Greenow: Um algoritmo de roteamento orientado a *workspace* para uma arquitetura de Internet do Futuro

Natal Vieira de Souza Neto

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

Uberlândia
2016

Natal Vieira de Souza Neto

Greenow: Um algoritmo de roteamento orientado a *workspace* para uma arquitetura de Internet do Futuro

Dissertação de mestrado 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 Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Pedro Frosi Rosa
Coorientador: Flávio de Oliveira Silva

Uberlândia
2016

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

S729g 2016

Souza Neto, Natal Vieira de, 1990-

Greenow : um algoritmo de roteamento orientado a workspace para uma arquitetura de Internet do Futuro / Natal Vieira de Souza Neto. - 2016.

154 f. : il.

Orientador: Pedro Frosi Rosa.

Coorientador: Flávio de Oliveira Silva.

Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.

Inclui bibliografia.

1. Computação - Teses. 2. Roteamento (Administração de redes de computadores) - Teses. 3. Internet - Teses. 4. Redes de computadores - Teses. I. Rosa, Pedro Frosi. II. Silva, Flávio de Oliveira, 1970-. III. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

CDU: 681.3

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Greenow: Um algoritmo de roteamento orientado a *workspace* para uma arquitetura de Internet do Futuro**" por **Natal Vieira de Souza Neto** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 12 de Fevereiro de 2016

Orientador: _____
Prof. Dr. Pedro Frosi Rosa
Universidade Federal de Uberlândia

Co-orientador: _____
Prof. Dr. Flávio de Oliveira Silva
Universidade Federal de Uberlândia

Banca Examinadora:

_____
Prof. Dr. Renan Gonçalves Cattelan
Universidade Federal de Uberlândia

Eduardo Cerqueira

_____
Prof. Dr. Eduardo Coelho Cerqueira
Universidade Federal do Pará

Este trabalho é dedicado aos bons, que trabalham
de forma honesta para o desenvolvimento do nosso país.

Agradecimentos

Agradeço à minha família, em especial minha mãe Luzia e minha avó Ilídia, pelo apoio incondicional em todos os caminhos que resolvi percorrer no âmbito pessoal, profissional e acadêmico. No fim aprendemos que família e conhecimento são as maiores riquezas.

Agradeço aos colegas do projeto EDOBRA, Doughert e Enrique, pela troca de experiências e aprendizado. Também aos colegas do grupo MEHAR, em especial ao meu amigo Flávio Silva, meu coorientador, que mostrou como precisamos de pessoas desbravadoras e de personalidade única.

Gostaria de deixar um agradecimento especial aos grandes amigos Caio Ferreira e Maurício Gonçalves, companheiros desta jornada e da vida. Este trabalho seria menos produtivo e certamente menos divertido sem vocês.

Agradeço aos docentes e técnicos da Faculdade de Computação pelo ensinamento e suporte prestados nos últimos anos. É uma honra poder dizer que, após esses anos, faço parte de uma comunidade tão importante.

Agradeço à Algar Telecom pelo apoio para a participação em congressos, e aos colegas de trabalho pela motivação que sempre proporcionaram. Neste âmbito, agradeço em especial ao João Henrique de Souza Pereira, amigo pesquisador e grande incentivador, pelos conselhos e diretrizes tão relevantes durante todo este período.

Por fim gostaria de deixar um agradecimento especial ao meu amigo Pedro Frosi Rosa, orientador deste trabalho e responsável por me ensinar que um bom café entre amigos pode resultar em grandes ideias que não surgiriam nas reuniões em salas fechadas.

"Explicar a emoção de ser palmeirense, a um palmeirense, é totalmente desnecessário.
E a quem não é palmeirense... É simplesmente impossível!"
(Joelmir Betting)

Resumo

A Internet cresceu muito nas últimas décadas e novos requisitos começaram a fazer parte da rede, tais como Mobilidade, Tráfego *Multicast*, *Multihoming*, Garantia de Banda etc. A arquitetura atual da Internet não suporta esses requisitos e portanto novas arquiteturas foram propostas. A ETArch (*Entity Title Architecture*) é uma delas e tem como objetivo garantir que requisitos e capacidades das aplicações sejam considerados para se estabelecer a comunicação da melhor forma. O Roteamento desempenha papel fundamental na Internet, sendo responsável por decidir o caminho pelo qual as primitivas contendo dados serão encaminhadas ao longo da rede. Além disso, todos os requisitos da Internet do Futuro dependem da atuação do roteamento. Na ETArch, o Roteamento é uma questão em aberto. O objetivo deste trabalho é prover algoritmos de roteamento intra e inter-domínio para a ETArch. Como hipótese, assumiu-se que para satisfazer os requisitos das aplicações a rota completa deve ser definida antes que os dados comecem a ser trafegados. As Redes Definidas por Software e tecnologias como OpenFlow viabilizam esta hipótese. Na Internet, o roteamento possui dois papéis distintos: (i) a parte algorítmica, responsável por decidir a melhor rota para a comunicação; e (ii) a parte de encaminhamento, responsável por encaminhar as primitivas de dados para o enlace correto. Nota-se que na arquitetura tradicional da Internet ambos os papéis são realizados toda vez que um pacote chega a um roteador. Este trabalho permite que a rota completa seja definida previamente, ou seja, gasta-se um tempo maior na configuração da rota, porém uma vez configurada a parte algorítmica não é mais realizada, similar ao que acontece na telefonia. Neste trabalho, foi definido o Roteamento para a ETArch e, também, foram realizados experimentos que demonstraram a viabilidade de se escolher rotas previamente. Uma vez que a rota é estabelecida, pode-se garantir que os requisitos das aplicações sejam atendidos e a partir daí apenas encaminhamento de primitivas é necessário, poupando-se tempo de processamento nos elementos de rede (antes necessário para rotear).

Palavras-chave: Roteamento. Internet do Futuro. Redes Definidas por Software.

Abstract

Current and future applications pose new requirements that Internet architecture is not able to satisfy, like Mobility, Multicast, Multihoming, Bandwidth Guarantee and so on. The Internet architecture has some limitations which do not allow all future requirements to be covered. New architectures were proposed considering these requirements when a communication is established. ETArch (Entity Title Architecture) is a new Internet architecture, clean slate, able to use application's requirements on each communication, and flexible to work with several layers. The Routing has an important role on Internet, because it decides the best way to forward primitives through the network. In Future Internet, all requirements depend on the routing. Routing is responsible for deciding the best path and, in the future, a better route can consider Mobility aspects or Energy Consumption, for instance. In the dawn of ETArch, the Routing has not been defined. This work provides intra and inter-domain routing algorithms to be used in the ETArch. It is considered that the route should be defined completely before the data start to traffic, to ensure that the requirements are met. In the Internet, the Routing has two distinct functions: (i) run specific algorithms to define the best route; and (ii) to forward data primitives to the correct link. In traditional Internet architecture, the two Routing functions are performed in all routers everytime that a packet arrives. This work allows that the complete route is defined before the communication starts, like in the telecommunication systems. This work determined the Routing for ETArch and experiments were performed to demonstrate the control plane routing viability. The initial setup before a communication takes longer, then only forwarding of primitives is performed, saving processing time.

Keywords: Routing. Future Internet. Software Defined Networking.

Lista de ilustrações

Figura 1 – Roteamento Estado de Enlace 37
Figura 2 – Roteamento Vetor de Distância 37
Figura 3 – Funcionamento do Protocolo BGP 39
Figura 4 – OpenFlow 42
Figura 5 – Controlador Floodlight 44
Figura 6 – RouteFlow 48
Figura 7 – *Handover* 56
Figura 8 – Arquitetura Internet x ETArch 59
Figura 9 – Exemplo de Topologia ETArch 61
Figura 10 – *Workspace* 62
Figura 11 – Autômato Finito para o ETCP 65
Figura 12 – DTSA do Projeto EDOBRA 67
Figura 13 – Diagrama de sequência para a primitiva WORKSPACE_ATTACH . . 69
Figura 14 – Movimentação de *workspace* em um cenário de *handover* 70
Figura 15 – Topologia SMART 71
Figura 16 – Controle na topologia de um *Master*-DTSA 79
Figura 17 – Controle na topologia Inter-*Master*-DTSA 81
Figura 18 – Configuração de DTSA vizinho 82
Figura 19 – Sequência de comunicação entre dois DTSAs 82
Figura 20 – DTS em níveis e Roteamento em larga escala 85
Figura 21 – Roteamento Intra-DTSA 89
Figura 22 – Algoritmo de roteamento Intra-DTSA 91
Figura 23 – Roteamento Inter-DTSA 94
Figura 24 – Lista de DTSAs em uma rota 95
Figura 25 – Algoritmo de roteamento Inter-DTSA - *Lookup* 96
Figura 26 – Algoritmo de roteamento Inter-DTSA - Configuração 97
Figura 27 – FSM para o DTSA 98
Figura 28 – Autômato para o *Master*-DTSA 99

Figura 29 – Topologia do experimento intra-DTSA 105
Figura 30 – Rotas escolhidas no experimento intra-DTSA 106
Figura 31 – Etapas do experimento intra-DTSA 107
Figura 32 – Topologia do primeiro experimento inter-DTSA 109
Figura 33 – Rotas escolhidas no primeiro experimento inter-DTSA 110
Figura 34 – Topologia do segundo experimento inter-DTSA 111
Figura 35 – Rotas escolhidas no segundo experimento inter-DTSA 112
Figura 36 – Rotas escolhidas em cenários com vários *Workspaces* 114
Figura 37 – Tempo de *attach* com vários *Workspaces* simultâneos 115

Lista de tabelas

Tabela 1 – Características dos controladores OpenFlow 46
Tabela 2 – Primitivas relacionadas ao roteamento 92
Tabela 3 – Tempos do experimento intra-DTSA topologia linear 108
Tabela 4 – Tempo para extensão de *Workspace* inter-DTSA 113

Lista de siglas

AICT Advanced International Conference on Telecommunications

AP Access Point

API Application Programming Interface

AS Autonomous System

ATM Asynchronous Transfer Mode

BGP Border Gateway Protocol

DB Database

DHT Distributed Hash Table

DNS Domain Name System

DTS Domain Title Service

DTSA DTS Agent

DTSCP DTS Control Procotol

ECMP Equal Cost MultiPath

EDOBRA ExtenDing Ofelia in BRAzil

ETArch Entity Title Architecture

ETCP Entity Title Control Protocol

FIB Forwarding Information Base

FSM Finite State Machine

HTTP HyperText Transfer Protocol

ICN[1] Information-Centric Networking

ICN[2] International Conference on Network

IP Internet Protocol

IPC Inter-Process Communication

ISP Internet Service Provider

JVM Java Virtual Machine

MAC Media Access Control

MDTSA Master DTSA

MIH Media Independent Handover

MPLS Multiprotocol Label Switching

MPT Modified Path Tree

MTU Maximum Transmission Unit

NaaS Network as a Service

NE Network Element

NDN Named Data Networking

NFaaS Network Functions as a Service

NFV Network Functions Virtualization

NIRA New Internet Routing Architecture

OFELIA OpenFlow in Europe: Linking Infrastructure and Applications

ONOS Open Network Operating System

OSPF Open Shortest Path First

QoE Quality of Experience

QoS Quality of Service

RA Resource Adapter

REST Representational State Transfer

RINA Recursive InterNetwork Architecture

SBB Service Building Block

SBRC Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos

SDN Software Defined Networking

SIP Session Initiation Protocol

SMART Suporte de Sessões Móveis com Alta Demanda de Recursos de Transporte

SNAC Simple Network Access Control

SPoF Single Point of Fail

SPT Shortest Path Tree

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TI Tecnologia da Informação

ToS Type of Service

UFU Universidade Federal de Uberlândia

WPEIF Workshop de Pesquisa Experimental da Internet do Futuro

VP Virtual Path

XML eXtensible Markup Language

Sumário

1 Introdução 25
1.1 Motivação 26
1.2 Objetivos e Desafios da Pesquisa 27
1.3 Hipótese 27
1.4 Contribuições 28
1.5 Organização da Dissertação 28
2 Fundamentação Teórica 31
2.1 Aspectos Conceituais de Roteamento 32
2.1.1 Roteamento e Encaminhamento de Primitivas 32
2.1.2 Circuito Lógico e Datagrama 33
2.1.3 Algoritmos de Roteamento 35
2.1.4 Roteamento Intra/Inter-domínio 38
2.2 Redes Definidas por Software 40
2.2.1 OpenFlow 41
2.2.2 Controladores OpenFlow 42
2.3 Propostas para Internet do Futuro 46
3 Uma arquitetura para Internet do Futuro 55
3.1 ETArch: Aspectos conceituais 55
3.1.1 Entidade 59
3.1.2 Título 59
3.1.3 *Domain Title Service* 60
3.1.4 *Workspace* 61
3.2 Implementações 65
3.2.1 Projeto EDOBRA 65
3.2.2 SMART 71
4 Proposta de Roteamento Intra/Inter-domínio orientado a *Workspace* 73
4.1 DTS: Aspectos Conceituais e Arquiteturais 75
4.1.1 DTS *Agents* 76

4.1.2 *Master*-DTSA . . . 77
4.1.3 *Workspace* de Controle . . . 78
4.1.4 Organização do DTS . . . 83
4.2 Roteamento ETArch: *Status Quo* . . . 86
4.3 Roteamento Intra Domínio . . . 87
4.4 Serviços de Roteamento Orientado a *Workspaces* . . . 91
4.5 Roteamento Inter-Domínio . . . 92
4.5.1 Algoritmo de Roteamento . . . 93
4.5.2 Autômato de roteamento no DTSA . . . 97
4.5.3 Autômato de roteamento no *Master*-DTSA . . . 98
4.6 Roteamento em Larga Escala . . . 100
5 Experimentos e Análise dos Resultados . . . 103
5.1 Método para a Avaliação . . . 104
5.2 Descrição dos Experimentos . . . 105
5.2.1 Experimentos intra-DTSA . . . 105
5.2.2 Experimentos inter-DTSA . . . 108
5.3 Avaliação dos Resultados . . . 116
6 Conclusão . . . 117
6.1 Principais Contribuições . . . 119
6.2 Trabalhos Futuros . . . 121
6.3 Contribuições em Produção Bibliográfica . . . 122
Referências . . . 125

Apêndices 133
APÊNDICE A Codificação das topologias utilizadas . . . 135
A.1 Script Python do experimento intra-DTSA . . . 135
A.2 Script Python do primeiro experimento inter-DTSA . . . 137
A.3 Script Python do segundo experimento inter-DTSA . . . 139
APÊNDICE B Fluxos configurados no Mininet pelos experimentos . . 143
B.1 Fluxos no experimento intra-DTSA . . . 143
B.2 Fluxos no primeiro experimento inter-DTSA . . . 146
B.3 Fluxos no segundo experimento inter-DTSA . . . 148

Capítulo 1

Introdução

O roteamento, presente nas redes de computadores e telefonia (WETHERALL; TANENBAUM, 2013), passou por diversas mudanças ao longo dos anos, buscando-se melhorias para atender requisitos antes não existentes. Com o surgimento das Redes Definidas por Software (SDN – *Software Defined Networking*)(KREUTZ et al., 2015), o plano de controle passa a ser separado logicamente do plano de dados, o que possibilita novas abordagens para roteamento.

Nas redes de computadores, foco deste trabalho, o roteamento é responsável por estabelecer o melhor caminho entre uma origem e determinado(s) destino(s) (WETHERALL; TANENBAUM, 2013). Portanto, trabalhos relacionados a roteamento têm uma especial importância no âmbito geral das redes de computadores, devido à influência que o roteamento possui no desempenho de uma rede.

Dos últimos anos até os dias atuais, diversos tipos de comunicação têm feito uso da Internet de forma massiva. Com isso, novos requisitos surgiram e necessitam atenção especial, como por exemplo: tráfego *Multicast*, eficiência energética, mobilidade, *multihoming*, segurança, QoS (*Quality of Service*), QoE (*Quality of Experience*) etc (LIBERAL; FAJARDO; KOUMARAS, 2009) (SWIATEK et al., 2012). A questão é que as aplicações muitas vezes exigem esses requisitos, mas deixam a cargo da rede o melhor cenário para satisfazê-los.

O roteamento esperado para as redes futuras está relacionado aos requisitos acima citados. A escolha da melhor rota para determinado tipo de comunicação deverá levar em consideração o que se espera por parte das aplicações. Como exemplo, cita-se uma aplicação que exija baixo consumo de energia, onde o roteamento deve optar por rotas com links e elementos de rede que gastem menos energia.

Na arquitetura Internet tradicional, o roteamento é realizado por roteadores (*hops*) ao longo do caminho (de um *host* de origem até um *host* de destino). Os dados são encaminhados em pacotes IP (*Internet Procotol*) e cada pacote é analisado a cada salto, em cada roteador, que analisa o destino para definir por qual link encaminhará o pacote. Dessa forma, o algoritmo de roteamento deve ser executado sempre que um pacote passa

por um roteador.

O presente trabalho pretende oferecer uma abordagem na qual o roteamento seja realizado no plano de controle de uma rede, antes que as primitivas contendo dados comecem a ser encaminhadas. Nesse tipo de abordagem, considerando-se que não se espera roteamento *inband*, abre-se mão dos roteadores, e os elementos de rede passam a necessitar apenas do papel exclusivo de encaminhar primitivas (*switching*).

SDN permite não utilizar roteadores, uma vez que primitivas para roteamento são competência do plano de controle. Os controladores SDN podem ser utilizados para executar algoritmos que definem rotas em cada comunicação. Espera-se com esse mecanismo reduzir o custo dos elementos de rede e flexibilizar (através de software) o roteamento.

A relevância deste trabalho está em apresentar um protocolo de roteamento que atue no plano de controle, factível de ser utilizado em uma arquitetura de Internet do Futuro que atenda aos requisitos atuais e futuros relacionados à Internet.

Como arquitetura de Internet do Futuro, pretende-se utilizar a ETArch (*Entity Title Architecture*)(SILVA, 2013). Essa arquitetura abre mão do conceito de *hosts* de origem e destino. Nela, entidades que desejem se comunicar passam a fazer parte de um barramento lógico, que independe da topologia, denominado *Workspace*, pelo qual primitivas são encaminhadas. Os requisitos das entidades (aplicações) em relação às comunicações são considerados ao se estabelecer cada *Workspace*.

1.1 Motivação

Na arquitetura Internet tradicional, o processamento realizado para roteamento é feito *inband*. Basicamente, toda vez que uma primitiva chega a um roteador, é realizada uma consulta na tabela de roteamento do roteador para determinar a rota pela qual a primitiva será conduzida. Essa abordagem não é flexível para tratamento de requisitos diferentes das aplicações. Além disso, o algoritmo de roteamento é feito para todas as primitivas e em todos os roteadores da rede.

Inerente à ETArch está o tratamento de requisitos das entidades (aplicações, usuários, *hosts* etc). Com isso, espera-se que essa arquitetura venha a ser utilizada em redes de grande porte. Na ETArch, o roteamento intra e inter-domínio não está especificado de forma clara. Fundamenta-se uma investigação mais específica na ETArch, empenhada em tratar roteamento para esta arquitetura.

Com a ETArch, que é compatível com conceitos SDN, é possível efetivar o roteamento prévio para que rotas sejam estabelecidas antes de se iniciar cada comunicação, em um funcionamento similar ao que acontece na telefonia. Com o roteamento prévio, é possível atender aos requisitos especificados para a comunicação.

Na ETArch as entidades são nomeadas por um título único e não ambíguo. Além disso, para cada comunicação é realizado o estabelecimento de *workspaces* responsáveis

por encaminhar primitivas de dados entre entidades. A ETArch é naturalmente *Multicast*, o título não referencia a localização (facilitando a reconfiguração para o deslocamento de entidades móveis) e requisitos de QoS e QoE podem ser levados em consideração no estabelecimento dos *workspaces*. Dado essas circunstâncias, afirma-se que aplicações do tipo *Multicast*, multimídia ou *mobile* são mais fortes na ETArch do que na arquitetura tradicional.

Este trabalho definirá o mecanismo de roteamento orientado a *workspace* (*workspace-driven*), sendo que nesse tipo de roteamento há espaço para considerar diferentes objetivos no cálculo das rotas. Porém, os experimentos realizados focaram no objetivo tradicional (menor distância).

1.2 Objetivos e Desafios da Pesquisa

O objetivo geral deste trabalho é propor um mecanismo de roteamento intra/inter-domínio na ETArch para que diferentes parâmetros (distância, largura de banda, QoS etc) possam ser avaliados para a escolha do melhor caminho, estabelecendo-se a rota completa antes que as primitivas de dados comecem a ser encaminhadas. Para se alcançar o objetivo supracitado propõe-se os objetivos específicos:

- ❏ Criação de um algoritmo para escolha da melhor rota, que defina previamente a rota completa, baseado nos requisitos das aplicações;
- ❏ Implantação do algoritmo em um ambiente baseado na Arquitetura ETArch;
- ❏ Demonstração e realização experimentos com o algoritmo proposto;
- ❏ Análise da utilização do novo método em cenários experimentais com comunicação passando por elementos de rede presentes em diferentes domínios e comparar com métodos de roteamento atuais.

