada requisito, conforme itens a seguir:

- *SEND/RECEIVE*: O *switch* deve manter em hardware as tabelas de *workspaces*, mecanismos de encaminhamento e estado atual das conexões. Este é um requisito particular porque está mais relacionado ao provimento de funções básicas do que com a parametrização de funcionalidades. Todos os outros contam com o funcionamento próprio deste para que suas operações tenham alguma significação semântica.

- *MOBILITY*: Switch deve dar suporte à mudança de ponto de conexão de uma

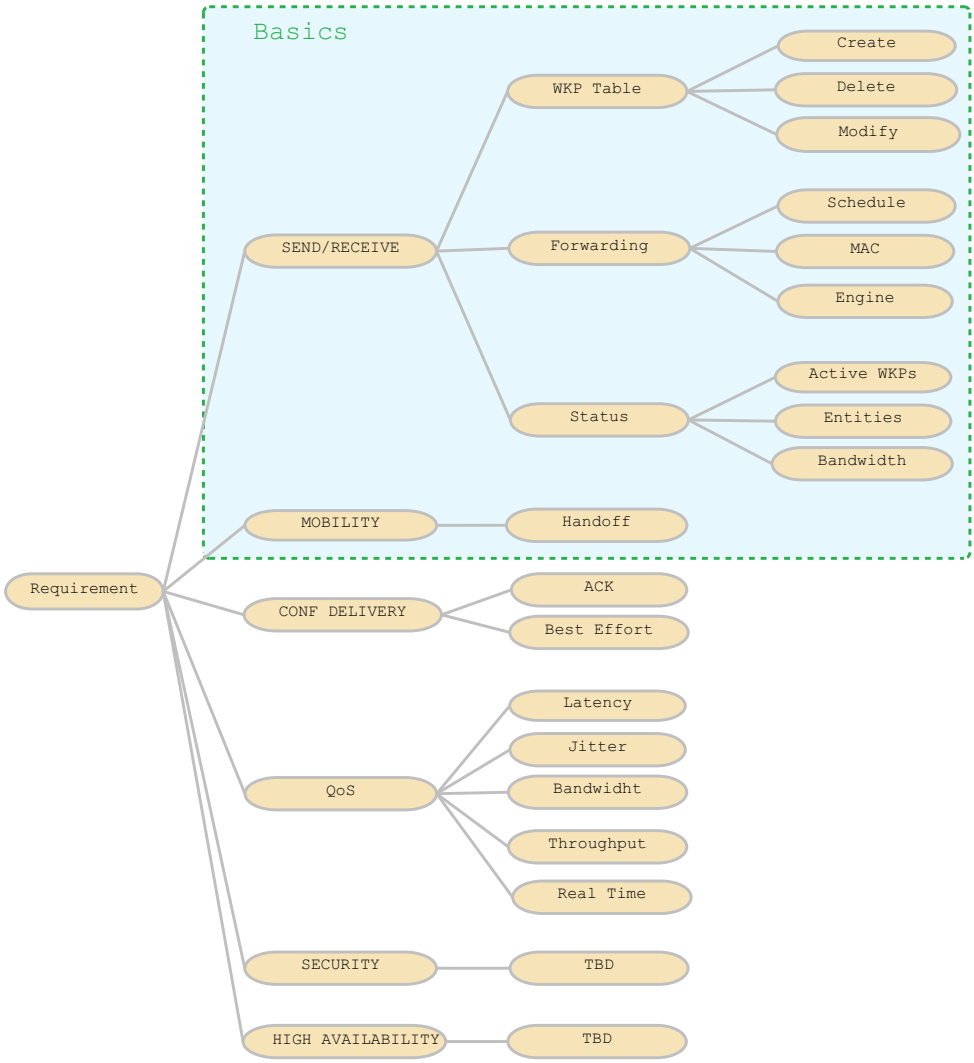


Figura 12: Conjunto de Requisitos - Caracterização de *Workspace*

entidade de forma transparente.

- *CONFIRMATION DELIVERY*: A entrega de mensagens pode ou não ser confirmada, depende da natureza da conexão estabelecida.

- *QUALITY OF SERVICE*: Requisito mais complexo, pois envolve uma série de parâmetros que no futuro exigirão grandes esforços do hardware. Controle de latência, jitter, largura de banda, vazão e exigências temporais no nível do enlace compõem um conjunto de parâmetros suficiente para extrair graus de QoS de aplicações. É pela parametrização destes itens que uma aplicação exigirá infraestrutura diferente, por exemplo, para transmissão de arquivos, *streamings* e audio/videoconferências.

- *SECURITY*: Depois de um esforço inicial, optou-se por destinar uma pesquisa paralela a este quesito segurança, já que é condição necessária nas comunicações e

no projeto da rede, mas envolve discussão desde o nível físico até as políticas de segurança da rede.

- *HIGH AVAILABILITY*: Da mesma forma que o requisito anterior, a alta disponibilidade requer um estudo mais específico, pois o paradigma cliente-servidor foi substituído pelo entidade-entidade, modificando protocolos e cenários típicos de ambientes de alta disponibilidade. Não se pode afirmar, sem exaustiva investigação, que os protocolos *First Hop Router Protocol (FHRP)* (CISCO, 2018) ainda se adaptam a estas novas arquiteturas de rede. O protocolo *High Availability Router Protocol (HARP)* foi especificado por este grupo de pesquisa no trabalho de Oliveira, Mesquita e Rosa (2013), implementado diretamente em hardware, mas em redes IPv4 que operam na filosofia mestre-escravo. Não se pode afirmar que elas serão aplicáveis neste cenário.

Até então a discussão se atenta aos requisitos, logo, deve-se partir para o projeto dos mecanismos no *switch* e definir o papel do hardware no provimento deles. Nem todos eles serão implementados de imediato. Como exposto anteriormente, o projeto e implementação do suporte em hardware acontecerá para o requisito fundamental de *SEND/RECEIVE*. A Mobilidade também será implementada já que seu suporte em hardware depende apenas do reuso dos blocos já implementados nos mecanismos de encaminhamento.

A Figura 13 acrescenta mais um nível à hierarquia e ilustra o papel do *switch* para os requisitos destacados. Requisitos em verde são os que estão implementados nesta pesquisa. A implementação da tabela de títulos e *Workspaces*, do mecanismo de encaminhamento em si e do mecanismo para informar os estados das conexões é vista no nível do hardware como uma combinação de blocos de hardware independentes e isolados - neste caso, referentes ao quinto nível (mais a direita) da árvore da Figura 13.

Como explicado anteriormente, o suporte a mobilidade fica resolvido com os mecanismos de *send-receive*. Ademais, implementar a parametrização e priorização de *Workspaces* significa dar um suporte inicial à exigência de QoS, por isso da representação não usual do requisito QoS na Figura 13.

A implementação dos elementos referentes ao *Status* do *switch*, bem como das suas conexões, é um ponto de interseção entre esta pesquisa e outra pesquisa do mesmo grupo que envolve co-design das funções do *switch*. A leitura de estatísticas da operação será feita de forma co-processada em sistema com sistema operacional embarcado, mas os mecanismos para coleta de dados já estarão implementados desde o presente trabalho.

O conjunto de mecanismos que atende o *SEND/RECEIVE* traz para o hardware o plano de dados/encaminhamento da ETArch e deve observar (e implementar) uma série de especificações condizentes com tal arquitetura. Estas especificações serão mapeadas para o hardware:

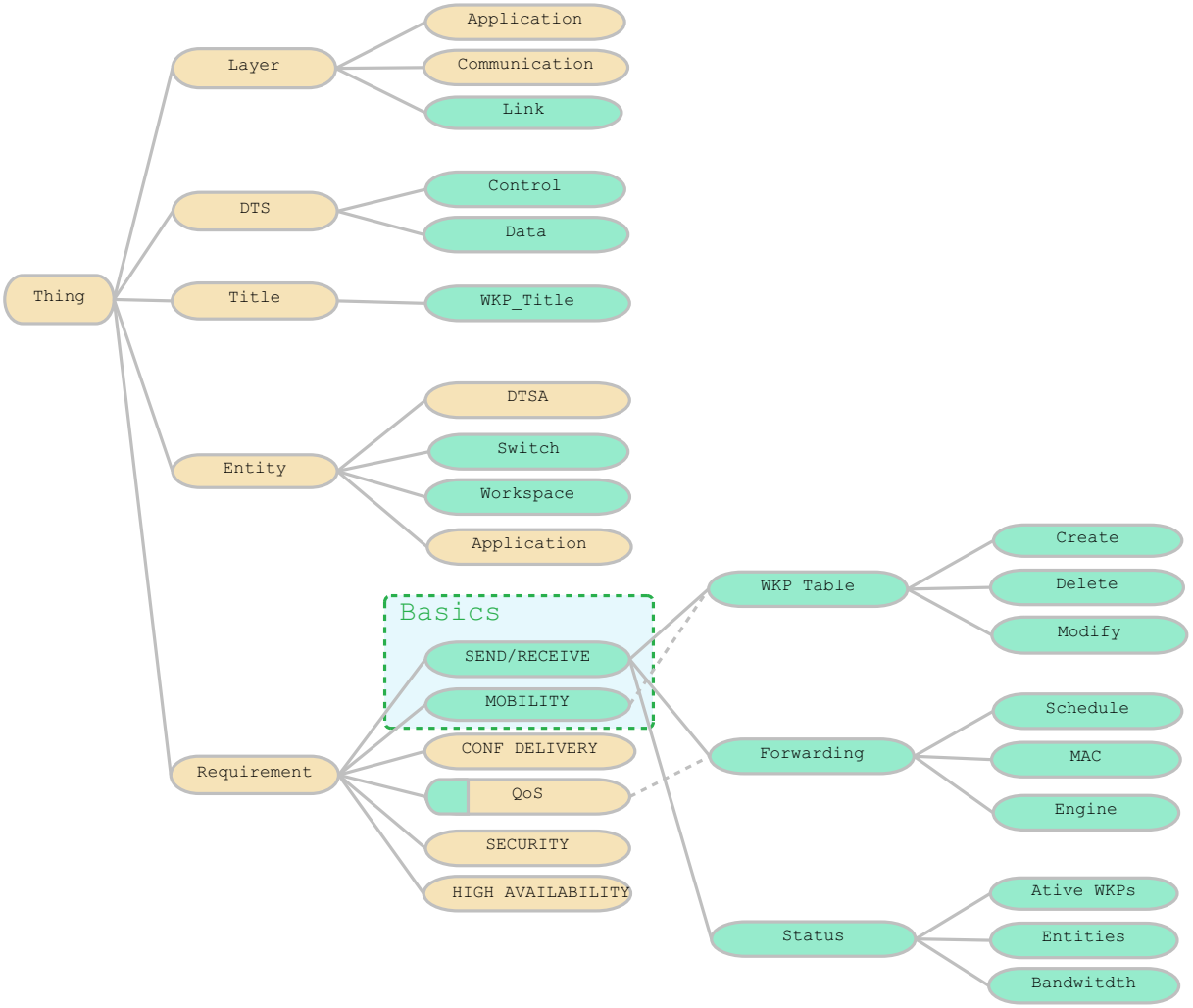


Figura 13: O suporte do hardware para os mecanismos de encaminahmento

1. Receber e enviar mensagens entre os entes comunicantes (que envolvem o *switch*):
   - ❏ Entidade → DTSA
   - ❏ Entidade → Workspace
   - ❏ DTSA → DTSA
   - ❏ DTSA → Workspace
   - ❏ DTSA → Entidade
   - ❏ DTSA → Switch
   - ❏ Switch → DTSA
2. Preparar para reconhecer títulos.
3. *Switch* deve associar portas pertencentes a cada *Workspace*.

4. Camada de abstração para abstrair o PHY que, embora necessário para a comunicação, está um nível abaixo da camada que acomoda o mecanismo de controle de acesso ao meio.

5. Construção da tabela de *Workspaces* a partir de mensagens de configuração do DTSA. Parâmetros: *Nome, Portas, Requisitos.*

6. O alcance de entidades específicas pelo DTSA fica a cargo do algoritmo de roteamento, que é *out-of-band*, i.e., executado no âmbito do DTSA.

7. O suporte ao roteamento se dá através dos *reports* do switch em relação a entidades conectadas e portas, definidos pelo protocolo de *bootstrapping* e sistema coprocessado[1]. Por exemplo, parâmetros como número de *Workspaces* ativos e lista de títulos de *Workspaces* ativos podem ser coletados para envio quando o protocolo de gerenciamento assim solicitar.

Já no caso da mobilidade *(MOBILITY)*, em se tratando mobilidade de *Workspace* um DTSA pode querer mover um *Workspace* de um switch a outro, por exemplo, para atender requisitos de QoS com base na largura de banda que está sendo possível alocar naquele momento ou por questões de disponibilidade e políticas de segurança. Isso é feito através do protocolo de configuração. Para o básico do suporte à mobilidade, o fluxo de operações é:

1. DTSA solicita remoção do *Workspace* ou extensão de *Workspace* a outra porta do Switch;

2. DTSA contacta outro *switch* e solicita criação do workspace com o mesmo título;

3. Para saber como alcançar cada entidade, o algoritmo de roteamento deve ser executado novamente (operação não dependente do *switch*).

Para mobilidade de entidade com transferência *stateless*:

1. Entidade e/ou DTSA optam pelo *dettach* da entidade daquele *Workspace* (não envolve *switch*)

2. DTSA solicita ao *switch* modificação de Workspace

3. Entidade e DTSA negociam novo *attach* (não envolve controle no *switch*)

4. DTSA solicita ao *switch* modificação de Workspace

[1] Ambas são pesquisas paralelas a esta que utilizarão dos mecanismos aqui prototipados para a coleta de dados de gerenciamento

4.2 Protocolo de Configuração do Switch ETArch

O *ETArch Switch Configuration Protocol* (ETSCP) é um protocolo desenvolvido para cuidar do processo de configuração de um Switch ETArch a partir de mensagens de controle enviadas pelo DTSA. Sua operação é baseada na troca de mensagem entre estas duas entidades e, sendo assim, ambas devem operar conforme regras do ETSCP. Vale ressaltar que este trabalho cuida da especificação do comportamento do Switch perante o recebimento de mensagens de configuração mas não dita ao DTSA regras que não sejam sobre o formato destas mensagens, pois o gerenciamento da rede e tomada de decisão no DTSA, conforme mencionado anteriormente, está em discussão e sendo modelado por uma pesquisa paralela a esta.

A especificação de um protocolo consiste de cinco partes distintas (HOLZMANN, 1991). Para ser completa, cada especificação deve incluir:

1. As suposições sobre o ambiente em que o protocolo é executado;
2. Os serviços a serem providos pelo protocolo;
3. O vocabulário de mensagens usado para implementar o protocolo;
4. O formato de cada mensagem no vocabulário; e
5. As regras de procedimento que garantem a consistência da troca de mensagens.

Assim, esta Seção traz diretivas sobre o ambiente em que o ETSCP se enquadra, apresenta o vocabulário e os serviços oferecidos, mostra o formato das mensagens e explica os serviços relacionando cada serviço e seu grupo de mensagens envolvidas.

4.2.1 Premissas sobre o ambiente

O ETSCP opera em cada *Switch* da rede como uma instância independente da outra. Os dois lados da comunicação são o DTSA e o Switch. Não há envolvimento de entidades do tipo aplicação nesta comunicação, posto que entidades não realizam operação de configuração no Switch, isso fica a cargo apenas do controlador DTSA.

É um protocolo para comunicação ponto a ponto. Se um DTSA quer realizar operações de configuração ao mesmo tempo em dois Switches diferentes, ele deve enviar mensagens independentemente umas das outras a cada Switch, diferenciando-as pelos parâmetros dos seus campos. As mensagens chegam a cada Switch na forma de quadros e são encaminhados à sua Unidade de Controle.

É um protocolo que opera diretamente na camada *Link* da ETArch, não tendo implicação direta nas comunicações abstratamente acima disso.

4.2.2 Serviços e Vocabulário

O ETSCP, até então, engloba um total de quatro serviços e oito mensagens em seu vocabulário. Naturalmente, as mensagens são trocadas para prover os serviços. Para que o leitor tenha uma visão geral do protocolo, estes dois elementos são introduzidos e relacionados pela Tabela 3, sendo retomados detalhadamente na Subseção 4.2.4. As mensagens obedecem à taxonomia *Request/Indication* e *Response/Confirmation*, i.e., uma *MSG Request* (MSG_REQ) tem o mesmo conteúdo que a *MSG Indication* (MSG_IND), diferenciando-se semanticamente no momento em que elas aparecem (IETF, 2008b).

Tabela 3: Serviços e Vocabulário do ETSCP

Nome	**Mensagens**	**Descrição**
Create Workspace (ADD WKP)	*CreateWkp Request (AddWkpREQ)* *CreateWkp Response (AddWkpRESP)*	Serviço confirmado. Utilizado para a criação de novos *Workspaces* de dados e controle em cada Switch.
Edit Workspace (EDIT WKP)	*EditWkp Request (EditWkpREQ)* *EditWkp Response (EditWkpRESP)*	Serviço confirmado. Utilizado para modificar parâmetros de um *Workspace*.
Remove Workspace (REM WKP)	*RemWkp Request (RemWkpREQ)* *RemWkp Response (RemWkpRESP)*	Serviço confirmado. Utilizado para solicitar a exclusão de determinado *Workspace* da tabela de *Workspaces*.
Change Frequency (EDIT FREQ)	*Change Freq Request (EditFreqREQ)* *Change Freq Response (EditFreqRESP)*	Serviço confirmado. Utilizado para modificar os valores referência dos divisores de frequência dos escalonadores do Switch.

A Seção 4.2.4 retoma esta discussão e detalha a execução de cada um dos serviços apresentados na Tabela 3, bem como a função das mensagem e seu devido uso. O Apêndice A mostra os dados de preenchimento dos campos para todas as mensagens apresentadas nesta seção. É importante ressaltar que o formato é de 64 bits, conforme adaptação para o sistema de demonstração desenvolvido.

4.2.3 Formato das Mensagens

O ETSCP é um protocolo de configuração cujas mensagens são quadros e não passam do nível do enlace. O formato do protocolo é detalhado pela Figura 14. É importante salientar que o formato do protocolo ETSCP tomou como base o comprimento do quadro Ethernet para que as placas de rede atuais possam também transmiti-lo sem maiores complicações. Por esta razão, a Figura 14 mostra uma comparação entre os padrões.

Também é importante ressaltar que embora o comprimento das sequências de bits correspondentes a endereço Ethernet e título ETArch sejam os mesmos, isso pode ser

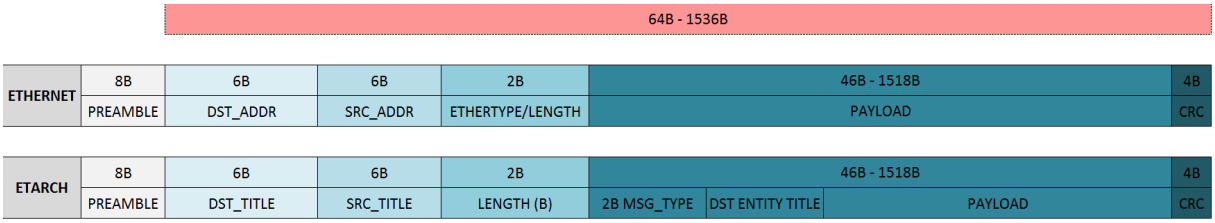


Figura 14: Formato da mensagem ETSCP

alterado quando houver interfaces de rede independentes do MAC Ethernet. Alterações no formato podem ser feitas conforme versões do protocolo, bastando, para isso, que o hardware também seja reconfigurado no Switch.

As mensagens ETSCP são essencialmente mensagens de controle. É, inclusive, por esta razão que no sistema de validação o cabeçalho e campo de dados possuem praticamente o mesmo comprimento.

O significado de cada campo pode ser conferido segundo a lista a seguir:

1. DEST_TITLE: Título do *Workspace* de destino.
2. SRC_TITLE: Título da entidade de origem.
3. LENGTH: Comprimento, em *Bytes*, do campo de dados.
4. MSG_TYPE: Indica qual o tipo de mensagem reconhecida pelo vocabulário que aquela primitiva está carregando.
5. DEST_ENTITY_TITLE: Título da entidade de destino dentro de um *Workspace*.
6. PAYLOAD: Campo de dados para parâmetros das mensagens.
7. CRC: Soma de verificação para detecção de erro;

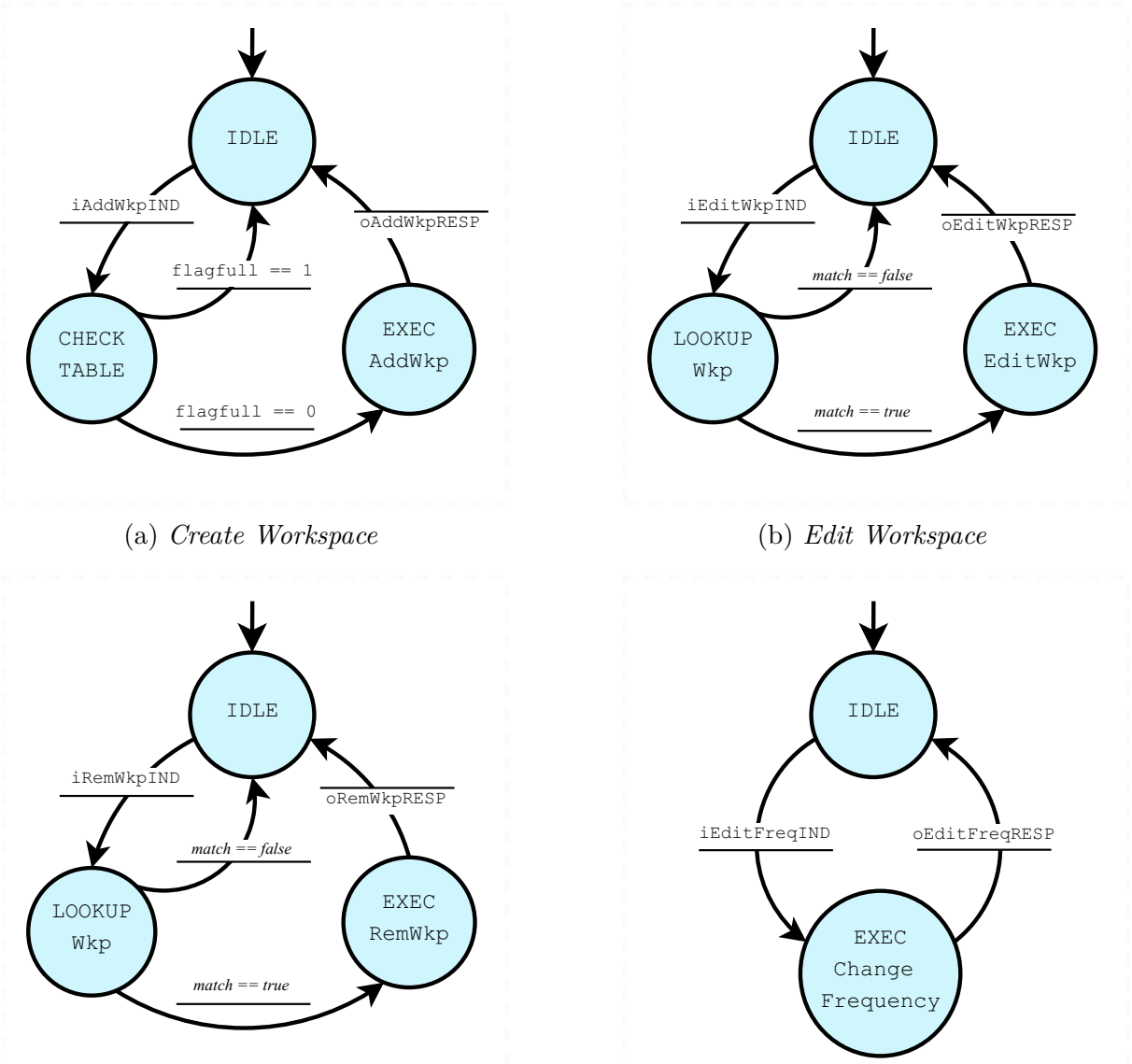
Há títulos que são chamados *well known titles* e se referem a *Workspaces* genéricos utilizados para controle e *bootstrapping* da rede.

4.2.4 Regras de Procedimentos

Cabe às regras de procedimentos explicar a dinâmica de troca de mensagens e o comportamento das máquinas de estados finitos (FSM) (HOPCROFT; MOTWANI; ULLMAN, 2000) do ETSCP durante a execução dos procedimentos dos serviços. Cada serviço apresentado na Tabela 3 é aqui representado como um autômato que especifica de forma não ambígua o comportamento da instância do ETSCP durante sua execução no Switch. Salienta-se aqui, mais uma vez, que as máquinas de estado dizem respeito às instâncias sendo executadas pelo Switch. O comportamento do DTSA perante o recebimento de

mensagem e as condições que o fazem gerar mensagem de configuração são discussões que, por ora, não estão no escopo deste trabalho.

São utilizados diagramas de estados para descrever os serviços. Nestes diagramas, cada transição é representada pela combinação de entrada e saída, na forma $\frac{entrada}{saida}$. A Figura 15 traz as FSMs referentes a cada serviço.



(a) *Create Workspace* (b) *Edit Workspace*

(c) *Remove Workspace* (d) *Change Frequency*

Figura 15: Máquinas de estados do Switch para cada serviço

As máquinas de estados são simples. Quando se deseja adicionar um *Workspace*, o DTSA deve enviar uma solicitação *AddWkpREQ* que será recebida na forma *AddWkpIND*, conforme Figura 15(a). Então, o Switch verifica se há espaço na tabela. Se a tabela estiver cheia, volta ao estado de *Idle*, senão, executa a inclusão no *Workspace* na tabela, envia confirmação *AddWkpRESP* ao DTSA e volta ao seu estado inicial.

Para a modificação de um *Workspace* já existente (Figura 15(b)), ao receber uma mensagem *iEditWkpIND*, o Switch verifica se aquela é uma entrada válida na tabela de *Workspaces*, caso não, volta ao estado inicial, caso sim, prossegue com a substituição da entrada, envia confirmação e retorna ao seu estado inicial. Se for para remover uma entrada da tabela, conforme Figura 15(c), o processo é praticamente o mesmo que quando de uma edição, com diferença apenas na implementação, pois haverá a exclusão da entrada na tabela e não sua modificação.

Por fim, ao receber uma solicitação de alteração nos parâmetros dos divisores de frequência dos escalonadores, conforme Figura 15(d), o Switch apenas executa ação, envia confirmação ao DTSA e retorna ao seu estado inicial.

Capítulo 5

Arquitetura e Organização do Switch ETArch

Tratar da arquitetura de um sistema computacional significa lidar com os parâmetros visíveis ao nível de abstração imediatamente acima do hardware e da distribuição dos componentes no nível lógico digital. A arquitetura trata de parâmetros visíveis ao programador, i.e., o conjunto de instruções, os registradores visíveis ao sistema operacional, o tamanho de memória, entre outros. A distribuição espacial e interconexão dos componentes digitais se referem aos aspectos organizacionais ou de implementação daquele sistema, conforme Patterson e Hennessy (2013). Este capítulo trata tanto dos aspectos arquiteturais quanto organizacionais do nível lógico digital do *Switch ETArch*.

Os *switches* comerciais atualmente obedecem ao padrão Ethernet. Os *switches* OpenFlow são, na maioria dos casos, também baseados em *switches* Eternet. Algo que todos eles têm em comum é o fato de que seu papel é encaminhar unidades de informação das portas de entrada para as portas de saída. O *Switch* ETArch também faz exatamente isso, mas seu mecanismo de encaminhamento introduz a ideia de um escalonamento parametrizável dos quadros porta-a-porta.

Observe a Figura 16. Ela nos dá uma visão geral do que se propõe para o *Switch*. Como se pode observar, o projeto é constituído por uma Unidade de Controle, Unidades de Memória e Unidades Operacionais (*Datapaths*) - quatro neste caso. A unidade de controle é responsável por controlar a execução das instruções e tomar as decisões relativas às solicitações do DTSA, que chegam através das mensagens de controle. As unidades de memória cuidam de manter uma tabela de *Workspaces* e um banco de registradores. Estes registradores, que têm dados com valor semântico de variáveis globais, são lidos tanto pela unidade de controle quanto pelas unidades operacionais. Eles mantém dados bem conhecidos como os endereços dos *Workspaces* de *bootstrapping* e controle, bem como seus respectivos *Bitmap*, *PortBitmap*, título do *Switch* e título do DTSA. A *WKP Table* mantém a tabela de *Workspaces* com seus Bitmaps. Já os *Datapaths* são os que tratam dos quadros que chegam pelas interfaces de transmissão.

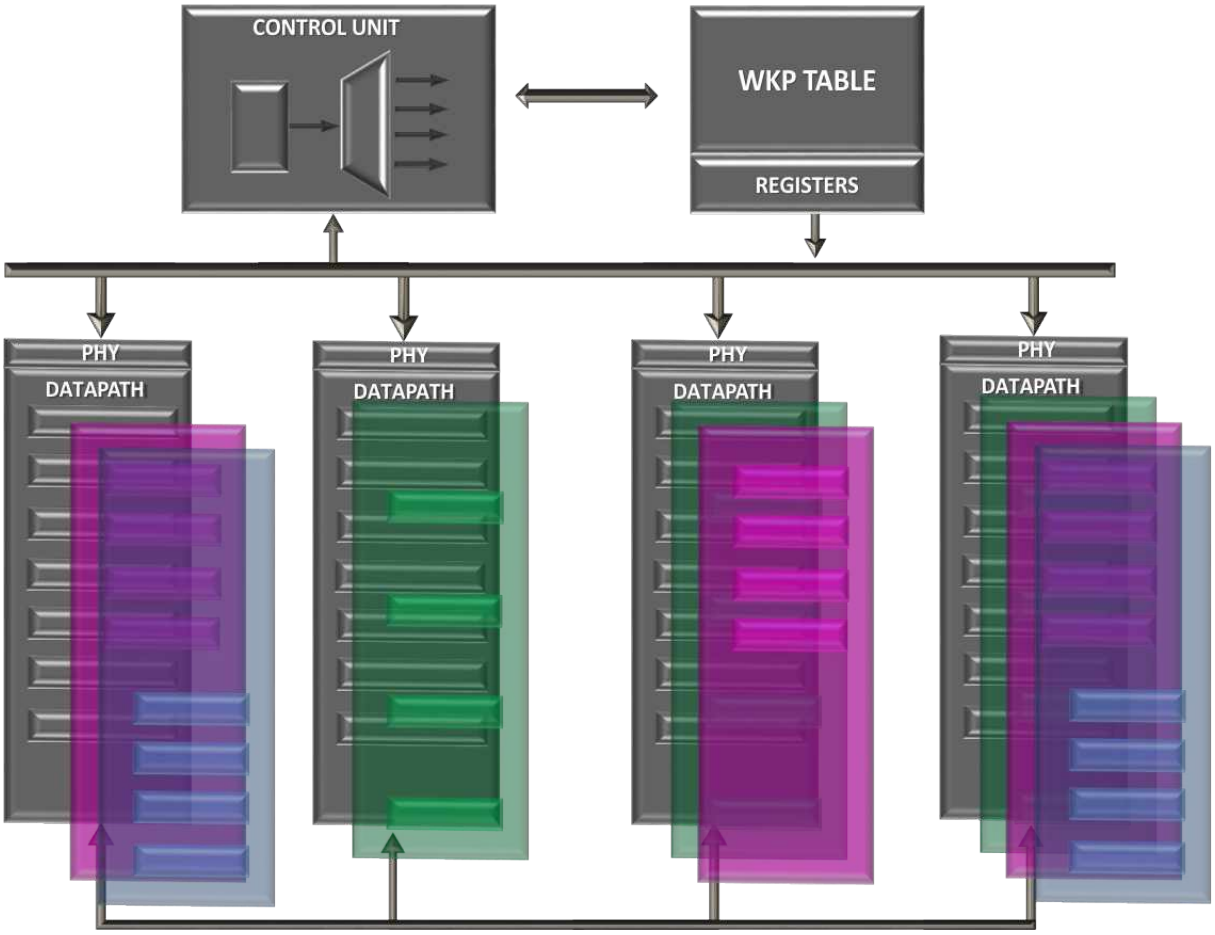


Figura 16: Os planos virtuais do Switch ETArch

A ideia principal é ter um *Switch* que possibilita diferentes métodos de acesso ao meio de acordo com as diferentes necessidades de comunicação e que permita a parametrização de cada *Workspace* sendo criado. A noção de separação de planos de dados e controle permanece, mas a programabilidade é também adicionada ao plano de encaminhamento com a possibilidade de mudar as configurações do hardware em tempo de execução.

A Figura 16 mostra como os *Workspaces* são implementados. Dentro do *Switch*, há os quatro *Datapaths* e seus respectivos PHYs. Cada *Datapath* físico pode ser usado de acordo com a necessidade de cada *Workspace* de forma que a Unidade de Controle configura um subconjunto das funções/núcleos implementados de cada *Datapath*, como representado pelos planos coloridos. Cada uma dessas funções/núcleos é detalhada nas Seções a seguir.

A chave para que um quadro possa passar por determinado *Datapath* com uma ou outra configuração está na inserção de um *Bitmap* de 16 bits que indicará quais os blocos implementados devem realizar alguma operação naquele quadro. Este *Bitmap*, que é um vetor de bits utilizado para seleção dos componentes, está detalhado na Tabela 4. Ao entrar pelo *Datapath* o quadro tem este *Bitmap* anexado a ele e o carrega até o último bloco antes de ir para a saída do *Switch*. Cada *Workspace* opera sob um *Bitmap* diferente.

Ao descer sutilmente o nível de abstração, deparamo-nos com as especificações or-

Tabela 4: *Bitmap* de definição de planos virtuais

BITMAP (15 *downto* 0)	
Bit	**Requeriment**
b_{15}	*Priority 0*
b_{14}	*Priority 1*
b_{13}	*Priority 2*
b_{12}	*Frequency 1*
b_{11}	*Frequency 2*
b_{10}	*Frequency 3*
b_9	*Hardware Cryptography*
b_8	*CRC*
b_7	*To be used*
b_6	*To be used*
b_5	*To be used*
b_4	*To be used*
b_3	*To be used*
b_2	*To be used*
b_1	*To be used*
b_0	*To be used*

ganizacionais que implementam a arquitetura do *switch*. Para termos uma ideia ainda mais detalhada desta implementação, a Figura 17 mostra uma visão completa de como as unidades funcionais estão implementadas para prover os subconjuntos que formam os planos virtuais, i.e., as diferentes combinações que dão suporte a coexistências de diferentes *Workspaces* em uma mesma porta. Esta Figura apresenta o caminho de dados e de controle do processamento em nível de transferência de registrador (*register transfer level* (RTL)).

É importante salientar, neste momento, que os *Datapaths* possuem uma estrutura de interconexão altamente paralela na saída dos quadros, o que é detalhado na Seção 5.5, diferenciando-se do mecanismos padrão.

Um quadro, ao ser recebido pela unidade física de conexão com a rede, é entregue ao *PHY Abstraction Layer* através do controlador de recebimento de quadros (*PHY Rx Controller*). Então, este quadro é direcionado ao Árbitro de Entrada de dados (*Input Arbiter*) que, por sua vez, deve orquestrar o recebimento de quadros vindos da interface de rede e da Unidade de Controle. Logo, o quadro segue para o *Tagger*, onde é inserido um *Bitmap* de 16 bits que indica em qual deve ser o processamento realizado bloco a bloco. De lá o quadro segue para o complexo de escalonamento que deverá selecionar a frequência pela qual o quadro alcançará o próximo árbitro com relação à priorização indicada pelo *Bitmap*. Logo, o *Queues Arbiter*, árbitro na saída do escalonador, recebe o frame e o encaminha ao módulo responsável por realizar processamento adicional, por exemplo, criptografando o seu conteúdo diretamente no hardware. Já na saída do *Customization*,

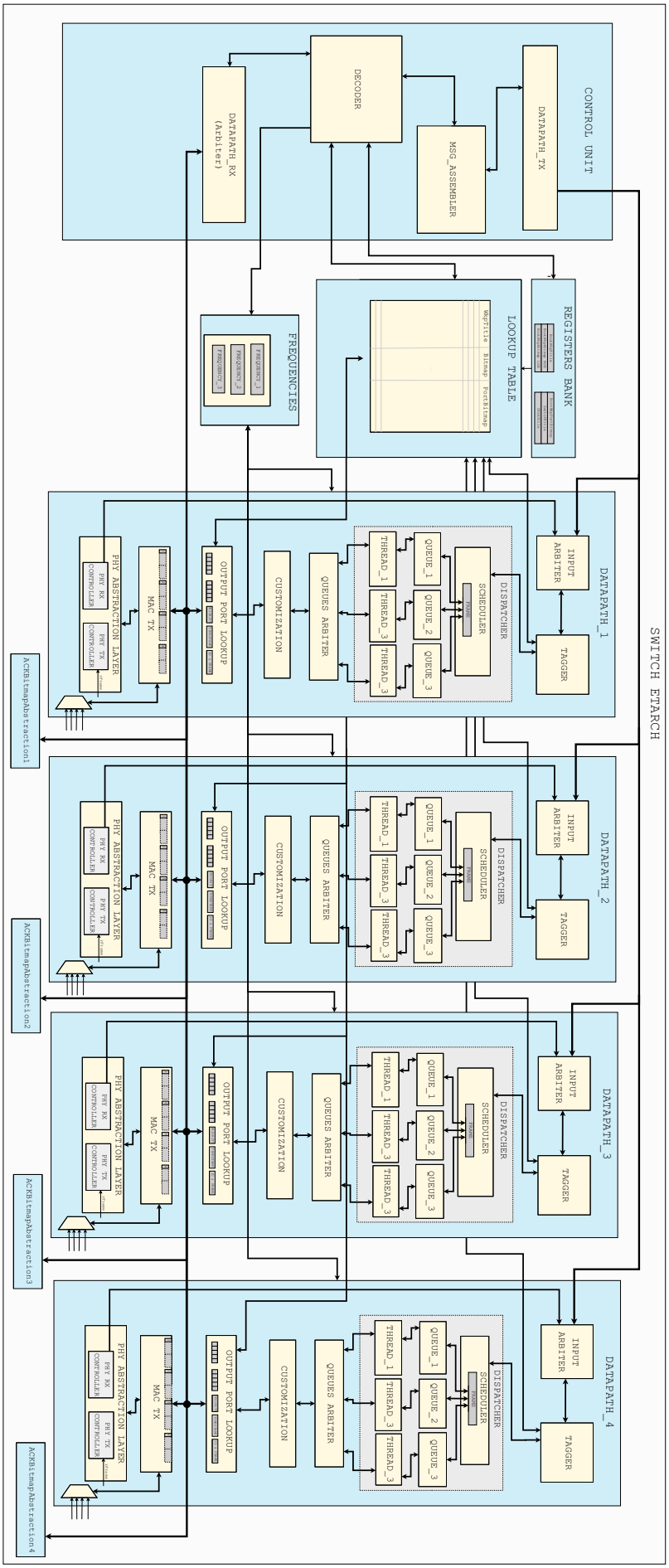


Figura 17: Arquitetura do Switch ETArch

o quadro encontra o *Output Port Lookup* que vai agora buscar na memória *Lookup Table* a informação sobre quais as portas pertecem ao *Workspace* daquele frame e dispara as solicitações de encaminhamento para os *Mac Tx* de todos os *Datapaths*. Cada *Mac Tx* que deve transmitir o quadro o encaminha a seu respectivo PHY e confirma esta transmissão ao *Output Port Lookup* de origem. Quando todas as confirmações forem recebidas, inclusive da Unidade de Controle, aquele *Datapath* estará liberado para dar vazão ao próximo quadro. As Seções a seguir detalham cada um destes módulos e transferências.

Um quadro ao entrar no pipeline de dados do *Datapath* é transmitido por vias de 64 bits de largura. Cabe salientar que este valor é parametrizável, i.e., para prototipagens com maiores capacidades em números de elementos lógicos, este valor pode ser em potências binárias, por exemplo, 128 ou 512 bits. No projeto inicial, optou-se por vias paralelas de 512 bits, já que esta largura proporciona espaço suficiente para todas as mensagens de controle dos planos de controle e gestão da ETArch bem como facilita a fragmentação de quadros Ethernet, já que este ainda é o comprimento padrão de transmissão pelas interfaces de rede mais capilares atualmente.

Quatro *Datapaths* foram adotados para o projeto por ser uma quantidade possível de demonstrar as *NxN* interconexões entre os módulos *Output Pot Lookup* e *Mac TX* - que cuidam do encaminhamento dos quadros para diferentes *Workspaces* - mantendo a complexidade de espaço possível de ser implementada em um FPGA de baixo custo. O projeto é expansível neste quesito, mas é importante lembrar que a quantidade de interconexões e recursos de hardware consumidos também aumenta.

Cada bloco visto na Figura 17 é implementado através de lógica sequencial tendo uma máquina de estados finitos como estrutura de dados principal. Para a implementação do sistema de validação, eles são ligados a um clock de 50 MHz. O módulo *PHY Abstraction Layer* também recebe um clock de 5 MHz, dividido no *Datapath* por um divisor de frequencia em virtude da velocidade de operação do *tranceiver* da interface de rede. Todo o circuito é implementado por lógica padrão, com exceção dos sinais de *reset* que, por uma questão de implementação, está ligado a botão de lógica complementar, sendo, então, ativado em nível lógico baixo.

Cada um dos blocos foi simulado e depois sintetizado. Todos eles foram integrados em um único projeto de hardware (ilustrado pela Figura 17) e prototipados no FPGA. A descrição destes módulos é dada nas subseções a seguir, dados de simulação também são apresentados no intuito de ilustrar a operação de cada um deles.

Uma vez que os módulos estão combinados organizacionalmente como na Figura 17, esta figura é repetida em cada subseção para apresentar os componentes e suas interfaces, bem como as estruturas de interconexão entre eles, com destaque para o módulo em questão.

A FSM de cada componente é explicada em sua respectiva subseção enquanto os diagramas de estados referentes às FSMs são apresentados no Apêndice B.

5.1 *Control Unit*

Para gerenciar o fluxo interno de dados e o instante preciso em que ocorrem as transferências entre uma unidade e outra são necessários sinais de controle. Esses sinais são fornecidos pela unidade de controle e atingem tanto memória (*Lookup Table*, quanto os registros *Frequencies*) e unidades operacionais *Datapaths*. Estes sinais são detalhados na Figura 18.

Cada sinal de controle comanda uma micro-operação que pode ser responsável pela realização de uma carga em um registrador (como os registradores de frequência), uma seleção de um dado para entrada em um determinado componente, uma edição de conteúdo da *Lookup Table* ou a habilitação de um dos circuitos lógicos dos *Datapaths*.

A unidade de controle do Switch ETArch é um conjunto de máquinas de estado finitas realizadas por lógica sequencial. A organização da unidade de controle é implementada por lógica convencional e não por lógica microprogramada, o que significa, de acordo com Weber (2009), que ela é composta por componentes digitais como flip-flops, contadores e decodificadores, que geram, sequencialmente e nos instantes de tempo adequados, todos os sinais de controle necessários à ativação da unidade operacional, do sistema de entrada e saída e da memória.

5.1.1 O Conjunto de Instruções

O conjunto de instruções atende às demandas de configuração do DTSA. Do ponto de vista do hardware, uma instrução é um conjunto de bits devidamente codificados que indica ao sistema que sequência de micro-operações ele deve realizar. As instruções do Switch ETArch não são buscadas numa memória interna. Como exposto anteriormente, elas são implementadas por lógica convencional e ativadas por mensagens de controle vindas do DTSA, recebidas pelos *Datapaths* e repassadas à decodificação pela Unidade de Controle.

O conjunto de instruções do compreende quatro instruções codificadas através de quatro bits. Nas mensagens de controle vindas do DTSA são os quatro bits do campo *MsgType* que contém o código da instrução (Figura 14). A Tabela 5 detalha tais instruções.

Na Tabela 5, as três primeiras instruções são relativas à modificações na tabela de *Workspaces*. Isso significa que podem ser realizadas as ações de incluir uma entrada que contém o título do *Workspace*, o *Bitmap* relativo ao seu processamento no Switch e o *Port Bitmap* referente às portas pertencentes àquele *Workspace* para encaminhamento do quadro. Estes parâmetros podem também ser editados para modificação e expansão de *Workspaces* bem como podem ser excluídos da tabela. Já a última instrução, de *Opcode* 0x07, realiza mudanças nos registradores que armazenam os valores de referências para as frequências dos escalonadores.

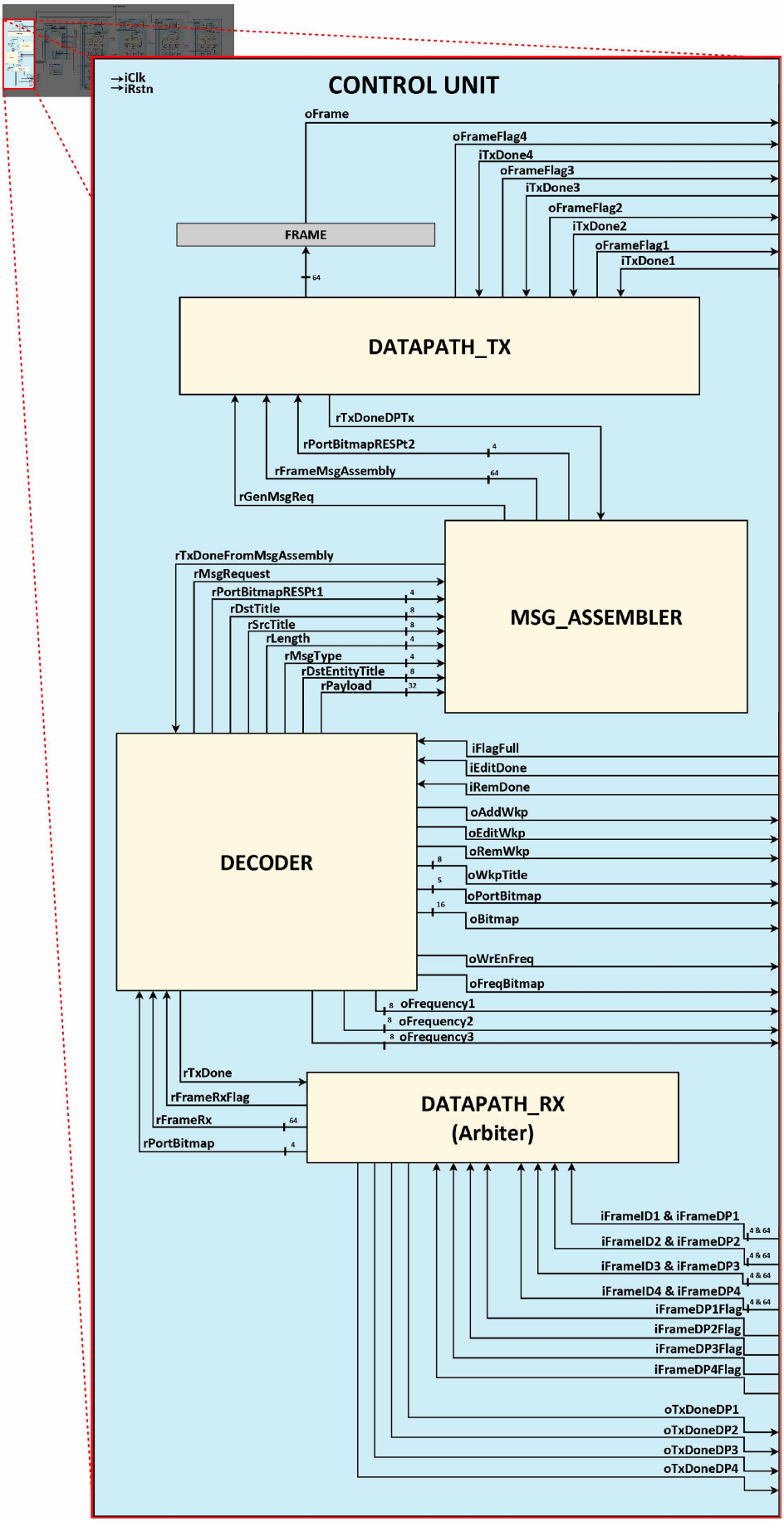


Figura 18: Unidade de Controle

Tabela 5: Conjunto de Instruções

Código	**Instrução**	**Comentário**
0001	CREATE WKP	Adiciona nova entrada na tabela de *Workspaces*
0011	EDIT WKP	Edita entrada na tabela de *Workspaces*
0101	REMOVE WKP	Remove entrada da tabela de *Workspaces*
0111	CHANGE FREQ	Altera o valor da frequência de varredura das *Threads* dos escalonadores

5.1.2 *Datapath Rx*

Este é o orquestrador da interface entre e Unidade de Controle e os *Datapaths*. O *Datapath Rx* possui, de acordo com a Figura 18, quatro canais de 64 bits para receber os quadros vindos de cada *Datapath*. Para cada quadro na entrada, há também um barramento de 4 bits reservado à identificação deste quadro, que é o *FrameID*. O *FrameID* força que cada quadro seja processado apenas uma vez, ainda que por alguma razão ele permaneça nas entradas do *Datapath Rx*. Para cada quadro, há também seu respectivo indicador *iFrameFlag*, que indica que novo quadro está disponível para ser processado. Além disso, os sinais *oTxDone* confirmam aos *Datapaths* que a transmissão do quadro já foi realizada.

Ainda, e agora com interface ao interior da Unidade de Controle, há o *rFrameRx*, um barramento de 64 bits para o envio do quadro à unidade de decodificação, o *rPortBitmap*, um barramento de 4 bits para indicar por qual porta a resposta a este quadro (quando se aplica) deve voltar, o *rFrameRxFlag*, um sinal que indica a ocorrência de novo quadro a ser decodificado e o *rTxDone*, o sinal de entrada que confirma que a transferência ao *Decoder* já foi realizada. Os sinais de confirmação permanecem em nível lógico por apenas um ciclo do clock.

A máquina de estados do *Datapath Rx* implementa uma estrutura de fila circular que alterna entre os *Datapaths* de *DP1* a *DP4*. Ocorre uma alternância entre os estados da FSM de forma a identificar, sequencialmente, se há quadro nas entradas. Caso haja, a FSM é redirecionada a um dos estados que realiza a transmissão ao *Decoder* e espera confirmação de recebimento do quadro. Caso a confirmação não chegue, um *timeout* é obedecido para que as transferências não fiquem todas pendentes por este erro. Neste ponto, uma sinalização será emitida, embora não esteja implementada nesta versão.

5.1.3 *Decoder*

Ainda observando a Figura 18, nota-se que o *Decoder* vem logo em seguida na Unidade de Controle a partir do *Datapath Rx*. O *Decoder* é o componente que de fato decodifica a

mensagem de controle recebida do DTSA ou outro Switch e dispara as ações relacionadas à execução e envio de mensagem de resposta.

De acordo com a Figura 18, a interface inferior do *Decoder* se liga ao *Datapath Rx* para receber a mensagem a ser decodificada, conforme explicado na Subseção anterior. Já os sinais do lado direito são os que se ligam aos componentes fora da Unidade de Controle.

Os sinais e barramentos *iFlagFull*, *iEditDone*, *iRemDone*, *oAddWkp*, *oEditWkp* e *oRemWkp* são os que se comunicam com a *Lookup Table* para execução das instruções de inclusão, edição e remoção de *Workspaces* da tabela de *Workspaces*, conforme Tabela 5. Já os barramentos *oWkpTitle*, *oBitmap* e *oPortBitmap* são os parâmetros que são transferidos à tabela quando da execução destas instruções, com 8, 16 e 5 bits respectivamente.

No canto inferior direito do *Decoder* observam-se as conexões com o componente *Frequencies* para a configuração dos valores de frequência referências dos escalonadores. A execução da instrução *CHANGE FREQ* conta com o sinal *oWrEnFreq* para sinalizar a intenção de realizar a operação, com o barramento *oFreqBitmap* para indicar qual dos valores de frequência será alterado e com os barramentos *oFrequency1*, *oFrequency2* e *oFrequency3* para indicar quais são os novos valores a serem armazenados. Os barramentos são parametrizáveis e, neste caso obedecendo às restrições do FPGA utilizado no sistema de validação, são de 8 bits para os valores de frequência.

A FSM começa em um estado de *Fetch* e nele permanece até que haja mensagem a ser decodificada. Quando isso ocorre, ela vai ao estado *Decoding* para selecionar qual a operação a ser realizada. Em seguida, vai a um dos estados de execução das operações e deles retorna ao *Fetch* ou por confirmação do serviço ou por *timeout* em sua execução. Cabe um atenção especial à execução da instrução *CREATE WKP* que, implementada durante o estado *Exec_AddWkp* já retorna ao *Fetch* caso a memória esteja cheia.

Já em sua interface superior, o *Decoder* se comunica com *MSG Assembler* para solicitar mensagens de respostas à execução dos serviços ou responder mensagens de Switches vizinhos que se comunicam através do protocolo de *boostrapping*. O *Decoder* utiliza o sinal *rMSsgRequest* para solicitar a transmissão de nova mensagem e o *rTxDoneFromMsgAssemby* para verificar se a transmissão já ocorreu. Os outros barramentos conectados ao *MSG Assembler* são os parâmetros da mensagem de resposta que deve ser enviada de volta.

5.1.4 *MSG Assembler*

Este componente tem a função bem específica de montar mensagens para serem enviadas aos *Datapaths*, passando pelo *Datapath Tx*. Em sua interface com o *Decoder*, detalhada do seu lado esquerdo na Figura 18, ele recebe os parâmetros da mensagem e a solicitação de envio. Enquanto isso, sua interface superior entrega o novo frame ao *Datapath Rx* junto com o *rPortBitmapRESPt2* que indica para qual dos *Datapaths* aquele quadro deve ser entregue.

A máquina de estados é bem simples, ao receber solicitação via *rMsgRequest* a FSM vai ao estado *Build*, ativa o *rGenMsqReq* e aguarda pela confirmação de transmissão num estado de espera até que chegue sinalização via *rTxDoneDPTx*.

5.1.5 *Datapath Tx*

Por último, na Unidade de Controle, o componente *Datapath Tx* entrega a mensagem de volta às unidades operacionais já na forma de ser transmitida. Ainda observando a Figura 18, observa-se que a interface inferior do *Datapath Tx* é a conexão com o *MSG Assembler* enquanto a superior se conecta ao árbitro de cada *Datapath*.

Na interface com os *Datapaths*, há um canal *oFrame* de saída que leva o conteúdo do frame, enquanto os sinais *iTxDoneN* e *oFrameFlagN* sinalizam aos *Datapaths* correspondentes por onde o quadro deve sair e deles recebem as confirmações de que o quadro entrou no pipeline de dados.

A máquina de estados entra em um estado de *Demuxing* ao receber solicitação do *MSG Assembler* e de lá vai a um dos quatro estados de transmissão. Quando a confirmação é recebida, ela volta ao estado de *Idle* esperando nova solicitação. É, a grosso modo, uma estrutura de demultiplexação controlada por lógica sequencial.

5.2 *Lookup Table*

A *Lookup Table* é o componente que se pode comparar à tabela de encaminhamento dos switches Ethernet ou tabela de fluxos em switches OpenFlow. É nela que se encontra a tabela de *Workspaces*. Cada entrada da tabela é uma tupla de {*Workspace Title* | *Bitmap* | *PortBitmap*}. Em outras palavras, cada entrada da tabela tem o título de um *Workspace* salvo aqui em 8 bits, o respectivo *Bitmap* em 16 bits para indicar qual será a configuração do hardware daquele *Workspace* e o *PortBitmap* em 5 bits indicando por qual porta os quadros daquele *Workspace* devem ser enviados.

O formato do *PortBitmap* é tal que que cada um dos 5 bits representa uma porta, na forma $b_0b_1b_2b_3b_4$. O bit b_0 indica a situação da porta local, i.e., a unidade de controle, enquanto os bits de b_1 a b_4 indicam os *PHYs* de 1 a 4, respectivamente. Por exemplo, um byte recebido no *Decoder* com a sequência $0x18$ tem a sequência binária 00011000. Pelo formato do protocolo detalhado pela Figura 43, os três bits MSBs assumirão valor lógico *don't care*, o que torna a sequência $xxx11000$ e então o quadro deve ir à Unidade de Controle ($L0$) e ser transmitido pela porta do *Datapath1*.

Pela Figura 19, a interface esquerda do *Lookup Table* se conecta ao *Decoder* para execução do conjunto de instruções. São estes sinais que habilitam as transições de estado da FMS principal da *Lookup Table*, que tem um estado para cada execução de instrução da Tabela 5 que trata de *Workspaces* (*ADD*, *EDIT* e *REMOVE*).