Os desafios do presente trabalho estão relacionados à independência do protocolo IP. Na ETArch, por definição, primitivas não são direcionadas a um destino. Com isso é impossível utilizar o endereço final de determinado *host*, sendo, contudo, possível estabelecer comunicações *unicast*. O roteamento nesta arquitetura deve ser construído de forma a otimizar extensões de *Workspaces* ao longo da rede.

1.3 Hipótese

Com a utilização da ETArch, as entidades, desejosas de se comunicar, solicitam juntar-se (*Attach*) a determinado *Workspace*. Com isso, o controlador da rede é utilizado para definir o caminho pelo qual tal *Workspace* será estendido até a entidade solicitante. A

abordagem de definição prévia das rotas torna-se possível no momento que determinada entidade solicita se juntar a algum *Workspace*: o caminho pelo qual o *Workspace* será estendido até a entidade é determinado no exato momento que ela faz a requisição.

Workspaces na ETArch são criados especificando-se determinados requisitos. Dessa forma, pode-se trabalhar com os requisitos das entidades de forma inerente. O roteamento, realizado pelo controlador, tomará decisão de rotas e informará aos elementos de rede para se configurarem de forma a encaminhar as primitivas ao longo do *Workspace* pré-estabelecido.

Para o roteamento inter-domínio, onde diferentes controladores gerenciam topologias distintas, pretende-se expandir o conceito de *Workspace* para o plano de controle. Dessa forma, tem-se a intenção de criar *Workspaces* de Controle: barramentos lógicos no plano de controle por onde diferentes controladores SDN se comunicam.

Acredita-se que o roteamento pelo plano de controle poderá atender os requisitos das aplicações de forma plena e com menor *overhead* de comunicação do que o roteamento IP.

1.4 Contribuições

O trabalho apresenta uma proposta de roteamento no plano de controle, capaz de determinar caminhos que atendam aos requisitos especificados por aplicações. Portanto, a função de roteador deixa de existir, pois a abordagem proposta possibilita aos elementos de rede apenas encaminhar primitivas, uma vez que a rota é estabelecida *out-of-band* (pelos controladores do plano de controle da rede).

O roteamento proposto no trabalho contribui para qualquer arquitetura SDN, inclusive ETArch. As rotinas de roteamento que serão definidas permitem o uso de objetivos diferentes e foram construídas, em sua totalidade, *out-of-band*. Dessa forma, toda arquitetura que separe o plano de controle da rede de dados pode utilizar este trabalho como referência.

As demais contribuições do trabalho estão relacionadas à ETArch que, até a conclusão deste projeto, não dispunha de roteamento. O presente trabalho propõe formas de roteamento intra e inter-domínio para a ETArch, possibilitando assim trabalhos futuros e evolução da arquitetura.

1.5 Organização da Dissertação

Esta dissertação está organizada em seis capítulos, sendo que o Capítulo 2 apresenta os aspectos conceituais de roteamento, explanando sua função nas redes, e o funcionamento de algoritmos e protocolos de roteamento mais utilizados. Além disso, são introduzidas as Redes Definidas por Software, que serão utilizadas neste trabalho. Por fim apresenta-se o

estado da arte de pesquisas em Internet do Futuro, com foco especial naquelas referentes a roteamento.

O Capítulo 3 apresenta os aspectos conceituais e principais implementações da arquitetura ETArch. Inicia-se com um detalhamento dos componentes principais e posteriormente é mostrado como cada componente foi desenvolvido para um ambiente computacional que implanta a arquitetura.

O Capítulo 4 apresenta a proposta principal deste trabalho, o roteamento orientado a *workspace*. Nesse capítulo, os aspectos conceituais e arquiteturais da ETArch são explorados minuciosamente, e é apresentada a situação atual de roteamento da arquitetura. Na sequência, a proposta para roteamento intra e inter domínio e os serviços de roteamento são descritos, detalhando-se os algoritmos e autômatos utilizados para compor o roteamento orientado a *workspace*.

O Capítulo 5 apresenta os experimentos realizados tanto em topologia de mesmo domínio quanto em topologias em diferentes domínios. São apresentados os resultados (rotas definidas pelo algoritmo proposto) e a avaliação do algoritmo.

O Capítulo 6 faz uma conclusão apresentando as contribuições do trabalho, os objetivos alcançados e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Em uma rede de computadores o termo roteamento pode ser entendido como "a definição do caminho que determinado dado seguirá desde a sua origem até alcançar um ou mais destinos" (WETHERALL; TANENBAUM, 2013). Dessa forma, pode-se dizer que roteamento está diretamente relacionado aos primórdios da telefonia e das redes de computadores no geral.

Nas redes telefônicas, uma chamada é estabelecida por meio de dois terminais (telefones) interconectados por diversos elementos de rede ao longo do caminho e, portanto, o meio físico por onde os dados passam é considerado um circuito (lógico). Nas redes computacionais, dois computadores trocam dados entre si e, da mesma forma, um meio físico entre diversos elementos deve ser interconectado para formar um caminho para que os dados possam ser transmitidos.

Na Internet, não poderia ser diferente: quando dois ou mais usuários desejam estabelecer uma comunicação é necessário que os dados sigam por uma determinada rota, pois os computadores dos usuários normalmente não estão ligados diretamente, podendo inclusive estar em continentes diferentes, interligados por vários nós intermediários (WETHERALL; TANENBAUM, 2013) (FOKKINK, 2013).

O roteamento nas telecomunicações é diferente daquele utilizado na Internet, mesmo que algumas técnicas criadas para uso nas telecomunicações tenham sido utilizadas na Internet, em mecanismos mais específicos, como por exemplo o MPLS (*Multiprotocol Label Switching*). Algumas dessas técnicas foram utilizadas pela telefonia há décadas e aproveitadas na Internet apenas recentemente, como é o caso da separação do plano de controle e dados.

Na Internet pesquisas evolucionárias foram realizadas para aprimorar o roteamento, sendo que a maioria manteve o uso do protocolo IP (Internet *Protocol*) (POSTEL, 1981). Em contrapartida, pesquisas não evolucionárias (*clean slate*) também foram realizadas, sendo que algumas propõem arquiteturas não dependentes do IP.

O presente capítulo tem como objetivo apresentar uma introdução a aspectos conceituais do roteamento na Internet, descrevendo os algoritmos e protocolos mais usuais, tanto

para redes sob o mesmo domínio quanto para redes em domínios diferentes. Conceitos de roteamentos na telefonia, apesar de fazê-lo diferentemente, serão apresentados, pois muitos desses conceitos foram motivadores para o trabalho ora proposto.

Os aspectos conceituais de Redes Definidas por Software também serão apresentados, mostrando os principais trabalhos e implementações dessa área. Por fim, as novas propostas de modelos e arquiteturas para Internet do Futuro serão relatadas, com foco nos trabalhos de roteamento.

2.1 Aspectos Conceituais de Roteamento

Em redes de computadores a palavra roteamento significa decidir as rotas que serão utilizadas para que os dados possam ser transferidos da origem ao(s) destino(s). Em uma rede, um *host* (computador, *smartphone*, *tablet*, etc.) envia e recebe informações de outros *hosts*. Dessa forma o destino de um dado pode ser um (*Unicast*) ou mais (*Multicast* ou *Broadcast*) *hosts*.

Topologias de redes podem ser representadas por grafos, nos quais roteadores são representados por vértices e enlaces (*links*) entre roteadores são representados por arestas. A capacidade de um roteador se interconectar a um ou mais roteadores, permitindo que uma comunicação tenha múltiplas opções de rotas, pode ser representada pela capacidade de um vértice poder ter uma ou mais arestas.

As redes de computadores podem ser arranjadas em topologias variadas tais como barramentos, anéis, estrelas, árvores etc, que podem ser rearranjadas de diversas formas, permitindo o uso de diversos algoritmos. Por exemplo, a topologia de uma rede poderia ser transformada em uma árvore que permitiria aplicar algoritmos tradicionais de busca em profundidade.

2.1.1 Roteamento e Encaminhamento de Primitivas

De um modo geral, o termo roteamento enseja as ações de rotear e encaminhar. Entretanto, o roteamento, propriamente dito, é o procedimento realizado pelos roteadores, para manter informações sobre as possíveis rotas da rede.

O encaminhamento é realizado quando uma primitiva de dados chega ao roteador, há o escrutínio da tabela de rotas em busca do destino e um enlace de saída é escolhido para o envio da primitiva. À presente pesquisa interessa o roteamento e não o encaminhamento. De toda forma, para melhor distinção do interesse, encaminhamento será abordado introdutoriamente.

Na arquitetura Internet, baseada no modelo OSI/ISO (DAY; ZIMMERMANN, 1983) (PARK; SHIRATORI, 1994), a camada de rede é a responsável pelo roteamento e o termo Pacote (*Packet*) é utilizado para nomear primitivas de dados.

O tempo de processamento total, desde a fila de entrada até a fila de saída (enlace) do roteador, considera também o tempo dispendido no encaminhamento, haja visto que todo pacote enviado consome tempo nos nós intermediários, que gastam tempo de processamento para decidir a rota que será utilizada e para encaminhar o pacote ao enlace de saída.

Na Internet, todo pacote passa de um roteador (*hop*) para outro até chegar ao destino. Os roteadores analisam cada pacote para determinar a qual enlace de saída (porta) ele deverá encaminhar o pacote. Em um roteador, as portas estão associadas a enlaces que têm por finalidade interligar os roteadores, sendo que o *host* de destino está ligado a um ou mais roteadores da rede.

Em uma rede, o tempo de roteamento pode ser considerável, pois, devido a fatores tais como o tamanho máximo de *frames*[1], um pacote pode ser fragmentado gerando diversos pacotes[2]. Portanto, o tempo de roteamento de um pacote depende do número de *hops* no caminho (rota) e também da quantidade de fragmentos que eventualmente serão gerados.

2.1.2 Circuito Lógico e Datagrama

Como mencionado anteriormente, a forma como o roteamento é feito na Internet (*In-Band*)[3] difere-se daquela usada na telefonia (*Out-of-Band*)[4]. A diferença nestes casos está no momento em que uma rota é definida, o que introduz os conceitos de Datagrama e Circuito Lógico. O primeiro é amplamente utilizado na Internet, enquanto o segundo é aquele implantado na telefonia.

Um Circuito Lógico é utilizado na telefonia para estabelecer um canal de comunicação entre dois terminais (telefones). Na prática, quando um assinante (*Caller*) deseja realizar uma ligação para outro (*Called*), ele deve identificar o número do terminal de destino (*Called Id*). O conceito de Circuito Virtual utilizado pelo protocolo TCP (*Transmission Control Protocol*) da camada de transporte da Arquitetura Internet é análogo ao conceito de Circuito Lógico.

Para que uma ligação seja estabelecida, a central telefônica responsável pelo terminal de origem (*Caller Id*) deve descobrir a localização do terminal de destino (*Called Id*). Após a localização do terminal de destino, um Circuito Lógico deve ser estabelecido pelas centrais de ambos os terminais. Esse circuito é um canal que tem como *end-points* as centrais telefônicas de origem e de destino, passando por diversos elementos de rede (centrais telefônicas) ao longo do circuito lógico (ITU, 1994).

Durante o estabelecimento de uma chamada, cada central ao longo do caminho deve ser configurada para fazer parte do circuito lógico. Após as configurações serem feitas,

[1] Definido pelo enlace lógico (MTU – *Maximum Transmission Unit*).
[2] Para o roteamento IP, cada fragmento é tratado como um pacote.
[3] *In-Band* é uma abordagem na qual transmissões de controle e de dados compartilham a mesma rede.
[4] *Out-of-Band* é uma abordagem na qual transmissões de controle não compartilham a mesma rede de dados.

i.e. o circuito lógico ser estabelecido, a chamada é liberada nos terminais e o canal estabelecido (rota) é mantido até o término da chamada (LIN; CHENG, 1993) (GAWLICK; KALMANEK; RAMAKRISHNAN, 1996) (MCKENZIE, 2003).

A primeira característica relevante do circuito lógico é o tempo de configuração (LIN; CHENG, 1993). Como a rota é pré-determinada, i.e., antes que os pacotes de voz sejam transmitidos, o tempo de estabelecimento do circuito pode ser considerável. Todavia, uma vez estabelecido o circuito lógico, os dados serão transmitidos sem que haja roteamento nas centrais durante o tempo de transferência. Se houver uma nova chamada, independentemente de ser os mesmos números de telefone, a rota é configurada novamente.

Outra característica relevante do circuito lógico é a reserva de banda (a telefonia padronizou trabalhar com largura de 4kHz). Uma vez que um circuito está sendo estabelecido entre dois telefones (em princípio para transmissão de voz), as centrais telefônicas selecionadas para fazer parte da rota devem reservar banda. Na literatura essa prática é conhecida como estabelecimento do canal, e, uma vez estabelecido, o canal está garantido para o circuito.

Na prática, o estabelecimento do canal significa que a comunicação necessita de uma determinada taxa de transferência (vazão) e as centrais ao longo do circuito lógico devem garanti-la. Caso essa taxa não seja respeitada, o usuário de um telefone pode notar falhas na voz do outro usuário. O estabelecimento do canal com reserva banda é um dos principais argumentos usados pelos pesquisadores em telefonia para manter o roteamento por circuito lógico (GAWLICK; KALMANEK; RAMAKRISHNAN, 1996) (ITU, 1997).

A desvantagem do circuito lógico está relacionada à disponibilidade, pois, se a chamada for interrompida, caberá ao usuário refazê-la. Em caso de queda de enlace, por exemplo, a central deverá definir uma nova rota para a próxima chamada. Portanto, a telefonia garante disponibilidade sem estado.

Circuito lógico é relevante para este trabalho, pois há relação entre conceitos desse mecanismo com esta proposta. Para o roteamento orientado a *workspace*, rotas serão pré estabelecidas, de modo análogo ao circuito lógico. O intuito é eliminar o tempo de processamento de rotas entre os elementos de rede ao longo do caminho, durante a transmissão de *streams*, e atender requisitos de aplicações, tais como reserva de banda.

Para o roteamento orientado a *workspace*, não será obrigatória a reserva de banda, diferentemente do circuito lógico da telefonia. Contudo, considerando que há vários tipos de aplicações, com diferentes requisitos, o roteamento aqui proposto deverá ser capaz de atendê-los, sendo que reserva de banda, por exemplo, passa a ser umas das possibilidades de requisitos.

Em relação à arquitetura Internet, uma característica da camada de Transporte é a orientação a conexão (*Connection Oriented*) no caso do protocolo TCP[5], enquanto na camada de Rede, o Protocolo IP (*Internet Protocol*) não é orientada a conexão (*Connec-*

[5] O protocolo TCP define as conexões de tranportes como Circuitos Virtuais (*Virtual Circuit*)

tionless) (WETHERALL; TANENBAUM, 2013). O argumento dos pesquisadores é que a camada de rede deve oferecer apenas o serviço de roteamento (além da transmissão de pacotes), e os demais serviços devem ser tratados pelas camadas superiores.

Na Internet, ordenação de pacotes, controle de fluxos e erros, retransmição de dados, etc são funcionalidades das camadas superiores à camada de Rede. Essa abordagem é contra-argumentada por pesquisadores adeptos das redes de circuitos lógicos, que defendem não ser possível garantir tráfego com qualidade de serviço caso a camada de rede não realize algumas dessas funcionalidades. No entanto, a Internet evoluiu ao longo dos anos mantendo-se fiel ao protocolo IP em relação a outras arquiteturas, como a ATM (*Asynchronous Transfer Mode*), por exemplo.

A principal característica das redes datagramas (*Connectionless*) está relacionada ao tratamento que um pacote recebe enquanto é enviado de roteador para roteador. Quando um *host* deseja se comunicar, é necessário conhecer o identificador do *host* de destino. Dessa forma, os dados são encapsulados em pacotes que contém o endereço IP (identificador) de origem e o endereço IP de destino, sendo que cada endereço IP identifica um *host*[6].

À medida que um pacote trafega pela Internet, baseado no endereço IP de destino do pacote a ser retransmitido, cada roteador deve consultar sua tabela local para escolher a rota de saída do pacote. Observe-se a diferença da transmissão de pacotes na Internet em relação a circuitos lógicos: nenhum circuito foi pré-estabelecido, ou seja, em cada roteador haverá uma consulta para determinar a rota e durante a comunicação o roteador pode decidir alterar a sua rota baseado nos algoritmos de roteamento que serão descritos mais adiante (WETHERALL; TANENBAUM, 2013).

Redes datagramas não necessitam de estabelecimento de circuito, podendo, portanto, iniciar suas transmissões à medida que pacotes sejam disponibilizados pelas aplicações. Circuitos lógicos consomem tempo para estabelecimento de conexões, antes que quaisquer pacotes possam ser transmitidos. Por outro lado, redes datagramas consomem tempo de roteamento a cada roteador, sendo que circuitos lógicos apresentam tempo de processamento significativamente menor a cada retransmissão.

Nas redes datagramas, a descoberta das rotas é feita no momento em que um pacote está trafegando na rede. O roteamento orientado a *workspace* proposto no presente trabalho, não usará o conceito de roteamento *in-band*, ou seja, as rotas serão estabelecidas antes que os dados comecem a ser trafegados. Para isso serão utilizadas características das redes definidas por software.

2.1.3 Algoritmos de Roteamento

A Internet utiliza diversos algoritmos para realizar roteamento, contando com diferentes abordagens para cada tipo de ambiente. O ambiente pode ser um mesmo domínio,

[6] Quando se tratar de endereço IP *Unicast*

com topologias conhecidas ou entre redes pertencentes a diferentes domínios. No geral, todos os algoritmos de roteamento mais utilizados pertencem a duas classes: Vetor de Distância (*Distance Vector*) ou Estado de Enlace (*Link State*).

A diferença entre as classes de roteamento está no nível de conhecimento que cada roteador armazena, podendo ser informações a respeito da topologia completa da rede ou apenas parcial. Outros algoritmos são variantes que utilizam as duas classes em conjunto, como por exemplo, algoritmos para determinar o menor caminho ou para enviar um *broadcast*[7] para todos os nós da rede.

Essa subseção vai apresentar o funcionamento das classes de roteamento e a próxima subseção apresentará os protocolos utilizados mais comumente, os quais apresentam características de uma das classes.

A primeira classe, Estado de Enlace, permite que cada nó da rede armazene informações sobre toda a topologia, sabendo qual a melhor rota entre todos os nós (MOY, 1998), o que pode trazer algumas vantagens na prática. O primeiro passo para que o estado de enlace funcione é determinar o custo entre vizinhos. Na prática o custo é diretamente proporcional à distância entre dois nós.

Normalmente pode-se estabelecer a distância entre dois nós de diversas formas, sendo as mais comuns: distância física entre os nós (em quilômetros, por exemplo); e, o tempo médio de resposta (em ms, por exemplo) para que um pacote trafegue de um nó a outro. Em contrapartida, pode-se utilizar o número de *hops* para a distância completa, nesse caso o custo entre arestas deve ser sempre um.

Uma vez determinado o custo dos enlaces, cada roteador armazena o endereço de seus vizinhos e o custo para se chegar a cada vizinho. Dessa forma, o passo seguinte, de cada roteador no estado de enlace, é enviar um pacote para seus vizinhos, contendo os custos aprendidos. Assim, todos os roteadores passam a saber sua distância para os vizinhos de seus vizinhos, e assim sucessivamente.

No passo final do estado de enlace, cada roteador calcula o menor caminho para se chegar a todos os outros. O cálculo do menor caminho pode ser feito utilizando-se qualquer algoritmo tradicional, sendo o proposto por Dijkstra o mais utilizado (WETHERALL; TANENBAUM, 2013).

A Figura 1 apresenta uma topologia de rede e um exemplo de como os nós podem enviar informações aprendidas dos vizinhos para os demais nós. Na Figura 1, observa-se quatro roteadores: A, B, C e D. Entre cada roteador há um peso, atribuído baseado na distância entre eles. Como exemplo, pode-se citar o peso de valor 2 entre os nós A e B. Acima de cada roteador há uma tabela contendo os nós de saída e seus pesos. O roteador então deve enviar essa tabela aos seus vizinhos e cada vizinho aos demais, seguindo os passos do algoritmo.

[7] Enviar uma mensagem para todos os receptores ao mesmo tempo, i.e., enviar um pacote para todos os nós da rede.

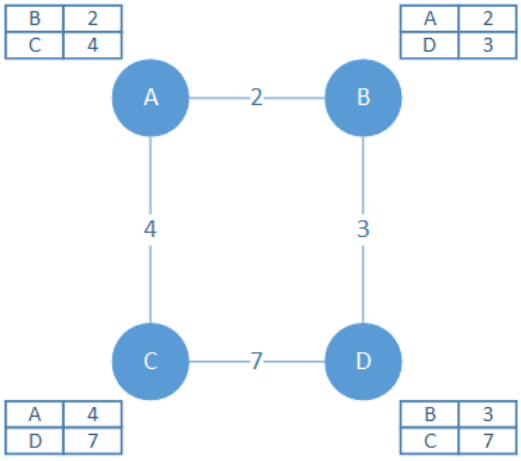

Figura 1 – Roteamento Estado de Enlace

A segunda classe de roteamento, Vetor de Distância, também é utilizada na Internet e permite que os roteadores armazenem a melhor saída para os demais. A tabela de melhor saída é atualizada dentro de certo tempo, baseando-se em troca de mensagens entre vizinhos de cada roteador. A desvantagem desse algoritmo é o tempo gasto para conversão, que pode ser um problema quando algum enlace está *down*.

A Figura 2 mostra uma topologia e a tabela com os custos recebidos de cada vizinho armazenados em um determinado roteador. Na Figura 2 as duas tabelas estão armazenadas no roteador A. Observe-se na tabela da esquerda que o roteador A armazena o custo que cada um de seus vizinhos possui para chegar até todos os roteadores na topologia. Após esse cálculo, o roteador A gera a segunda tabela, mais a direita na figura, que contém o custo e a saída que o roteador A deve seguir para enviar informações a qualquer nó da rede. Assim como o roteador A, os demais roteadores possuem tabelas similares (XU; DAI; GARCIA-LUNA, 1997) (REKHTER; LI; HARES, 2006).

	B	C
A	2	4
B	0	6
C	6	0
D	3	7

Dest	Custo	Saída
A	0	-
B	2	B
C	4	C
D	5	B

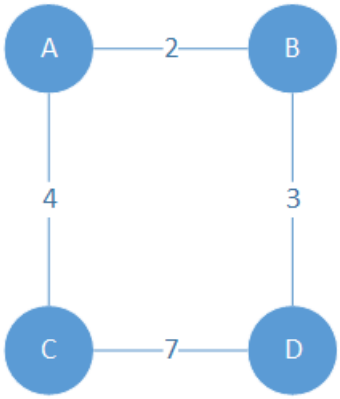

Figura 2 – Roteamento Vetor de Distância

O roteamento orientado a *workspace* tem a tendência de usar características multi-

objetivo de ambas as classes de roteamento descritas acima, e essas características, bem como os algoritmos propostos serão descritos no Capítulo 4.

2.1.4 Roteamento Intra/Inter-domínio

Um dos pontos chave para se realizar roteamento na Internet é decidir qual a melhor forma de rotear intra e inter-domínio. O roteamento intra-domínio, também conhecido como *intranetworking*, é aquele realizado em uma topologia conhecida e controlada dentro de uma mesma rede, como por exemplo a rede de uma empresa, universidade ou operadora. O roteamento inter-domínio é realizado quando se faz necessário que os pacotes trafeguem entre diferentes redes. A arquitetura utilizada como base para o presente trabalho depende de roteamento intra e inter-domínio.

A vantagem do roteamento intra-domínio está no fato de que a topologia da rede é conhecida, portanto um grafo completo da rede pode ser montado, facilitando o uso dos mais diversos tipos de algoritmos, controle de congestionamento nos enlaces, identificação do estado desses enlaces, etc.

Um domínio na Internet é conhecido como AS (*Autonomous System*) e, dentro de um AS, é mais comum o uso de algoritmos Estado de Enlace, sendo o protocolo mais utilizado o OSPF (*Open Shortest Path First*) (MOY, 1989) (MOY, 1998). O OSPF é um protocolo de uso geral, aberto na comunidade, que consegue se adaptar às mudanças na topologia e fazer balanceamento de carga, evitando congestionamentos nos enlaces.

O protocolo OSPF mapeia a rede em um grafo e utiliza o algoritmo Estado de Enlace, que calcula o menor caminho entre todos os roteadores. Caso sejam encontrados dois ou mais caminhos de mesmo tamanho, nenhum é descartado e é utilizada uma técnica chamada de ECMP (*Equal Cost MultiPath*) (THALER; HOPPS, 2000a), que balanceia a carga entre os caminhos de mesmo tamanho.

Uma característica relevante no protocolo OSPF é que um grande AS pode ser dividido em vários subdomínios e o próprio OSPF possui versatilidade para trabalhar com essa divisão: ele é preparado para enviar os pacotes aos roteadores de borda que ficam entre os subdomínios, podendo assim chegar dentro dos subdomínios e o estado de enlace executar normalmente no subdomínio.

O protocolo OSPF especifica ligeira diferença do estado de enlace, que evita enviar muitas mensagens de controle na rede, e para isso faz com que os roteadores não precisem se comunicar com todos os seus vizinhos, apenas com roteadores designados para essa função (THALER; HOPPS, 2000b).

O roteamento inter-domínio (*internetworking*) permite conectar um AS a outros e tem uma importância significativa na Internet atual, principalmente no que diz respeito a políticas. A característica principal é que um AS pode não querer seu tráfego passando por determinados AS e, portanto, para chegar a um destino em outra rede, uma determinada rota pode ter que ser evitada.

Um determinado AS pode não quer expor a topologia de sua rede para outro(s), como é o caso de operadoras de telecomunicações, que podem simplesmente não querer que suas concorrentes conheçam sua topologia. Para que exista a comunicação entre diferentes ASs, é necessário um protocolo único de roteamento, que todos os ASs respeitem para que a interconexão entre eles seja feita. Atualmente esse protocolo é o BGP (*Border Gateway Protocol*) (REKHTER; LI; HARES, 2006).

O BGP é executado nos roteadores de borda, que são responsáveis por interconectar diferentes ASs. Um exemplo de interconexão é quando clientes de um determinado ISP (*Internet Service Provider*) se comunicam com clientes de outro ISP. Nesse caso, o AS do primeiro ISP conecta-se com o AS do segundo ISP para oferecer o tráfego de ponta a ponta.

Um AS não conhece a topologia de outro, e para isso utiliza o BGP. Outro caso de utilização do BGP é quando um AS intermediário é usado para oferecer conexão entre dois ASs. Nesse caso um AS compra tráfego de outro, pois não deseja se conectar diretamente com um determinado AS.

A comunicação entre um cliente doméstico e seu ISP não necessita do protocolo BGP, pois o enlace de saída para Internet é único. Entretanto, quando o cliente é pessoa jurídica, e decide utilizar a técnica de *multihoming*[8] (WETHERALL; TANENBAUM, 2013), é aplicável o uso do protocolo BGP.

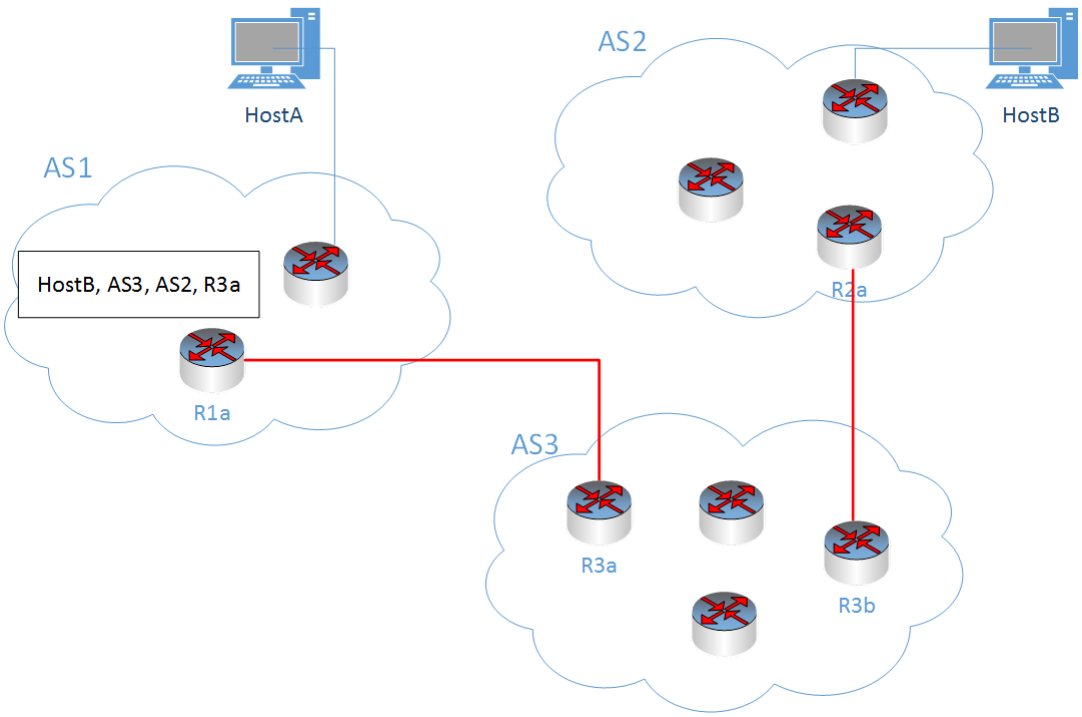

Figura 3 – Funcionamento do Protocolo BGP

[8] Múltiplos enlaces de saída para garantir confiabilidade quando um link cair

O protocolo BGP utiliza roteamento por vetor de distância e, então, cada roteador de borda conhece os demais roteadores de borda no seu próprio AS. Dessa forma, um roteador armazena registros contendo os AS pelos quais ele passa até alcançar um destino, incluindo o próximo *hop* (roteador de borda) (REKHTER; LI; HARES, 2006).

A Figura 3 representa a interconexão de três ASs diferentes (AS_1, AS_2 e AS_3). O AS_1 não está diretamente conectado ao AS_2, os roteadores de borda são apenas R_{1a}, R_{2a}, R_{3a} e R_{3b}, sendo que não há conexão entre roteadores de AS_1 e AS_3. Observa-se na figura que caso o $Host_A$ (ligado ao AS_1), queira enviar uma informação para o $Host_B$, o tráfego deverá sair pela borda de AS_1, que é o roteador R_{1a}.

Na Figura 3, acima de R_{1a}, há um retângulo contendo o registro que R_{1a} possui em relação ao $Host_B$. Nesse registro há o caminho de AS necessário para chegar até o $Host_B$, além do próximo *hop* (R_{3a}). O AS_1 conhece apenas R_{3a} em AS_3, os demais roteadores e topologia de AS_3 não são divulgados.

Para determinar por qual caminho os pacotes serão enviados, o protocolo BGP utiliza a menor rota, onde uma rota é menor que outra considerando-se a quantidade de ASs pelo caminho. Isso pode ser um caminho mais longo caso um AS maior esteja na mesma rota, mas no geral o funcionamento é o melhor. Além disso um AS não sabe a quantidade de roteadores que o pacote vai trafegar dentro de outro AS (WETHERALL; TANENBAUM, 2013).

2.2 Redes Definidas por Software

As Redes Definidas por Software são conhecidas por sua sigla em inglês SDN (*Software Defined Networking*) (FARHADY; LEE; NAKAO, 2015) (KREUTZ et al., 2015), e representam uma forma diferente de operar uma rede. A principal característica nessas redes é a separação do plano de controle do plano de dados.

Através das seções anteriores é possível notar que a Internet funciona no modelo *in-band*, como por exemplo no roteamento, onde um pacote trafegando dados possui também informações para controle da rede, e cada roteador processa no mesmo plano o controle da rede (determinar a rota sainte) e os dados (encaminhar o pacote ao enlace de saída).

Em uma rede SDN o controle é feito *out-band*, ou seja, feito por elementos diferentes daqueles que realizam encaminhamento dos dados. O algoritmo de roteamento proposto no presente trabalho funciona tendo como base uma arquitetura de Internet do Futuro totalmente SDN, e por isso nessa seção serão descritos os principais aspectos conceituais de SDN, suas implementações disponíveis no mercado e os protocolos já consolidados na comunidade científica.

2.2.1 OpenFlow

A tecnologia OpenFlow (KERNER, 2012) (MCKEOWN et al., 2008) pode ser considerada uma implementação de SDN, consolidada e utilizada por diversos pesquisadores e fabricantes. O conceito de roteamento orientado a *workspace* independe da tecnologia OpenFlow para funcionamento, porém o estudo de caso deste trabalho foi feito utilizando alguns componentes dessa tecnologia.

O termo OpenFlow se refere ao *switch* OpenFlow, ao protocolo OpenFlow e ao controlador OpenFlow, os quais possuem papéis diferentes. Um *switch* OpenFlow é uma construção SDN feita para permitir a separação do plano de controle do plano de dados. Nesse tipo de *switch*, um dado tem seu controle feito por um elemento externo ao *switch*, conhecido como controlador, e a comunicação entre *switch* e controlador é feita através do protocolo Openflow.

A Figura 4 representa os elementos da Tecnologia OpenFlow. O *Switch* OpenFlow estabelece uma associação segura para comunicação com o Controlador, que contém tabelas de fluxos (*Flow Tables*). Um *frame* recebido pelo *switch* é condicionado à análise na tabela de fluxos. Caso exista um fluxo para aquele *frame*, ele é encaminhado de acordo com o fluxo, caso contrário, ele é enviado ao Controlador, que decidirá seu destino.

A comunicação do *switch* OpenFlow com o controlador é feita por meio do protocolo OpenFlow (KONTESIDOU; ZARIFIS, 2009), que utiliza uma associação segura SSL (*Secure Sockets Layer*). Pode-se dizer que o primeiro *frame* de cada *stream* é enviado ao controlador, que estabelecerá o respetivo fluxo (se houver) a ser utilizado por todos os demais *frames*.

Quando há uma regra de *switching* definida, o controlador configura o *switch*, que insere a regra em sua tabela de fluxos (i.e., cria um fluxo no *switch*), e, deste modo, os demais *frames* dessa *stream* são chaveados de acordo com a regra recém configurada e não mais são encaminhados ao controlador OpenFlow.

Na referência original do OpenFlow, o procedimento determinado pelo controlador e armazenado nos *swiches* é chamado de Ações. Um exemplo de ação poderia estar relacionado ao endereço MAC (*Media Access Control*) do *frame*, isto é, todos os *frames* com determinado MAC são destinados a uma porta específica do *switch*, e assim sucessivamente.

Na versão 1.0 do OpenFlow, usada no presente trabalho, os possíveis parâmetros a serem usados na criação de ações são: Porta de entrada, Endereço Ethernet de Origem, Endereço Ethernet de Destino, Ethernet *Type*[9], VLAN id, VLAN *priority*, Endereço IP de Origem, Endereço IP de Destino, *Protocol*[10], TOS (*Type of Service*), Portas[11] de Origem e de Destino (KONTESIDOU; ZARIFIS, 2009). O presente trabalho faz uso das ações

[9] Este campo do *frame* Ethernet é frequentemente referenciado como *EtherType*

[10] Refere-se ao campo *Protocol* do cabeçalho do Protocolo IP

[11] Refere-se ao campo *Port* da interface abstrata *Sockets* utilizada pelos protocolos TCP/UDP

OpenFlow, baseadas no endereço Ethernet, para implementar o conceito de *Workspace*, que será apresentado no Capítulo 3.

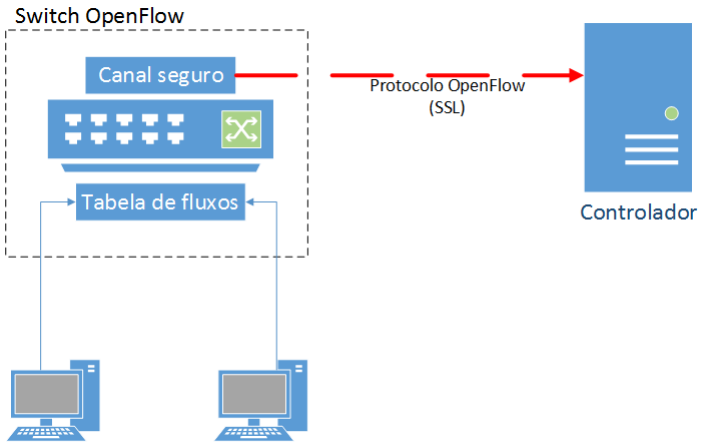

Figura 4 – OpenFlow

A tecnologia OpenFlow oferece flexibilidade para testar novos protocolos de rede utilizando a própria rede em produção. Uma empresa pode não permitir que novos protocolos sejam testados no seu ambiente, com receio de ter problemas na rede de produção. Com o OpenFlow, pode-se determinar que apenas certo tipo de tráfego será alterado, como por exemplo, pode-se criar um novo protocolo e testá-lo criando fluxos no *switch* onde pacotes com endereço IP de origem começando com determinado padrão devem receber um tratamento específico, de acordo com o novo protocolo sendo testado.

Além do conceito de Ação, que é utilizado nos diversos *switches*, o roteamento proposto neste trabalho serve a uma arquitetura com conceitos semelhantes a um controlador OpenFlow. Por este motivo, os principais controladores OpenFlow apresentados na literatura e no mercado serão brevemente descritos.

2.2.2 Controladores OpenFlow

Controladores são um elemento fundamental na arquitetura OpenFlow pois materializam a separação dos planos de controle e de dados. A ideia por trás do controlador é simples: todo tráfego de controle deve ser enviado a um controlador, dessa forma ele poderá decidir o que será feito com o tráfego.

O Controlador OpenFlow é basicamente um servidor, que pode ser implementado por meio de computadores de propósito geral, sendo que todo *switch* em operação na rede deve estar ligado fisicamente a um controlador. Nessa seção, alguns controladores serão descritos, pois eles utilizam conceitos similares aos da arquitetura usada para desenvolvimento do trabalho desta dissertação.

Dentre os primeiros controladores apresentados destaca-se o NOX (NOXREPO, 2014), implementado em linguagem C++, com características de velocidade e I/O assíncrono.

O NOX foi testado em três importantes distribuições Linux, utilizadas mundialmente, tais quais Ubuntu, Debian e RHEL (*Red Hat Enterprise Linux*). A topologia da rede, ou seja, o grafo de *switches* e respectivos enlaces, é disponibilizada pelo NOX, dessa forma os pesquisadores podem fazer uso dessas características do controlador, visto que é disponibilizada uma API (*Application Programming Interface*) em C++, para uso geral (GUDE et al., 2008).

A equipe do NOX também é responsável por outro controlador, o POX (NOXREPO, 2013) desenvolvido em linguagem Python,[12] que foi testado em ambientes Linux, MacOS e Microsoft Windows. Descoberta da topologia da rede também está presente no POX, dentre outras ferramentas do NOX. A desvantagem está apenas na latência e *throughput*, se comparado ao NOX.

Outro controlador, além do NOX e POX, o Floodlight foi desenvolvido na linguagem Java (Big Switch Networks, 2014) e merece especial atenção, pois foi o controlador utilizado nas implementações do presente trabalho. Tal como os controladores já citados, o Floodlight foi criado para ser utilizado em ambientes com *switches* OpenFlow físicos ou virtuais.

Segundo seus criadores, o Floodlight foi desenvolvido para ser um controlador de alto desempenho fácil de ser usado, que conseguisse manipular redes OpenFlow e redes não OpenFlow simultaneamente. É utilizado por importantes iniciativas, tais como produtos comerciais da Big Switch Networks e pelo OpenStack, ambos apresentados oportunamente.

O Floodlight não é apenas um controlador OpenFlow, mas também um conjunto de aplicações construídas sob o controlador (Big Switch Networks, 2012). A Figura 5 exemplifica a flexibilidade concedida pela API, inclusive com interface REST (*Representational State Transfer*) aberta para conexão com outras plataformas e sistemas.

Na Figura 5 observa-se diversos blocos do controlador que podem ser aproveitados pelos programadores, como por exemplo *Topology Manager*, que contém informações da topologia dos *switches* sob seu controle e algoritmos tradicionais usados em roteamento, como por exemplo menor caminho. A utilização do *Topology Manager* e demais módulos do controlador são de fácil aprendizado para programadores Java.

Os códigos do projeto Floodlight estão sob licença *open source*, podendo ser utilizados pela comunidade em geral. Além do controlador Floodlight, outros exemplos de controladores em Java, similares, são Beacon (ERICKSON, 2014) e Maestro (CAI, 2014).