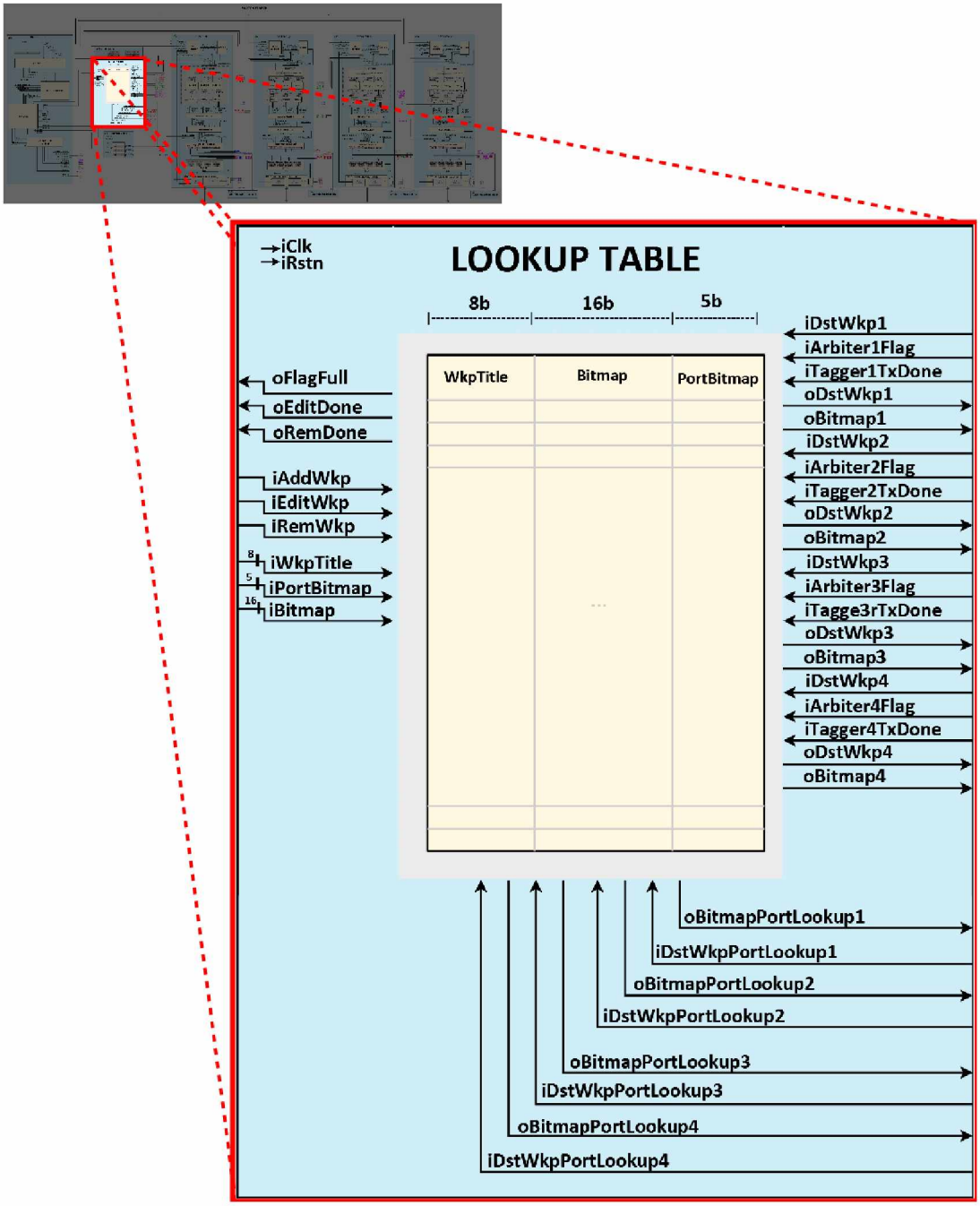


Figura 19: *Lookup Table* com tabela de *Workspaces*

Além disso, na interface direita da *Lookup Table*, observam-se quatro conjuntos de sinais que se comunicam com o *Input Arbiter* e o *Tagger* dos *Datapaths*. Isso se deve ao fato de que quando um quadro chega no *Datapath* vindo do PHY, ele deve receber um *Bitmap* de acordo com o título do *Workspace* de destino que ele carrega. Então, os primeiros componentes de cada *Datapath* tratam de recuperar esta informação da *Lookup Table*.

Em cada conjunto destes sinais, o *iArbiterFlag* sinaliza que um novo quadro chegou da interface de rede e o *iDstWkp* indica, em 8 bits, qual é o título a ser encontrado. Quando percorrida a tabela, as respostas são colocadas em *oDstWkp* e *oBitmap*, com 8 e 16 bits, respectivamente. Quando o *iTaggerTxDone* sinaliza que a transmissão foi feita, os dados são retirados da saída.

Ainda observando a interface do *Lookup Table*, a parte inferior trata das comunicações com os *Output Port Lookups* dos *Datapaths* mas desta vez para buscarem o *PortBitmap*, que indica portas de saída do Switch. Também são quatro pares de barramentos. O *iDstWkpPortLookup* indica com seus 8 bits o título que precisa ser pesquisado e o *oBitmapPortLookup* indica com seus 5 bits o *Port Bitmap* respectivo - ou zero caso não seja encontrado.

No protótipo desenvolvido para este trabalho, o tamanho da tabela é de 8x29 bits (8 entradas de 29 bits cada). Cabe lembrar que os estes oito mecanismos de busca que se comunicam com os *Datapaths* são independentes entre si e implementados por FSMs simples com apenas 3 estados: *Idle, Fetch, Done*.

5.3 *Frequencies*

Uma das instruções implementadas pelo Switch é alteração da frequência do escalonador nos pipelines de dados. Isso reflete diretamente a capacidade de *Workspaces* terem diferentes taxas de transferência de acordo com a capacidade da máquina. O componente *Frequencies* é que fornece aos *Datapaths* qual o valor da frequência que cada escalonador deve operar ou, ainda em mais detalhes, qual frequência cada *Thread* do escalonador deve operar. A Figura 20 deixa isso bastante detalhado.

Pode-se observar que a interface esquerda os pinos e barramentos são de entrada e vêm da Unidade de Controle. O sinal *iWrEnFreq* avisa, em um ciclo de clock, que a operação de escrita deve ser realizada enquanto o *iFreqBitmap* indica em qual dos registradores a operação deve ocorrer. Os barramentos *iFrequency* trazem o novo valor referência a ser armazemnado nos registradores. Então, os barramentos *oFrequency*, do lado direito da interface, deixam estes parâmetros disponíveis para leitura pelos *Datpaths*.

Uma FSM simples é implementada para este controle e transita entre os estados: *Idle, CheckInput, SaveFrequency, Done*. Quando há operação de escrita a ser realizada, os valores são salvos e a FSM retorna a seu estado inicial.

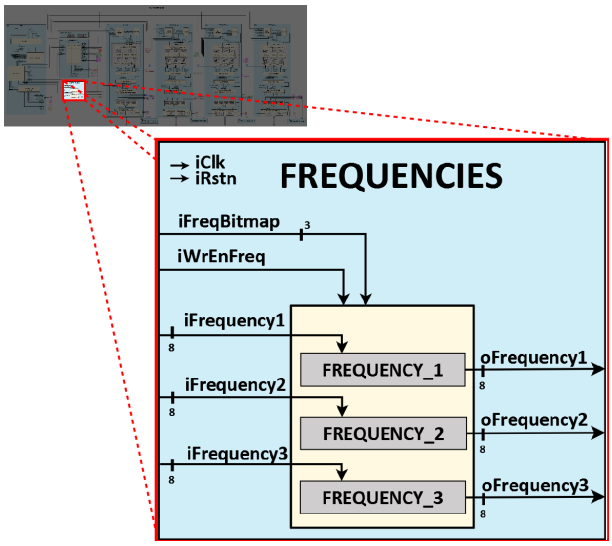


Figura 20: *Frequencies*

5.4 *Datapath*

Os *Datapaths* são os conjuntos de Unidades Operacionais por onde cada quadro deve passar durante seu processamento desde que chegou pela PHY até as portas de saída. Neste projeto há quatro *Datapaths* e este número pode ser tanto maior quanto for a capacidade do hardware que o suporta. A Figura 21 detalha as entidades e as suas interconexões.

Neste ponto, é interessante observar que esta estrutura da Figura 21 se repete em cada *Datapath*, mas o três últimos blocos, que tratam de entregar o quadro aos outros *Datapaths*, são alterados conforme aumenta o número de *Datapaths* no Switch.

Um quadro que chega e é bufferizado pelo *PHY RX Controller*, na parte inferior da Figura 21, é entregue ao *Input Arbiter* e de lá deve seguir até atingir o *MAC Tx*, de onde será direcionado aos PHYs das portas respectivas, conformes *PortBitmaps* dos *Workspaces*.

Este é o momento de observar a possível segregação de tráfego do ponto de vista do hardware. Não se pode dizer qual será o caminho exato do quadro que entra neste pipeline, em termos de circuitos, até que o *Bitmap* seja acoplado a ele. Por exemplo, cada *Datapath* possui uma unidade de escalonamento com diferentes frequências de varredura para enviar os quadros a frente e a utilização de cada uma delas varia de acordo com a configuração de cada *Workspace*. Isso se também é percebido com o componente *Customization*, que realiza processamento adicional. E é este mecanismo que permite a noção de planos virtuais implementando *Workspaces* independentes que utilizam o mesmo pipeline de dados no hardware, conforme Figura 16.

As Subseções a seguir explicam detalhadamente cada componente dos *Datapaths*, bem como as estruturas de interconexão que tornam o Switch capaz de ter diferentes *Workspaces* fluindo por uma mesma porta e diferentes portas pertencendo a um mesmo *Workspace*.

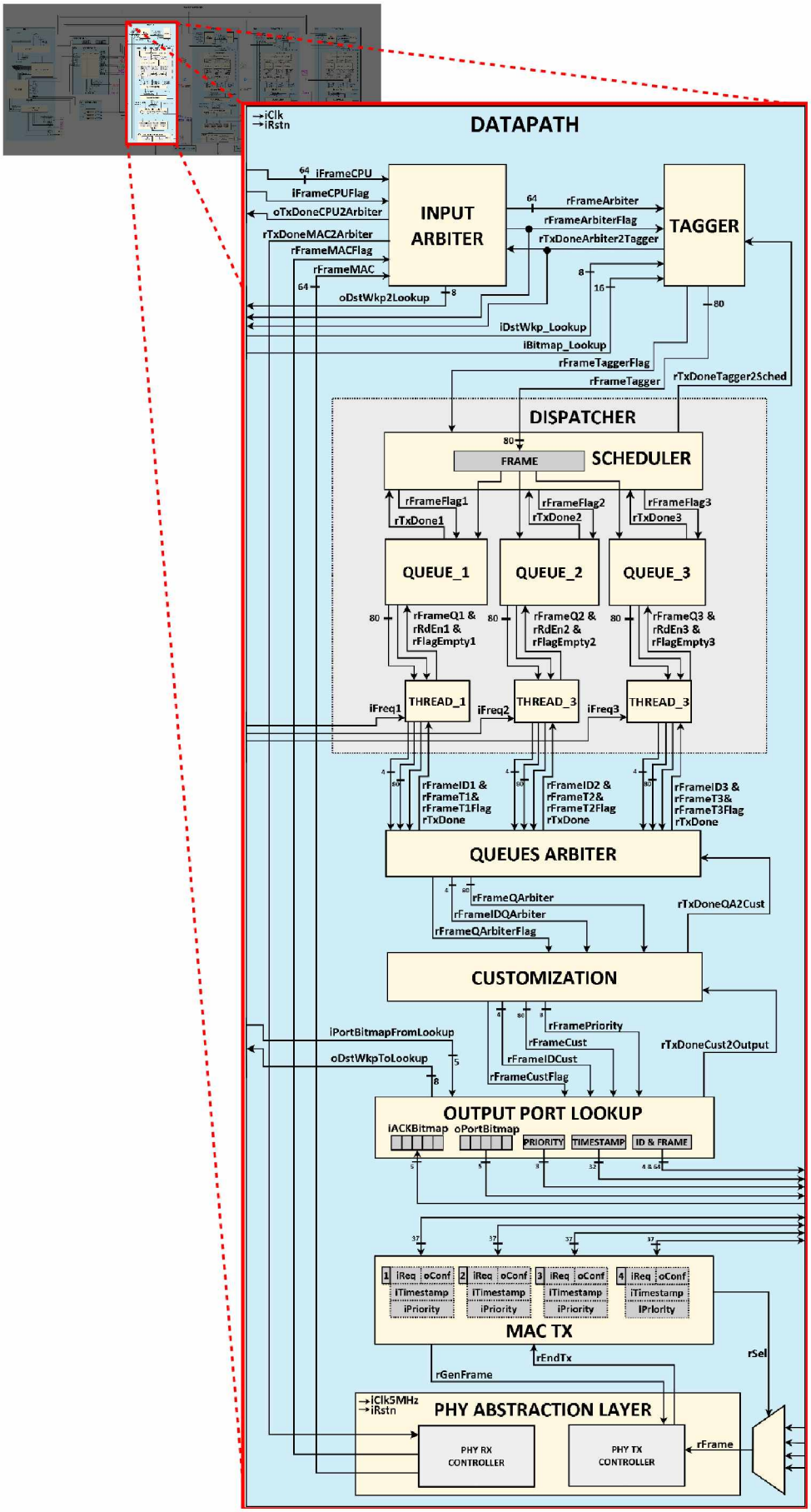


Figura 21: *Datapath*

5.4.1 *Input Arbiter*

O *Input Arbiter* é o árbitro que orquestra o recebimento dos quadros da Unidade de Controle e da PHY para aquele *Datapath*. Sua estrutura de interconexão pode ser observada com detalhes na Figura 22.

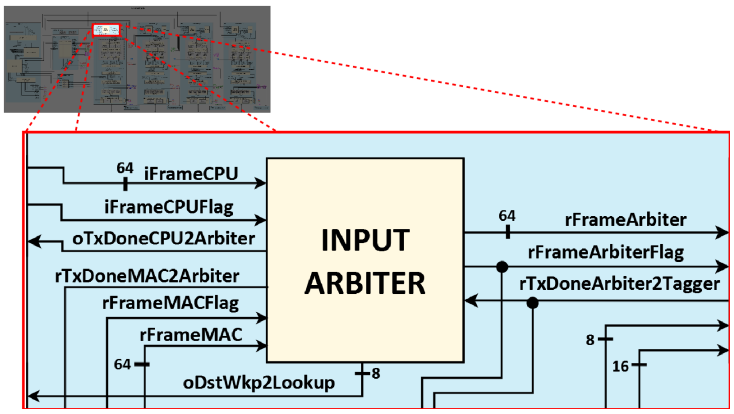


Figura 22: *Input Arbiter*

Ele recebe o *iFrameCPU* por um canal de 64 bits vindo do *Datapath TX* da Unidade de Controle acompanhado do *iFrameCPUFlag*, que indica que o novo quadro está disponível. O mesmo ocorre com a interface de rede, o quadro chega pelo barramento *rFrameMAC* com indicação pelo sinal *rFrameMACFlag*. As confirmações de recebimento são enviadas pelos sinais *oTxDoneCPU2Arbiter* e *rTxDoneMAC2Arbiter* e ocorrem quando o quadro foi copiado.

Há também o canal *oDstWkp2Lookup* que entrega à *Lookup Table* o título do *Workspace* cujo *Bitmap* deve ser devolvido ao bloco *Tagger*. Isso é feito para adiantar a busca e aproveitar esta vantagem oferecida pelo processamento paralelo do hardware.

Na interface do lado direito, há os sinais de comunicação com o *Tagger*: um sinalizador de novo quadro, o canal do quadro e a confirmação de transferência.

A implementação da FSM também segue a estrutura de uma fila circular que começa sempre pela CPU, já que as mensagens de controle não têm a mesma taxa de ocorrência das mensagens de dados e devem ter priorização em seu atendimento.

5.4.2 *Tagger*

Com o *Tagger* é que o quadro recebe o *Bitmap* que ditará por quais componentes ele deve passar. Pela Figura 23, observa-se que o *iBitmap_Lookup* vem de fora do *Datapath* junto com o *iDstWkp_Lookup* que indica o título relativo àquele *Bitmap*. Estes barramentos estão vindo da *Lookup Table*.

A interface com o bloco *Input Arbiter* foi discutida na subseção anterior. Na parte inferior do *Tagger*, a interface é com o módulo de escalonamento e os pinos da sua interface

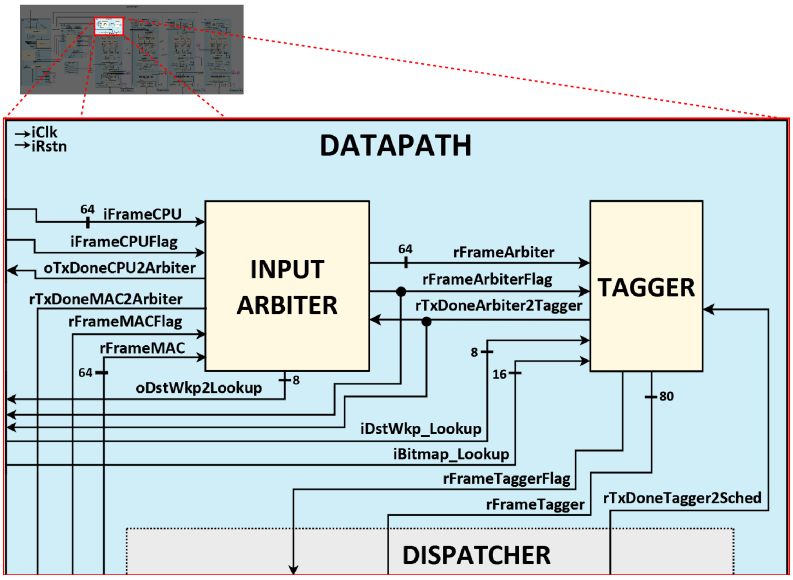


Figura 23: *Tagger*

são, da mesma forma, o quadro recebido, o sinalizador de novo quadro disponível e a confirmação de transferência.

A FSM do *Tagger* é elementar, não necessitando mais que os estados de: *Idle, AttachBitmap, TransferToScheduler*. Caso o título de destino do quadro não seja encontrado na tabela de *Workspaces*, o quadro é descartado e o *Tagger* continua sua operação a partir do estado *Idle*. Neste ponto também pode ser emitida notificação já que a transferência de estado ocorre com condição especial.

5.4.3 *Dispatcher (Scheduling)*

Ao sair do *Tagger* com o *Bitmap* acoplado, o quadro passa então ao estágio do escalonamento. Identificado na Figura 21 como *Dispatcher*, a estrutura de escalonamento do *Datapath* é detalhada pela Figura 24. Como se pode observar, um quadro que chega à entrada no escalonador pode sair por três vias. Ele pode ser entregue a umas das três filas e, em seguida, será retirado por uma das três *Threads*.

Quando o quadro está na saída no componente *Scheduler*, o próprio componente marca a flag da fila correspondente, assim o sinal *rFrameFlag* funciona como um *Write Enable* para a fila. Na fila, os quadros são organizados conforme prioridade. O *Bitmap* desta versão prevê três níveis de priorização. Se quadros consecutivos chegam com a mesma prioridade, fica na frente aquele que chegou primeiro. Todas as filas possuem a mesma estrutura. No protótipo, as filas armazenam quatro quadros em cada uma.

A *Thread* é um mecanismo de busca de quadros na fila e despacho para o árbitro que os encaminha à saída. Os quadros chegam às diferentes filas com a mesma frequência devido ao *Scheduler*, ocorre que cada uma das três *Threads* é alimentada por um clock diferente. Os valores de referência para cada frequência de varredura vêm do componente

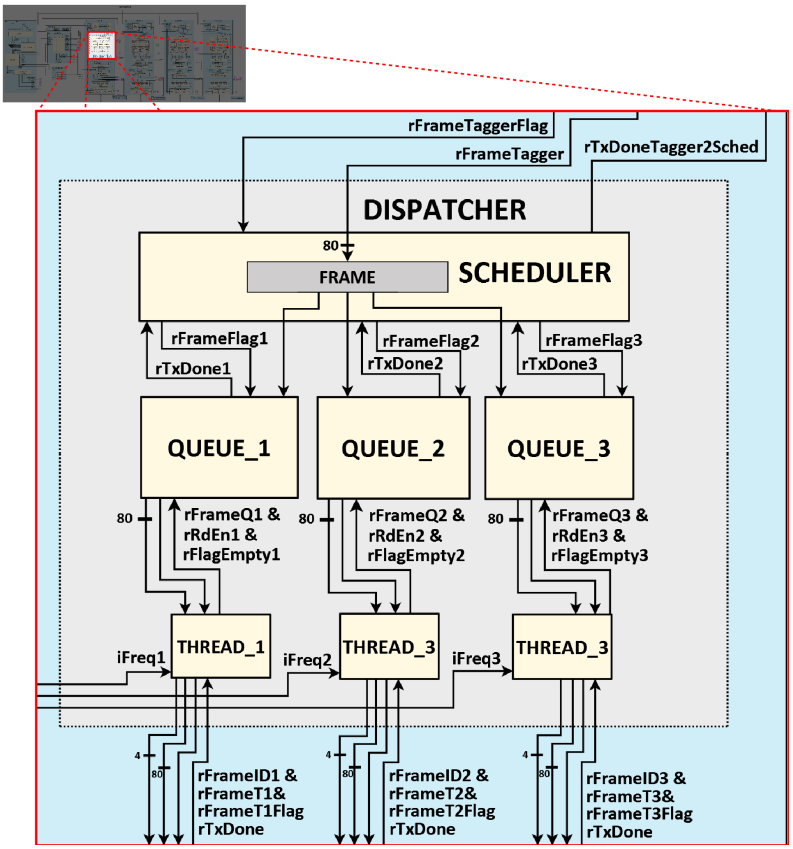


Figura 24: *Scheduling*

Frequencies e cada *Thread* implementa um divisor de clock interno que faz com que sua operação ocorra de forma assíncrona se comparada às outras duas *Threads*.

Desta forma, embora os quadros cheguem de forma síncrona à entrada do escalonador, eles são liberados em diferentes momentos para o restante do processamento. Os valores que serão usados para dividir o clock são indiferentes ao circuito, eles obedecem a políticas que vão de acordo com a necessidade dos *Workspaces*, já que podem ser reconfigurados sob demanda através da Unidade de Controle.

A FSM do *Scheduler*, basicamente, obedece à sequência de copiar mensagem, setar o sinal *rFrameFlag* e aguardar a confirmação de transmissão para a respectiva fila. Cabe salientar que caso a fila esteja cheia é aqui que ocorre o descarte de quadros. Cada *Queue* implementa uma memória tipo FIFO, já que as mensagens são organizadas por ordem de prioridade, e tem capacidade 4x80bits (4 entradas de 80 bits cada).

Paralelamente, a FSM das *Threads* obedece à sequência de checar fila, copiar quadro e aguardar transmissão ao árbitro de saída. Vale pontuar que antes de checar a fila, a FSM fica no estado que marca que a divisão de frequência está ocorrendo.

5.4.4 *QueuesArbiter*

Quando os quadros são buscados nas FIFOs pelas *Threads*, eles ficam a espera de serem transferidos para o *Queues Arbiter*. Este componente é o orquestrador dos canais que disputam a saída das *Threads*. Como mostra a Figura 25, o *Queues Arbiter* recebe de cada *Thread* o quadro pelo *rFrameTn*, o indicador *rFrameTnFlag* e o identificador *rFrameID* e confirma o recebimento pelo *rTxDone*.

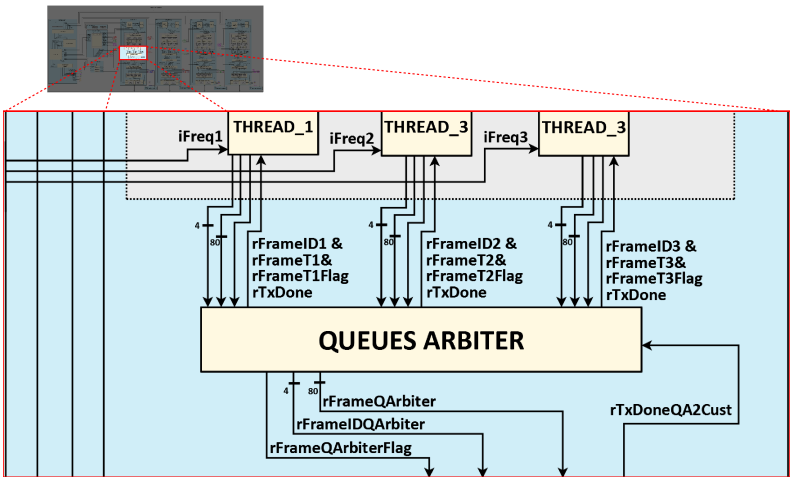


Figura 25: *Queues Arbiter*

A FSM deste componente também implementa uma conferência circular, i.e., confere a primeira entrada, em seguida a segunda e por último a terceira. Caso haja quadro para ser transmitido em alguma delas, a FSM migra a um estado de transmissão ao *Customization* e então seque para a conferência da próxima *Thread* e não para o início. Isso garante o atendimento a todos. Vale lembrar também que já que os quadros são liberados nas *Threads* em frequências diferentes, aqui uma política de equilíbrio é atender em fila circular, para evitar a condição de *starvation*.

Caso as *Threads* estejam liberando os quadros com mesma frequência e prioridade, então o *Queues Arbiter* os transmite sequencialmente por ordem de chegada. Caso haja diferentes configurações nas *Threads*, estes quadros já chegarão nas saídas em momentos diferentes.

Ainda observando a Figura 25, a interface inferior do *Queues Arbiter* se comunica com o *Customization* transmitindo quase todos os parâmetros recebidos das *Threads*, exceto pelo *quadroID* que é gerado novamente em sua FSM para serializar esta contagem que vem de fontes paralelas.

5.4.5 *Customization*

Quando chega ao *Customization*, o quadro agora passará por algum processamento opcional. Este é o ponto em que funções adicionais podem ser inseridas para incluir, por

exemplo, mais segurança a um quadro que deixa o Switch. De acordo com o *Bitmap* o quadro recebido pode passar por processo de criptografia, ter ou não CRC inserido e outras funções que forem planejadas para as próximas versões.

A interface do *Customization* é detalhada pela Figura 26. Na parte superior estão os canais que se comunicam com o *Queues Arbiter* e que foram explicados na Subseção anterior. Já a interface inferior apresenta praticamente os mesmos canais, porém agora com interface com o *Output Port Lookup* e, somando-se a eles, o canal *rFramePriority* também é entregue separadamente ao *Output Port Lookup*.

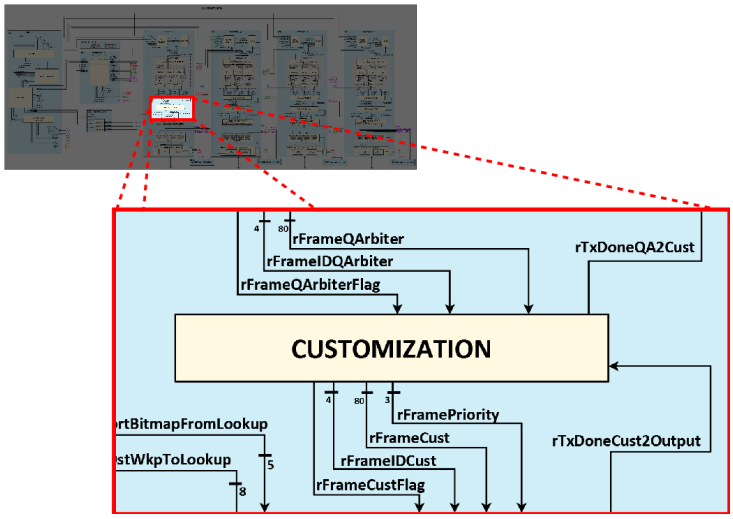


Figura 26: *Customization*

A FSM não apresenta transições que não sigam à ordem: verifica entrada, copia quadro, (possivelmente) realiza customização, transfere ao próximo bloco, limpa todos os registradores e volta ao estado de conferir as entradas.

5.5 Estrutura de interconexão entre *Datapaths*: *Output Port Lookup* e *Mac Tx*

É conveniente apresentar os três componentes que encaminham os quadros às saídas do Switch nesta única seção. Isso se explica pela forte estrutura de interconexão entre eles. Pela Figura 27, pode-se observar o detalhamento dos dois componentes que encaminham os quadros às transmissões.

Primeiramente, o *Output Port Lookup* em cada *Datapath* recebe o quadro do *Customization* pronto para encaminhamento. Perceba que o parâmetro prioridade também é entregue pelo *rFramePriority*. É necessário também observar que o *PortBitmap* referente às portas de saída vem da *Lookup Table* e é entregue ao *Output Port Lookup*. Assim, o quadro recebido está pronto para ser retirado e transmitido.