Além de SDN, a comunidade vem desenvolvendo muitos trabalhos relacionados ao conceito de NFV (*Network Functions Virtualization*) (HAN et al., 2015), que prega a virtualização de funções da rede. As operadoras de telecomunicações têm interesse nos resultados dessas pesquisas, visto que hoje os elementos de rede dessas operadoras são as chamadas caixas pretas, isto é, a operadora compra uma plataforma ou central de um

[12] Incluindo uma API em Python para facilitar o uso do controlador

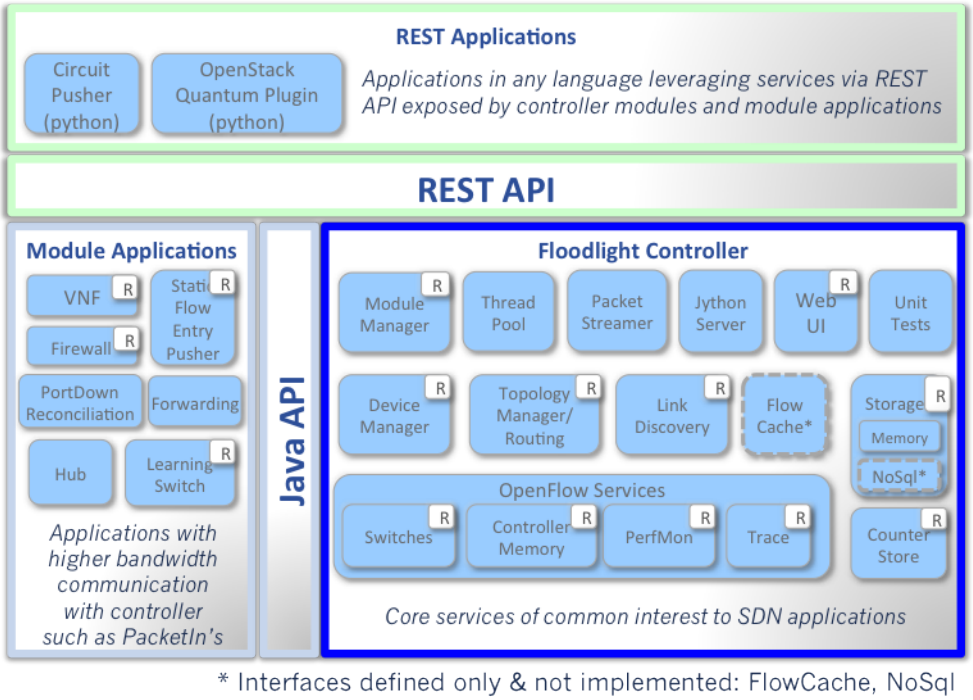

Figura 5 – Controlador Floodlight

Fonte: (Big Switch Networks, 2015)

grande fornecedor e a mantém até que se evolua para outro tipo de tecnologia, e então a operadora compra uma nova caixa preta para a nova tecnologia, e assim sucessivamente.

NFV permite que a operadora virtualize suas plataformas e centrais, não precisando comprar novo hardware sempre que uma nova tecnologia surgir. Muitos pesquisadores consideram SDN e NFV como tecnologias complementares (WOOD et al., 2015), sendo que vários projetos foram criados visando ambas.

Um dos principais projetos para SDN e NFV é o OpenDaylight (OpenDaylight, 2014b), criado para ser uma plataforma de uso geral, contendo conceitos para uma rede programável. É licenciado como um projeto *open source* tendo sido usado com sucesso em diversas redes universitárias, além de ser patrocinado por importantes fornecedores de software e hardware relacionados a redes de computadores (OpenDaylight, 2014a) (MATSUMOTO, 2014).

Como características importantes do OpenDaylight pode-se destacar uma API REST e alta disponibilidade através de *clustering*. O OpenDaylight pode ser utilizado pelo OpenStack para gerenciamento da rede. O OpenStack (OpenStack, 2015) é um sistema de operação em *cloud* aberto que fornece recursos para controle de rede, *storage* e computação para *datacenters*, de uso global.

A arquitetura de referência para o presente trabalho possui características *Carrier Grade*, que também estão presentes no OpenDaylight. Outro trabalho baseado em SDN

de destaque com funções *Carrier Grade* é o projeto ONOS (*Open Network Operating System*), desenvolvido para fornecer alta disponibilidade, desempenho e escalabilidade horizontal (ON.LAB, 2014a) (ON.LAB, 2014b).

Do mesmo modo, o ONOS trabalha como um controlador OpenFlow, com característica de permitir mais de uma instância para o controlador, criando um núcleo distribuído. O ONOS trás um conceito chamado *intent*, que é similar ao de requisitos (utilizado neste trabalho e que será descrito no Capítulo 3).

O conceito de *intent* permite que os *hosts* possam traduzir múltiplos objetivos para a rede, visando usar determinadas políticas na configuração. O ONOS pode ser considerado como um NFaaS (*Network Functions as a Service*), podendo interoperar com conceitos de NFV e sendo assim um projeto de referência para o futuro em operadoras de telecomunicações.

ONOS introduziu uma melhora no protocolo MPLS (*Multiprotocol Label Switching*) (ROSEN; VISWANATHAN; CALLON, 2001) para que o roteamento seja feito no plano de controle, característica essa que é a base para o roteamento orientado a *workspace* proposto no presente trabalho (ON.LAB, 2014b).

Além do ONOS, outro projeto com características de sistema distribuído é o Onix (KOPONEN et al., 2010), proposto pelos mesmos criadores do NOX. O Onix foi criado como uma plataforma de controle distribuído para ser utilizado em redes de larga escala.

Segundo Koponen et al. (2010), os maiores desafios de uma plataforma para redes em larga escala são: generalidade, escalabilidade, confiabilidade, simplicidade e desempenho do plano de controle. A arquitetura do Onix basicamente possui uma infraestrutura física, contendo roteadores e *switches* que são ligados ao módulo Onix através de uma infraestrutura de conectividade.

Um controle lógico é utilizado para verificar falhas no Onix. Com ele é possível alterar a instância que controla os elementos de rede ou manter mais de uma instância controlando os elementos, ao mesmo tempo. Para tratar de escalabilidade são usados os conceitos de particionamento e agregação, que podem ser usados futuramente no roteamento orientado a *workspace*, para armazenamento e manutenção de grandes topologias com muitos *workspaces*.

Este trabalho também se propõe a fornecer suporte futuro para requisitos de operadoras de telecomunicações, tais como segurança ou escalabilidade. Nesse cenário se destaca o projeto Big Network Controller (NETWORKS, 2014), que é um controlador OpenFlow comercial, baseado no Floodlight. A criação do Big Network Controller se deu baseada nos requisitos encontrados em *datacenters*, como necessidade de monitoramento e segurança, além de escalabilidade. Esse controlador foi usado por aplicações como Big Virtual Switch e o Big Tap, tendo como característica principal a flexibilidade.

Nos dias atuais, a virtualização é uma realidade nas grandes empresas e, por isso, um trabalho importante buscando construir um *switch* virtual multicamadas é o Open

vSwitch (Open vSwitch, 2012). A ideia do projeto é permitir que programadores escrevam programas para controlar a rede, sendo de fato o mesmo conceito de SDN.

A equipe do Open vSwitch afirma que a computação tem evoluído muito nas últimas décadas, porém as redes de computadores não evoluíram na mesma proporção (Open vSwitch, 2012). Um exemplo é que há 15 anos atrás a configuração de um novo servidor em um *datacenter* precisava da instalação de sistema operacional e infra, enquanto nos dias atuais uma nova instância de máquina virtual é criada em poucos minutos. Dessa forma a virtualização de um *switch* é o ponto central do projeto, o que está relacionado com questões de pesquisas recentes com temas no âmbito de NFV.

O Open vSwitch pretende incorporar outras plataformas, como o OFConnect, uma biblioteca *open source* para redes OpenFlow. Com o OFConnect é possível criar um ambiente adicionando novos *switches* e controladores OpenFlow, escalando de forma horizontal novos *switches* para lidar com requisitos de desempenho. Questões especificamente relacionadas ao roteamento também são discutidas nos congressos do Open vSwitch, como por exemplo o MPLS LDP.

A Tabela 1 apresenta um resumo das características dos controladores citados. É possível observar na tabela a similaridade entre as funcionalidades desses controladores.

Tabela 1 – Características dos controladores OpenFlow.

Controlador	**API**	***OpenSource***	**Linguagem**	**Controle distribuído**
NOX	Sim	Sim	C++	Não
POX	Sim	Sim	Python	Não
Floodlight	Sim	Sim	Java	Não
OpenDaylight	Sim	Sim	Java	Não
ONOS	Sim	Sim	Java	Sim
Onix	Sim	Sim	C++	Sim
Big Network Controller	Sim	Não	Java	Não
Open vSwitch	Sim	Sim	C	Não

2.3 Propostas para Internet do Futuro

Atualmente pode-se considerar que SDN é uma realidade, visto que muitos projetos e plataformas comerciais já vêm com módulos para utilização de controladores para gerenciar a rede. Além disso, NFV tende a ter uma grande importância nas próximas gerações de telecomunicações, visto que cada vez mais o ambiente de telefonia converge para TI (Tecnologia da Informação).

À luz dos conceitos apresentados por SDN e NFV, várias propostas foram criadas nos últimos anos para suportar os requisitos do futuro. Sabe-se que a Internet atual não consegue trabalhar bem com alguns desses requisitos, como mobilidade, segurança, eficiência energética, voz e vídeo em tempo real, dentre outras. Para preparar a rede para lidar com essas questões, pesquisadores vêm trabalhando em novos modelos e arquiteturas, inclu-

sive *clean slate*, protocolos e *frameworks*. Serão apresentadas algumas dessas propostas, principalmente aquelas relacionadas a roteamento.

Um dos principais projetos relacionados à pesquisa em Internet do Futuro dentro da área de roteamento é o RouteFlow (NASCIMENTO et al., 2011), um projeto *open source*, com intuito de fornecer roteamento virtualizado em ambiente OpenFlow.

No RouteFlow, um controlador OpenFlow, tal como Open vSwitch, Floodlight ou Nox, por exemplo, é usado para controlar roteadores virtuais e um elemento extra chamado *Routing Engine* é responsável pode criar uma FIB (*Forwarding Information Base*) de acordo com os protocolos OSPF ou BGP (os quais podem ser configurados). Após isso, processos RouteFlow coletam essas regras e as aplicam nos *switches* OpenFlow para que esses façam o encaminhamento dos *frames*.

A Figura 6 mostra a topologia do RouteFlow. Importantes características do projeto são: não modificar a pilha dos protocolos de roteamento, visto que os pacotes para atualização da FIB rodam em ambiente virtualizado; é fácil usar múltiplos controladores; e independe do hardware no plano de dados, sendo assim possível usar hardware de diferentes fabricantes.

As ações que o RouteFlow utiliza nas tabelas de fluxo dos *switches* OpenFlow são determinadas pelos endereços MAC e IP de destino, juntamente com a máscara de sub rede. Observe-se que o RouteFlow não é uma arquitetura de rede completa, mas sim uma combinação de tecnologias bem consolidadas, podendo ser considerado um exemplo de NaaS (*Network as a Service*).

Em princípio, o RouteFlow possui similaridade com o roteamento proposto neste trabalho, dependendo, contudo, das redes legadas, i.e., possui grande dependência do protocolo IP. A ideia por trás do projeto RouteFlow é justamente otimizar o IP através de SDN. Algumas limitações do RouteFlow também serão observadas futuramente nos trabalhos com roteamento orientado a *workspace*, sendo uma delas a limitação em escala consequente do tamanho da tabela de fluxos dos *switches* OpenFlow.

Outra abordagem com conceitos similares à arquitetura de Internet do Futuro usada no presente trabalho é o FlowVisor (SHERWOOD et al., 2009), que permite diversos controladores controlarem a rede ao mesmo tempo. O projeto pode ser considerado como um controlador OpenFlow, agindo de forma transparente para que a rede seja dividida em diferentes partes (*slices*), que serão controladas por diversos controladores.

Assim como no RouteFlow, o FlowVisor também utiliza o conceito de virtualização da rede. Entre *switches* OpenFlow e diversos controladores entra em cena o módulo FlowVisor que armazena diferentes políticas baseadas em regras que serão aplicadas em partes da rede. Dessa forma, um mesmo *switch* conversa com mais de um controlador para definir os fluxos de encaminhamento, pois o FlowVisor trata de fatiar a rede através de suas políticas.

A ideia do módulo FlowVisor é ser totalmente transparente, assim as políticas no

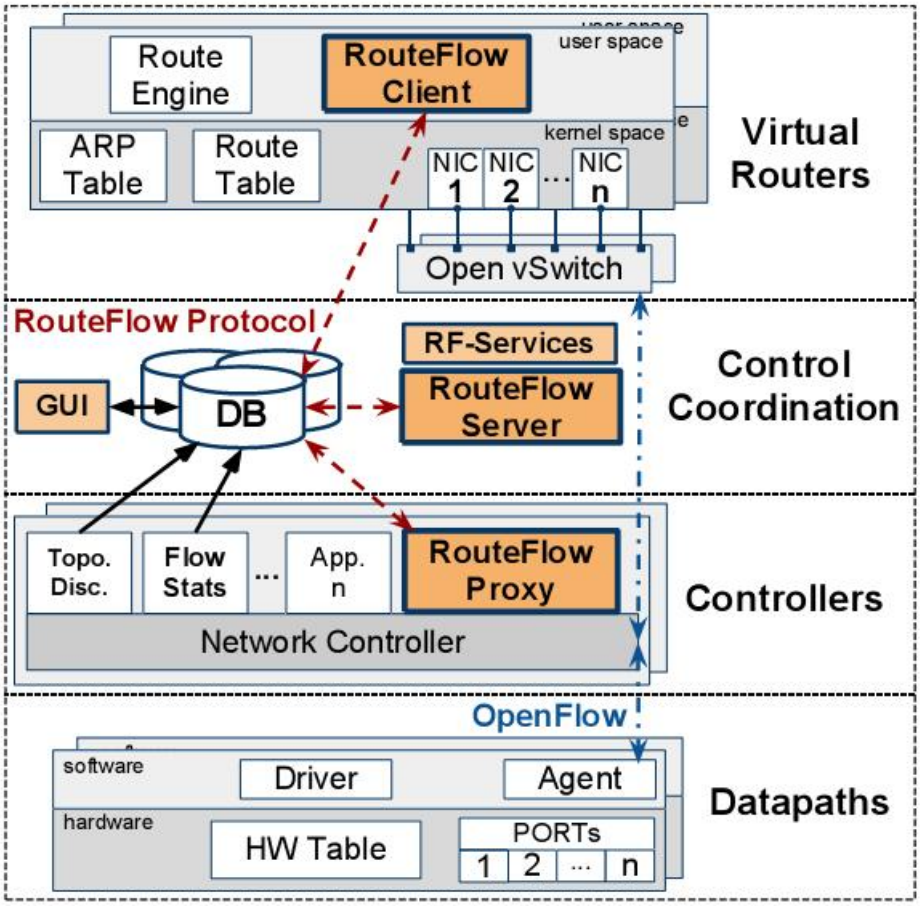

Figura 6 – RouteFlow

Fonte: (RouteFlow, 2015)

ambiente de produção em uma rede não são afetadas, e um *switch* OpenFlow não terá seu funcionamento alterado, pois ele é configurado para enviar suas mensagens de controle para um controlador específico. O FlowVisor intercepta as mensagens de controle, alterando suas regras.

Outra característica do FlowVisor é ser totalmente isolado, assim, um controlador FlowVisor pode controlar diversos *switches* ao mesmo tempo. Também outro controlador FlowVisor pode controlar o primeiro controlador FlowVisor, e assim por diante, em um sistema recursivo, onde controladores FlowVisor interceptam e aplicam suas políticas antes que a mensagem de controle chegue ao controlador OpenFlow original.

A característica de módulos FlowVisor, interceptando outros módulos FlowVisor, pode ser aplicada em um grafo de inter-domínio na arquitetura do trabalho aqui proposto, onde os controladores do roteamento orientado a *workspace* podem possuir outros controladores intermediários com os quais haverá comunicação até chegar ao controlador *Master*, que resolverá questões de busca de *workspaces* na rede inter-domínio.

A arquitetura para construção de um roteamento orientado a *workspace* possui grande

ligação com o conceito de ontologia na rede. Em trabalhos iniciais da arquitetura pensava-se que no futuro o roteamento poderia ser baseado na própria ontologia, que cria classificadores na rede, e quando um novo *workspace* é criado ou uma nova entidade (*host* ou aplicação) entra na rede ela é automaticamente colocada na topologia, visto que os classificadores tratariam de reconfigurar os *switches* para tal.

O conceito de se utilizar um elemento externo para gerenciar a rede não foge muito do que é proposto pela comunidade. Um exemplo é o SNAC (*Simple Network Access Control*) (OpenFlow, 2015), o qual utiliza um gerenciador de políticas baseado na web para determinar o funcionamento da rede. Nessa proposta, novos *hosts* que entram na rede são direcionados para o SNAC com intuito de serem gerenciados.

No SNAC, os administradores da rede, através de uma interface gráfica pela web, podem determinar como será o roteamento dos novos *hosts*, baseando-se em controladores OpenFlow, como o NOX. A diferença para o roteamento orientado a *workspace* é que esse não será feito baseado nos *hosts*, mas sim no ambiente de comunicação entre eles, o *workspace*.

Um conceito que foi explorado nos últimos anos e pode ser um objetivo importante relacionado a roteamento é a eficiência energética. Pesquisadores em todo o mundo estão buscando formas de economizar energia pensando em sustentabilidade no futuro, além do próprio custo financeiro da energia.

O roteamento na Internet tem ligação com alto custo energético, visto que a rota por onde os pacotes passam pode consumir muita energia. Basicamente, as pesquisas na área de eficiência energética têm dois objetivos: permitir escolher rotas que gastem menos energia; e desligar links que não são muito utilizados, para economizar energia.

Um dos primeiros projetos relacionados à eficiência energética foi o Green OSPF (CIANFRANI et al., 2010), um algoritmo de roteamento do tipo estado de enlace para poupar energia. Segundo os proponentes, mais de 60% dos *links* avaliados pelo algoritmo puderam ser desligados sem maiores problemas. O Green OSPF roda em três passos:

1. Alguns roteadores são escolhidos como *exportadores*, isto é, calculam suas SPTs (*Shortest Path Tree*), que é uma *spanning tree* partindo de si mesmo até chegar em todos os outros roteadores;

2. Os roteadores *não exportadores* também calculam suas SPTs e, usando Dijkstra, determinam uma MPT (*Modified Path Tree*), avaliando as SPTs dos roteadores exportadores, fazendo dessa forma com que alguns enlaces possam ser descartados; e,

3. Neste último passo, é feita uma otimização recalculando os caminhos desses roteadores não exportadores, sendo que os enlaces que puderem ser descartados são desligados, poupando energia.

Uma questão relevante segundo Januraio (2014) é a de monitoramento. Em redes que desligam *links*, para gastar menos energia, fica difícil para os gerenciadores da rede descobrirem se um *link* caiu ou simplesmente foi desligado com intuito de poupar energia.

Além do Green OSPF, há outras abordagens para economizar energia. Dentre elas, pode-se destacar o ElasticTree (HELLER et al., 2010), usado para poupar energia em *datacenters* com grande carga de dados. O ElasticTree tem relação com este trabalho por ter comparado *switches* OpenFlow de diferentes fabricantes em produção, o que pode ser usado no futuro como objetivo do roteamento orientado a *workspace*. A desvantagem é o tempo de resposta, que depende do tempo de se levantar um enlace e do cálculo das rotas.

Outra proposta apresentada na comunidade foi (DU, 2004), que usa roteamento para economia de energia em redes *ad hoc* móveis. Basicamente cada roteamento é feito utilizando menos nós da rede, dessa forma os demais nós não têm gasto de energia naquele roteamento específico.

Outros trabalhos focados em abordagens para poupar energia podem ser citados dentre os quais (CHENG; JIA, 2005), (CHEN et al., 2012) (focados em redes *Wireless*), (KIM; NOEL; TANG, 2013) e (LI; SHANG; CHEN, 2014). Muitas dessas propostas focam em *datacenters*, que são grandes consumidores de energia, devido ao crescimento da carga de dados internamente.

Existe relação entre roteamento e eficiência energética, pois boa parte das propostas para poupar energia buscam aprimorar o roteamento nas redes. (RODRIGUE et al., 2012) inclusive propõe um modelo matemático para analisar o consumo de energia dos protocolos de roteamento em redes *wireless*. Além das propostas acima citadas para economizar energia, (WANG et al., 2014) deixa clara a importância de SDN para esse fim, visto que o controle lógico centralizado nesse tipo de rede permite trabalhar com a questão de consumo de energia dos roteadores.

Não apenas voltadas para eficiência energética, soluções para roteamento usando SDN, tal como a que será proposta no presente trabalho, foram apresentadas na comunidade. Dentre elas, pode-se citar (JIA; WANG, 2013), que propõe um algoritmo unificado para roteamento *unicast* e *multicast* para ser usado em redes de *datacenters*. Em (LEE et al., 2014), o algoritmo para roteamento proposto é efetivo para o problema do roteamento *multicast*, que pode ter várias rotas em aplicações como *streaming* de vídeo.

Em um trabalho mais recente, Lin et al. (2014) propõe um *framework* para roteamento em SDN, utilizando o plano de controle, de forma centralizada, para rotear, conceito muito similar ao roteamento orientado a *workspace*. Os autores apresentam algumas vantagens e desvantagens.

Como desvantagens, Lin et al. (2014) cita o ponto único de falha (SPoF – *Single Point of Fail*), o controlador centralizado, e o já citado anteriormente problema do limite de tamanho da tabela de fluxos em um *switch* OpenFlow. Como vantagens, Lin et al. (2014)

menciona o uso do roteamento no plano de controle, que permite novas configurações serem inseridas facilmente, além da facilidade dos usuários em construir *building blocks* para implantar seus próprios algoritmos.