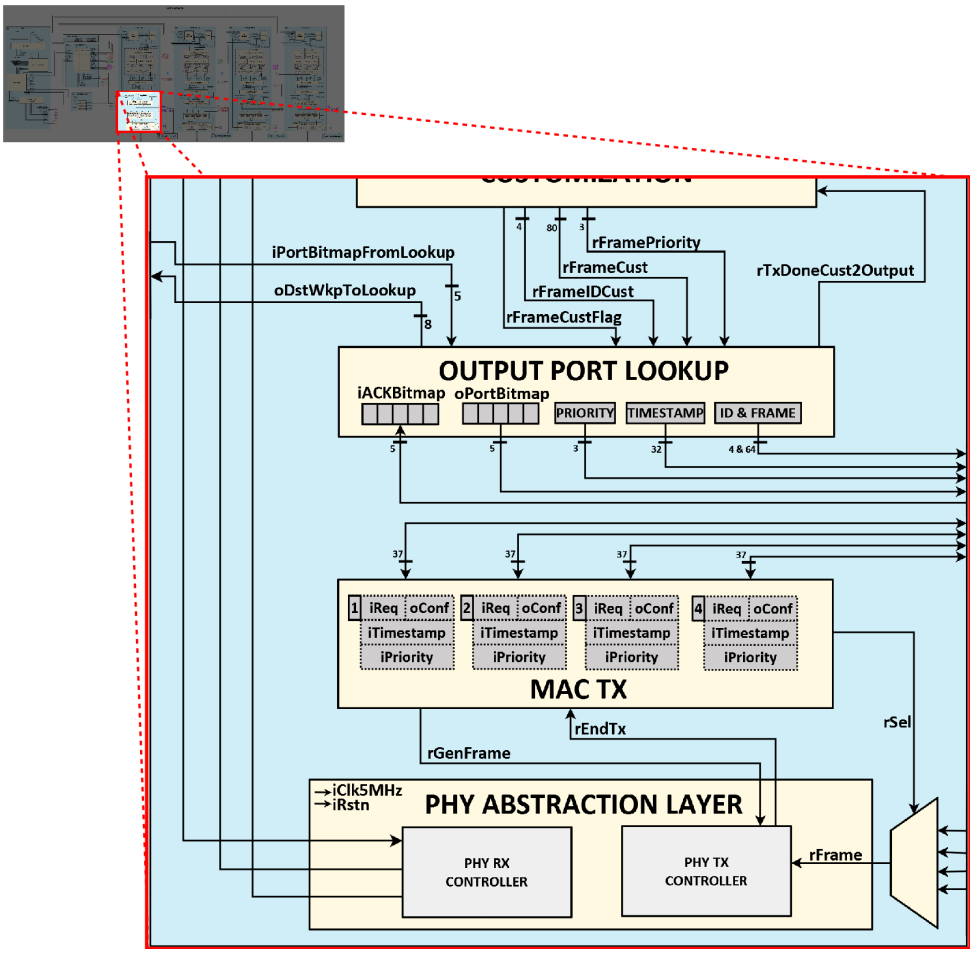


Figura 27: *Mac Tx*

Quando o *Output Port Lookup* coloca o quadro na saída, uma série de canais divulgam informações sobre este quadro. Na interface inferior do componente todos os barramentos ali se conectam aos componentes *Mac Tx* de todos os outros *Datapaths* numa rede de conexões conforme a figura 28.

O *Output Port Lookup* coloca o quadro na saída junto com seu identificador pelo barramento *ID&Frame*. Além disso, os parâmetros *Timestamp* e *Priority* também ficam à disposição das unidades que solicitarão as retransmissões - os *Mac Tx*.

Observa-se também que há dois registradores na saída: o *oPortBitmap* e o *iACKBitmap*. Na verdade, eles são lidos bit a bit por cada *Datapath* conectado. O *oPortBitmap* indica quais *Datapahts* devem transmitir aquele quadro por pertencerem ao *Workspace* de destino. Esta informação vem da *Lookup Table* na forma $b_0b_1b_2b_3b_4$. O *iACKBitmap*, também na forma $b_0b_1b_2b_3b_4$, recebe as confirmações de transmissão de cada *Datapath*, assim o *Output Port Lookup* detecta quando todas as transmissões já ocorreram e pode tirar o quadro da sua saída.

Enquanto isso, o *MAC Tx* lê os dados de todos os quatro *Output Port Lookups* do switch. A cada ciclo, este componente verifica se há quadros aguardando para serem transmitidos por ele e, caso sim, ele faz a cópia do quadro, confirma ao *Output Port*

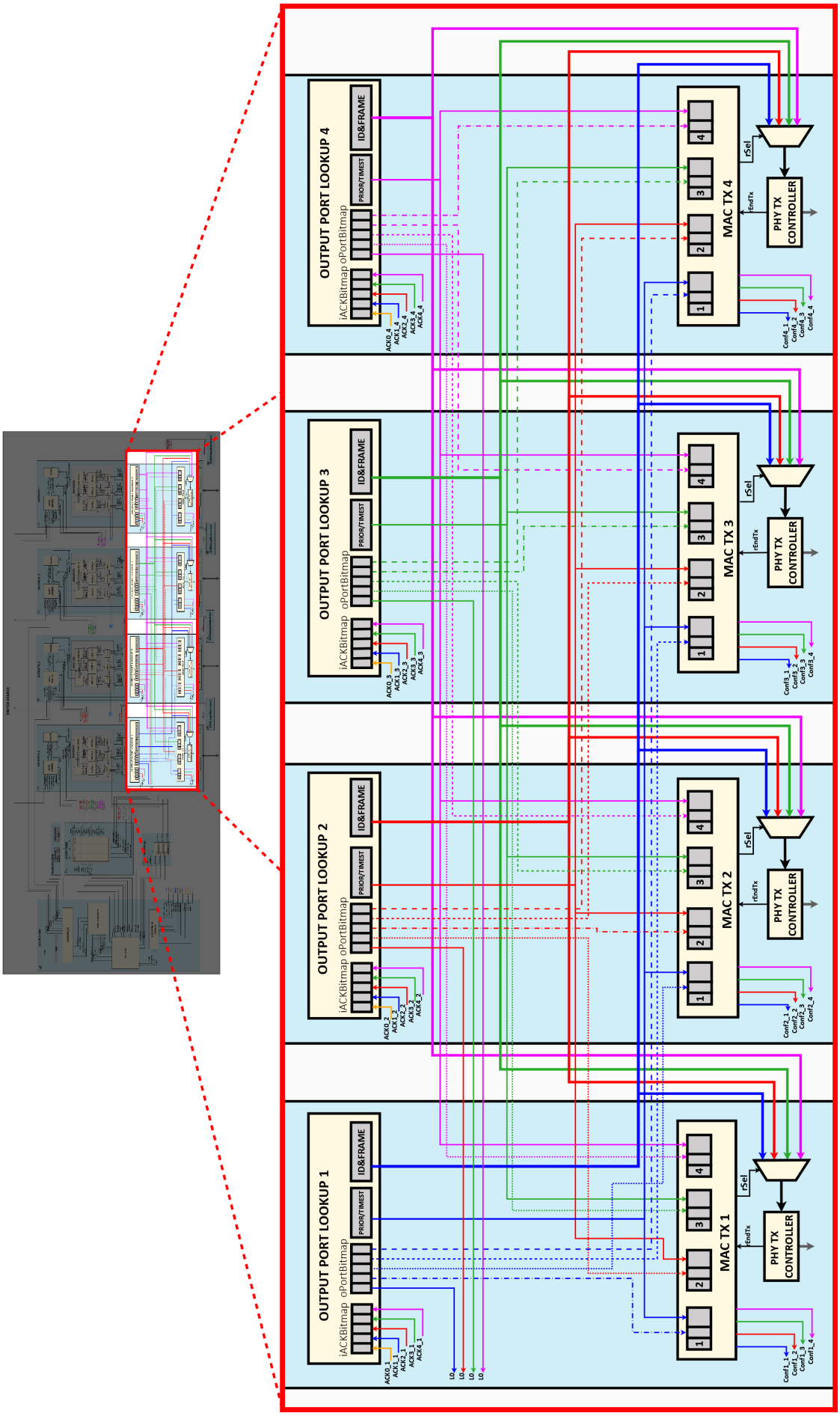


Figura 28: Interconexão entre *Output Port Lookups* e *Mac Txs*

Lookup de origem e solicita ao seu *PHY Tx Controller* que cuide da transmissão. Quando ocorre de ter mais de um quadro a ser transmitido para o mesmo *Datapath*, aí o *MAC Tx* faz a comparação e seleciona aquele que tiver maior prioridade e, em caso de empate, seleciona o que está há mais tempo esperando par ser transmitido.

A Figura 28 deixa esta estrutura bem detalhada. Por exemplo, o *Output Port Lookup 1* emite sinais a partir do seu *oPortBitmap* na cor azul que chegam ao espaço reservado por cada *MAC Tx* à solicitação de transmissão vinda do *Datapath1*. Da mesma forma, o *iACKBitmap* ainda do DP1 recebe confirmações de todos os *Datapaths* e da Unidade de Controle.

Os sinais solicitando transmissão do quadro do *Datapath1* à Unidade de Controle e aos *Datapaths* são, respectivamente: *iReq1_0, iReq1_1, iReq1_2, iReq1_3* e *iReq1_4*. Analogamente, as confirmações saem da Unidade de Controle e *Datapaths* na forma: *iConf0_1, iConf1_1, iConf2_1, iConf3_1* e *iConf4_1*. Os sinais *iACK* percebidos em cada *Output Port Lookup* não vêm direto das confirmações pelos *MAC Tx*. Devido à necessidade de bufferização das confirmações até que a última transmissão tenha sido confirmada, foi necessário incluir o componente *ACK Bitmap Abstraction* para garantir a montagem do vetor de confirmação já que as confirmações ocorrem em instantes de tempo diferentes.

A FSM do *Output Port Lookup* baseia-se na sequência: confere entrada, copia quadro, coloca na saída, transfere aos *MAC Tx's*, limpa todos os registros. Um *timeout* de confirmação também foi estabelecido aqui para que o Switch não ficasse com as portas travadas por erros de transmissão. Apesar de não informar o erro nesta versão, as transições são separadas e permitem que o switch gere notificação quando a transição por *timeout* ocorrer.

A FSM do *MAC Tx*, projetada para a comparação de Prioridade e Tempo de espera, está prototipada para esta versão de demonstração, na forma de fila circular, i.e., confere sequencialmente cada *Datapath* e transmite se houver quadro a ser transmitido. Na volta, a FSM volta ao estado relativo à conferência do próximo *Datapath* e não do estado inicial.

5.6 *ACK Bitmap Abstraction*

O *ACK Bitmap Abstraction* está mais relacionado a uma necessidade da implementação em hardware do que ao projeto do Switch em si. Cada confirmação de quadro transmitido que um *MAC Tx* envia ao *Output Port Lookup* fica em nível alto por apenas um ciclo de clock, ou seja, quando o *MAC Tx* recebe confirmação do controlador do PHY, ele avisa imeditamente ao emissor do quadro e já volta ao seu estado inicial.

Ocorre que um *Output Port Lookup* espera até cinco confirmações devido ao seu *oPortBitmap*, conforme a estrutura mostrada na Figura 27. Estas cinco confirmações chegam de forma independente umas das outras, pois os *MAC Tx* agem como se fossem nós de um

sistema distribuído operando paralelamente. Apenas em uma situação muito excepcional estas confirmações estariam setadas ao mesmo tempo.

Por isso, o *ACK Bitmap Abstraction* recebe as confirmações *iConfM_N* e as bufferiza até montar todo o vetor de ACKs e entregar ao *Output Port Lookup* para que ele o compare com o *oPortBitmap*.

Observe a Figura 29 que detalha as interfaces tomando como exemplo o *ACK Bitmap Abstraction* relacionado ao *Datapath1*. O componente recebe dois barramentos: o *iPortBitmap1* que tem o *PortBitmap* indicando quais as unidades devem retransmitir aquele quadro e o *rACKbitmap1* que recebe as confirmações de cada transmissão. Sua saída é dada pelo *rACKBitmapBuffer1* que indica todos que confirmaram a retransmissão.

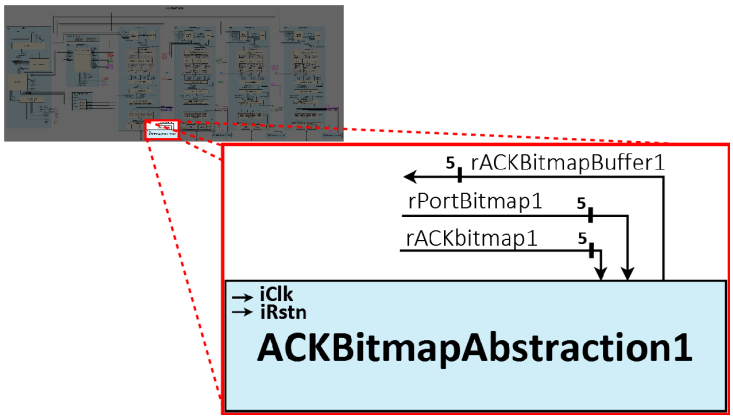


Figura 29: *ACK Bitmap Abstraction*

Os barramentos são da forma:

rPortBitmap1 <= rReq1_0 & rReq1_1 & rReq1_2 & rReq1_3 & rReq1_4;

ACKbitmap1 <= rConf0_1 & rConf1_1 & rConf2_1 & rConf3_1 & rConf4_1;

rACKbitmapBuffer <= rACK0_1 & rACK1_1 & rACK2_1 & rACK3_1 & rACK4_1;

Cada *ACK Bitmap Abstraction* instancia cinco componentes que implementam, cada um, uma FSM que espera pelo sinal de confirmação e mantém uma flag setada até que todas as confirmações tenham sido recebidas. Quando isso ocorre, ele transita a um estado que indica que todas as transmissões foram feitas e retorna ao seu estado inicial.

5.7 *Parallel Sorter*

As filas do escalonador nos *Datapaths* são FIFOs que organizam suas mensagens por ordem de prioridade e chegada. Da mesma forma, quando um *MAC Tx* verifica que tem mais que um quadro a ser transmitido, ele precisa decidir qual *Datapath* atender primeiro. Esta decisão quadro tem maior prioridade e está a mais tempo esperando *timestamp*. A Figura 30 mostra o projeto do ordenador construído para escolher qual quadro será priorizado nos *MAC Tx's*.

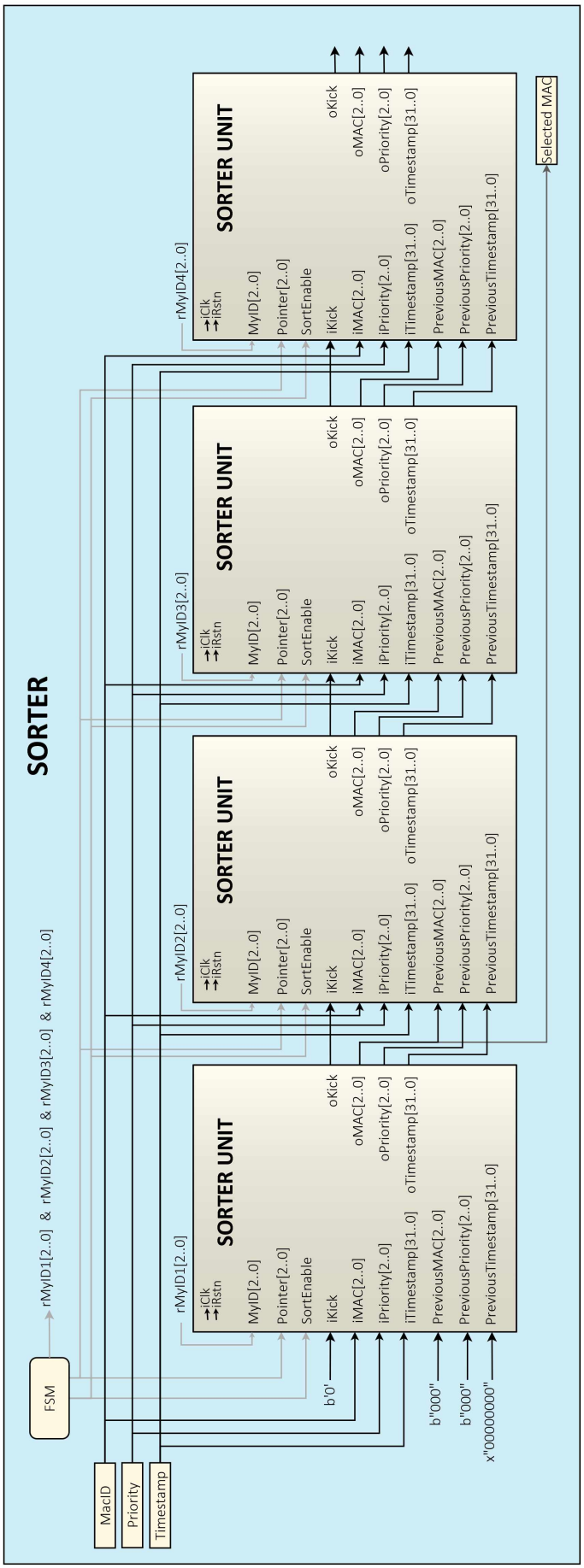


Figura 30: Sorter

O *Sorter* instancia quatro unidades elementares de ordenação. Cada uma das unidades *Sorter Unit* recebe seu identificador *MyID* e o barramento *Pointer* controla a quantidade de interações do algoritmo e quais os módulos participarão da ordenação. Uma fila de apenas três entradas não precisa de quatro unidades em execução. O sinal *SortEnable* é que habilita as comparações e a atualização da saída. Quando um novo dado chega à primeira *Sorter Unit*, ele é naturalmente transferido para a saída dos registradores, pois os valores referência estavam todos zerados.

Quando o segundo conjunto de dados chega ao primeiro e segundo componentes, cada um deles o compara com os valores atuais dos seus registradores. Se este valor na entrada for prioritário em relação ao que já está armazenado, ele transfere o seu atual conteúdo para a saída e ativa o sinal *oKick* para que o próximo componente saiba que este conteúdo armazenado agora deve ser deslocado para frente. Daí, a unidade verifica sua entrada *iKick* e, se ela estiver ativa, então os novos dados estão vindo do componente anterior (que passou pelo mesmo processo). Se o *iKick* estiver inativo, a unidade deve receber o novo conjunto de dados das linhas de entrada. E assim, automaticamente, os dados vão sendo inseridos já na sua forma ordenada e ao fim da execução, as saídas da primeira *Sorter Unit* já informará o *MAC Tx* prioritário.

Implementação em software de algoritmos de ordenação podem ter complexidade da ordem $\theta(nlog(n))$ e $\theta(n^2)$ (Osama; Omar; Badr, 2016). Quando se trata de hardware, a solução para este problema pode ser mais eficiente, conforme exposto por (Song et al., 2016). A complexidade aqui fica em $\theta(n)$, mas com n contado em ciclos de clock do hardware. Comparar este valor com uma execução em software tem que considerar o formato e modo de endereçamento das instruções do GPP utilizado, mas em nenhum caso será menor que esta complexidade contada em ciclos de clock e não de instrução.

A FSM é simples. Ela sai do *Idle* para o estado *Sort* quando há dados novos e nele permanece por tantos ciclos quantos forem o número de elementos a serem comparados. Então, volta ao seu estado inicial.

5.8 A aplicabilidade do FPGA

Hardware reconfigurável são amplamente utilizados para implementar dispositivos de rede, conforme Corsa (2019), Jose et al. (2015) e Bosshart et al. (2013). Aplicações implementadas em *Field-Programmable Gate Array (FPGA)* normalmente alcançam maior desempenho, menor latência e menor consumo de energia em comparação com implementações com CPUs de uso geral.

Atendendo ao *trade off* entre programabilidade e desempenho, Sezer et al. (2013) colocam o FPGA em um nível intermediário entre processadores de uso genérico e o *Application Specific Integrated Circuit (ASIC)* e afirmam que esta tecnologia é ideal para implementar funções de rede altamente paralelas. Com FPGA o pipeline do plano de

encaminhamento chega a atingir atualmente processamento acima de 200 Gbps por dispositivo, conforme a Figura 31.

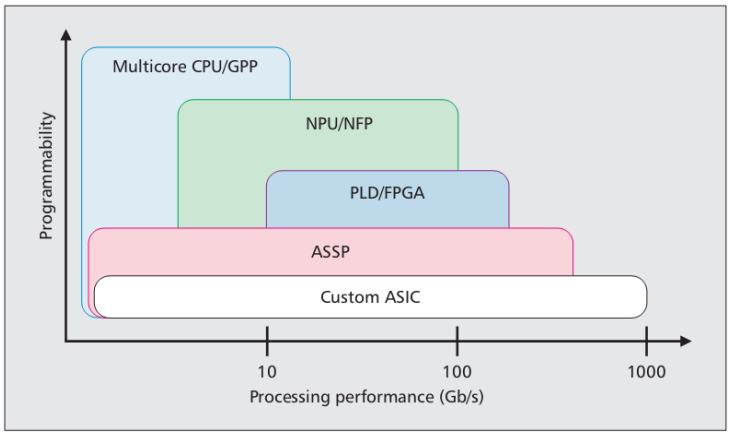


Figura 31: Processamento de rede: Desempenho *vs.* Programabilidade.
Fonte: (SEZER et al., 2013)

A adoção especificamente de FPGA como hardware do Switch é coerente e promissora, uma vez que FPGAs possibilitam um longo ciclo de vida de um produto devido à sua reconfigurabilidade, além de terem desempenho muito superior ao software. Seu *Non Recurring Engineering (NRE)* é consideravelmente mais baixo que de um ASIC, i.e., o custo inicial para pesquisar, desenvolver, projetar e testar um novo produto é bem menor. Além disso, o tempo de projeto e custos de desenvolvimento são mais baixos se comparados aos ASICs, conforme Hauck e DeHon (2010) e Singh (2018).

Utilizar uma plataforma de hardware reconfigurável no processo de desenvolvimento significa adotar um modelo de projeto baseado em prototipagem rápida. A prototipagem é um processo que dá ênfase à construção de protótipos durante o desenvolvimento e isso permite ter *feedback* mais rápido deste processo, conforme reafirmam Radatz (1997) e Tioh et al. (2008). O *feedback* mais rápido é benéfico para identificar problemas que podem reduzir os custos de produção, o que tem feito esta tendência de protótipos crescer desde o surgimento da reconfigurabilidade.

Assim, combinando uma proposta de Internet do Futuro e Computação Reconfigurável, este trabalho volta a atenção para o hardware que implementa o plano de dados da arquitetura ETArch e como ele pode cooperar com o software para prover uma solução flexível e ágil para dar suporte a requisitos das aplicações operando sobre esta arquitetura. O projeto e implementação de um equipamento reconfigurável especialmente projetado para a comutação na filosofia de orientação a *Workspace* motivam este trabalho.

5.9 Resumos dos dados de prototipagem

A implementação apresentada neste capítulo se refere à proposta do Switch ETArch em sua primeira versão. A largura dos barramentos pode sofrer alteração conforme for a capacidade da plataforma de prototipagem. Vale ressaltar que o projeto inicial foi feito com vetores de 512 bits, pois este valor é o tamanho mínimo compatível com o comprimento do quadro Ethernet. Ademais, tráfego de todas as mensagens de controle do Switch poderia ocorrer sem necessidade de fragmentação, bem como 512 bits são suficientes para grande parte dos pacotes que trafegam na rede em domínios multicast, por exemplo, para áudio conferências.

Ocorre que a plataforma de prototipagem impõe limites no uso de recursos de hardware. O FPGA utilizado possui 33216 elementos lógicos disponíveis. Quando o projeto foi sintetizado e o Quartus II v13.1 gerou a *Netlist Post-Fitting*, i.e., a lista de recursos utilizados para prototipagem depois de distribuir o protótipo no FPGA, o switch precisaria de aproximadamente quatro vezes a capacidade deste FPGA.

Apenas para um *Datapath*, com uma FIFO reduzida, o uso ficava entre dezenove mil e vinte mil elementos lógicos. Logo, ao se combinar os quatro *Datapaths* com a Unidade de Controle, *Lookup Table* e *Frequencies* este valor estaria na casa dos cem mil elementos lógicos. Então, o projeto foi readaptado para uma versão com o quadro em 64 bits, conforme Figura 44.

Embora cada módulo receba dados dos outros módulos, os processamentos em cada um deles é diferentes entre si, o que implica em diferentes arranjos lógicos para sua implementação. Em virtude disso, eles são limitados por diferentes valores de frequência máxima de operação. O mesmo ocorre com a área ocupada no FPGA. A Tabela 6 lista a ocupação do hardware por cada componente e suas limitações de frequência no FPGA Cyclone II.

A Tabela 6 mostra os dados coletados ao se trabalhar com cada componente separadamente, i.e., no momento em que cada componente é projetado, simulado e sintetizado independentemente dos outros. Quando as conexões são realizadas, os valores ficam diferentes devido a caminhos que se juntam, pinos de entrada e saída que se conectam via sinais intermediários e possíveis lógicas redundantes. No Capítulo 6 estes resultados são comparados aos dados de prototipagem do sistema completo.

Já se pode notar que os componentes que mais usam elementos lógicos são *Phy Abstraction Layer*, *Lookup Table*, *ACK Bitmap Abstraction* e as *Queues*. Esse é um dado importante, já que trata especialmente dos blocos que podem ficar fora do chip do processador principal do switch. Outra observação é em relação à frequência máxima de operação que também é limitada por baixo pelo *Phy Abstraction Layer*.

Tabela 6: Dados de prototipagem dos componentes

Component	**Logic Elements (occupied area)**	**Combinational Functions**	**Registers**	F_{MAX} **(MHz)**	**Pins**
DATAPATH					
Input Arbiter	142	77	138	326,05	210
Tagger	246	216	119	195,31	176
Scheduler	100	98	96	275,63	173
Queue 1	426	416	406	188,29	172
Queue 2	426	416	406	188,29	172
Queue 3	426	416	406	188,29	172
Thread 1	197	195	136	209,78	180
Thread 2	197	195	136	209,78	180
Thread 3	197	195	136	209,78	180
Queues Arbiter	297	295	203	173,37	342
Customization	228	228	179	198,57	160
Output Port Lookup	476	437	325	191,28	201
Mac Tx	366	365	161	181,72	167
Phy Abstraction	1799	1233	1226	173,34	1126
CONTROL UNIT					
Datapath Rx	249	249	93	180,44	347
Decoder	258	258	194	218,72	217
MSG Assembler	75	73	75	428,45	145
Datapath Rx	94	94	78	386,4	147
ACKBitmap Abst.	1009	994	379	183,62	25
LOOKUP TABLE	1864	1863	529	139,08	243
FREQUENCIES	52	48	28	401,77	38

Capítulo 6

Análise de Resultados

Em se tratando de resultados, salienta-se que os principais resultados desta tese foram apresentados nos Capítulos 4 e 5. A especificação do novo protocolo de configuração do Switch e sua arquitetura, por si só, são as máximas deste trabalho.

Este capítulo se destina a apresentar uma análise da operação do *Switch* no sistema de demonstração montado, bem como faz considerações sobre quais as medidas necessárias para que o ele possa operar em redes Gigabit. Primeiramente, a discussão é voltada para a execução dos serviços de controle do *Switch* (Seção 6.1). Em seguida, são analisados o tempo gasto para o processamento das mensagens e a frequência de operação do hardware (Seções 6.3 e 6.4). Por último, a atenção se volta à prototipagem e uso do FPGA (Seção 6.2).

6.1 Trocas de Mensagens e Execução dos Serviços de Controle

Retomando a discussão sobre a operação do *Switch*, convém observar a Tabela 2. São listados quatro processos para avaliação e que demonstram uma operação funcional. Os resultados destes processos são apresentados na forma de diagramas de sequência que descrevem exatamente as trocas de mensagens durante a execução dos serviços.

As trocas de mensagens detalhadas a seguir são relativas às mensagens recebidas sem erros. Este é um detalhe importante de observar para o sistema de demonstração construído, pois não foi implementado controle de erro para as interfaces de rede, apenas detecção. O sistema que gerou os resultados aqui apresentados é descrito na Subseção 3.4.

A Figura 32 detalha a troca de mensagens que ocorre durante a execução do serviço de criação de novo *Workspace*. Antes que um primeiro *Workspace* fosse adicionado à tabela de *Workspaces* uma entidade envia uma mensagem de dados destinada a um *Workspace* qualquer. Em um dos exemplos foi usado o título de destino 0xBB. A mensagem chega ao

switch, é vista nos displays de sete segmentos da placa com conteúdo íntegro e nenhuma ação adicional ocorre. A mensagem não chega a nenhum destino. Isso porque o *Tagger* do *Datapath* não recebeu na tabela de *Workspaces* nenhum *Bitmap* para prosseguir o processamento. Deste ponto a mensagem foi ignorada.