Tanto as vantagens quanto as desvantagens enumeradas no trabalho de Lin et al. (2014) se aplicam também ao roteamento orientado a *workspace*. A diferença entre as duas propostas está na arquitetura *clean slate* por trás do último.

Owens e Durresi (2014) apresenta um outro problema de SDN em relação ao roteamento no plano de controle, que poderá no futuro ser um problema para o roteamento orientado a *workspace*, pois o controle centralizado poderá ter um problema de escalabilidade caso muitos eventos sejam enviados para processamento no plano de controle. A ideia no caso do roteamento orientado a *workspace* é criar uma distribuição para escalar horizontalmente os controladores SDN.

Outro tipo de mecanismo utilizado no roteamento orientado a *workspace* é o conceito inter-domínio, na prática conhecido como inter-AS. Na literatura, Bennesby et al. (2012) propôs um componente para roteamento inter-AS utilizando SDN. O controlador NOX originalmente introduziu o roteamento para uma topologia interna. A proposta de Bennesby et al. (2012) é expandir esse controlador OpenFlow para rotear inter-AS.

O componente proposto por Bennesby et al. (2012) utiliza muitos conceitos do protocolo BGP, mas em um ambiente no plano de controle. O procedimento desse novo componente para inter-AS é o seguinte: o OpenFlow segue seu procedimento normal, porém, quando um destino não é encontrado no domínio, o componente inter-AS é chamado para verificar se algum outro domínio alcança o prefixo solicitado. O roteamento orientado a *workspace* também utiliza um procedimento parecido quando um *workspace* não é encontrado no domínio local.

O roteamento orientado a *workspace* possui características de *lookup* quando um *workspace* não é encontrado no domínio local. Como um *workspace* não é um *host*, pode-se considerá-lo similar a um conteúdo sendo buscado (similar apenas na busca, não sendo esse o seu conceito).

Alguns trabalhos propõem algoritmos para *lookup* através do roteamento, como é o caso de Behdadfar et al. (2009), que cria uma busca escalável, devido ao grande crescimento das tabelas nos roteadores da Internet, que devem armazenar os prefixos alcançáveis. No trabalho, o armazenamento dos prefixos não é mais feito usando-se um *range*, o que pode trazer benefícios como custo de memória para armazenamento.

No caso específico de busca de conteúdo, algumas propostas giram em torno da DHT (*Distributed Hash Table*), para armazenamento distribuído dos conteúdos, e algumas propostas, como Sluijs et al. (2009), criam uma estratégia de *cache* entre DHT e a camada de aplicação. O uso de DHT não é descartado pelo roteamento orientado a *workspace*.

Os trabalhos acima relacionados têm buscado resolver questões que serão requisitos na Internet do Futuro, como eficiência energética, uso de SDN para roteamento inter-AS

e escalabilidade para *lookup*. Algumas arquiteturas mais completas, focadas em atender a maioria dos requisitos, também foram propostas nos últimos anos, destacando-se a NIRA (*New Internet Routing Architecture*), a RINA (*Recursive InterNetwork Architecture*), a MobilityFirst e a ETArch (*Entity Title Architecture*). Essas arquiteturas serão relacionadas nos próximos parágrafos, visto a semelhança de alguns conceitos com o trabalho desta dissertação.

A Internet atual, com o uso do BGP, faz com que os pacotes de um usuário entrem em um AS e sejam roteados conforme ordene esse AS. A NIRA (YANG; CLARK; BERGER, 2007) propõe algo um pouco diferente, onde o usuário pode determinar por onde quer que seu pacote trafegue. Dessa forma, o usuário pode decidir a sequência de provedores através dos quais suas primitivas passarão.

Na NIRA, cada usuário pode escolher diferentes provedores dependendo da aplicação que deseja utilizar, o que é uma abordagem interessante para ser seguida também no escopo deste trabalho, que tem a preocupação de permitir que requisitos das aplicações sejam levados em consideração no roteamento, para QoS (*Quality of Service*), e que futuramente podem se transformar em requisitos do próprio usuário, para QoE (*Quality of Experience*).

Além do próprio usuário escolher, a rede poderia encaminhar primitivas através de diferentes caminhos ao mesmo tempo ou o cabeçalho das primitivas poderia conter informações de novas rotas para o caso de falha na rota preferencial escolhida (GANICHEV, 2011). Arquiteturas como a NIRA possuem similaridades com o trabalho aqui proposto, pois o roteamento tratando elementos de rede controlados por diferentes controladores pode ser considerado *inter-domínio*.

A diferença entre o que é proposto na NIRA e no roteamento orientado a *workspace* está no fato de que o esse último assume que o controlador é quem vai decidir as rotas, e não o usuário, visto que entende-se que no futuro as topologias de diferentes domínios não permitirão essa flexibilidade, por questões puramente políticas.

Já na arquitetura RINA (VRIJDERS et al., 2014) (RINA, 2015), a rede é do tipo IPC (*Inter-Process Communication*), eliminando o modelo de camadas da Internet atual. Dessa forma, um número diferente de camadas pode ser usado para gerenciar largura de banda e QoS.

Na RINA, o roteamento é feito na interface do *host* de cada usuário, dessa forma, um nome completo de endereçamento deve ser respeitado para se atingir um destino, fugindo da dependência do IP. Com um *namespace* específico, as características de *multihoming* e mobilidade são naturais para a RINA. A arquitetura do presente trabalho também altera a forma de endereçamento na rede, como será explorado no Capítulo 3.

Outra arquitetura apresentada nos últimos anos na abordagem *clean slate* é a MobilityFirst (VENKATARAMANI et al., 2014), focada em mobilidade. A ideia por trás da arquitetura é separar nome e endereçamento para que não haja um localizador fixo

quando um usuário fizer mobilidade, mudando seu ponto de acesso.

Uma característica importante na MobilityFirst é o uso de um servidor global para resolução de nomes, o que pode ser similar para o roteamento orientado a *workspace*. Este último não usa um serviço global, mas a busca por um *workspace* pode tomar proporções globais no caso de grandes topologias.

Além das arquiteturas acima, cabe destacar a ETArch (SILVA, 2013), que é a arquitetura utilizada no presente trabalho. Nela, o conceito *hosts* de origem e destino não existe mais. Quem se registra para ser controlado no controlador é chamado de entidade, que pode ser um *host*, usuário, o próprio controlador etc. Após isso, as entidades criam *workspaces* que fornecerão um canal para comunicação, flexível o suficiente para atender requisitos das aplicações. Como exemplo de requisitos pode-se citar largura de banda, QoS, segurança, mobilidade etc.

O próximo capítulo vai detalhar a ETArch, pois a primeira versão do roteamento orientado a *workspace* roda à luz dos conceitos dessa arquitetura.

Capítulo 3

Uma arquitetura para Internet do Futuro

A Internet é considerada como o maior projeto de engenharia em produção (KUROSE; ROSS, 2010), visto a quantidade de usuários que a usam e o número de roteadores espalhados mundialmente para fornecer comunicação entre esses usuários. Ao longo dos anos, a rede evoluiu mantendo a dependência do IP, o qual identifica e localiza os diversos *hosts*.

Identificar um *host* significa dar nome ao mesmo, de forma a distingui-lo dos demais. Localizar um *host* significa endereçá-lo de modo que seja possível chegar até ele. Com o IP exercendo ambos os papéis fica difícil trabalhar com alguns requisitos que não existiam quando a Internet teve início. Pensando nisso pesquisadores criaram um novo modelo, o Modelo de Títulos (PEREIRA et al., 2011) (PEREIRA, 2012), eliminando a hierarquização presente no IP.

O Modelo de Títulos deu origem a uma nova arquitetura, a ETArch (*Entity Title Architecture*) (SILVA, 2013). O presente foi desenvolvido para a ETArch, aproveitando sua flexibilidade. Inicialmente neste capítulo serão citados alguns requisitos que fazem parte do futuro da Internet. Logo após será detalhado o Modelo de Títulos, seguido da descrição dos conceitos por trás da ETArch e, por fim, as principais implementações realizadas.

3.1 ETArch: Aspectos conceituais

A ETArch é uma arquitetura que se propõe a oferecer requisitos da Internet do Futuro, tais como: mobilidade, *multihoming*, segurança, eficiência energética, QoS, QoE, largura de banda, dentre outros. Todavia, alguns desses requisitos já são realidade na rede atual, sendo que o roteamento tem papel fundamental, haja visto que requisitos podem influir na escolha do caminho (rota).

Mobilidade, por exemplo, não estava presente nos primórdios da Internet, visto que os *hosts* eram ligados fisicamente, através de enlaces, a *switches*, roteadores e outros *hosts*.

Nos dias atuais, um *host* não é apenas um computador pessoal do tipo *Desktop*, podendo ser um *notebook* ou *smartphone*, os quais não estão fixos em um local, e nem sequer ligados fisicamente em outros elementos de rede (NE – *Network Element*).

A mobilidade ocorre quando um *host* muda de localização. Inicialmente, um *host* está ligado a um AP (*Access Point*), que o conecta fisicamente à Internet. Como exemplo de AP pode-se citar um *switch* ou roteador *wireless*. Como exemplo de mobilidade pode-se citar um *smartphone* ligado a um AP se comunicando pela Internet. Quando o ganho do sinal *wireless* de outro AP torna-se mais forte que o corrente AP, o *smartphone* vai se registrar no novo AP.

No mundo IP, o problema da mobilidade reside na troca de AP, pois quando isto ocorre, um *smartphone*, por exemplo, troca de endereço IP, recebendo, portanto, um novo IP. Assim sendo, todas as conexões do *smartphone* com outros *hosts* da Internet devem ser refeitas. A Figura 7 exemplifica os momentos nos quais ocorre a movimentação do usuário e seu *smartphone*.

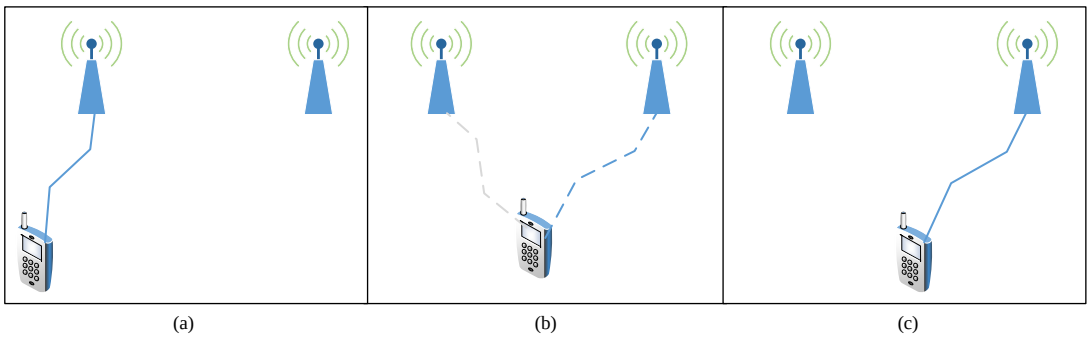

Figura 7 – *Handover*

Na Figura 7(a) observa-se um *smartphone* e dois APs e uma conexão entre o *smartphone* e o AP da esquerda. A Figura 7(b) ilustra o momento em que o *smartphone* se desloca, recebendo sinais de ambos os APs. Nesse momento, considera-se que a conexão com o AP da esquerda apresenta ganho de sinal menor e uma conexão com o AP da direita começa a se estabelecer. Na Figura 7(c) o *handover*[1] é finalizado, permanecendo apenas a conexão com o AP da direita.

Se o endereço IP é o identificador e o localizador do *host* na rede Internet, então, é obrigatório o estabelecimento de uma nova conexão, pois houve a troca de endereço de um dos *endpoints* do canal de comunicação. Algumas propostas, como por exemplo Atkinson e Bhatti (2012), separam a identificação da localização, mantendo, porém, a dependência do endereço IP.

Multihoming é outro requisito estudado pelos pesquisadores nas últimas décadas. Essa técnica é utilizada por empresas que dependem da Internet para operar. Elas compram

[1] Quando um *handset* muda de ponto de acesso (AP), eventualmente com outra banda de frequências

mais de um enlace, normalmente de dois provedores (ISP – *Internet Service Provider*) diferentes, com saída para Internet. Dessa forma se um enlace cair o serviço da empresa não é afetado, pois os dados passam pelo outro enlace.

O problema do *multihoming* está na forma como os roteadores escolhem o enlace de saída. Na arquitetura IP, essa escolha é feita através do prefixo do endereço IP de destino, pois o roteador escolhe sempre o endereço IP com prefixo mais longo cadastrado em sua tabela de rotas. O *multihoming* está diretamente relacionado ao roteamento orientado a *workspace*, que leva em consideração requisitos diferentes das aplicações. Portanto, a escolha do melhor enlace, estando em operação, por exemplo, dois diferentes ISPs, deverá levar em conta os requisitos que esses ISPs podem atender.

Segurança é um requisito que, nos dias atuais, não é discutido nas camadas inferiores[2], sendo responsabilidade das aplicações. Atualmente, segurança na Internet se dá pela força bruta, sendo cada pacote escrutinado individualmente. O perfil de um usuário da rede não é conhecido pela própria rede, sendo, assim, impossível distinguir se um usuário está bem ou mal intencionado. Com o roteamento orientado a *workspace* busca-se inserir segurança como requisito no nível de rede.

O tema sustentabilidade (mundialmente em pauta) e a questão financeira (com gasto de energia) trouxe um novo requisito para a Internet: eficiência energética. As redes atuais não se preocupam em medir consumo de energia ou desligar enlaces ou equipamentos de rede. Grandes *datacenters* preferem algumas vezes manter seus equipamentos ligados, alegando considerável consumo de energia no *boot* das máquinas.

Um exemplo de eficiência energética é quando uma aplicação possui duas opções de rotas A e B para chegar até determinado conteúdo na Internet, como por exemplo uma página web. Suponha-se que o tempo de resposta pela rota A seja $100ms$ e pela rota B seja $200ms$. Suponha-se ainda que a rota A consuma mais energia que a rota B. A aplicação poderia simplesmente preferir a rota B, mesmo gastando o dobro do tempo, mas economizando energia. Como o tempo de $200ms$ é imperceptível para o usuário que busca uma página web, essa tática de escolha da rota se torna praticável.

O roteamento orientado a *workspace* pretende utilizar eficiência energética como requisito, visto que determinadas aplicações podem ter características independentes da taxa de transferência ou latência, preferindo escolher rotas pelo consumo energético. Além disso, o controlador da rede também pode decidir por rotas mais eficientes em termos de energia, como no caso de topologias de *datacenters*, que possuem alta taxa de consumo.

Os conceitos de QoS e QoE (*Quality of Experience*) estão relacionados, sendo ambos requisitos presentes na Internet, principalmente no que diz respeito ao uso da rede para tráfego de voz e *streaming* de vídeo em tempo real. A QoS é a qualidade de serviço oferecida que determinadas aplicações precisam. Um exemplo é o *streaming* de vídeo, que

[2] De acordo com o Modelo de Referência OSI, camadas inferiores são aquelas abaixo da camada de Transporte

precisa da alta taxa de transferência e baixa latência (*release jitter*) para não ter perda de pacotes.

O conceito de QoE refere-se a qualidade de experiência dos usuários ao usar determinada aplicação. Um exemplo é que determinados pacotes no *streaming* de vídeo possuem mais importância que outros. Caso pacotes desse tipo sejam perdidos o usuário tem uma experiência ruim ao assistir o vídeo. Para QoS, a telefonia, implementando cada vez mais voz sobre IP, vai exigir QoS para a transferência do áudio sem cortes. O roteamento também está relacionado com QoS e QoE, pois a definição da rota pode levar em conta enlaces que atendam esses requisitos.

Além dos requisitos citados acima, largura de banda, presente desde as primeiras implementações de protocolos na Internet, que está relacionada à vazão dos enlaces. Os principais protocolos de roteamento, como o OSPF, possuem táticas para controle de congestionamento. O roteamento no futuro não abrirá mão desse requisito, mas deve aprimorá-lo para trabalhar em conjunto com os demais.

Os requisitos mencionados acima demonstram que uma mudança na estrutura da Internet deveria ocorrer. Assim, Pereira et al. (2011) propuseram o Modelo de Títulos. Um modelo é uma abstração de um sistema complexo em algo mais simples, mas que consegue representar bem o sistema complexo. A Internet, com todas as tecnologias necessárias para funcionar, pode ser considerada um sistema com alta complexidade de operação.

O Modelo de Títulos separa a identificação de uma entidade de sua localização (endereço), mundando basicamente a forma de endereçamento. O endereçamento de *hosts* feito pelo endereço IP é hierárquico, atrelando fortemente com a sub-rede (local) ao qual pertence. Pereira (2012) propõe uma forma de endereçamento horizontal, onde os diversos elementos na rede são identificados por um nome único. Isso é similar ao que acontece no DNS (*Domain Name System*), com a diferença que o Modelo de Títulos não possui características hierárquicas. A tese é ter todos os *hosts* dispersos em um plano na Internet.

Baseada no Modelo de Títulos foi proposta a arquitetura ETArch, que implementa o endereçamento horizontal, eliminando a camada de rede no que tange a funções de roteamento. Na ETArch, o conceito de camadas não é mais aquele especificado na Arquitetura Internet. A Figura 8 apresenta uma comparação dos modelos de camadas. Do lado esquerdo está a Arquitetura Internet, dependente das camadas de transporte e rede, com a pilha de comunicação criada pelos processos nos *hosts*. Do lado direito, essas camadas são substituídas por uma camada de comunicação que possui tamanho flexível.

A representação não usual da camada Comunicação indica a flexibilidade introduzida pela ETArch que permite a cada aplicação especificar os requisitos que necessita para se comunicar, ou seja, uma aplicação, por exemplo, pode requerer confiabilidade e largura de banda.

Dentre os conceitos do Modelo de Títulos e implementados pela Arquitetura ETArch,

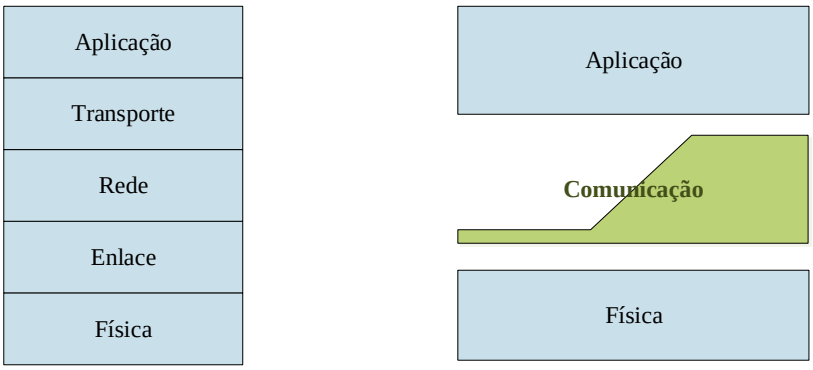

Figura 8 – Arquitetura Internet x ETArch

alguns importantes para a compreensão do presente trabalho serão descritos nas próximas subseções.

3.1.1 Entidade

Entidade é um ente fundamental no Modelo de Títulos e pode ser definida como tudo que tem capacidade de se comunicar. Dessa forma, entidades podem ser *hosts*, *smartphones*, NEs, usuários, controladores SDN, enlaces, aplicações, dentre outros. A diferença de concepção em relação ao modelo tradicional está na forma de definição dos requisitos (PEREIRA, 2012).

Um requisito é tudo que a entidade pode exigir da rede, como largura de banda, QoS, QoE, eficiência energética, comunicação orientada à conexão, etc. Dessa forma, antes de iniciar uma comunicação, a entidade passa para a rede seus requisitos para que a comunicação comece. A rede deve ser capaz de satisfazer os requisitos. O roteamento está diretamente ligado aos requisitos, pois determinar uma rota deve levar em consideração quais requisitos a entidade exigiu naquela comunicação.

Outro ponto importante a se destacar é que uma entidade que queira se comunicar, na ETArch isso é natural, define seus requisitos e a rede é preparada para fornecer o palco para que uma comunicação exista.

3.1.2 Título

Título é um identificador único, não ambíguo e independente da topologia da rede (PEREIRA, 2012). A ETArch usa o conceito de Título para a identificação de entidades. Isso significa que cada entidade recebe um Título que a identificará univocamente em seu espaço de nomes (*Namespace*). Vale ressaltar que na ETArch não existe a hierarquia criada no mundo IP, ou seja, um identificador de uma entidade a acompanha mesmo se houver mudança de localização.

Na prática, o uso do Título significa que qualquer entidade que estivesse na Internet e desejasse se comunicar deveria possuir um nome válido, e esse nome a acompanharia mesmo se ela mudasse de AP, ou mude até mesmo sua posição geográfica em proporções continentais.

A abordagem deste modelo, baseado em títulos, faz com que o roteamento e demais rotinas de controle da rede trabalhem para que as entidades possam trocar informações independente do local físico que se encontrem. Portanto o foco é buscar e trocar informações, sendo que o conhecimento da localização real de uma entidade não é objetivo do roteamento.

O Título, na arquitetura em questão, também se aplica aos *Workspaces*. A nomeação dos *Workspaces* pode ser considerada semelhante à nomeação de informações nas arquiteturas do tipo ICN[1] (*Information-Centric Networking*) (XYLOMENOS et al., 2014). Note-se que o conceito de *Workspace* será explicado na seção 3.1.4 e não possui nenhuma relação ou semelhança ao conceito de informação (a relação está apenas no fato de nomear aquilo que será buscado em determinada comunicação).

A utilização de Título na ETArch permite desacoplar o conteúdo da comunicação de sua origem, separando-se localizador e identificador. Quando uma requisição para comunicação é solicitada por uma entidade o controle da rede é responsável por configurar elementos de rede da topologia para satisfazer a requisição. Todavia, a posição geográfica das entidades que fazem parte da comunicação em questão não é relevante, similar ao que acontece nas arquiteturas ICN[1].

Uma arquitetura de ICN[1] que utiliza conceitos semelhantes ao de Título aqui proposto é a NDN (*Named Data Networking*) (SONIYA; KUMAR, 2015). Nessa arquitetura as primitivas carregam o nome dos dados ao invés do endereço de origem e destino. A justificativa é que o crescimento na distribuição de conteúdo deixa o IP com problemas de escalabilidade, visto que foi projetado para comunicação entre dois pontos. NDN separa os pacotes nos tipos interesse e dados enquanto a ETArch separa suas primitivas nos tipos controle e dados.

3.1.3 *Domain Title Service*

A ETArch é uma arquitetura que se encaixa na definição de SDN, na qual o Plano de Controle é separado do Plano de Dados. A ideia dessa separação no Modelo de Títulos surgiu com base nas redes ATM e de telefonia que sempre fizeram o controle (sinalização) separadamente do plano de dados. Na ETArch, o responsável por controlar o ambiente distribuído de entidades (e respectivos Títulos) interconectadas é denominado DTS (*Domain Title Service*). DTS é um sistema distribuído composto por agentes denominados DTSA (DTS *Agent*), que são superconjuntos de controladores SDN.

DTSAs são entidades responsáveis por comunicações de controle e monitoração com um ou mais NEs (*switches*) e, também, com outros DTSAs. A topologia de uma rede

ETArch é constituída de duas partes: topologia de interconexão de NEs, que para este trabalho são *switches* OpenFlow, responsáveis por encaminhamento no plano de dados; e topologia de interconexão de DTSAs. A Figura 9 ilustra as duas topologias.

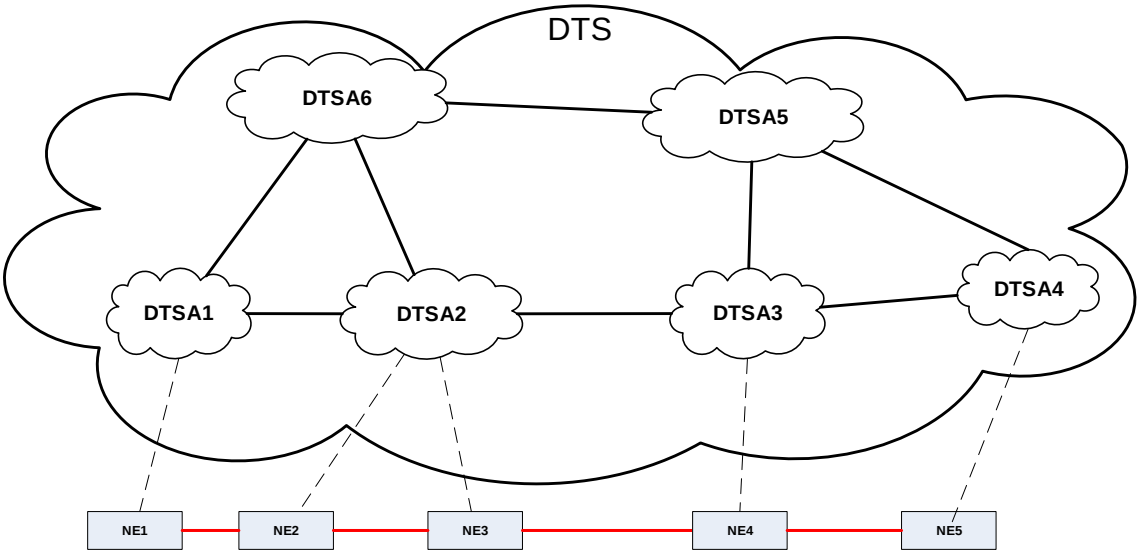

Figura 9 – Exemplo de Topologia ETArch

Na Figura 9, o grafo, composto pelos nós NE_1 a NE_5 e arestas em vermelho, representa a topologia do Plano de Dados e a nuvem, composta pelos $DTSA_1$ a $DTSA_6$, representa o Plano de Controle (DTS). Note-se que um DTSA pode controlar 1 ou mais NEs, sendo que na Figura 9 essa ligação é representada pelas linhas tracejadas.

3.1.4 *Workspace*

O conceito de *Workspace* é o ponto fulcral para comunicações na arquitetura ETArch. Conceitualmente, *workspace* é um barramento lógico para a comunicação de uma ou mais entidades, independente das topologias e interconexões de redes subjacentes. Posto de outro modo, para se comunicar uma entidade precisa de um *Workspace*. Uma entidade que deseje se comunicar pode criar um *Workspace*, se ele ainda não existir, ou fazer uso de um *Workspace* existente.

O *Workspace* oferece suas capacidades de comunicações através de unidades de comunicação codificadas em Primitivas. As comunicações no Plano de Controle dar-se-ão através de Primitivas de Controle e no Plano de Dados por meio de Primitivas de Dados. O *Workspace* é responsável por entregar primitivas a todas as entidades que dele façam uso. Por outro lado, uma entidade envia suas primitivas ao *Workspace* do qual faz uso. Este é um ponto fundamental para a mobilidade.

Uma vez que as primitivas são enviadas ao *Workspace* e é sua responsabilidade entregá-las (primitivas) às entidades que dele (*Workspace*) fazem uso, então, ao se movimentar,

uma entidade provoca deformações no *Workspace*, via DTS, para estendê-lo à sua presente localização.

Note-se que esta concepção provoca a aludida mudança no conceito de endereçamento. Não mais faz sentido em se falar em endereçamento *Unicast*, *Broadcast* ou *Multicast*. O destino das primitivas não é (são) mais entidade(s) de destino(s) – é o *Workspace*. Podemos dizer que o *Workspace* é intrinsecamente *Multicasting*, sendo os demais casos particulares de endereçamento.

A Figura 10 representa o funcionamento de um *workspace*. Uma entidade Servidor de Vídeo cria o *workspace* com título $Workspace_1$ para realizar o *streaming* de um vídeo, como por exemplo uma partida de futebol ao vivo. As entidades $Application_1$ e $Application_2$ decidem que querem fazer parte da comunicação e, portanto, solicitam ao DTS[3] para fazer parte do $Workspace_1$. Uma vez solicitado, o DTSA (re)configura seus NEs de forma que todas as primitivas recebidas pelo $Workspace_1$ sejam encaminhadas para as portas de saída dos respectivos NEs, nas quais estão ligadas as $Application_1$ e $Application_2$.

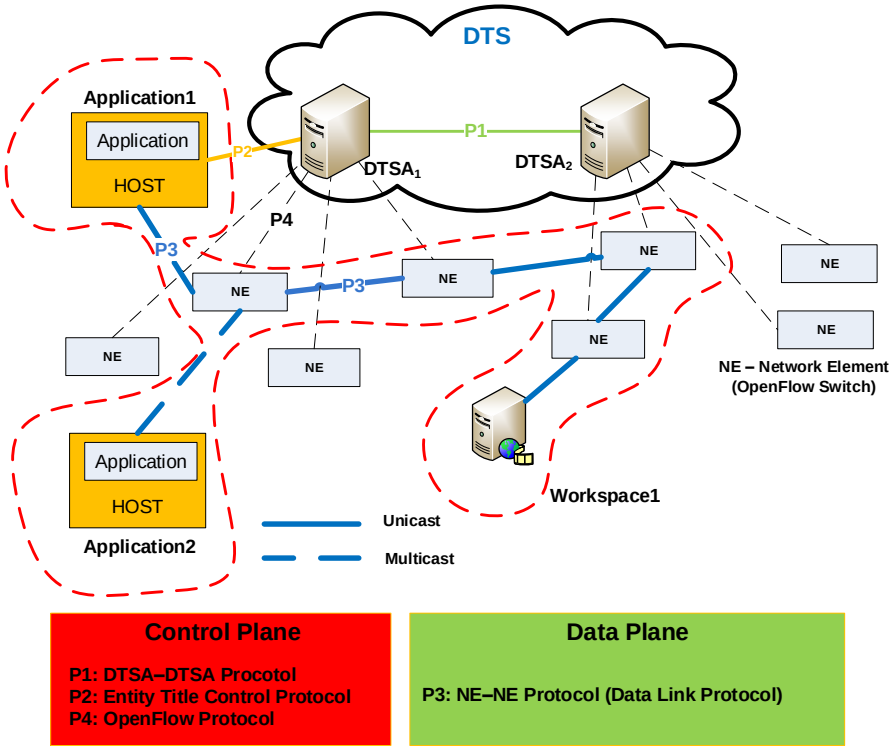

Figura 10 – *Workspace*

Na Figura 10 a linha tracejada em vermelho representa logicamente o *workspace* criado. Dessa forma, no plano de dados, as ligações P3 da figura mostram os enlaces pelos quais as primitivas de dados trafegam pelo $Workspace_1$. A figura também mostra as ligações

[3] De fato a solicitação é feita ao DTSA ao qual estão sob controle

P1 (entre dois DTSAs) e P2 (entre NE e DTSA), ambas representando protocolos da camada de controle.

Na Figura 10 observa-se que uma vez estabelecido o *workspace* os NEs são configurados para encaminhar primitivas daquele *workspace* pelas portas de saída de forma que sigam os caminhos para chegar até as entidades que fazem parte da comunicação.

É importante mencionar que uma entidade, ao criar um *workspace*, passa ao DTSA os requisitos para o funcionamento apropriado. Por exemplo, se a entidade é uma aplicação de telefonia, o requisito de QoS deverá ser informado para que o DTSA configure um *workspace* de forma que os enlaces ao longo do caminho atendam a característica de áudio (tempo real). Outro exemplo, se a entidade é uma aplicação de chat, então o *workspace* não precisa atender o requisito de tempo real.

Caso uma entidade resolva se juntar a um *workspace* cujo Título seja desconhecido, o seu DTSA de controle deverá solicitar informações a outros DTSAs. Isso é uma das funcionalidades do roteamento inter-domínio e será descrita no Capítulo 4.

Dois protocolos do Plano de Controle foram propostos para a implementação do conceito de *Workspace*: DTSCP (*Domain Title Service Control Protocol*) e ETCP (*Entity Title Control Protocol*), representados na Figura 10 respectivamente como P_1 e P_2. Ambos foram criados para trabalhar no controle da rede, sendo o ETCP, um protocolo para comunicação entre entidades e DTSAs, e o DTSCP, um protocolo para comunicação entre dois ou mais DTSAs. Os serviços do protocolo ETCP estão descritos a seguir:

a) ENTITY_REGISTER: serviço através do qual uma entidade requisita seu registro no DTS (de fato DTSA), informando seu título, requisitos e capacidades, sendo o DTSA responsável por armazenar essas informações e, também, por garantir a unicidade de seu título;

b) ENTITY_UNREGISTER: serviço que remove o registro de uma entidade do DTS;

c) WORKSPACE_CREATE: serviço utilizado por uma entidade registrada, informando ao DTSA que um novo *workspace* identificado por seu Título está disponível, sendo este o primeiro passo para o estabelecimento de comunicações;

d) WORKSPACE_ATTACH: serviço requisitado por uma entidade registrada indicando que quer tomar parte nas comunicações suportadas por um *Workspace* existente criado pelo serviço WORKSPACE_CREATE, serviço este que requererá que o DTSA configure os NEs ao longo do caminho, permitindo que o *Workspace* seja estendido até a entidade requisitante;

e) WORKSPACE_DELETE: serviço que indica ao DTSA que o *workspace* identificado por seu título deve ser removido e as respectivas regras de encaminhamento devem ser retiradas dos NEs;

f) WORKSPACE_DETACH: serviço através do qual uma entidade registrada resolve não mais fazer parte de *workspace* identificado por seu título, implicando em que

as regras de encaminhamento devem ser retiradas dos NEs, para que os dados não sejam trafegados sem motivo, economizando assim recursos nos NEs e no próprio DTSA;

g) WORKSPACE_MODIFY: serviço que permite modificar um determinado *workspace* identificado por seu título, por exemplo, modificar as capacidades que o *workspace* possui para suportar determinados requisitos.

O funcionamento do ETCP é mostrado na Figura 11. Observa-se que uma entidade inicialmente está no estado *unregistered* e obrigatoriamente deve se registrar (junto a um DTSA). No registro, através da primitiva ENTITY_REGISTER, a entidade informa seu título. De posse do título o DTSA passa a conhecer e controlar aquela entidade. Assim que recebe a confirmação de registro a entidade passa para o estado *registered* e pode se comunicar. Para se comunicar duas opções estão disponíveis: (i) a entidade pode criar um *workspace* através da primitiva WORKSPACE_CREATE; ou (ii) a entidade pode se juntar a um *workspace* existente através da primitiva WORKSPACE_ATTACH.

Caso uma entidade crie um *workspace* o estado *workspace-created* da Figura 11 permite apenas que a entidade delete o *workspace*. Caso a entidade se junte a um *workspace* existente o estado *workspace-attached* da Figura 11 permite que a entidade deixe o *workspace* (através da primitiva WORKSPACE_DETACH).

O protocolo DTSCP possui alguns serviços apresentados por Silva (2013). O trabalho aqui proposto adiciona novos serviços, que serão descritos no Capítulo 4. Os atuais são apresentados a seguir:

a) WORKSPACE_LOOKUP: serviço utilizado quando um *workspace* é especificado por uma entidade, cujo DTSA não possui informações desse *workspace*, então esse DTSA (que recebeu a solicitação do serviço) deve solicitar a outros DTSAs informações sobre o referido *workspace*, sendo que este serviço será detalhado no Capítulo 4, vista sua importância para o roteamento inter-domínio;

b) DTSA_REGISTER: serviço que permite a uma entidade DTSA se registrar em um tipo especial de DTSA denominado *Master*-DTSA, informando quais DTSAs conseguem resolver um *lookup* para esse DTSA, ou seja, para quais DTSAs ele enviará um WORKSPACE_LOOKUP caso não conheça um determinado *workspace*, sendo seu detalhamento no Capítulo 4;

c) WORKSPACE_ADVERTISE: serviço que permite a um *Master*-DTSA inserir informações sobre um *workspace* ao longo de sua árvore de DTSA;

d) DTS_MESSAGE: serviço que permite troca de mensagens genéricas entre diferentes DTSAs.

As implicações do protocolo DTSCP nas rotinas de roteamento orientado a *workspace* serão detalhadas no Capítulo 4.

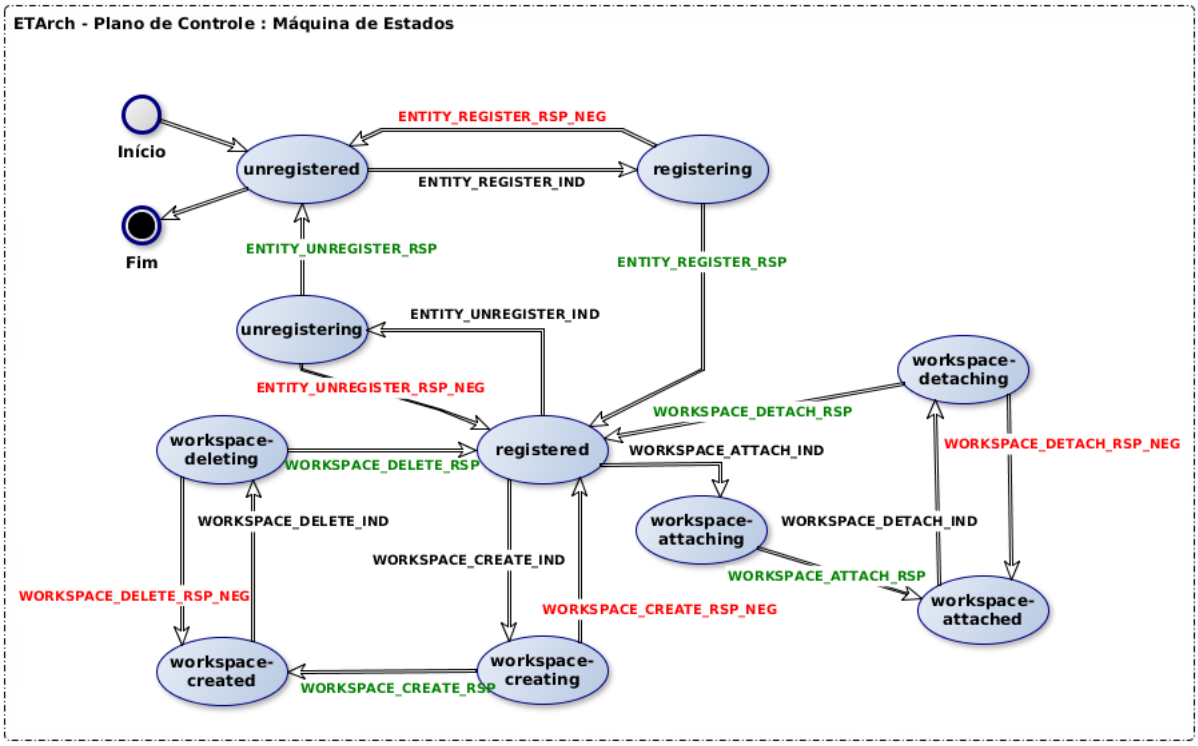

Figura 11 – Autômato Finito para o ETCP

Fonte: Gonçalves (2014)

3.2 Implementações

Alguns trabalhos foram realizados utilizando a ETArch em suas implementações, como por exemplo Silva et al. (2014), Guimaraes et al. (2014) e Lema (2014). Os requisitos abordados por todos os trabalhos variam, porém todos foram implementados utilizando o DTSA do Projeto EDOBRA ou através do SMART. Ambos os projetos serão descritos nas próximas subseções.

3.2.1 Projeto EDOBRA

Uma das principais implementações utilizando a ETArch foi o Projeto EDOBRA (*ExtenDing Ofelia in BRAzil*), que demonstrou o *Workspace* (*multicast*) e *Vertical Handover*. O projeto pode ser dividido em duas partes: criação de uma ilha Ofelia[4] no Brasil; e desenvolvimento de uma plataforma rodando em um ambiente ETArch.

Ofelia (*OpenFlow in Europe: Linking Infrastructure and Applications*) é uma iniciativa da comunidade européia com objetivo de criar um ambiente experimental para que pesquisadores possam testar seus trabalhos relacionados principalmente a redes de computadores

[4] http://www.fp7-ofelia.eu/ofelia-facility-and-islands

e OpenFlow. A iniciativa conta com investimento da União Europeia e participação de diversas universidades e centros de pesquisas europeus, dos mais diversos países.

Através do EDOBRA, o Brasil, mesmo estando em outro continente, passou a ser um dos países participantes do Ofelia, composto por várias ilhas, que são locais contendo infraestrutura completa de *switches*, servidores físicos e links para que pesquisadores e estudantes possam remotamente rodar suas aplicações em cada ilha.

A criação da ilha Ofelia no Brasil se deu na Faculdade de Computação da UFU (Universidade Federal de Uberlândia). Essa ilha contém um servidor para controle, servidores para criação de máquinas virtuais, três *switches* OpenFlow, algumas NETFPGAs e fibra óptica diretamente conectada à Europa.

A segunda parte do EDOBRA foi a implementação da ETArch propriamente dita. O desenvolvimento se deu em linguagem de programação Java, utilizando o protocolo OpenFlow através do controlador Floodlight e o conceito de JAIN SLEE[5](FERRY, 2014). Foi desenvolvido um DTSA, modificando o Floodlight e usando-o para comunicação com *switches* OpenFlow na ilha de Uberlândia.

JAIN SLEE abstrai a camada de comunicação, necessária para o desenvolvimento de um protocolo computacional. Na prática, o desenvolvedor não precisa se preocupar com a codificação dos protocolos, que são transformados em serviços. Outra vantagem no uso de JAIN SLEE é seu bom desempenho para lidar com disponibilidade e escalabilidade (FEMMINELLA et al., 2011).