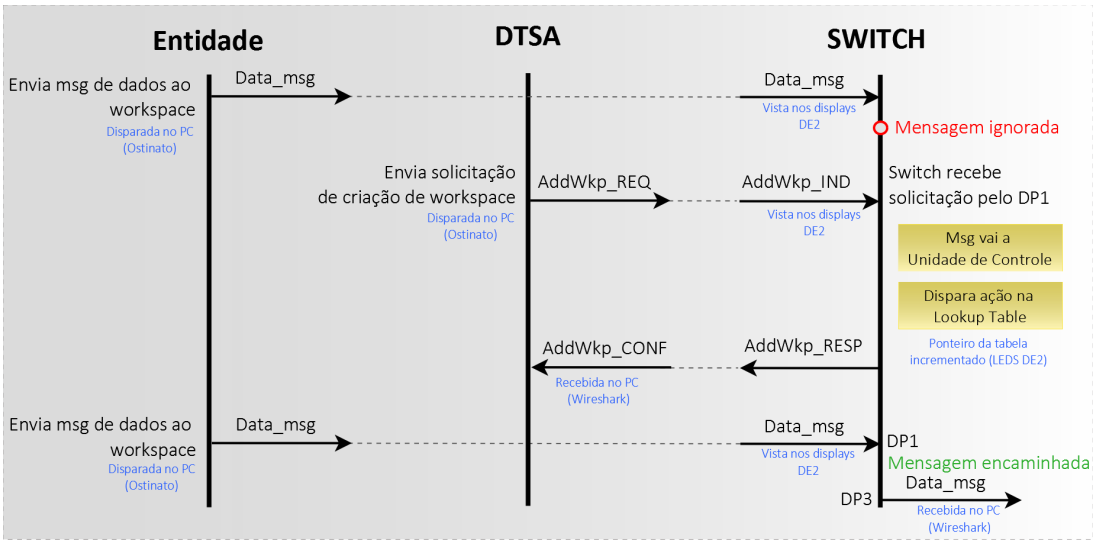


Figura 32: Serviço de Criação de *Workspace*

Depois de algum tempo, foi gerada outra mensagem no simulador de DTSA com tipo 0x01 solicitando a criação de um *Workspace*, naquele caso, de título $0xBB$. O campo *Portmap* tinha valor b"00010" para que a mensagem saísse pelo *Datapath3* A mensagem foi recebida pela placa, exibida nos displays e o ponteiro da tabela de *Workspaces* foi incrementado, indicando que alguma entrada tinha sido inserida na tabela. Pelo mesmo *Datapath*, voltou uma mensagem de tipo $0x02$ com os mesmos parâmetros da mensagem recebida, significando que a confirmação do serviço estava sendo transmitida de volta.

Então, solicitou-se o reenvio da mensagem de dados, que foi recebida pelo *Datapath1* e reencaminhada pelo *Datapath3* e recebida no PC conectado à PHY3, onde foi detectada pelo Wireshark, confirmando, assim, que agora $0xBB$ era uma entrada válida de título de *Workspace*.

A Figura 33 detalha a troca de mensagens que ocorre durante a execução do serviço de edição de *Worksapce* existente na tabela. Para começar os testes de edição, uma mensagem foi enviada para um *Workspace* de dados como destino. A mensagem foi recebida pelo *Datapath1* e reencaminhada pelo *Datapath3*, já que o *PortBitmap* tinha valor b''00010". Isso atesta que o destino daquele *Workspace* era uma entrada válida na tabela. Então, uma mensagem com *MsgType* $0x01$, i.e., *EditWkp_Req* foi enviada ao ao Switch solicitando alteração do *PortBitmap* para b''00000". Ao ser recebida, ela foi à Unidade de Controle que enviou mensagem de confirmação pelo mesmo *Datapath* e esta confirmação foi percebida pelo Wireshark no PC que simula o DTSA.

Em seguida, a mesma mensagem de dados foi enviada ao *switch* e não foi reenca-

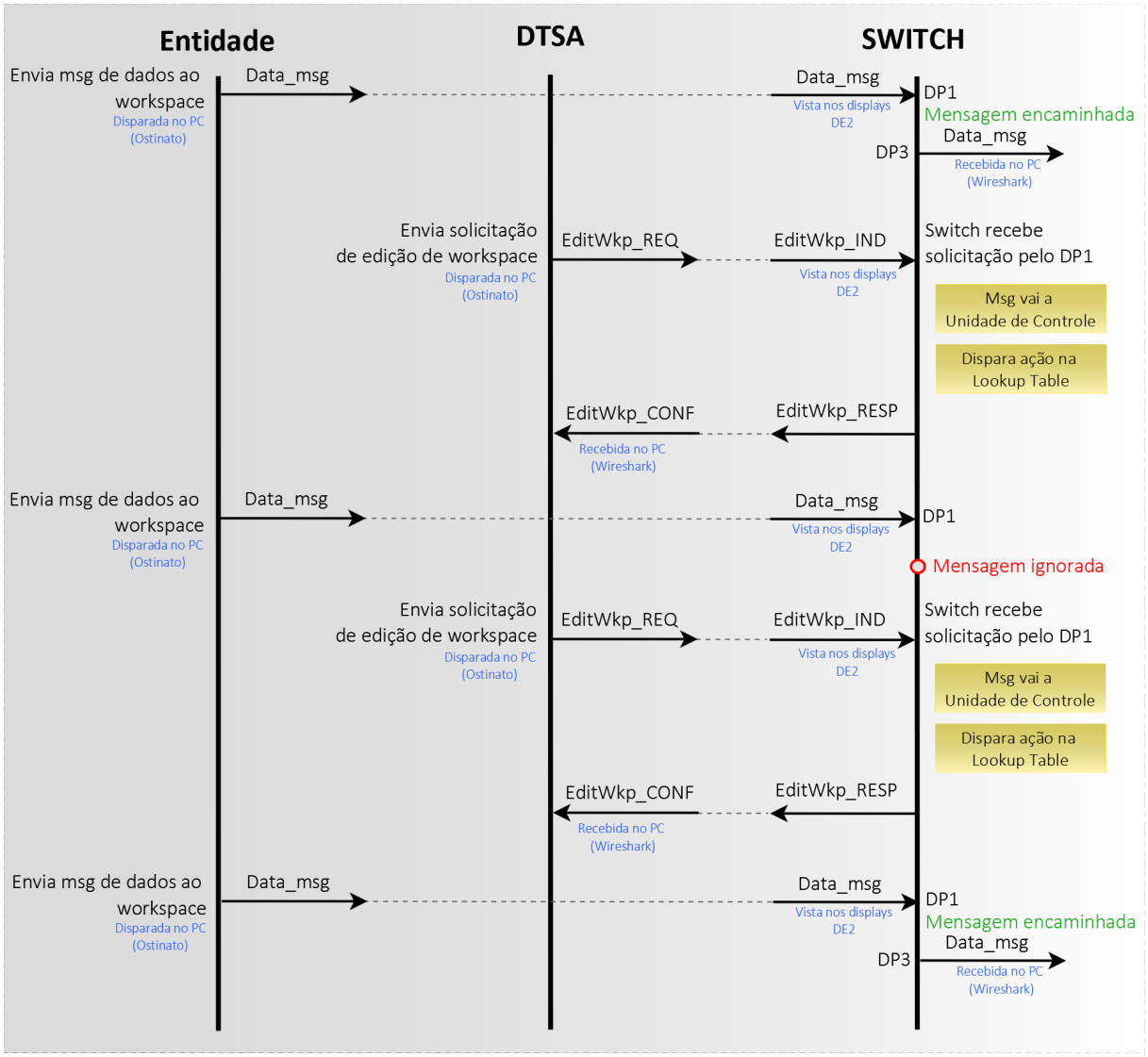


Figura 33: Serviço de Edição de *Workspace*

minhada por nenhuma saída. Isso demonstra que aquela entrada na tabela passou por modificação. Então, o PC simulando o DTSA enviou novamente mensagem de edição retornando ao *PortBitmap* de origem e recebeu a confirmação da placa. Quando a mesma mensagem de dados foi reenviada, ela foi reencaminhada ao *Datapath3*.

A Figura 34 detalha a troca de mensagens que ocorre durante a execução do serviço de exclusão de *Worksapce*. Neste caso, uma entrada na tabela deve ser excluída. Ao receber uma mensagem destinada a um *Workspace* de dados pelo *Datapath1*, esta mensagem foi reencaminhada pelo *Datapath3*, já que a entrada continua a mesma descrita no teste anterior, de edição de *Workspace*. Ocorre que o DTSA enviou mensagem do tipo $0x05$ para a exclusão do *Workapace* $0xBB$. Uma mensagem de confirmação foi percebida pelo Wireshark no DTSA, bem como o ponteiro da tabela foi decrementado nos LEDS do *debug*. Quando a mensagem de dados foi reenviada ao *switch*, ela foi ignorada.

Desta forma, as três Figuras 32, 33 e 34 demonstram configuração feita no FPGA,

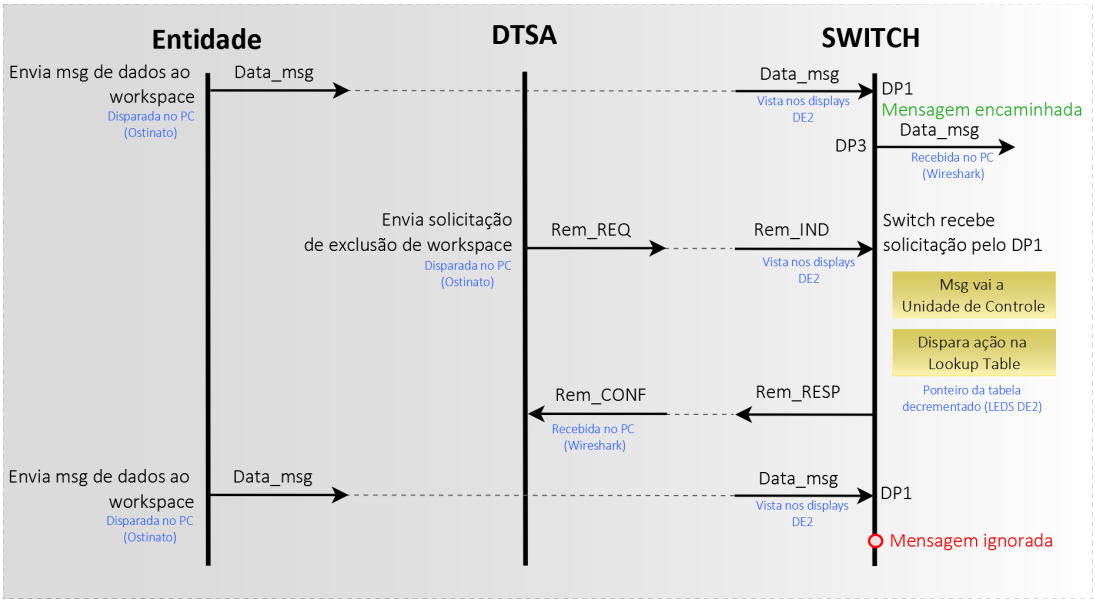


Figura 34: Serviço de Exclusão de *Workspace*

resolvendo a variável M1 da Tabela 2. Repetir o teste das Figuras 32 e 34 com diferentes *Workspaces* de destino resolveram a M2. Os testes da Figura 33 resolvem a M4 e todos os testes resolvem a M3.

Para este sistema de validação não foi possivel medir a taxa de erro de forma probabilística, pois não há fluxo intenso de mensagens em trânsito.

6.2 Análise da Prototipagem em FPGA

No Capítulo 5 os dados de uso componente a componente foram apresentados. Quando os blocos elementares são integrados na formação de um sistema final, neste caso o da Figura 17, alguns valores podem sofrer variações. Por isso, a Tabela 7 traz os dados do uso do FPGA pelo *switch*, componente a componente, e serve como referência para as discussões.

Neste mesmo contexto, a Figura 35 mostra o uso dos elementos lógicos do FPGA por cada componente prototipado no *switch*. Os componentes que se repetem quatro vezes nos *Datapaths* foram indicados apenas uma vez neste gráfico.

São seis medidas extraídas por componente. As três primeiras, nas cores vermelha, amarela e azul, indicam, respectivamente, o número de funções combinacionais, registradores dedicados e pinos de entrada e saída alocados quando o componente foi compilado e sintetizado individualmente. Já as três últimas, verde, laranja e lilás, indicam os mesmos parâmetros na compilação do sistema em que os componentes estão integrados, conforme Figura 17 e Tabela 7.

É interessante notar que o maior uso ocorre pelo *Lookup Table*, i.e., a tabela de *Workspaces*, e este valor se mantém praticamente inalterado nas duas situações. Em seguida,

Tabela 7: Dados de prototipagem do Switch ETArch 64 bits

Component	**Logic Elements**	**Combinational Functions**				**Registers**				F_{MAX} **(MHz)**
		DP1	**DP2**	**DP3**	**DP4**	**DP1**	**DP2**	**DP3**	**DP4**	
TOP SWITCH	22240	21812				13159				111,92
DATAPATH		3725	3710	3710	3710	2678	2669	2669	2669	
Input Arbiter		79	79	79	79	136	136	136	136	
Tagger		171	171	171	171	104	104	104	104	
Scheduler		81	81	81	81	78	78	78	78	
Queue 1		338	338	338	338	331	331	331	331	
Queue 2		337	337	337	337	331	331	331	331	
Queue 3		337	337	337	337	331	331	331	331	
Thread 1		175	175	175	175	117	117	117	117	
Thread 2		176	176	176	176	117	117	117	117	
Thread 3		176	176	176	176	117	117	117	117	
Queues Arbiterv		274	274	274	274	183	183	183	183	
Customization		221	221	221	221	173	173	173	173	
Output Port Lookup		397	397	397	397	254	254	254	254	
Mac Tx		361	361	361	361	156	156	156	156	
Phy Abstraction		401	392	392	590	250	242	242	242	
CONTROL UNIT		520				334				
Datapath Rx		164				65				
Decoder		234				170				
MSG Assembler		46				48				
Datapath Rx		69				51				
ACKBitmap Abst.		829	829	829	829	372	372	372	372	
LOOKUP TABLE		1843				529				
FREQUENCIES		31				27				
Mux 2Kx1K		32				32				
Mux 36Kx1K		1156				64				
Displays		56				0				

está o *Phy Abstraction*, o que cuida da comunicação com a interface de rede, mas para ele o uso diminuiu drasticamente quando ele foi integrado aos outros componentes, o que também ocorre com os seus pinos. É que agora estes pinos estão se conectando a sinais intermediários e os elementos lógicos já estão sendo contabilizados pelos outros componentes. Este comportamento pode ser observado em todos os componentes mas, nos outros, de forma mais discreta.

O *ACK Bitmap Abstraction* também usa boa parcela dos recursos, seguido das filas dos escalonadores (*Queues*). Isso permite concluir que os componentes que mais usam são aqueles ligados a armazenamento e não às atividades de processamento em si.

É oportuno observarmos as Figuras 36 e 37. Ela mostra o uso total de funções combinacionais por tipo de componente e dá uma ideia de proporcionalidade, da porção do

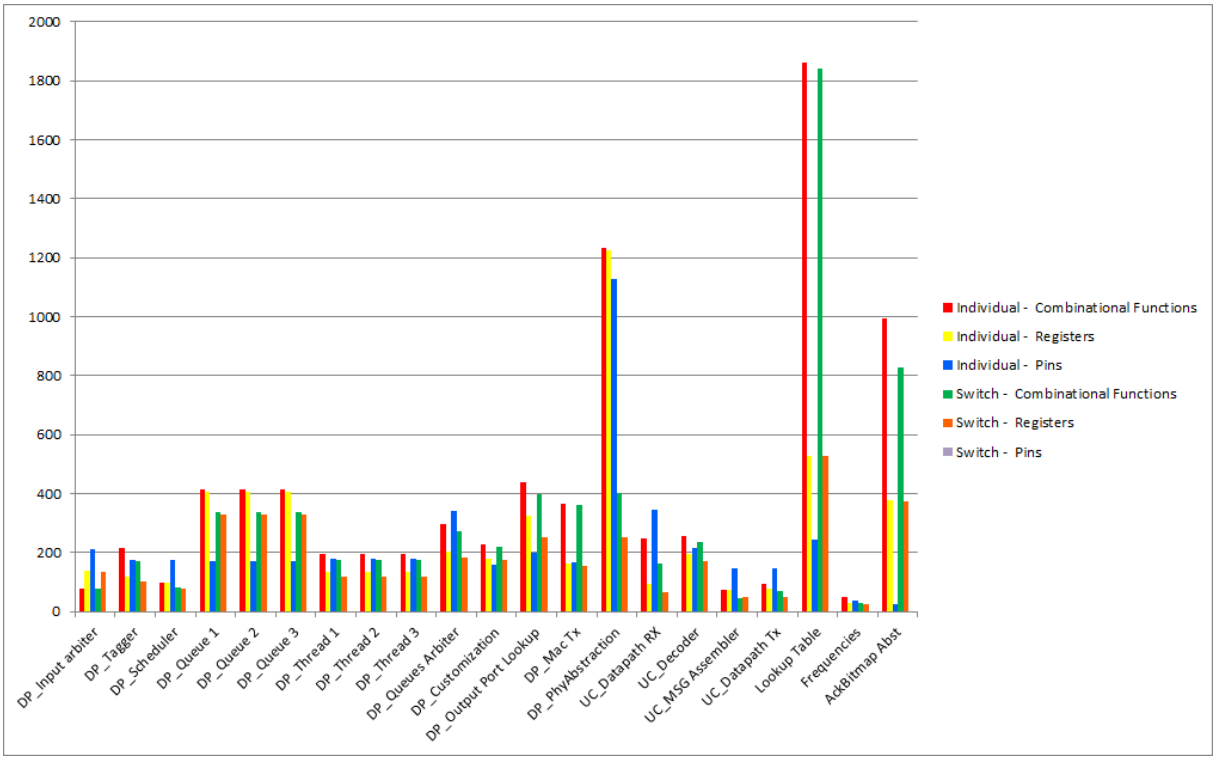


Figura 35: Uso de Elementos Lógicos por unidade de hardware

todo que cada um está consumindo.

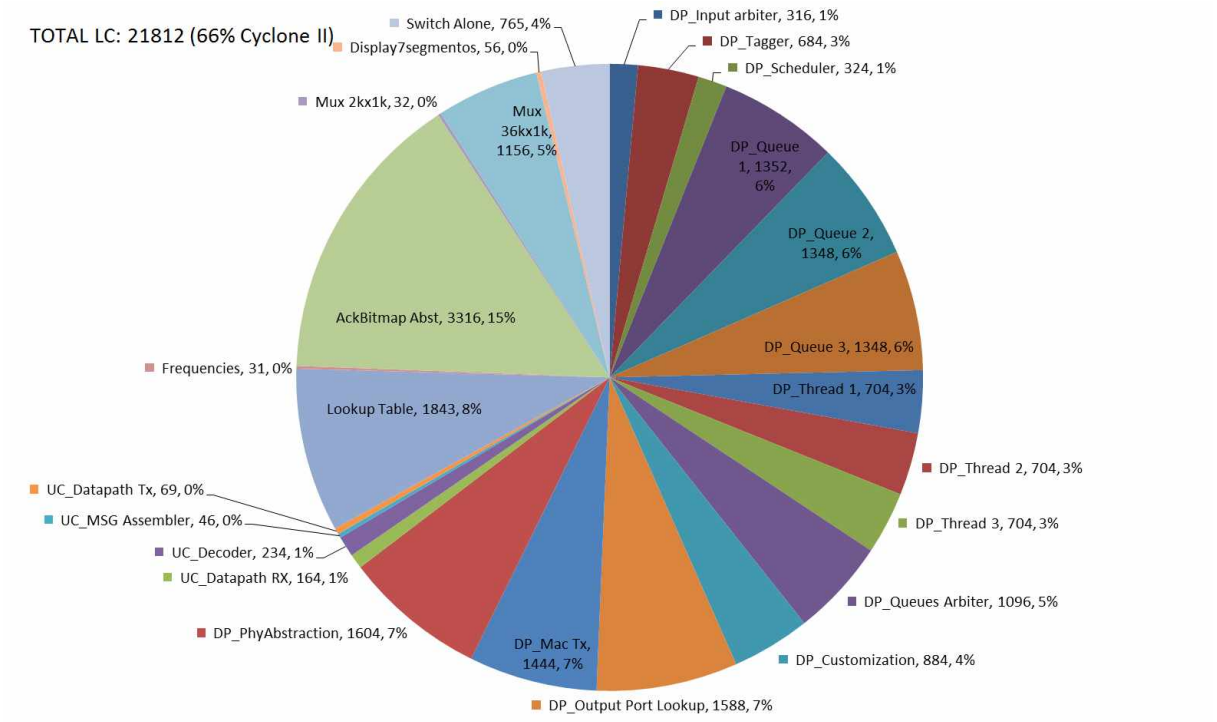


Figura 36: Uso de Lógica Combinacional

Na Figura 36, os componentes que se repetem no projeto já aparecem com seu valor de

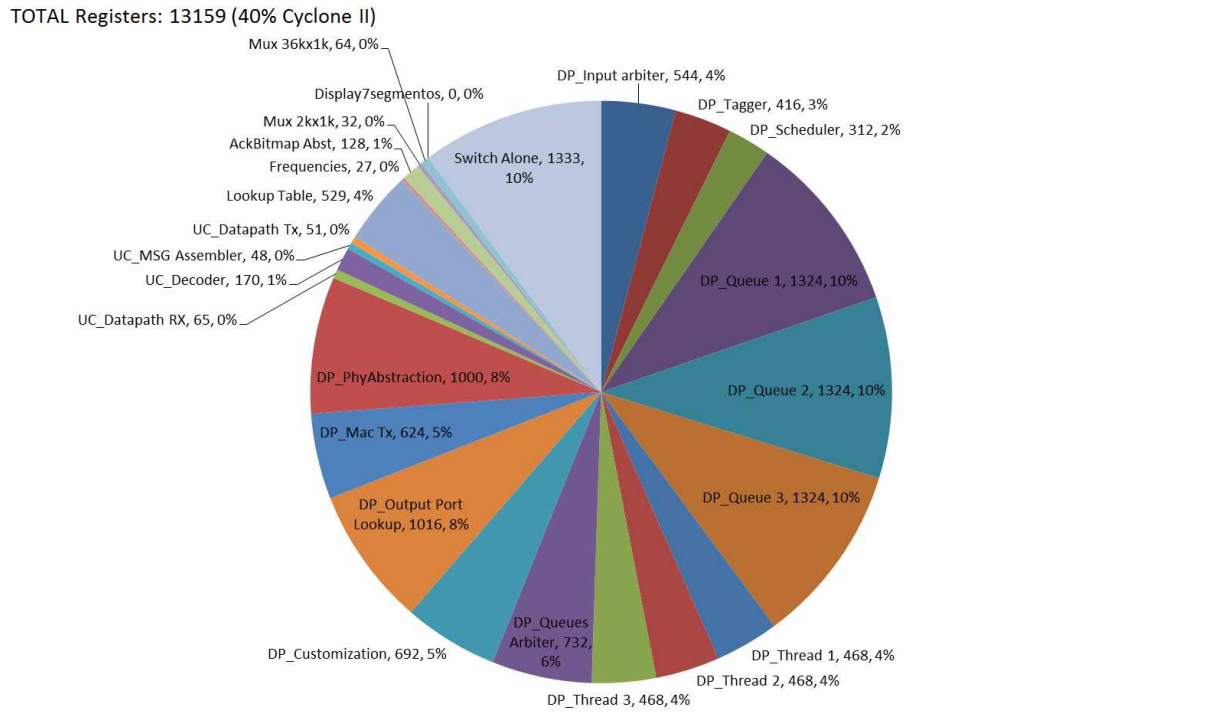


Figura 37: Uso de registradores dedicados

uso total. Por exemplo, o *Tagger* ocupava 216 funções combinacionais, conforme Tabela 6, e aqui já esta com o valor 684, sendo o valor 171 da Tabela 7 multiplicado por 4, que são as quatro instâncias, um em cada *Datapath*. O *switch* ocupou um total de 21812 elementos de lógica combinacional do FPGA. Isso dá um total de aproximadamente 66% de área ocupada no FPGA, que tem 33216 elementos lógicos disponíveis.

O *switch* por si só utiliza 4% disso na lógica de interligação de Unidades de Controle e operacionais entre si. Além disso, os multiplexadores (Mux) usam 5% dos elementos lógicos apenas para *debug* do protótipo. Estima-se que o total utilizado para sinais de debug seja de 8% contando que cada componente exporta um vetor de 4 bits para indicação da sua máquina de estados mais os 26 leds. A Unidade de controle fica com cerca de 3% do total. Os *Datapaths*, juntos, somam cerca de 67% dos elementos lógicos e a Lookup Table 9%. Os *AckBitmapAbstractions* usam cerca de 15%.

Como exposto anteriormente, todos os componentes foram prototipados utilizando elementos lógicos do FPGA. O que se tem de entrada e saída no sistema é estritamente para lidar com o controle da interface de rede e com *debug*. Para a prototipagem aplicada a um ambiente real de produção, a primeira medida de melhoria seria colocar os componentes de memória *LookupTable* e *Queues* em chips de memória, por exemplo, do tipo *TCAM*. Nessa migração, 28% de elementos lógicos seriam liberados.

Também, os *AckBitmapAbstractions* não precisam ficar no chip dos *Datapaths*, já que o desempenho de sincronizar as respostas depende muito mais da conversação distribuída dos componentes que da própria FSM. Como será visto na Seção 6.4, o tempo que um

quadro gasta no *PHY Abstraction Layer* está ordens de grandeza maior que o no *AckBitmapAbstractions*. Se eles também ficam como co-processamento, libera-se mais 15% do FPGA. Somando, então, com a liberação das memórias, será uma economia de 43% de elementos lógicos.

O controlador de rede também pode ir pra um chip atuando como co-processador e controlador de entrada e saída. Então, fala-se de mais 7% de elementos sendo liberados. Assim, somando-se todas as partes que poderiam ser retiradas do chip e classificadas como memória ou entrada e saída, há uma redução de 35% do total de elementos lógicos que podem ser usados para expansão do processamento,i.e., aumentar a largura dos barramentos e aumentar o número de Datapaths. Esses valores são aproximados, pois ao remover um componente toda a otimização na compilação é afetada.

Nesta mesma linha, observa-se a Figura 37 que trata do consumo de registradores e a discussão segue os mesmos padrões. Os que mais usam são os que ficam fora do chip do processamento principal, aqui seriam aproximadamente 42% de recursos liberados quando redistribuídas as memórias e controlador PHY.

Não se pode deixar de destacar que o FPGA Cyclone II é de baixo custo e baixa capacidade computacional se comparados aos FPGAs de famílias superiores tanto do mesmo fabricante como de outros. Então, os dados apresentados até aqui são mais importantes quando comparados em conjunto, em suas relações de proporcionalidade do que os valores absolutos em si. Projetos para um ambiente real de produção devem considerar FPGAs mais robustos e capazes de operar a frequências mais altas.

6.3 Considerações de Frequência

Outra discussão que se deve ter acerca da operação de equipamentos de rede é a velocidade de operação que tal equipamento pode ser submetido. É importante salientar no caso deste trabalho que o protótipo construído objetiva principalmente a prova de conceito e, apesar de a codificação considerar métodos que levem ao melhoramento de desempenho aproveitando o paralelismo do hardware, ele não foi codificado obedecendo restrições de tempo e frequência neste momento.

A utilização de um FPGA de baixo custo se aplica neste contexto de prova de conceito, conforme o objetivo deste trabalho, mas não é indicado em ambiente de operação massiva para altas velocidades. Pela Figura 38, pode-se observar os valores limites de frequência que cada componente é capaz de operar neste FPGA e os valores que o *switch* deve obedecer para garantir a consistência dos dados. As barras em azul são dos componentes individualmente. Elas foram extraídas do processo de compilação e síntese de cada componente individualmente. Já as barras em vermelho e verde se referem aos limites do *switch* operando com todos os componentes interligados.

Individualmente, a *Lookup Table* foi quem limitou a frequência por baixo. O valor

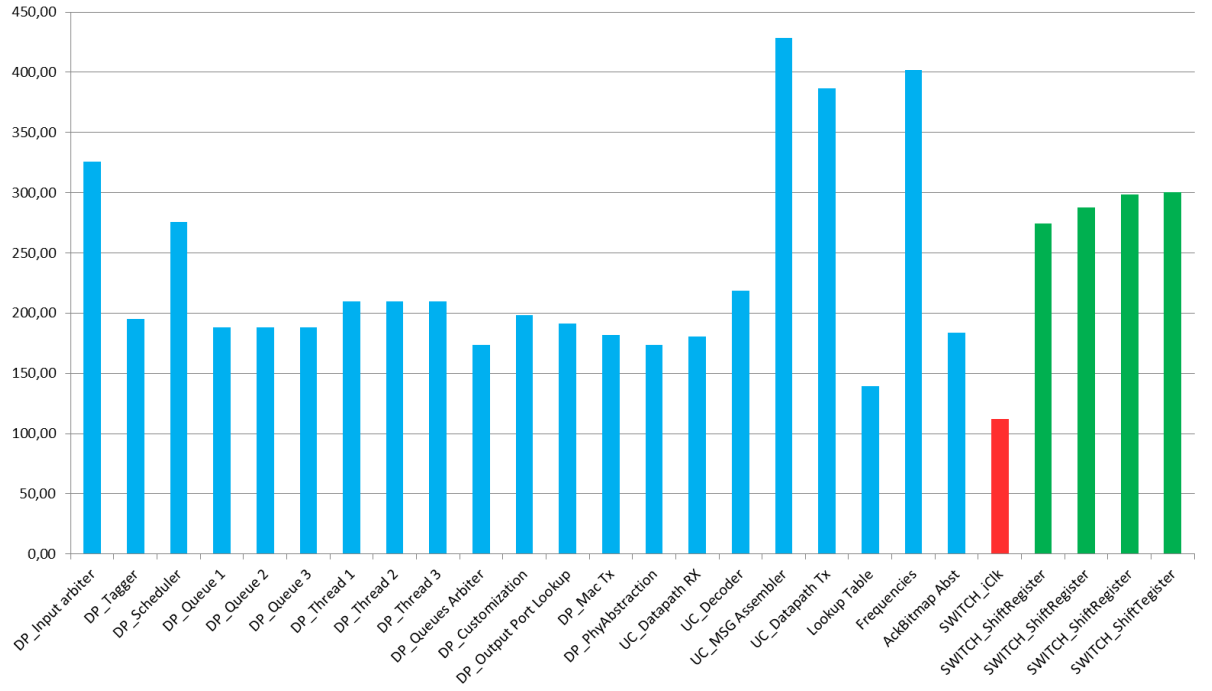


Figura 38: Frequência máxima de operação por elemento de hardware

máximo que ela é capaz de operar é de 139,08 MHz. Dentro do *Datapath*, o *Queues Arbiter* (osquestrador da saída do escalonador) é que apresentou ser o mais lento, operando a 173,37 MHz. Quando se trata do *switch*, cinco valores de frequência foram reportados. Os quatro últimos, entre 274,05 MHz e 300,66 MHz são referentes a um bloco chamado *Shift Register*, que é um componente que fica dentro do controlador da interface de rede, implementado para ser o controlador de recebimento de quadros.

A limitação do *switch* ocorre mesmo na rede de distribuição do *iClk*, que é, nos códigos VHDL, o sinal que distribui o clock referência para o sistema inteiro. Então, a operação fica limitada a uma frequência de 111,92 MHz.

Neste sentido, a Figura 39 traz também considerações importantes. Ela mostra os caminhos no FPGA onde estão os maiores atrasos de propagação de um elemento lógico a outro. Observando os caminhos criados, nota-se que quase todos eles são relacionados também aos componentes que devem ficar em outros chips, a *Lookup Table* e o Controlador PHY. Apenas onde as barras estão em amarelo, que são quatro entradas, é que o atraso ocorre em componente dentro do *Datapath*, o *Mac Tx*. Ainda assim, é em um sinal utilizado apenas para *debug* que atrasa a transição da FSM para que determinada ação seja percebida a olho nu e apenas nos bits 30 e 31 deste sinal (que é de 32 bits), que não são lidos na prática. Em suma, os atrasos críticos ocorrem apenas fora do *Datapath*.

O Cyclone II FPGA não é indicado especificamente para aplicações em rede justamente por não atingir altas taxas de transferência e ter baixos valores para a rede de propagação de clock para aplicações sequenciais em redes de alta velocidade. Se comparado, por

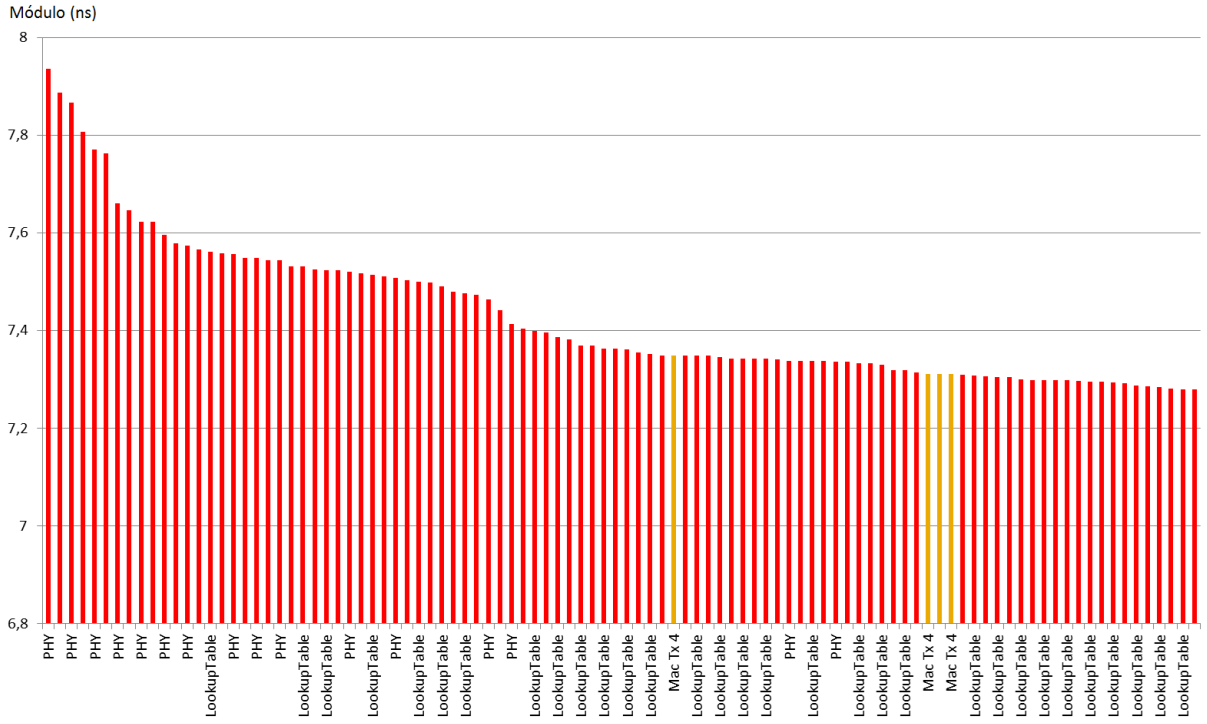


Figura 39: Atrasos na rede de distribuição de Clock

exemplo a um FPGA da família Stratix (Altera) ou Virtex (Xilinx), sua capacidade em recursos lógicos chega a ser menos de 20% no número de unidades programáveis.

Um Virtex 7 (XILINX, 2018), por exemplo, utiliza *trasnceivers* de alta velocidade que permitem comunicações de 10G-400Gbps com interfaces seriais. Com um FPGA desta família, os problemas apresentados neste trabalho não ocorreriam. A frequência nos elementos lógicos do Cyclone II não passa de 450 MHz de acordo com aplicações trazidas no *datasheet*. Assim, usar o Cyclone II é aplicável para prova de conceito mas não para assumir discussões sobre desempenho em redes.

6.4 Considerações de Tempo

Propor um novo método de acesso ao meio com modificações no nível do enlace reflete diretamente em diferentes implementações no hardware, pois, como exposto na Seção 2.1.4, é neste nível que o hardware e o software se encontram nas implementações para as camadas das redes.

Ao analisar novamente a Figura 17, observa-se que a estrutura de interconexão e escalonamento de quadros passa por mais etapas de processamento do que os *switches* Ethernet. É importante, então, fazer uma análise do custo computacional do processamento e discutir se as implementações são viáveis para as taxas de transmissão das redes atuais.

Embora a prototipagem tenha sido realizada em um hardware de baixa velocidade, al-

guns aspectos do nível lógico digital permanecem inalterados e independentes da frequência de referência que o circuito é submetido. Quando se analisa as máquinas de estado implementadas por cada componente do nível digital, consegue-se contabilizar, em termos de ciclos de clock, o tempo computacional requerido pelo processamento em cada *Datapath*. O primeiro passo é analisar os dados trazidos pela Figura 40.

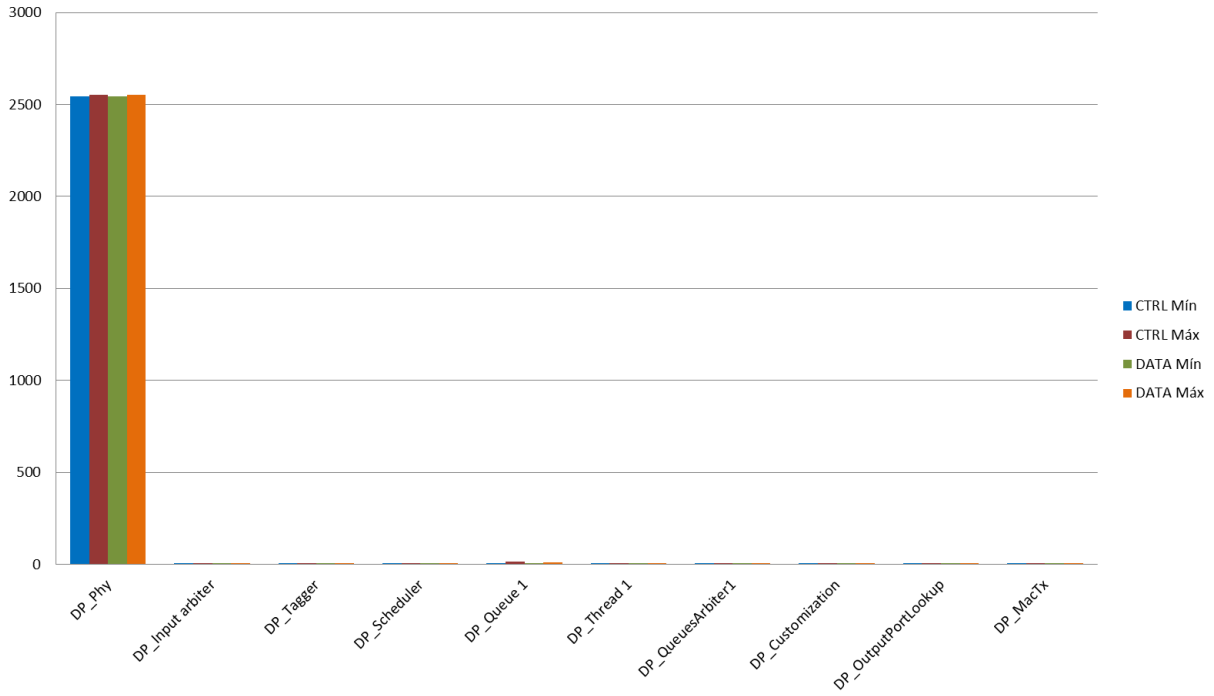


Figura 40: Ciclos de Clock por componente do *Datapath*

A Figura 40 mostra o total de ciclos de clock que o quadro permanece em cada componente do *Datapath*. São apresentadas quatro medidas, que são os valores mínimo e máximo de ciclos de clock que um quadro de dados e um de controle permanecem sendo processados em cada componente. Em cada componente a variação é muito discreta, em alguns casos chega a ser irrelevante. O que se destaca neste gráfico é o fato de o quadro ficar mais de 2500 ciclos na interface de rede desde que chegou até ser transmitido, enquanto os outro valores ficam algumas ordens de grandeza menores. Isso indica claramente que o gargalo de processamento está nas interfaces de rede, que são dependentes da implementação e não do projeto do *switch*. Cabe lembrar, que os chips Waveshare (Figura 9) operam a uma frequência de 10 MHz. No seu controlador, a frequência é de 5 MHz porque as transmissões ocorrem em *dibits*. Isso torna o tempo de transmissão muito alto se comparado a qualquer rede operando atualmente. Então, para voltar a atenção ao controle do aceso ao meio em si, as Figuras 41 e 42 analisam estes tempos sem o PHY.

Os ciclos de clock para um quadro de controle são vistos na Figura 41. Percebe-se que praticamente não há variação entre o número mínimo e o número máximo de ciclos em cada componente, exceto por aqueles no escalonador, *Queue* e *Thread*. No caso da

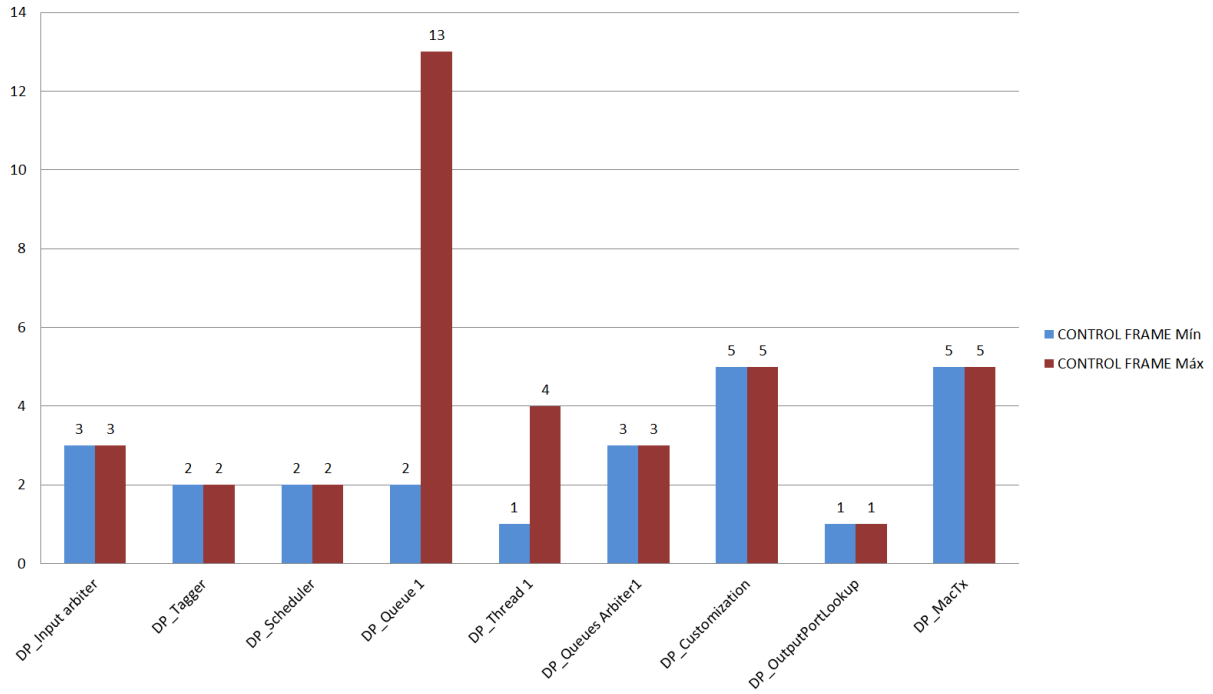


Figura 41: Ciclos de Clock por componente para quadro de controle

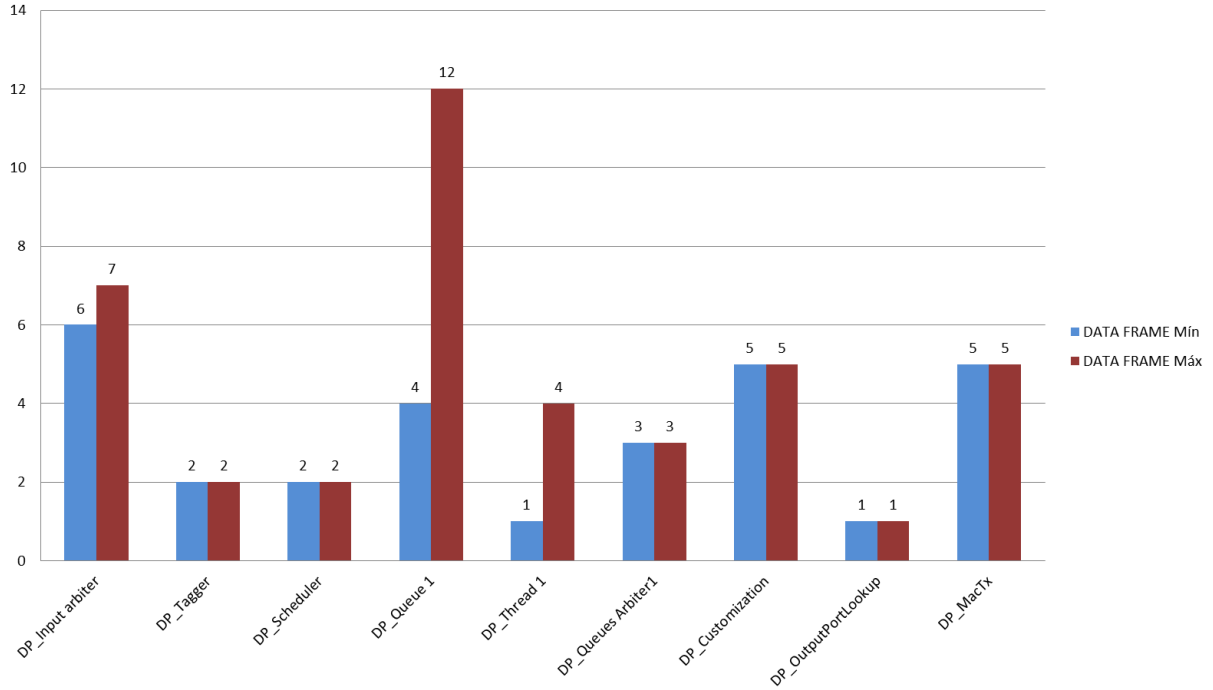


Figura 42: Ciclos de Clock por componente para quadro de dados

Queue, o gráfico indica o intervalo entre o momento em que o quadro chegou na entrada da *Fifo* e foi lido pela *Thread*. O intervalo é variável pois já que há divisor de frequência na *Thread*, às vezes ela confere a *Fifo* em menos que o tempo total da divisão. Deve-se

somar a isso o tempo operacional de transferência do quadro, que é de 2 ciclos + 2 de atrasos. O máximo pode ir até 14 ciclos utilizando um divisor de frequência de fator 10 aqui (para atingir 10MHz). Então, depende de o quanto a frequência será dividida.

Já no caso da *Thread*, foi percebido o mínimo de 1 ciclo e o máximo de 5 ciclos. Este é o tempo entre o quadro chegar na *Thread* e ser recebida pelo árbitro. Como o árbitro funciona como uma fila circular, para uma arquitetura com 3 *Datapaths*, o valor máximo será mesmo de 5 ciclos, que é 1 ciclo para transferência mais até 4 ciclos para rodar a fila.

Quando se observa a Figura 42, referente aos ciclos gastos por quadro de *Workspace* de dados, percebe-se que houve alteração também no *Input Arbiter*. Isso é esperado já que ele depende da confirmação do *Tagger* que, por sua vez, depende da busca do *Bitmap* do *Workspace* lá na *Lookup Table* e os endereços de controle e *bootstrapping* ficam em registradores especiais.

Outro ponto que merece destaque é o *Output Port Lookup* que, apesar de não ter apresentado variação, pode apresentar devido também ao tempo de busca na tabela de *Workspaces*. A busca começa já no primeiro ciclo quando o quadro chega ao componente e, enquanto a FSM continua a evoluir, a busca ocorre em paralelo. Então, neste caso, para poucas entradas na tabela, não foi possível perceber diferença. Este é um ponto clássico do roteamento, a busca nas tabelas. Ocorre que neste caso há uma pré-busca, pois o dado é passado antes de chegar no componente que o utiliza. Também o componente *Mac Tx* tem tempo fixo e que é da ordem de $O(n)$, especificamente $N + 1$ ciclos, onde N é o número de *Datapaths*, já que o organizador pode gastar até N ciclos de clock para selecionar qual o frame deve ser transmitido naquele momento.

Interfaces que operam a 10Mbps não são rápidas o suficiente para operar em rede atuais. Pelos padrões de mercado, até mesmo os computadores pessoais operam com placas com taxa 1000 Mbps. Quando se trata dos esquipamentos do núcleo da rede, estas taxas são na casa das dezenas ou centenas de Gbps. Assim, embora a prototipagem tenha se dado em um FPGA é importante fazer uma prospecção de requisitos e capacidades do *switch* quando colocado em operação em redes Gbps. A análise é feita com transmissões do tamanho de quadros Ethernet que é o padrão *de facto* de mercado e, para manter a compatibilidade com as redes atuais, o ETArch utiliza o mesmo tamanho de quadro (não formato, mas apenas comprimento da sequência de bits).

Numa rede operando a 1Gbps a taxa de transferência é de $10^9 bits/seg$. Um quadro no formato Ethernet tem entre 64 e 1536 Bytes, ou seja, entre 512 e 12288 bits. No caso máximo, com 12288 bits mais 96bits de *Interpacket Gap (IPG)*, por segundo podem ser transferidos

$$10^9/(12288 + 96) = 80749,35 frame/seg \tag{1}$$

No *switch* são blocos de 64b por fragmento, logo

$$12288/64 = 192 fragmento/frame \tag{2}$$

$$80749,35 frame/seg * 192 fragmento/frame = 15503876 frag/sec \quad (3)$$

Cada fragmento precisa de 20 ciclos, logo:

$$15503876 fragmento/seg * 20 ciclos/fragmento = 310077519 ciclos/seg \quad (4)$$

O período tem que ser no máximo 3,2 ns e a frequência maior maior que 312,5 MHz
Com um *switch* de 512 bits de largura de barramentos de dados:

$$10^9/(12288 + 96) = 80749,35 frame/seg \quad (5)$$

Blocos de 512 bits no *switch*, logo:

$$12288/512 = 24 fragmento/frame \quad (6)$$

$$80749,35 frame/seg * 24 fragmento/frame = 1937984 fragmento/seg \quad (7)$$

Cada fragmento precisa de 20 ciclos:

$$1937984 fragmento/seg * 20 ciclos/fragmwnto = 38759688 ciclos/seg \quad (8)$$

O período tem que ser no máximo 25,8 ns e a frequência maior que 38,8 MHz. O que indica que o *switch* implementado com uma largura de 512 bits em um FPGA da família Cyclone II operaria em redes na escala de 10 Gbps se ele tivesse capacidade de operar com *transceivers* nessa velocidade. Considerando que o ganho do pipeline foi desconsiderado neste cálculo, essa velocidade seria ainda maior.

Então, prototipando em um FPGA da família do Virtex 7, que possui capacidade de recursos lógicos 80% maior que o hardware aqui utilizado e uma taxa de transferência com interfaces seriais a 10G-400G, o circuito fica operacional em redes da escala de centenas de Gbps. O fato de precisar de cerca de 20 ciclos para percorrer o pipeline de dados se torna desconsiderável, pois de fato ocorre a propagação do quadro com paralelismo no nível de instrução e, quando o quadro passa o próximo bloco, o anterior já é buscado para ser executado. Quem vai limitar a operação é a capacidade de transmissão com o bloco de entrada e saída que, como foi visto nos gráficos, é o componente que consome mais tempo dentro do sistema.

Capítulo 7

Conclusão

Este trabalho apresentou a especificação de um *switch* para a *Entity Title Architecture*, sua implementação em linguagem de descrição de hardware com prototipagem em FPGA e a especificação de um protocolo de configuração que padroniza a comunicação DTSA/*switch*.

O *switch* apresenta novo método de acesso ao meio, diferente da subcamada de controle de acesso ao meio do Ethernet, que permite que *Workspaces* sejam criados, modificados e excluídos das tabelas do *switch* em tempo de execução. A programabilidade é conseguida com o processamento de um *Bitmap* que seleciona blocos de hardware do plano físico e não com nova síntese do hardware.

O protocolo ETSCP permite que o DTSA se comunique com o *switch* sob seu controle para operar com funções relativas ao gerenciamento da rede. As mensagens de controle são enviadas e encapsuladas diretamente nos quadros da camada de *Link* e têm formato com 512 bits, acompanhando o tamanho mínimo de transmissão pelo padrão IEEE 802.3, de forma a manter a compatibilidade com as interfaces de rede comerciais.

Embora o tamanho do quadro seja o mesmo que na Ethernet, o formato é diferente e implementa o Modelo de Endereçamento por Títulos. O protocolo traz em seu vocabulário oito mensagens correspondentes aos serviços de criação, modificação e exclusão de *Workspaces*, além da mudança dos parâmetros de divisão de frequência dos escalonadores. O vocabulário pode se expandir até 255 mensagens devido aos 8 bits reservados para o campo de tipo de mensagem.

Foram detalhadas a arquitetura e a organização de um *switch* para suporte a arquitetura de Internet do Futuro. A arquitetura se baseia fundamentalmente em uma nova proposta de método de acesso ao meio que permite parametrizar e configurar fluxos baseados em *Workspaces* diretamente na subcamada de acesso ao meio e com parametrização direta no hardware sem prejuízo à execucão, i.e., com tempo zero de indisponibilidade da rede.

A arquitetura apresenta uma estrutura de escalonamento paralelo de forma que se pode escolher qual a frequência de varredura em que os quadros serão encaminhados de

acordo com as necessidades de cada *Workspace*, que refletem diretamente os acordos de serviço tratados no DTSA.

O *switch* está prototipado através de uma unidade de controle, um banco de registradores, uma memória para tabela de *Wokspaces* e quatro *Datapaths* no pipeline de dados. O número de *Datapaths* que o *switch* é capaz de lidar depende da quantidade de recursos lógicos disponíveis no hardware utilizado para síntese. Sua estrutura de encaminhamento de dados, já na saída, conta com uma estrutura que conecta todos os *Output Port Lookups* entre si, em um esquema com interconectividade NxN.

É disponibilizado um mecanismo que permite a criação de planos virtuais para a implementação de diferentes *Workspaces* em um mesmo *Datapath* através de um *Bitmap* que é anexado ao quadro e lido em cada componente do hardware para escolher que tipo de tratamento aquele quadro deve receber. É exatamente este recurso que permite que vários *Workspaces* coexistam em uma mesma porta e que um mesmo *Workspace* possa existir em mais de uma porta.

O protótipo ocupa 66% da área de um FPGA de baixo custo, o Altera Cyclone II. Com melhorias no sistema de memória e interface com o enlace físico, mais de 40% destes recursos podem ser realocados ao migrar blocos lógicos de memória para chips de memória externa. A adoção de um FPGA para aplicações de alto desempenho permite prototipagem com largura de 512 bits e coloca o *switch* apto a operar em redes de 10-400 Gbps.

Algumas limitações importantes foram encontradas ao longo do desenvolvimento da pesquisa e merecem destaque: a taxa de transferência dos *transceivers* RJ 45 é muito baixa se comparada às redes Gigabit; o gerenciamento direto em hardware destas mesmas interfaces apresentou diversos entraves para manuseio da *media independent interface*; a taxa de erros também é muito alta no cenário de validação; o FPGA disponível para prototipagem é expressivamente limitado para a prototipagem de um *switch* e, do ponto de vista da implementação, a sincronização dos *Datapaths* apresentou grande complexidade para sua codificação. A título informativo, o código fonte está estruturado em 159 blocos de hardware escritos em 10k linhas de VHDL.

Para dar continuidade a esta pesquisa de forma a encontrar resultados ainda mais promissores, sugere-se, para desenvolvimentos futuros:

- ❏ Adicionar requisitos de tempo real no controle de acesso ao meio;
- ❏ Investigação de técnicas de aprendizado de máquinas incluídas no *switch*;
- ❏ Utilização de controle de erro e interfaces de transmissão que operam com taxas de transferência de Gbps, com protótipo em FPGA de alto desempenho, para testes em ambiente real com aferição de taxa de erros;

- ❏ Migrar parte do controle para um coprocessador, no nível do sistema operacional embarcado, bem como transferir o banco de registradores e a tabela de *Workspaces* para memórias externas;
- ❏ Inclusão no controlador do DTSA um *Resource Adapter* que controle a interface com o *switch* já se comunicando através do protocolo ETSCP;
- ❏ Inclusão de módulos compatíveis com arquitetura TCP/IP para interoperabilididade do equipamento com a rede atual; e
- ❏ Investigar como implementar a comunicação com alguma linguagem de representação ontológica.

Referências

AHLGREN, B. et al. A node identity internetworking architecture. In: **Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications**. [s.n.], 2006. p. 1–6. ISSN 0743-166X. Disponível em: <https://doi.org/10.1109/INFOCOM.2006.51>.

ALBERTI, A. M. A conceptual-driven survey on future internet requirements, technologies, and challenges. **Journal of the Brazilian Computer Society**, v. 19, n. 3, p. 291–311, Sep 2013. ISSN 1678-4804. Disponível em: <http://dx.doi.org/10.1007/s13173-013-0101-2>.

ALTERA. **DE2 Development and Education Board User Manual**. 2006. URL: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=183&No=30&PartNo=2>. Acesso em: 05.28.2019.

ANAND, A. et al. XIA: An architecture for an evolvable and trustworthy internet. In: **Proceedings of the tenth ACM Workshop on Hot Topics in Networks HotNets-X**. Cambridge, MA. USA.: [s.n.], 2011. Disponível em: <https://doi.org/10.1145/2070562.2070564>.

ANDERSON, T. et al. The nebula future internet architecture. In: _____. **The Future Internet: Future Internet Assembly 2013: Validated Results and New Horizons**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 16–26. ISBN 978-3-642-38082-2. Disponível em: <http://dx.doi.org/10.1007/978-3-642-38082-2_2>.

ANWER, M. B. et al. Switchblade: A platform for rapid deployment of network protocols on programmable hardware. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 40, n. 4, p. 183–194, Aug 2010. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1851275.1851206>.

ASHENDEN, P. **Digital Design - An Embedded Systems Approach Using VHDL**. [S.l.]: Morgan Kaufmann, 2007. ISBN 9780123695284.

BHATNAGAR, V. et al. An fpga software defined radio platform with a high-level synthesis design flow. In: **2013 IEEE 77th Vehicular Technology Conference (VTC Spring)**. [s.n.], 2013. p. 1–5. ISSN 1550-2252. Disponível em: <https://doi.org/10.1109/VTCSpring.2013.6691879>.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 3, p. 87–95, 2014. Disponível em: <http://doi.acm.org/10.1145/2656877.2656890>.