No DTSA desenvolvido pelo projeto EDOBRA, o Floodlight foi encapsulado em uma *lib* Java dentro do *container* Mobicents (MOBICENTS, 2014), que é uma implementação *opensource* da especificação JAIN SLEE. O JAIN SLEE apresenta dois conceitos principais: RA (*Resource Adapter*) e SBB (*Service Building Block*).

No JAIN SLEE, um RA é responsável por abstrair os protocolos de rede, transformando suas primitivas em serviços. O SBB é responsável por capturar os serviços de cada RA e lançá-los para outros SBBs, para que recebam tratamento específico.

O uso do JAIN SLEE pode ser exemplificado através das operadoras de telecomunicações que utilizam Mobicents. Uma operadora pode ter diversos RAs para diferentes protocolos de rede, como por exemplo RA para HTTP (*Hypertext Transfer Protocol*) (FIELDING et al., 1999), RA para SIP (*Session Initiation Protocol*) (ROSENBERG et al., 2002) e assim para quaisquer outros protocolos. Dessa forma, a operadora pode aproveitar os RAs disponíveis na comunidade e utilizá-los, tendo apenas o trabalho de construir SBBs específicos para suas regras de negócio.

A Figura 14 mostra o DTSA desenvolvido no projeto EDOBRA. Nele foram desenvolvidos dois RAs: OpenFlow RA e MIH[6] RA. O RA do OpenFlow tem por objetivo abstrair o protocolo OpenFlow, através da *lib* do Floodlight. O RA do MIH tem por objetivo a

[5] JAIN SLEE é uma especificação criada para desenvolvimento baseado em serviços, abstraindo protocolos de telecomunicações.

[6] MIH (*Media Independent Handover*) como é conhecido o padrão IEEE 802.21

comunicação entre o DTSA e a rede sem fio para questões de controle e *handover* das entidades móveis. Além dos dois RAs foram desenvolvidos quatro SBBs, representados na cor verde.

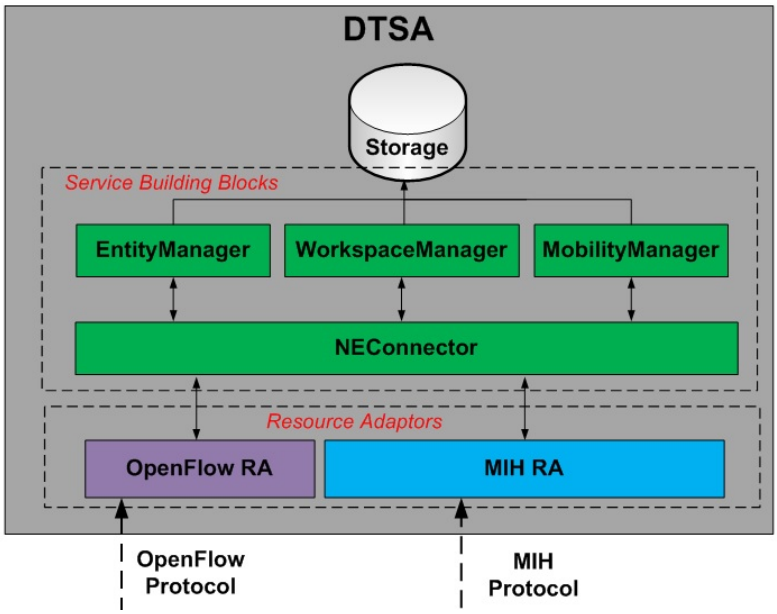

Figura 12 – DTSA do Projeto EDOBRA

Fonte: Silva (2013)

Conceitualmente, o SBB NEConnector, na Figura 12, é uma camada genérica de comunicação entre a rede e o DTSA. Atualmente, os protocolos que se comunicam com o NEConnector são OpenFlow e MIH. No futuro, novos protocolos podem ser criados, bem como o OpenFlow e MIH podem ser substituídos. Em ambos os casos, os SBBs acima do NEConnector podem ser reaproveitados.

Uma entidade, para se registrar, envia uma primitiva ENTITY_REGISTER, através do NE (*switch* OpenFlow) ao qual está ligada fisicamente. No NE, essa primitiva é encapsulada em um PACKET_IN e enviada ao controlador (DTSA) do NE, que é o funcionamento padrão do protocolo OpenFlow (KONTESIDOU; ZARIFIS, 2009). Ao chegar no DTSA, o RA OpenFlow (através do Floodlight) transforma o PACKET_IN em um serviço (e o lança a uma 'camada superior'). Para todas as outras primitivas do ETCP o funcionamento é similar.

O PACKET_IN lançado pelo RA OpenFlow é capturado pelo SBB NEConnector. O NEConnector analisa todas as mensagens PACKET_IN recebidas, para resgatar a primitiva ETArch nelas encapsuladas, que pode ser qualquer uma das primitivas do ETCP. Após realizar essa análise, o NEConnector não tem o que fazer com o conteúdo, então a lança para que outros SBB possam capturá-las.

Os outros três SBBs da Figura 12, acima do NEConnector, são responsáveis por captu-

rar os serviços lançados pelo próprio NEConnector. Primitivas relacionadas às entidades são capturadas pelo EntityManager. Tudo que está relacionado à criação e manutenção de um *workspace* é capturado pelo WorkspaceManager. As configurações para manutenção dos NEs para entidades móveis e *handover* são capturados pelo MobilityManager.

O SBB EntityManager recebe as primitivas para registro e remoção do registro de uma entidade. No EntityManager, essas primitivas recebem o tratamento adequado, como por exemplo registrar a entidade no *storage* mostrado na Figura 12, que é o banco de dados do DTSA. Além disso, o EntityManager é quem responde para o NEConnector informando que a operação (registrar ou remover o registro) foi realizada com sucesso.

O SBB WorkspaceManager é responsável por todas as primitivas relacionadas aos *workspaces*: criar ou excluir um *workspace*, inserir ou retirar uma entidade de um *workspace* e alterar as características de um *workspace*. O *storage* da Figura 12 também é manipulado pelo SBB WorkspaceManager. Além das funções descritas, cabe ao SBB WorkspaceManager manipular primitivas que serão enviadas a outros DTSAs.

O SBB MobilityManager é o responsável pelos serviços lançados pelo MIH RA. Portanto, quando uma entidade faz *handover* para um novo AP, o MIH RA receberá a primitiva e lançará ao NEConnector, que então lançará ao MobilityManager. O MobilityManager é responsável pelas ações para a primitiva, fazendo as alterações necessárias no *storage* do DTSA, além de avisar outros DTSAs, caso necessário, da movimentação dos *workspaces* que a entidade participa.

Dos quatro SBB da Figura 12, o único que se comunica com a rede é o NEConnector. Portanto, quando uma entidade estende um *workspace*, o NEConnector é quem vai solicitar aos *switches* OpenFlow as configurações das regras de encaminhamento, assim como quando um *workspace* é criado ou removido, ou uma entidade deseja sair da comunicação. A Figura 13 mostra o diagrama de sequência para a primitiva WORKSPACE_ATTACH, a qual utiliza os SBBs NEConnector e WorkspaceManager.

A sequência de trocas de primitivas do protocolo ETCP é similar, alterando o conteúdo das primitivas. Os passos de uma primitiva são: chegar em um dos dois RAs e ser abstraída pelo NEConnector; por fim ser lançada para cima e capturada pelo EntityManager, WorkspaceManager ou MobilityManager. Para exemplificar, os itens abaixo enumeram a sequência de passos (mesma da Figura 13) desde a entidade requisitar um WORKSPACE_ATTACH até receber a confirmação do DTSA). As demais primitivas possuem sequência similar.

1. Entidade solicita um WORKSPACE_ATTACH, enviando o título do *workspace*;

2. WORKSPACE_ATTACH chega até o NE ao qual a entidade está fisicamente ligada;

3. NE encapsula WORKSPACE_ATTACH na primitiva PACKET_IN e lança para o controlador;

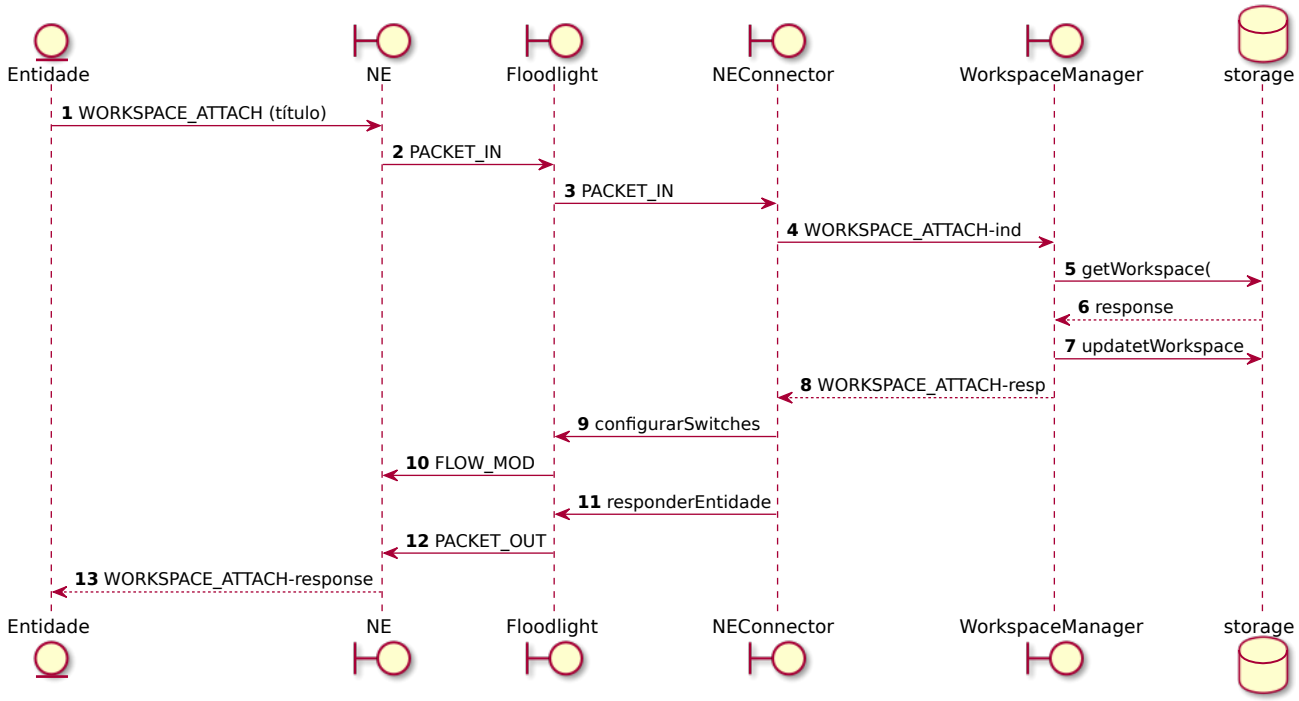

Figura 13 – Diagrama de sequência para a primitiva WORKSPACE_ATTACH

4. PACKET_IN chega no controlador Floodlight (lib dentro do DTSA);
5. Floodlight encaminha o PACKET_IN para o OpenFlow RA;
6. OpenFlow RA transforma o PACKET_IN em serviço e lança;
7. Serviço PACKET_IN é capturado pelo SBB NEConnector;
8. NEConnector faz análise do serviço PACKET_IN, recuperando a primitiva WORKSPACE_ATTACH;
9. NEConnector lança o serviço WORKSPACE_ATTACH-ind;
10. WorkspaceManager captura serviço WORKSPACE_ATTACH-ind;
11. WorkspaceManager recupera informações sobre o *workspace* requisitado no seu *storage*;
12. *WorkspaceManager* faz um *update* no *storage* acrescentando a entidade solicitante no *workspace*;
13. *WorkspaceManager* lança serviço WORKSPACE_ATTACH-resp contendo as informações de resposta;
14. NEConnector captura serviço WORKSPACE_ATTACH-resp;

15. NEConnector solicita ao Floodlight que configure regras, nos *switches* OpenFlow, necessárias para que o *workspace* seja estendido até a entidade solicitante;

16. Floodlight insere as regras na primitiva OpenFlow FLOW_MOD e a envia ao NE (*switches* OpenFlow);

17. NEConnector solicita ao Floodlight que responda para a entidade;

18. NEConnector encapsula a resposta na primitiva OpenFlow PACKET_OUT e a envia ao NE da entidade solicitante; e,

19. NE envia a resposta para a entidade.

Nos itens enumerados acima um *workspace* estava presente no *storage* do DTSA. Caso não estivesse, a sequência mudaria do item 11 até o final. Nesse caso, o DTSA deveria solicitar um *Workspace Lookup*. Esse serviço de *lookup* será detalhado no Capítulo 4.

Um importante experimento realizado no projeto EDOBRA foi a transmissão de um vídeo enquanto uma das entidades recebendo o vídeo faz *handover*. A movimentação do *workspace* nesse experimento é apresentado na Figura 14.

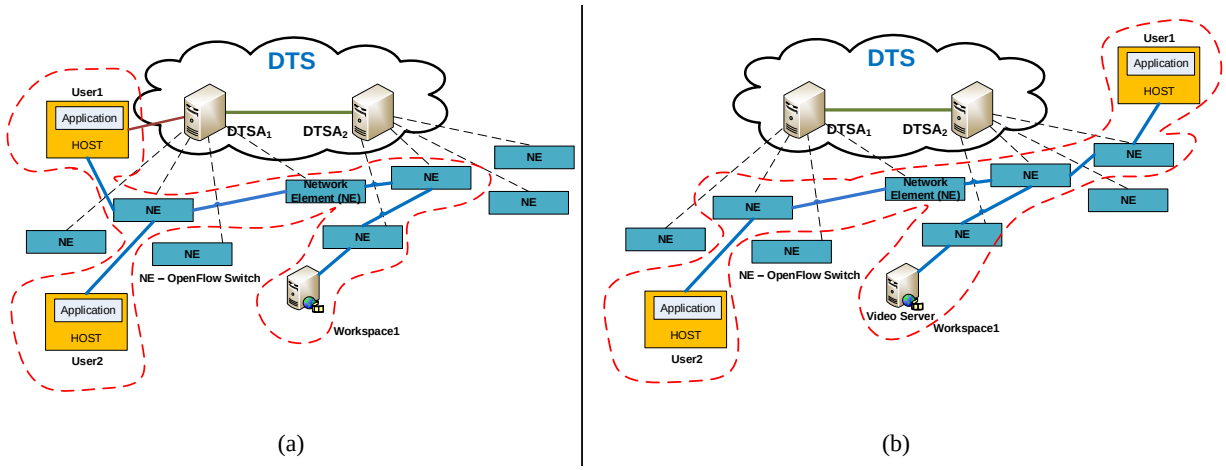

Figura 14 – Movimentação de *workspace* em um cenário de *handover*

Na Figura 14(a) é possível observar que as entidades $User_1$ e $User_2$ estendem o $Workspace_1$, dessa forma todos os dados enviados pelo servidor serão trafegados na forma *multicasted* para $User_1$ e $User_2$. Na Figura 14(b) $User_1$ se movimentou para outro NE. Portanto, os dois DTSAs da topologia na figura devem reconfigurar seus NEs de modo que agora os dados cheguem à nova localização de $User_1$.

A configuração em um NE é uma regra Openflow determinando que todos os dados que cheguem contendo o *hash* de determinado *workspace* sejam direcionados para uma ou mais portas. Essas portas são as conexões que fazem o *workspace* se estender através de todas as entidades que se juntaram a ele.

3.2.2 SMART

Além do EDBORA, outra implementação a partir da ETArch é o projeto SMART (Suporte de Sessões Móveis com Alta Demanda de Recursos de Transporte). O SMART pode ser considerado um *framework* que permite garantia de QoS para as aplicações, usando a arquitetura ETArch . O SMART não utilizou o JAIN SLEE, mas construiu um módulo similar a um SBB, o QoSManager.

O algoritmo utilizado pelo SMART permite que entidades enviem seus requisitos de QoS ao se registrar. Dessa forma, toda vez que uma entidade deseje se juntar a um *workspace*, é possível analisar se o *workspace* é capaz de atender aos requisitos de QoS da entidade. Em caso afirmativo, o DTSA deve apenas configurar os NEs ao longo do caminho de forma a estender o *workspace*. Caso contrário, o DTSA define os NEs que precisam de reajustes e, então, envia as configurações a esses NEs.

Além do algoritmo para tratar QoS, o projeto SMART também acrescenta o conceito de *switches core*. O conceito surgiu, pois alguns *workspaces* utilizam o mesmo caminho para encaminhamento de dados, então uma agregação de *workspaces* pode ser criada para diminuir o número de regras nos NEs. Os NEs que estão na borda não são alterados, mas os que estão no meio da rede podem utilizar esse conceito para ter uma tabela de encaminhamento menor, além de poupar recursos.

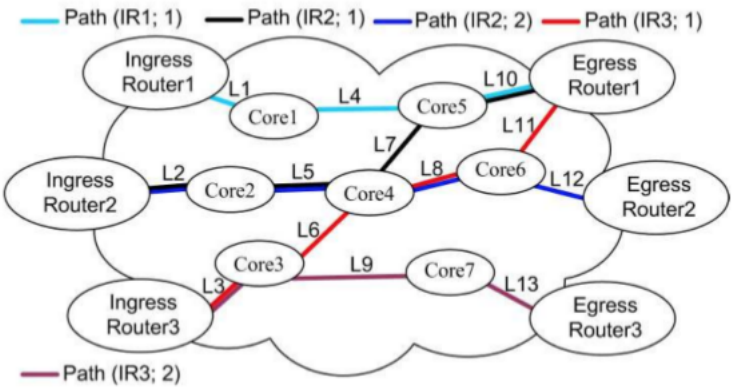

Figura 15 – Topologia SMART

Fonte: Lema (2014)

A Figura 15 apresenta uma topologia no SMART. Observa-se que quatro caminhos foram criados, ou seja, nos NE *core*, quatro agregações de *workspaces* são feitas. Para exemplificar pode-se considerar o caminho Path(IR2;1), de cor preta na figura. Qualquer comunicação entre o NE $IngressRouter_1$ e o NE $EgressRouter_1$ é realizada pelo Path(IR2;1). Dessa forma nos NEs podem ser configuradas apenas duas regras que satisfazem todos os *workspaces* que passarão tráfego entre $IngressRouter_2$ e $IngressRouter_1$.

Os três outros caminhos possuem comportamento similar, criando rotas únicas entre os NEs da borda.

Utilizando a ETArch através dos aspectos conceituais listados no presente capítulo, além das observações retiradas nas implementações do projeto EDOBRA e do SMART, é possível a criação de um mecanismo para rotear utilizando apenas o plano de controle. O próximo capítulo apresentará a proposta do roteamento orientado a *workspace*, um mecanismo rodando apenas no plano de controle, para roteamento intra e inter-domínio. O capítulo apresentará as vantagens e desvantagens desse roteamento e como ele pode ser implantado na ETArch.

Capítulo 4

Proposta de Roteamento Intra/Inter-domínio orientado a *Workspace*

O advento de SDN (*Software Defined Networking*) na área das redes de computadores altera a forma como alguns aspectos da comunicação de dados podem ser tratados. O SDN impõe que o aspecto relativo ao roteamento seja tratado pelo plano de controle, visto que tomar decisão sobre uma rota (roteamento propriamente dito) é uma função de controle da rede. O presente trabalho objetiva propor o uso do plano de controle para execução dos algoritmos de roteamento.

Duas abordagens são possíveis para tratar o roteamento no plano de controle. Na primeira, toda primitiva do plano de dados, ao chegar a um elemento de rede, é direcionada para o plano de controle, para que este decida qual a rota dessa primitiva[1]. Na segunda abordagem, antes que uma comunicação se inicie, o plano de controle resolve o roteamento (decide as rotas que primitivas seguirão naquela sequência de comunicação). Essa segunda abordagem interessa à proposta deste trabalho.

Com a decisão sobre as rotas acontecendo antes que as primitivas do plano de dados comecem a ser encaminhadas, todas as vantagens do uso de SDN podem ser aproveitadas (FARHADY; LEE; NAKAO, 2015). Diversos algoritmos podem ser implantados nos controladores SDN para contribuir na decisão de uma rota, aproveitando requisitos e capacidades das entidades que estão se comunicando.

Com o estabelecimento da rota *a priore* é possível garantir que os requisitos das aplicações sejam atendidos, sendo este o principal motivo da adoção desta abordagem (roteamento prévio). Como na telefonia, algumas desvantagens também são caracterizadas no uso dessa abordagem, como a questão da disponibilidade: uma vez que um elemento de rede ou link está com problemas a rota completa deve ser recalculada e todos os elementos reconfigurados.

[1] Esta abordagem tem muita semelhança com o mundo IP no qual o roteamento é feito a cada *hop*

O roteamento no mundo IP pode ser considerado do tipo *in-band*[2], onde os serviços e algoritmos relacionados ao controle da rede são executados no plano de dados, ou posto de outro modo, não há separação entre os planos de dados e de controle. Em abordagens *in-band*, o mesmo plano envolve os aspectos algorítimos e encaminhamentos do roteamento.