_____. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In: **Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM**. New York, NY, USA: ACM, 2013. (SIGCOMM '13), p. 99–110. ISBN 978-1-4503-2056-6. Disponível em: <http://doi.acm.org/10.1145/2486001.2486011>.

BREBNER, G. Programmable hardware for software defined networks. In: **2015 European Conference on Optical Communication (ECOC)**. [s.n.], 2015. p. 1–3. Disponível em: <https://doi.org/10.1109/ECOC.2015.7341956>.

CAMPBELL, A. T. et al. A survey of programmable networks. **ACM SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 29, n. 2, p. 7–23, apr 1999. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/505733.505735>.

CASADO, M. et al. Ethane: Taking control of the enterprise. **ACM SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 37, n. 4, p. 1–12, ago. 2007. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1282427.1282382>.

CISCO. **IP MultiLayer Switching Sample Configuration**. 2007. URL: <https://goo.gl/pAuTRZ>. Acesso em: 05.28.2019.

_____. **CAM (Content Addressable Memory) VS TCAM (Ternary Content Addressable Memory)**. 2017. URL: <https://goo.gl/orPSen>. Acesso em: 05.28.2019.

_____. **Switches**. 2017. URL: <http://www.cisco.com/c/pt_br/products/switches/index.html?stickynav=1>. Acesso em: 05.28.2019.

_____. **First Hop Redundancy Protocol (FHRP)**. 2018. URL: <https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipapp_fhrp/configuration/xe-16/fhp-xe-16-book/fhp-hsrp-mgo.html>. Acesso em: 28.05.2019.

CORSA. **SDN Done Right. For the Physical Network.** 2019. URL: <https://www.corsa.com/why-corsa-dataplanes/>. Acesso em: 05.28.2019.

DAY, J.; MATTA, I.; MATTAR, K. Networking is ipc: A guiding principle to a better internet. In: **Proceedings of the 2008 ACM CoNEXT Conference**. New York, NY, USA: ACM, 2008. (CoNEXT '08), p. 67:1–67:6. ISBN 978-1-60558-210-8. Disponível em: <http://doi.acm.org/10.1145/1544012.1544079>.

DIGITAL; INTEL; XEROX. **The Ethernet - A Local Area Network**. 1980. (Specification).

Einsiedler, H. J. et al. A new meta-model for future internet architectures. In: **2013 Future Network Mobile Summit**. [S.l.: s.n.], 2013. p. 1–11.

HAUCK, S.; DEHON, A. **Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation**. [S.l.]: Elsevier Science, 2010. (Systems on Silicon). ISBN 9780080556017.

HOLZMANN, G. J. **Design and validation of computer protocols**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1991. ISBN 0-13-539925-4.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. **Introduction to Automata Theory, Languages, and Computation**. 2nd. ed. Addison Wesley, 2000. Disponível em: <https://doi.org/10.1145/568438.568455>.

IEEE. Ieee standard for local and metropolitan area networks: Overview and architecture. **IEEE Standard 802-2001**, Feb 2001.

_____. Ieee standard for local and metropolitan area networks–media access control (mac) bridges and virtual bridged local area networks. **IEEE Std 802.1Q-2011 (Revision of IEEE Std 802.1Q-2005)**, p. 1–1365, Aug 2011.

_____. Ieee standard for ethernet. **IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)**, p. 1–5600, Aug 2018.

IETF. **User Datagram Protocol**. [S.l.]: IETF, 1980. URL <https://tools.ietf.org/html/rfc768>. (Request for Comments, 768).

_____. **Internet Protocol - DARPA Internet Program Protocol Specification**. [S.l.]: IETF, 1981. URL <https://tools.ietf.org/html/rfc791>. (Request for Comments, 791).

_____. **Transmission Control Protocol - DARPA Internet Program Protocol Specification**. [S.l.]: IETF, 1981. URL <https://tools.ietf.org/html/rfc793>. (Request for Comments, 793).

_____. **Host Identity Protocol**. [S.l.]: IETF, 2008. URL <https://tools.ietf.org/html/rfc5201>. (Request for Comments, 5201).

_____. **Unified Layer 2 (L2) Abstractions for Layer 3 (L3)-Driven Fast Handover. RFC 5184**. [S.l.]: IETF, 2008. URL: <http://tools.ietf.org/html/rfc5184>. (Request for Comments, 5184).

_____. **The Locator/ID Separation Protocol (LISP)**. [S.l.]: IETF, 2013. URL <https://tools.ietf.org/html/rfc6830>. (Request for Comments, 6830).

_____. **Software-Defined Networking (SDN): Layers and Architecture Terminologyl**. [S.l.]: IETF, 2015. URL <https://tools.ietf.org/html/rfc7426#ref-OF08>. (Request for Comments, 7426).

ISO. **Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model.** Geneva, Switzerland, 1994. Disponível em: <https://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip>.

ITU. **Key ICT indicators for developed and developing countries and the world (totals and penetration rates)**. 2018. URL: <http://www.itu.int/ITU-D/ict/statistics/index.html>. Acesso em: 05.28.2019.

JOSE, L. et al. Compiling packet programs to reconfigurable switches. In: **Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2015. (NSDI'15), p. 103–115. ISBN 978-1-931971-218. Disponível em: <http://dl.acm.org/citation.cfm?id=2789770.2789778>.

KALYAEV, A.; MELNIK, E. Fpga-based approach for organization of sdn switch. In: **2015 9th International Conference on Application of Information and Communication Technologies (AICT)**. [s.n.], 2015. p. 363–366. Disponível em: <https://doi.org/10.1109/ICAICT.2015.7338580>.

KERLING, P. **Design, Implementation, and Test of a Tri-Mode Ethernet MAC on an FPGA**. Tese (Doutorado) — technische universität ilmenau, Fakultät für Elektrotechnik und Informationstechnik, Ilmenau, Alemanha, 2018. Disponível em: <https://www.db-thueringen.de/receive/dbt_mods_00038245>. Acesso em: 5.26.2019.

KOHLER, E. et al. The click modular router. **ACM Trans. Comput. Syst.**, ACM, New York, NY, USA, v. 18, n. 3, p. 263–297, 2000. ISSN 0734-2071. Disponível em: <http://doi.acm.org/10.1145/354871.354874>.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a Internet: uma abordagem top-down**. [S.l.]: Pearson Addison Wesley, 2006. ISBN 9788588639188.

LABRECQUE, M. et al. Netthreads: Programming netfpga with threaded software. In: **In: NetFPGA Dev. Workshop'09. 2009**. [S.l.: s.n.], 2009.

LACY, L. **Owl: Representing Information Using the Web Ontology Language**. [S.l.]: Trafford, 2005. ISBN 9781412034487.

LAKATOS, E. M.; MARCONI, M. A. **Fundamentos de Metodologia Científica**. 5. ed. São Paulo, SP, Brasil: Atlas, 2003. ISBN 8522433976.

MARTIN, D.; VöLKER, L.; ZITTERBART, M. A flexible framework for future internet design, assessment, and operation. **Computer Networks: The International Journal of Computer and Telecommunications Networking.**, Elsevier North-Holland, Inc., New York, NY, USA, v. 55, n. 4, p. 910–918, mar. 2011. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2010.12.015>.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 2, n. 38, p. 69–74, 2008. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>.

MÜLLER, P.; REUTHER, B. Future internet architecture-a service oriented approach. **IT-Information Technology**, v. 50, n. 6, p. 383–389, 2009. Disponível em: <https://doi.org/10.1524/itit.2008.0510>.

NETO, N. V. de S. et al. Control plane routing protocol for the entity title architecture: Design and specification. In: **Proceedings of the ICN 2015 : The Fourteenth International Conference on Networks**. [S.l.: s.n.], 2015. p. 185 –190.

NIKANDER, P. et al. Towards software-defined silicon: Experiences in compiling click to netfpga. In: **In: 1st European NetFPGA Developers Workshop, Cambridge, UK. 2010**. [S.l.: s.n.], 2010.

OLIVEIRA, R.; MESQUITA, D.; ROSA, P. Harp: A split brain free protocol implemented in fpga. In: **Proceedings of the Ninth Advanced International Conference on Telecommunications, AICT 2013**. [S.l.: s.n.], 2013. v. 9, p. 197–203. ISSN 978-1-61208-279-0.

OLIVEIRA., R. D. et al. Workspace-based virtual networks: A clean slate approach to slicing cloud networks. In: INSTICC. **Proceedings of the 9th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,**. SciTePress, 2019. p. 464–470. ISBN 978-989-758-365-0. Disponível em: <https://doi.org/10.5220/0007753104640470>.

ONF. **Software-Defined Networking: The New Norm for Networks**. 2012. URL: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>. Acesso em: 05.28.2019.

_____. **OpenFlow-enabled Transport SDN**. [S.l.]: Open Networking Foundation, 2014. URL <https://goo.gl/KzwYWV>. (ONF Solution Brief).

_____. **OpenFlow Switch Specification. Version 1.5.1 (Protocol version 0x06)**. 2015. URL: <https://www.opennetworking.org/software-defined-standards/specifications/>. Acesso em: 05.28.2019.

_____. **BIR - B Intermediate Representation**. 2017. URL: <https://goo.gl/wjzSBF>. Acesso em: 28.05.2019.

_____. **Protocol Independent Forwarding**. 2017. URL: <http://opensourcesdn.org/projects/pif-open_source_intermediate_representation_for_datapaths/>. Acesso em: 28.05.2019.

_____. **Software-Defined Networking (SDN) Definition**. 2017. URL: <https://www.opennetworking.org/sdn-resources/sdn-definition>. Acesso em: 05.28.2019.

Osama, H.; Omar, Y.; Badr, A. Mapping sorting algorithm. In: **2016 SAI Computing Conference (SAI)**. [s.n.], 2016. p. 488–491. Disponível em: <https://doi.org/10.1109/SAI.2016.7556025>.

P4.ORG. **P4**. 2017. URL: <http://p4.org/>. Acesso em: 07.07.2017.

PACíFICO, R. D. G. et al. Roteador sdn em hardware independente de protocolo com análise, casamento e ações dinâmicas. In: **Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. Porto Alegre, RS, Brasil: SBC, 2018. ISSN 2177-9384. Disponível em: <https://portaldeconteudo.sbc.org.br/index.php/sbrc/article/view/2431>.

PAN, J.; PAUL, S.; JAIN, R. A survey of the research on future internet architectures. **IEEE Communications Magazine**, v. 49, n. 7, p. 26–36, July 2011. ISSN 0163-6804. Disponível em: <https://doi.org/10.1109/MCOM.2011.5936152>.

PATTERSON, D. A.; HENNESSY, J. L. **Computer Organization and Design, Fifth Edition: The Hardware/Software Interface**. 5th. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013. ISBN 0124077269, 9780124077263.

PEREIRA, J. d. S. et al. Title model ontology for future internet networks. In: DOMINGUE, J. et al. (Ed.). **The Future Internet**. Springer Berlin - Heidelberg, 2011, (Lecture Notes in Computer Science, v. 6656). p. 103–114. ISBN 978-3-642-20897-3. Disponível em: <https://doi.org/10.1007/978-3-642-20898-0_8>.

PEREIRA, J. H. de S. **Modelo de título para a próxima geração de Internet**. Tese (Doutorado) — Universidade de São Paulo, São Paulo, SP, 2012. Disponível em: <doi.org/10.11606/T.3.2012.tde-13062013-113304>.

PMI, P. M. I. **PMBOK Guide**. 5. ed. [S.l.: s.n.], 2013. PMI Standard.

RADATZ, J. **The IEEE Standard Dictionary of Electrical and Electronics Terms**. 6th. ed. New York, NY, USA: IEEE Standards Office, 1997. ISBN 1559378336.

RECH, J. **Ethernet: Technologien und Protokolle für die Computervernetzung**. [S.l.]: Heise Zeitschriften Vlg G, 2014. ISBN 978-3-944099-04-0.

REXFORD, J.; DOVROLIS, C. Future internet architecture: Clean-slate versus evolutionary research. **Commun. ACM**, ACM, New York, NY, USA, v. 53, n. 9, p. 36–40, set. 2010. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1810891.1810906>.

SESKAR, I. et al. Mobilityfirst future internet architecture project. In: **Proceedings of the 7th Asian Internet Engineering Conference**. New York, NY, USA: ACM, 2011. (AINTEC '11), p. 1–3. ISBN 978-1-4503-1062-8. Disponível em: <http://doi.acm.org/10.1145/2089016.2089017>.

SEZER, S. et al. Are we ready for sdn? implementation challenges for software-defined networks. **IEEE Communications Magazine**, v. 51, n. 7, p. 36–43, July 2013. ISSN 0163-6804. Disponível em: <https://doi.org/10.1109/MCOM.2013.6553676>.

SHAHBAZ, M. et al. Pisces: A programmable, protocol-independent software switch. In: **Proceedings of the 2016 ACM SIGCOMM Conference**. New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 525–538. ISBN 978-1-4503-4193-6. Disponível em: <http://doi.acm.org/10.1145/2934872.2934886>.

SHARMA, S. et al. Openflow: Meeting carrier-grade recovery requirements. **Comput. Commun.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 36, n. 6, p. 656–665, 2013. ISSN 0140-3664. Disponível em: <http://dx.doi.org/10.1016/j.comcom.2012.09.011>.

SILVA, F. de O. **Endereçamento por título: uma forma de encaminhamento multicast para a próxima geração de redes de computadores**. Tese (Doutorado) — Universidade de São Paulo, Uberlandia, MG, 2013. Disponível em: <doi.org/10.11606/T.3.2013.tde-22092014-111409>.

SILVA, F. de O. et al. On the analysis of multicast traffic over the entity title architecture. In: **2012 18th IEEE International Conference on Networks (ICON)**. [S.l.: s.n.], 2012. p. 30–35. ISSN 1531-2216.

_____. Enabling future internet architecture research and experimentation by using software defined networking. In: **2012 European Workshop on Software Defined Networking**. [S.l.: s.n.], 2012. p. 73–78. ISSN 2379-0350.

SINGH, R. **FPGA vs ASIC: Differences between them and which one to use?** 2018. URL: <https://numato.com/blog/differences-between-fpga-and-asics/>. Acesso em: 05.28.2019.

Song, W. et al. Parallel hardware merge sorter. In: **2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)**. [s.n.], 2016. p. 95–102. Disponível em: <https://doi.org/10.1109/FCCM.2016.34>.

SRIVATS, P. **Ostinato Packet Generator**. 2019. URL: <https://ostinato.org/>. Acesso em: 05.28.2019.

TIOH, J.-W. et al. Reprogrammable high-speed platform : Bridging the gap between research, education and engineering. In: **Proceedings of the IEEE International Conference on Electro/Information Technology, 2008. EIT 2008.** [S.l.: s.n.], 2008. p. 145 –147.

UNNIKRISHNAN, D. C. **Reconfigurable Technologies for Next Generation Internet and Cluster Computing**. Dissertação (Mestrado) — University of Massachusetts, 2013. Disponível em: <http://scholarworks.umass.edu/open_access_dissertations/823/>. Acesso em: 7.7.2017.

WANG, H. et al. P4fpga: A rapid prototyping framework for p4. In: **Proceedings of the Symposium on SDN Research**. New York, NY, USA: ACM, 2017. (SOSR '17), p. 122–135. ISBN 978-1-4503-4947-5. Disponível em: <http://doi.acm.org/10.1145/3050220.3050234>.

WAVESHARE. **DP83848 Ethernet Board**. 2015. URL: <https://www.waveshare.com/dp83848-ethernet-board.htm>. Acesso em: 05.28.2019.

WAZLAWICK, R. S. **Metedologia de Pesquisa para Ciência da Computação**. 2. ed. Rio de Janeiro, RJ, Brasil: Elsevier, 2009. ISBN 9788535266436.

WEBER, R. **Fundamentos de Arquitetura de Computadores - Vol.8: Série Livros Didáticos Informática UFRGS**. [S.l.]: Bookman Editora, 2009. ISBN 9788540701434.

WIRESHARK. **Wireshark**. 2019. URL: <http://www.wireshark.org/>. Acesso em: 05.28.2019.

XILINX. **SDNet**. 2017. URL: <https://www.xilinx.com/products/design-tools/software-zone/sdnet.html>. Acesso em: 05.28.2019.

_____. **7 Series FPGAs Data Sheet: Overview**. 2018. URL: <http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf>. Acesso em: 05.28.2019.

ZHANG, L. et al. **Named Data Networking Tech Report 001**. 2010. (Technical Report).

Apêndices

APÊNDICE **A**

Formato e Vocabulário do ETSCP

Nas próximas páginas, duas figuras tratam do formato do protocolo. A primeira, a Figura 43, relaciona as mensagens do protocolo ETSCP e detalha os valores de cada campo conforme as especificações de formato da Seção 4.2.3. Já a segunda, a Figura 44, detalha como foi o processo de adaptação de formato de 512 para 64 bits em virtude da implementação.

FRAME ETARCH DISSECTION (64 bits)

1. ETARCH FRAME FORMAT

63 62 61 60 59 58 57 56	55 54 53 52 51 50 49 48	47 46 45 44	43 42 41 40	39 38 37 36 35 34 33 32	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
DST_TITLE	SRC_TITLE	LENGHT	MSG_TYPE	DST_ENTITY_TITLE	PAYLOAD	
WKP_TITLE	*BITMAP*				*000 & PORT_BITMAP (4 .. 0)*	CREATE_WKP_REQUEST
WKP_TITLE	*BITMAP*				*000 & PORT_BITMAP (4 .. 0)*	CREATE_WKP_RESPONSE
WKP_TITLE	*BITMAP*				*000 & PORT_BITMAP (4 .. 0)*	EDIT_WKP_REQUEST
WKP_TITLE	*BITMAP*				*000 & PORT_BITMAP (4 .. 0)*	EDIT_WKP_RESPONSE
WKP_TITLE						REMOVE_WKP_REQUEST
WKP_TITLE						REMOVE_WKP_RESPONSE
b1b2b3 & 00000	*F1*	*F2*	*F3*			CHANGE_FREQUENCY_REQUEST
b1b2b3 & 00000	*F1*	*F2*	*F3*			CHANGE_FREQUENCY_RESPONSE

2. WORKSPACE TABLE ENTRY

28 27 26 25 24 23 22 21	20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5	4 3 2 1 0
WKP_TITLE	BITMAP	PORTMAP

2. BITMAP VIEW

79 78 77	76 75 74	73	72	71 70 69 68 67 66 65 64
PIORITY	FREQUENCY	crip	crc	

Figura 43: *ETArch Frame Format*

ADAPTAÇÃODO FRAME ETHERNET 512 bits PARA ETARCH 64 bits NAS TRANSMISSÕES DO SISTEMAS DE DEMONSTRAÇÃO

Campos ETArch

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Ethernet																																																		
DstAdd	511	510	509	508	507	506	505	504	503	502	501	500	499	498	497	496	495	494	493	492	491	490	489	488	487	486	485	484	483	482	481	480	479	478	477	476	475	474	473	472	471	470	469	468	467	466	465	464
	DST_ADDR to WKP_DST_ADD																																																
SrcAdd	463	462	461	460	459	458	457	456	455	454	453	452	451	450	449	448	447	446	445	444	443	442	441	440	439	438	437	436	435	434	433	432	431	430	429	428	427	426	425	424	423	422	421	420	419	418	417	416
	SRC_ADDR to WKP_SRC_ADD																																															
EthType	415	414	413	412	411	410	409	408	407	406	405	404	403	402	401	400																																		
	ETHType to LENGTH (B)																																															
Payload	399	398	397	396	395	394	393	392	391	390	389	388	387	386	385	384																																		
	PAYLOAD to 2B MSG_TYPE																																															
	383	382	381	380	379	378	377	376	375	374	373	372	371	370	369	368	367	366	365	364	363	362	361	360	359	358	357	356	355	354	353	352	351	350	349	348	347	346	345	344	343	342	341	340	339	338	337	336
	PAYLOAD to DST ENTITY TITLE																																															
	335	334	333	332	331	330	329	328	327	326	325	324	323	322	321	320	319	318	317	316	315	314	313	312	311	310	309	308	307	306	305	304	303	302	301	300	299	298	297	296	295	294	293	292	291	290	289	288
	287	286	285	284	283	282	281	280	279	278	277	276	275	274	273	272	271	270	269	268	267	266	265	264	263	262	261	260	259	258	257	256	255	254	253	252	251	250	249	248	247	246	245	244	243	242	241	240
	239	238	237	236	235	234	233	232	231	230	229	228	227	226	225	224	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208	207	206	205	204	203	202	201	200	199	198	197	196	195	194	193	192
	191	190	189	188	187	186	185	184	183	182	181	180	179	178	177	176	175	174	173	172	171	170	169	168	167	166	165	164	163	162	161	160	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144
	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32																																		
	PAYLOAD to PAYLOAD																																															
CRC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
	CRC to CRC																																															

Figura 44: Adaptação de Formato: 512 para 64 bits

APÊNDICE **B**

Máquinas de Estados dos componentes

A seguir são apresentadas as FSM de cada componente conforme implementado. As condições de transições estão explicadas no Capítulo 5.

- Control Unit
  - DatapathRx: Figura 45
  - Decoder: Figura 46
  - MSGAssembler: Figura 47
  - DatapathTx: Figura 48
- Lookup Table: Figura 50
- Frequencies: Figura 51
- Datapath
  - InputArbiter: Figura 52
  - Tagger: Figura 53
  - Scheduler: Figura 54
  - QueuesArbiter: Figura 55
  - Customization: Figura 56
  - OutputPortLookup: Figura 57
  - Sorter: Figura 60
- Ack Bitmap Abstraction: Figura 59

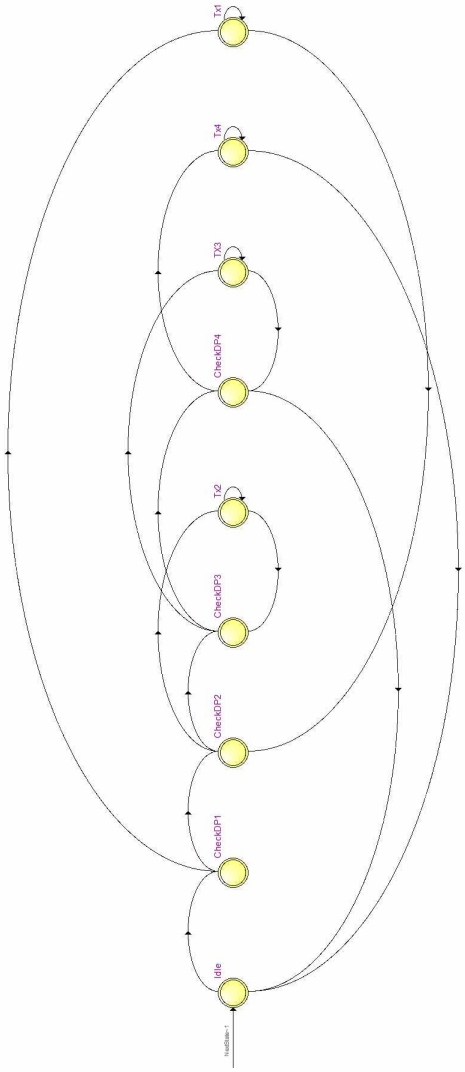


Figura 45: FSM: Control Unit - Datapath Rx

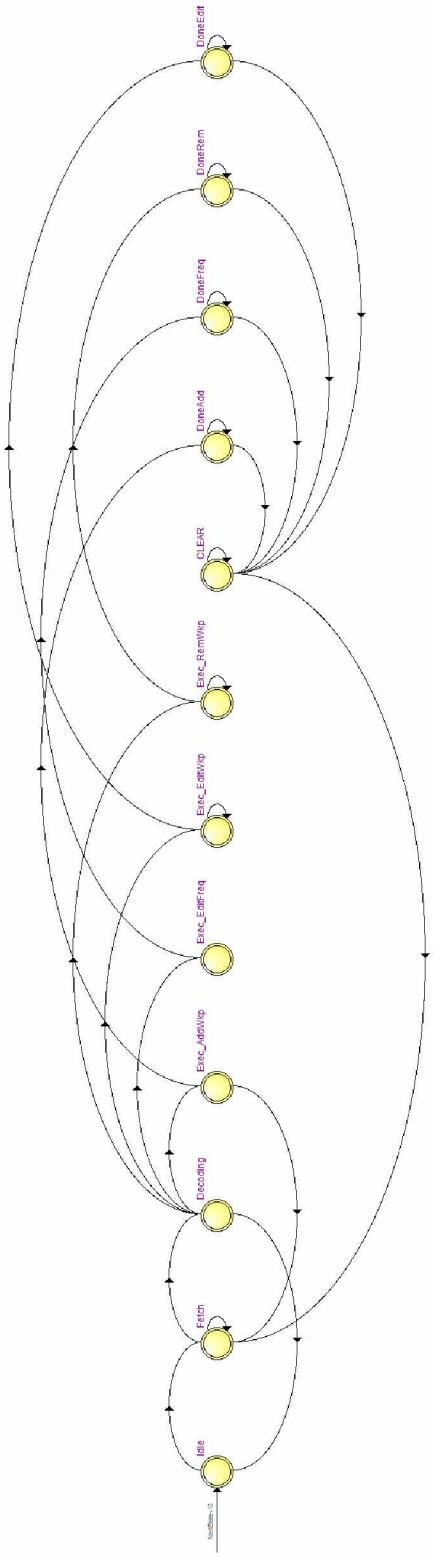


Figura 46: FSM: Control Unit - Decoder

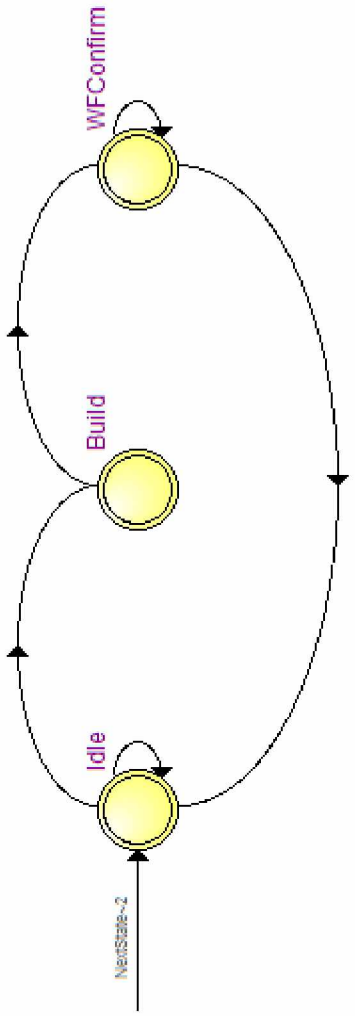


Figura 47: FSM: Control Unit - MSG Assembler

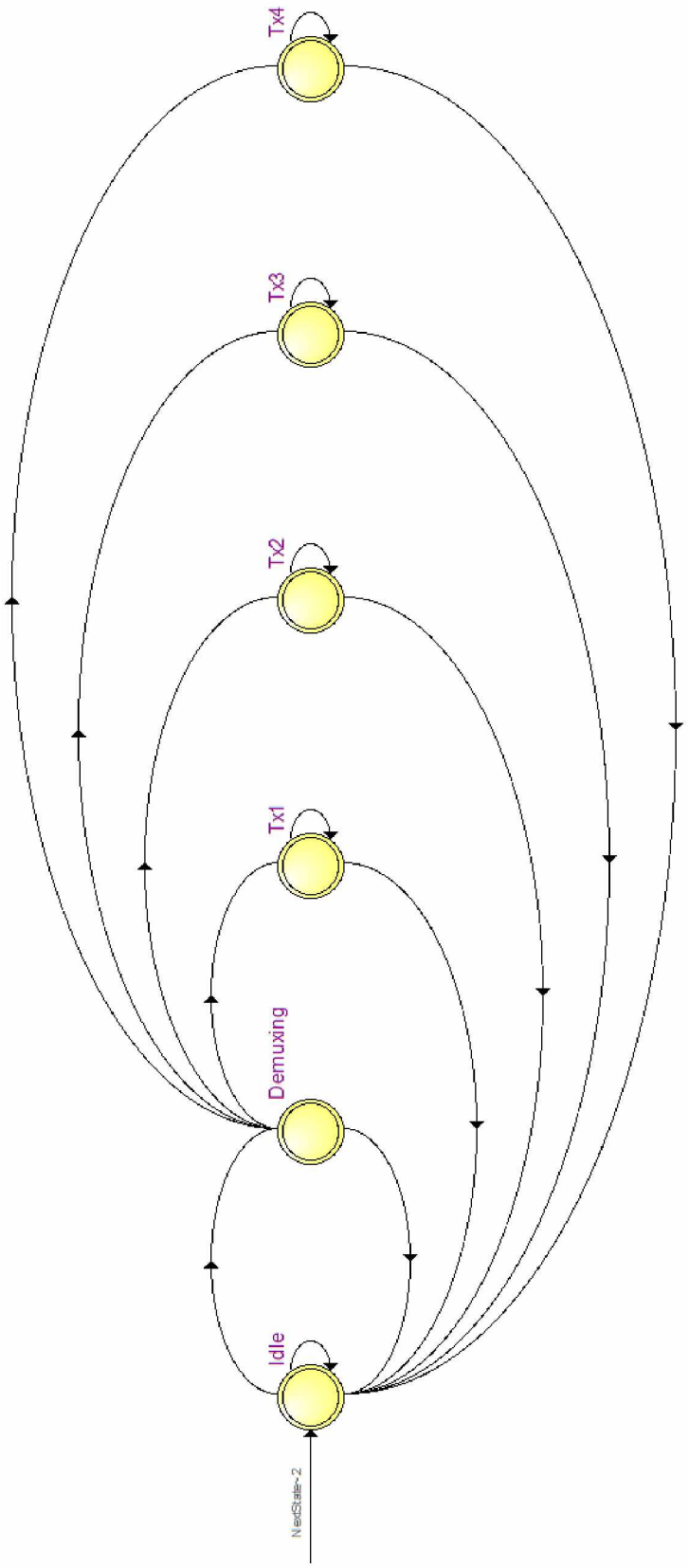


Figura 48: FSM: Control Unit - Datapath Tx

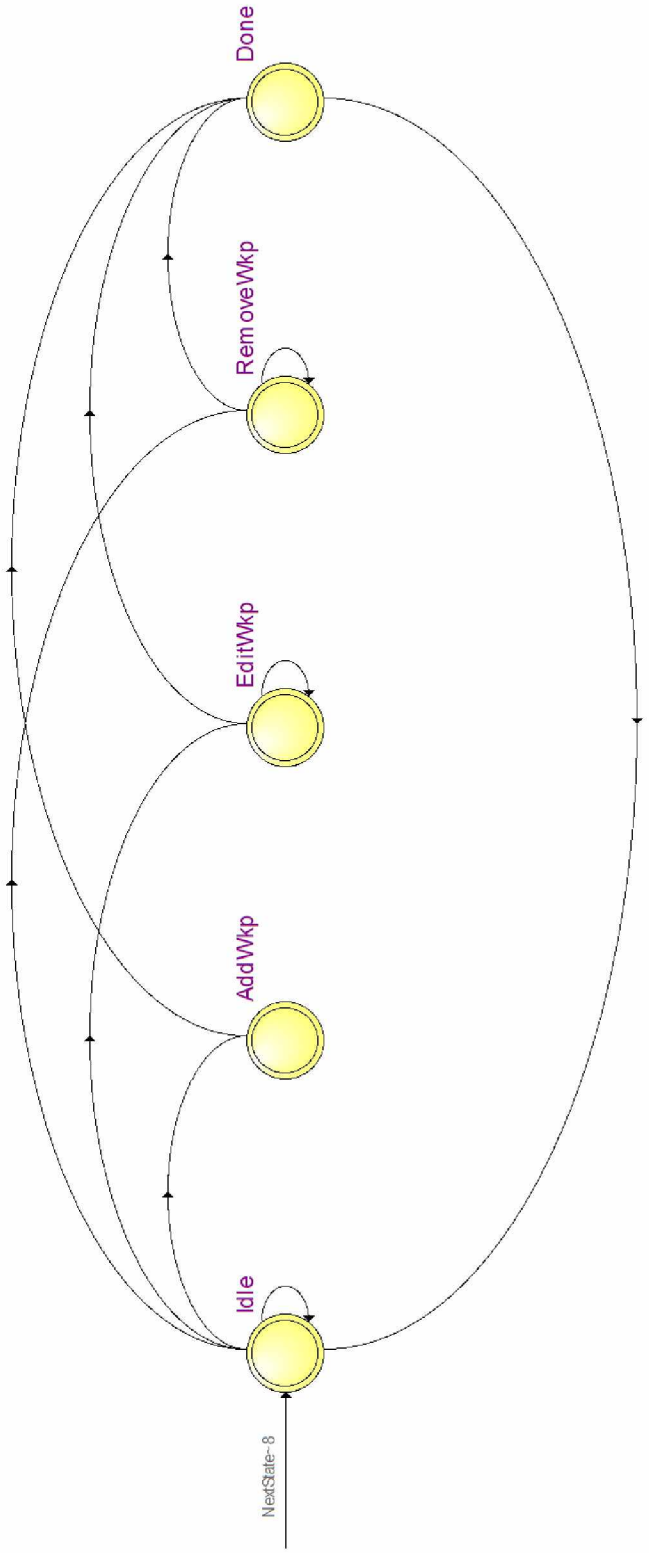


Figura 49: FSM: Lookup Table

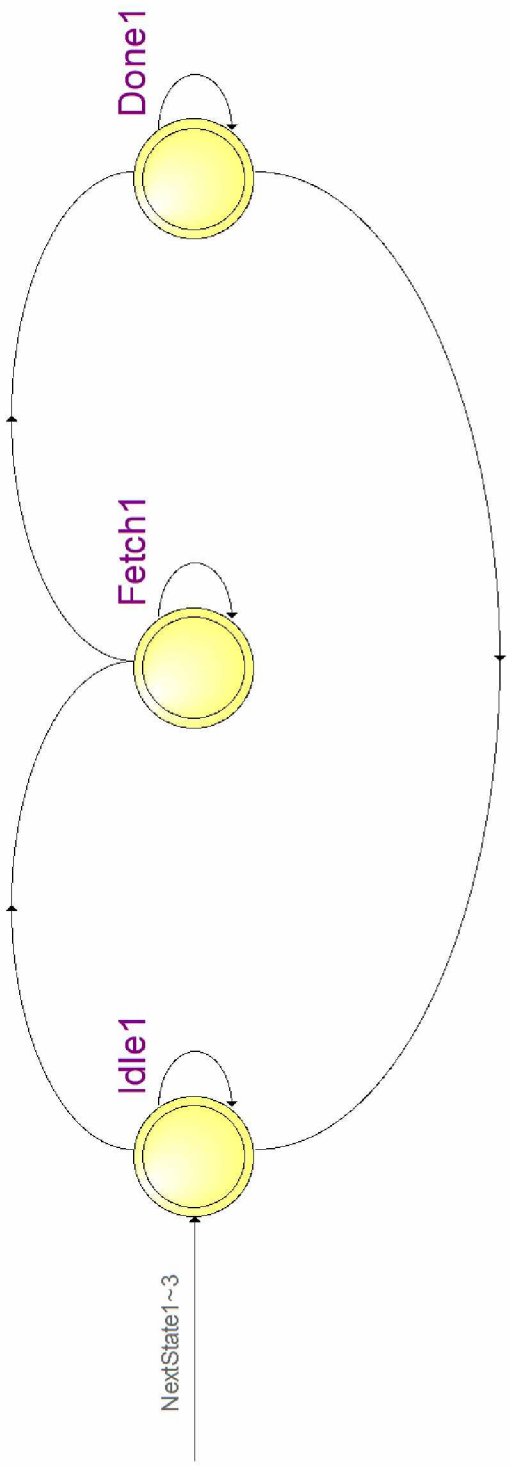


Figura 50: FSM: Lookup Table - Search

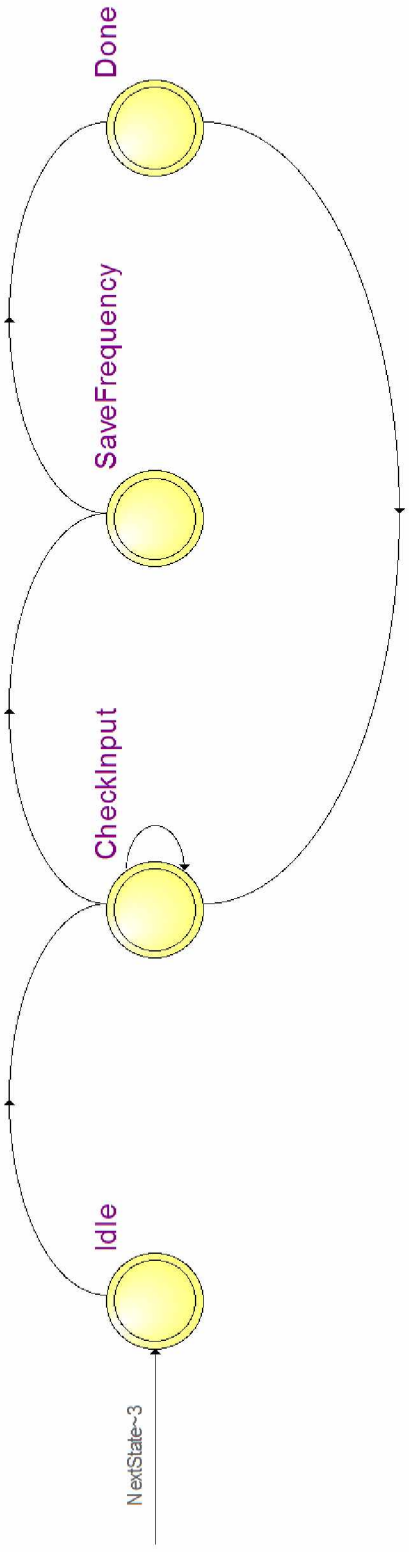


Figura 51: FSM: Frequencies

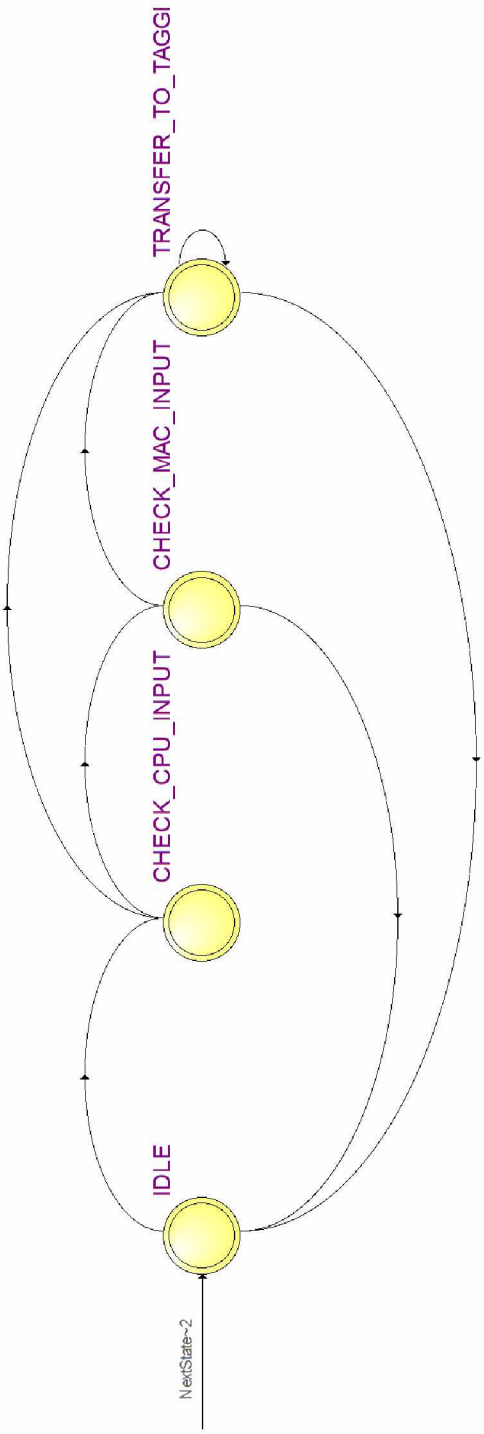


Figura 52: FSM: Datapath - Input Arbiter

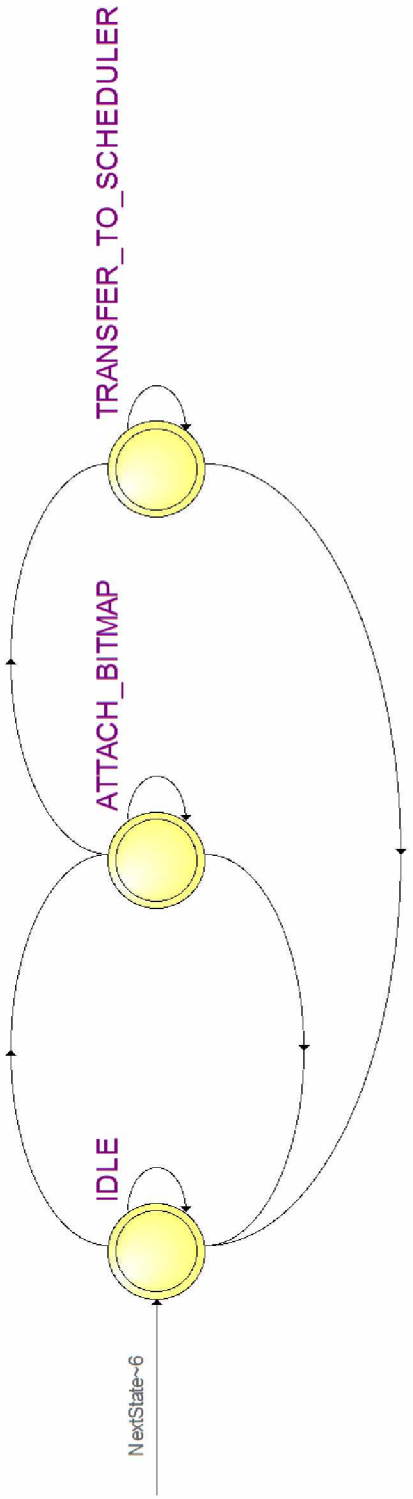


Figura 53: FSM: Datapath - Tagger

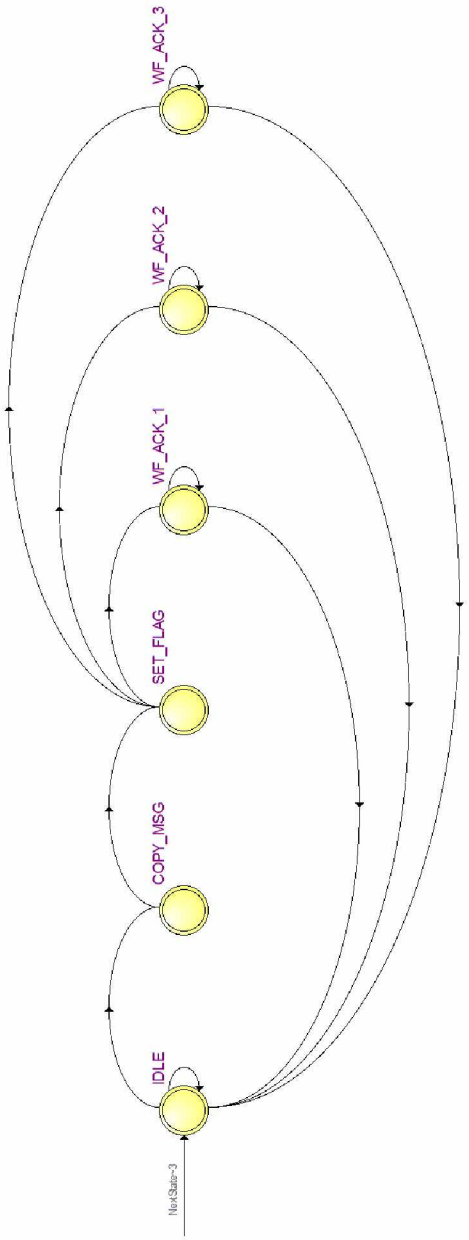


Figura 54: FSM: Datapath - Scheduler

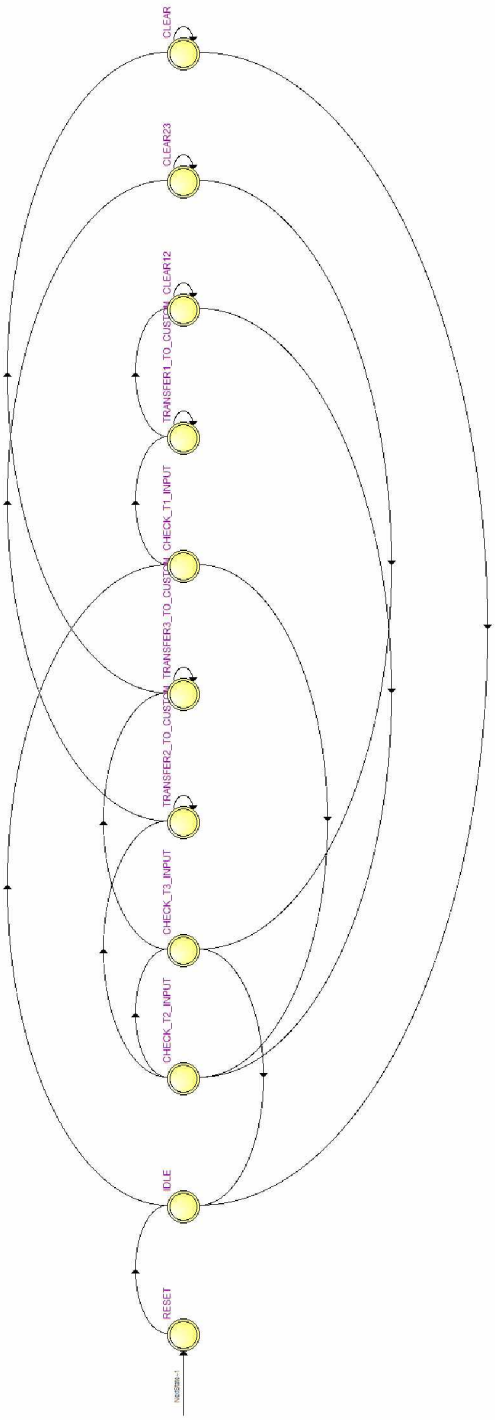


Figura 55: FSM: Datapath - Queues Arbiter

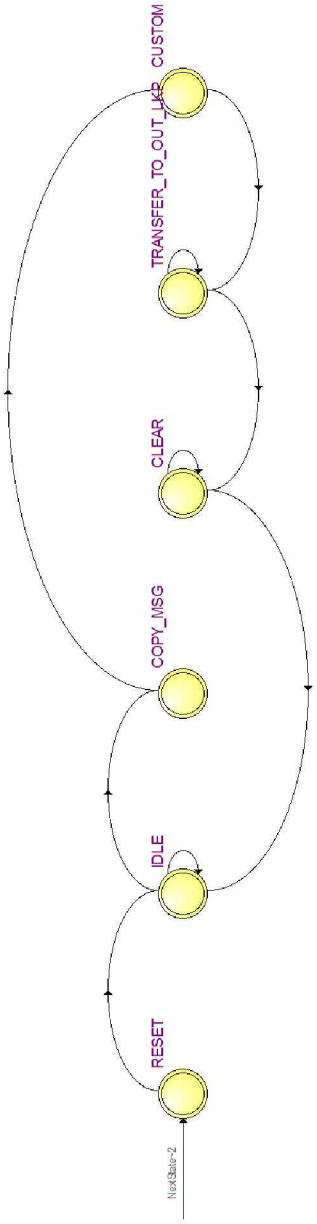


Figura 56: FSM: Datapath - Customization

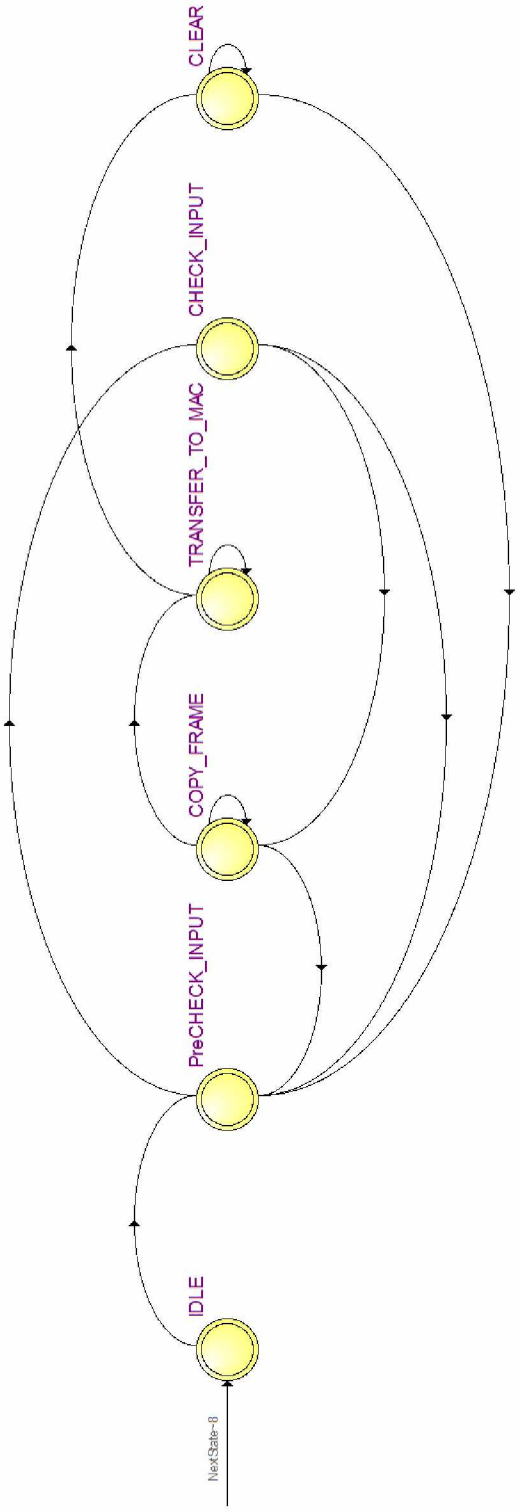


Figura 57: FSM: Datapath - Output Port Lookup

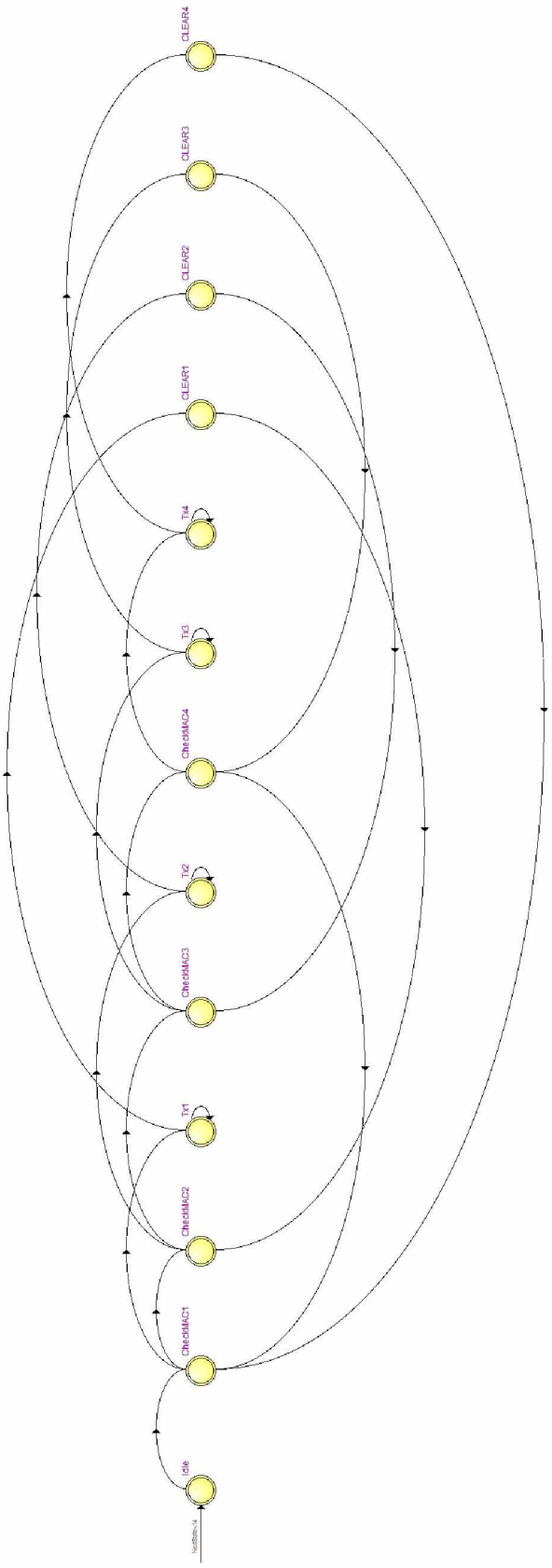


Figura 58: FSM: Datapath - Mac Tx

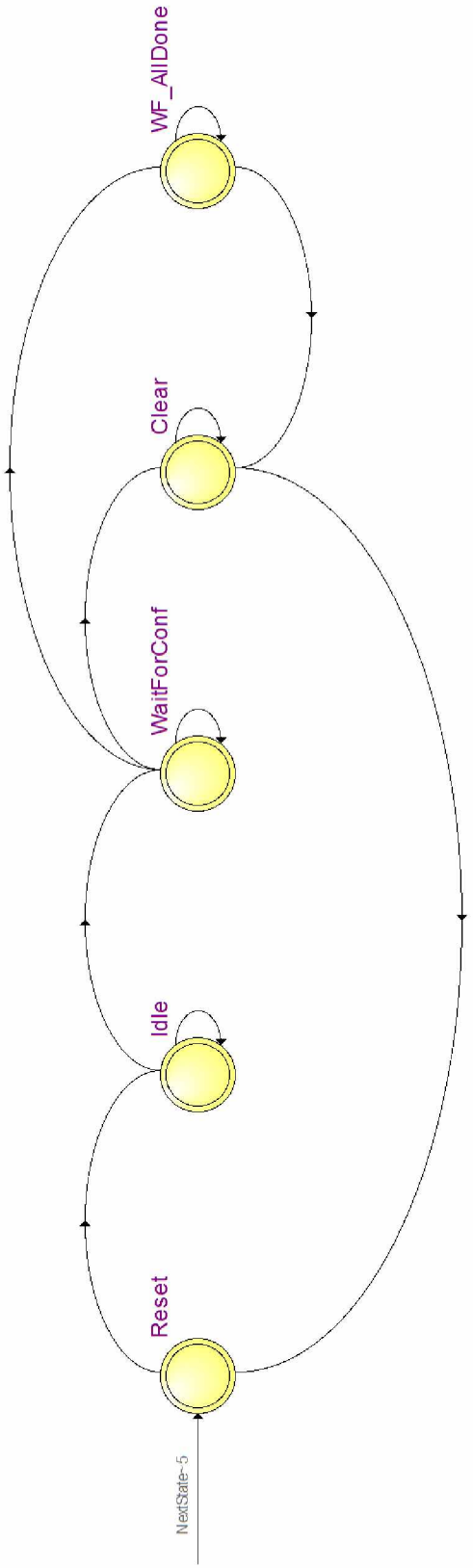


Figura 59: FSM: ACK Bitmap Abstraction

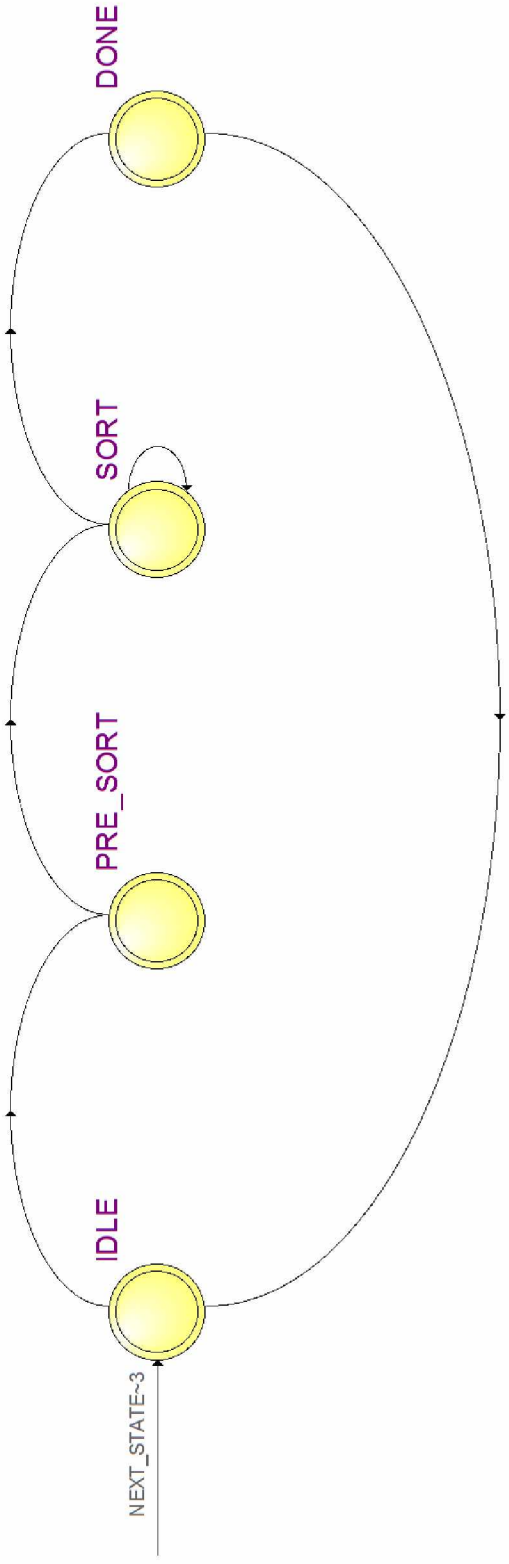


Figura 60: FSM: Sorter