O aspecto algorítimo é o esforço realizado pelos protocolos de roteamento, tais como BGP (*Border Gateway Protocol*) e OSPF (*Open Shortest Path First*), para decidir as rotas para encaminhar as primitivas contendo os dados. Em arquiteturas convencionais usadas na Internet, esses protocolos executam em roteadores e analisam metadados de outros (nós vizinhos) para tomar decisão sobre qual rota utilizar ou para controle de congestionamento da rede.

O aspecto do encaminhamento é o envio de um pacote propriamente dito (na arquitetura Internet exercido pelo protocolo IP). No modelo tradicional da Internet, o encaminhamento é realizado por *switches* ou roteadores. Ressalte-se que esses tipos de nós apenas buscam em tabelas internas as informações para encaminhar os dados, nenhuma decisão é tomada no passo do encaminhamento.

O roteamento muda radicalmente utilizando-se arquiteturas SDN (como a ETArch – *Entity Title Architecture*), não sendo executado *in-band*. Nessas arquiteturas, o roteamento é feito *out-of-band*[3], onde o controle é realizado em um plano separado daquele no qual as primitivas de dados serão encaminhadas.

Com o uso de SDN, o roteamento deixa de ser *in-band*, visto que conceitualmente os aspectos algorítmicos residem no âmbito de controle. Note-se que os protocolos de roteamento devem trabalhar *out-of-band*, deixando que o software utilize informações variadas para decisão das rotas. Na ETArch, esse software tende a utilizar requisitos e capacidades das entidades para resolver o encaminhamento de primitivas.

O presente capítulo pretende apresentar uma proposta de roteamento orientado a *workspaces* (*Workspace-driven*). Esse conceito objetiva construir a infraestrutura algorítmica no plano de controle para a escolha da melhor rota. Como na ETArch uma rota não é um caminho entre dois *hosts*, o roteamento tem como papel escolher uma rota por onde serão estendidos (passarão) os *workspaces*.

Um roteamento orientado a *workspace* deve considerar requisitos e capacidades das entidades, sendo essas as informações que serão analisadas em momentos que há tomada de decisão (quando é necessário decidir entre duas ou mais rotas). Uma entidade, ao criar um *workspace*, especifica os requisitos que devem ser atendidos por todas as entidades que porventura desejem fazer parte daquele *workspace*.

Na arquitetura Internet, rotas são definidas considerando a distância entre *hosts* (por exemplo atraso médio ou número de *hops*), sendo essa a análise feita (*in-band*). No roteamento orientado a *workspace* esse tipo de análise será realizado *out-of-band*, considerando

[2] Informações de controle da rede são transmitidas no mesmo canal usado para transmissão de dados
[3] Informações de controle são transmitidas e tratadas em um plano específico de controle

os requisitos para comunicação: rotas que os atendam terão prioridade em detrimento às demais.

A abordagem orientada a *workspace*, ao executar algoritmos de roteamento no plano de controle, define que os NEs não possuem mais a função de rotear, apenas encaminhar primitivas no plano de dados. A primeira implicação dessa abordagem é a flexibilidade de alterar os algoritmos de roteamento, visto que todos os seus protocolos trabalham em controladores SDN, não precisando mais implantar novas funcionalidades em todos os NEs da rede.

A outra implicação da abordagem orientada a *workspace*, em relação aos NEs, está no desempenho. Como o papel de um NE passa a ser apenas encaminhar primitivas no plano de dados (*switching*), não há tempo de processamento dispendido para rotear (*routing*). Entende-se que o tempo total de processamento será apenas o tempo gasto no encaminhamento e o tempo de roteamento (escolha da rota) deverá ser zero (uma vez que a comunicação está estabelecida).

O tempo de roteamento no início de uma comunicação (quando um *workspace* está sendo estendido) é um pouco maior, visto que toda a rota é definida antes que os dados comecem a inundar o *workspace* (*Multicast* proporcionado pela ETArch). A vantagem ocorre quando o *workspace* está estabelecido e a entidade começa a enviar e receber primitivas (não há mais roteamento nesse caso).

Inicialmente o presente capítulo mostrará os aspectos conceituais e arquiteturais do DTS (*Domain Title Service* – elemento fulcral na ETArch) e seus desdobramentos para o roteamento a ser proposto. A segunda seção mostrará a situação atual do roteamento na ETArch. A terceira seção detalhará como é realizado o roteamento *Intra*-DTSA e possíveis melhorias no mesmo para atender requisitos das entidades. Na quarta seção serão definidos os novos serviços necessários para o roteamento Inter-DTSA. Por fim será apresentado o algoritmo de roteamento Inter-DTSA proposto.

4.1 DTS: Aspectos Conceituais e Arquiteturais

Esta seção apresenta uma visão geral do DTS explanando aspectos conceituais, arquiteturais e topológicos fundamentais para o entendimento do roteamento no plano de controle. A ETArch (arquitetura base para roteamento orientado a *workspace*) utiliza o DTS em suas definições e é preparada para ampliar seus serviços de modo a considerar a proposta do presente capítulo.

O DTS é um serviço de domínio (espaço) que gerencia Títulos (nomes) de entidades por meio de serviços específicos de monitoramente e controle. Um Título identifica, univocamente, sem ambiguidade e independentemente da topologia, uma entidade em seu espaço de nomes (*namespace*).

A tarefa de roteamento possui serviços que conceitualmente se encaixam no plano

de controle da rede. Na ETarch, os algoritmos de roteamento são projetados para ser executados como um módulo do DTS, capaz de gerenciar (criar, estender, remover etc) *Workspaces* que ofereçam, a partir de requisitos e capacidades especificados, o melhor caminho em um grafo de NEs.

Uma entidade tem requisitos e capacidades que devem ser geridos pelo DTS na criação e manutenção de *Workspaces*. Inicialmente, suponha uma entidade que deseje se comunicar, seja ela uma aplicação ou um usuário. Para cada instância, as necessidades de comunicação da entidade servem de base para a configuração dos NEs na rede.

O DTS oferece um ambiente capaz de gerenciar as comunicações de entidades, incluindo o roteamento. O ambiente oferecido pelo DTS é composto de agentes (DTSA – DTS *Agents*) que fazem a gestão local de NEs interconectados por topologias variadas. Por exemplo, a Universidade Federal de Uberlândia, na qual um DTSA poderia controlar as comunicações através das redes espalhadas em diversos *campi*.

Um subconjunto de DTSAs pode ser controlado por um tipo especial de agente denominado MDTSA (*Master* DTSA). DTSAs e MDTSAs serão definidos respectivamente nas seções 4.1.1 e 4.1.2. Também serão explanados os conceitos de *Workspaces* de Controle (seção 4.1.3) e os serviços relativos à tarefa de roteamento.

4.1.1 DTS *Agents*

Os DTSAs são agentes distribuídos no plano de controle, que compõem o DTS, responsáveis por controlar elementos de rede, *Workspaces*, entidades, requisitos de QoS/QoE e energia, entre outros. Os controles executados pelo DTSA são feitos através de módulos gerenciadores tais como *Entity Manager*, QoS *Manager*, *Workspace Manager* etc. O presente capítulo propõe um módulo para o gerenciamento de rotas (*Routing Manager*) no plano de controle de uma rede SDN.

O Modelo de Títulos, materializado pela Arquitetura ETArch, introduziu o conceito de *Workspace* (seção 3.1.4) que redefine o conceito de endereçamento. O destino de uma primitiva não é mais uma (*Unicast*) ou várias (*Multicast* ou *Broadcast*) entidades, mas o *Workspace* especificado. Uma primitiva destinada ao *Workspace* especificado é entregue a todas as entidades a ele ligadas (*Attached*).

Uma vez que o DTSA, por meio dos serviços de controle, gerencia os *Workpaces* através dos elementos de rede, isto justifica que as funções de roteamento sejam realizadas pelo DTSA. Entidades e NEs no plano de dados não mais necessitam da função de roteamento uma vez que a configuração da rede para suportar o *Workspace* é feita a *priori* pelo plano de controle. Um DTSA deve conhecer todos os nós (elementos de rede) sob seu controle, bem como as arestas (enlaces) que os ligam.

O DTS (seção 4.1) é um ambiente distribuído composto de um ou mais DTSAs, sendo que, no plano de controle, cada DTSA é um nó do grafo que o (DTS) representa. Um DTSA controla determinada parte da rede exclusivamente, i.e., não controla, nem mesmo

detém informação, sobre a rede controlada por outro DTSA. Um DTSA conhece apenas os DTSAs vizinhos que possui e o caminho para se chegar até esses vizinhos através de NEs denominados NEs de borda.

O DTSA gerencia[4] *Workspaces*, incluindo a seleção de NEs (rota) que tomarão parte no suporte às comunicações. O DTS, ao estender um *Workspace*, reconfigura os NEs selecionados com regras de encaminhamento. Uma vez estabelecido um *Workspace*, o DTSA (agente do DTS efetivamente responsável pela referida extensão) aguarda serviços de controle eventualmente requeridos por outros DTSAs.

O roteamento proposto neste trabalho preconiza que o caminho (rota) das primitivas (*Workspace*) deve ser estabelecido antes que a comunicação se inicie. Ressalte-se que funções de roteamento podem ser invocadas, após o estabelecimento das comunicações, para reconfigurar o *Workspace* (estender ou alterar rotas).

4.1.2 *Master*-DTSA

Um DTSA é um agente capaz de controlar um número limitado de elementos de redes e outros tais da arquitetura ETArch (Seção 4.1.1), sendo, portanto, recomendado para ambientes em escalas de redes locais. Um DTSA constitui um espaço contendo elementos exclusivamente sob seu controle. Uma rede local pode ser vista como uma rede com elementos localizados próximos entre si, como por exemplo uma unidade acadêmica de um campus ou um andar do prédio de uma empresa. Essa limitação do ambiente em 'rede local' é devido a quantidade de entidades que são gerenciadas por um DTSA ao mesmo tempo.

Uma região geográfica – uma cidade, área metropolitana etc – pode conter diversos agentes DTSAs. Uma rede de campus, como por exemplo a rede da UFU, poderia conter vários DTSAs, sendo um para cada unidade acadêmica. Deste modo, o tráfego de controle numa determinada região contendo diversos DTSAs pode ser considerável. Faz-se necessário, portanto, introduzir um mecanismo que seja capaz de otimizar os tráfegos de controle quando houver comunicações entre entidades em regiões distintas. Doravante, denominar-se-á Domínio para o que se chamou de região neste parágrafo.

A arquitetura ETArch propõe um tipo especial de agente, denominado MDTSA (*Master* DTSA), capaz de fazer a interface entre domínios distintos em que cada domínio hospede um ou mais DTSAs. Considerando que *Workspaces* podem ser estabelecidos envolvendo DTSAs existentes em dois ou mais domínios, haverá necessidade de roteamento entre dois ou mais MDTSAs. Observe-se que o tráfego de controle inter domínios será feito essencialmente entre MDTSAs. MDTSAs, assim como DTSAs, são agentes do plano de controle e, portanto, não fazem uso do plano de dados.

Por exemplo, a UFU teria um MDTSA responsável por internalizar primitivas de controle externas e também de externalizar primitivas oriundas de DTSAs sob seu domínio.

[4] Gerenciamento envolve a criação, extensão, reconfiguração e remoção de *Workspaces*

Note-se que o MDTSA é responsável por conhecer a topologia do domínio, ou seja, o MDTSA mantém um grafo de DTSAs (nós) e as respectivas ligações (arestas).

Todas as informações acerca de *Workspaces* presentes em um DTSA devem ser conhecidas pelo MDTSA. Isso significa que, quando uma entidade cria ou estende um *Workspace*, o DTSA, que controla localmente a referida entidade, deverá informar ao MDTSA do domínio. Um MDTSA conhecer as informações sobre *Workspaces*, mantidos por DTSAs sob seu domínio, facilita a tarefa de roteamento. Se um $MDTSA_1$ precisa estender, para um de seus DTSAs, um *Workspace* existente em outro domínio, controlado por $MDTSA_2$, o $MDTSA_1$ precisa armazenar informações sobre o *Workspace* estendido, pois isto otimizará futuras extensões.

Um MDTSA pode receber requisições indicando procura por *Workspaces*. Se o MDTSA acusa a existência do *Workspace*, então ele devolve uma resposta com as informações relativas ao *Workspace* especificado. Em caso negativo, o MDTSA encaminha a requisição de modo que outro(s) MDTSA(s) receba(m) a solicitação relativa ao *Workspace* especificado.

As funcionalidades de MDTSAs foram concebidas para ser um superconjunto daquelas especificadas para um DTSA e, portanto, pode-se afirmar que um MDTSA pode desempenhar o papel de um DTSA, mas a recíproca não é verdadeira. Deste modo, na inicialização da rede, o DTS elege quais DTSAs farão o papel de *Masters*, sendo que os demais DTSAs serão configurados para 'apontar em direção ao seu' *Master*.

Considerando que as comunicações entre DTSAs são exclusivamente relativas ao plano de controle da rede, com características de tráfego (tamanho de primitivas, vazão, temporalidade etc) especiais, então, na inicialização da rede serão estabelecidos *Workspaces* exclusivos para os tráfegos de primitivas de controle denominados '*Workspaces* de Controle' (Seção 4.1.3).

4.1.3 *Workspace* de Controle

Workspace de Controle é um tipo de *Workspace* (Seção 3.1.4) exclusivo para as comunicações de controle da rede. *Workspaces* são essencialmente dinâmicos, criados/removidos ou estendidos sob demanda de DTSAs, sendo que as sessões de comunicações começam após um intervalo de tempo necessário para seu estabelecimento.

Considerando que as comunicações de controle apresentam características de tráfego especiais (poder-se-ia dizer 'QoS de Controle'), considerando que eventualmente essas comunicações não poderão esperar pelo intervalo de tempo requerido para o estabelecimento de *Workspaces* sob demanda e considerando que as entidades (agentes) comunicantes são conhecidos *a priore*, então os *Workspaces* de Controle são criados na inicialização da rede e mantidos durante seu funcionamento.

O *Workspace* de Controle é utilizado para o roteamento Inter Domínio, pois as comunicações entre DTSAs são a razão para a existência desse tipo de *Workspaces*. A Figura

16 apresenta, em azul tracejado, um *workspace* de controle. Observa-se que ele deve ser configurado na inicialização da rede.

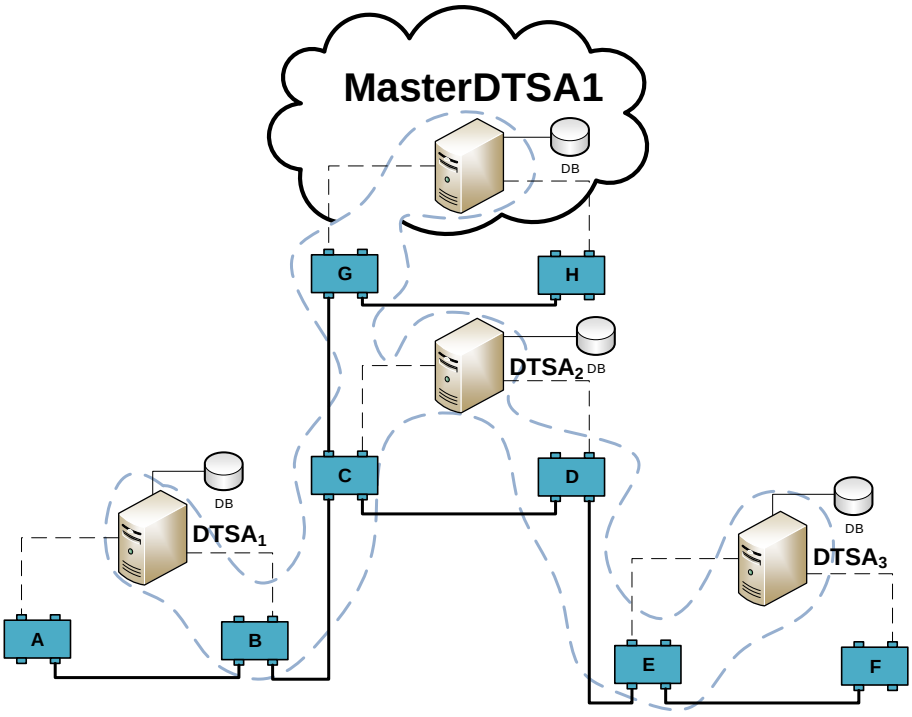

Figura 16 – Controle na topologia de um *Master*-DTSA

Uma entidade pode requisitar a busca por um *Workspace* de Dados que nesse momento não seja conhecido por seu DTSA. Nesse cenário há três possibilidades de comunicações no(s) plano(s) de controle: i) entre DTSAs; ii) entre um DTSA e seu respectivo MDTSA; e, iii) entre MDTSAs.

Observe que os casos (i) e (ii) acontecem sob o domínio de um MDTSA e que o caso (iii) envolve mais de um MDTSA e, portanto, diversos domínios. Por este motivo, a arquitetura ETArch introduz dois tipos de *Workspaces* de Controle: Privado, para atender às demandas de comunicações dos casos (i) e (ii); e, Público, para atender ao caso (iii). Observe-se que o termo 'Público' não quer dizer acesso irrestrito, mas tem analogia como as redes *Carriers* em telecomunicações, i.e., é um tipo de *Workspace* de Controle que tem a capacidade de suportar as interações de controle inter domínios.

O *Workspace* de Controle pode ser composto de dois ou mais DTSAs e MDTSAs. Na prática cada DTSA é incluído em *Workspaces* de Controle através de elementos de rede (*switches*), similar a roteadores de borda utilizados por arquiteturas tradicionais (como no protocolo BGP, por exemplo). Dessa forma, cada DTSA possui informação sobre qual de seus elementos de rede consegue comunicação para determinado *Workspace* de Controle, ou seja, para qual elemento de rede e porta ele deve enviar suas primitivas de controle.

A Figura 16 apresenta o *Workspace* de Controle de um *Master* DTSA. Observa-se que os $DTSA_1$ e $DTSA_3$ não possuem comunicação direta com $MasterDTSA_1$ e, portanto, precisam requisitar o serviço *Workspace_Lookup*[5] por meio do $DTSA_2$. O $MasterDTSA_1$, conhecedor da topologia, consegue responder as solicitações corretamente. Além disso, um dado *Workspace*, que esteja sob algum DTSA na topologia do $MasterDTSA_1$, pode ter mais de um caminho para ser estendido.

Na Figura 16, se o $DTSA_1$ não possui informações sobre um *Workspace* especificado, ele deverá requerer o serviço WORKSPACE_LOOKUP contendo o título do *Workspace* especificado para o *switch* B, que é um NE de saída (borda) para o $DTSA_2$. A primitiva será encaminhada para a porta de saída conectada ao *switch* C, que a encaminhará para a porta de saída no *switch* G e, então, será encaminhada ao $MasterDTSA_1$.

Na Figura 16, é possível notar que todo DTSA ou MDTSA possui capacidade de armazenamento representada por DB (*Database*). O DB de um DTSA contém as informações relativas a *Workspaces* criados por entidades controladas pelo DTSA, além de informações de *Workpsaces* estendidos a partir outros DTSAs. Cada implementação pode definir a melhor maneira de implantar o DB de seu DTSA, podendo utilizar bancos relacionais, em memória, NoSQL etc.

Ao receber a indicação de uma primitiva WORKSPACE_LOOKUP, o MDTSA verifica se algum DTSA de seu domínio tem as informações sobre o *Workspace* especificado na primitiva recebida (armazenado previamente quando foi estendido ou criado).

A Figura 16 apresenta (tracejado) um *Workspace* de Controle Privado, utilizado para comunicação entre DTSAs sob o mesmo MDTSA (apenas um MDTSA é utilizado nesta figura).

Workspaces de Controle Públicos são necessários, pois um dado *Workspace* de Dados pode ser requerido e não existir no domínio de um dado MDTSA. Nesse caso, o MDTSA deve entrar em contato com outros MDTSAs, por meio de *Workspaces* de Controle Públicos, como mostra a Figura 17.

As comunicações entre MDTSAs, analogamente ao que acontece na topologia do domínio de um MDTSA, são realizadas através de NEs de borda que os conectam. A Figura 17 mostra a comunicação entre os $MasterDTSA_1$ e $MasterDTSA_2$. Na figura, o *Workspace* de Controle Público é representado pela linha tracejada na cor verde.

O conhecimento geral da topologia é feito na inicialização da rede. Portanto, arquivos de configuração são necessários para especificar a topologia de rede envolvendo diferentes DTSAs. Cada DTSA deve conhecer seus vizinhos e a forma como eles se conectam. Assim como acontece no protocolo BGP, um DTSA deve conhecer qual NE está na borda do seu domínio.

Um arquivo de configuração para a Figura 16 é apresentado na Figura 18. Esse arquivo, contendo uma especificação XML (*eXtensible Markup Language*), mostra quais

[5] *Workspace_Lookup* é um serviço que procura por *Workspaces* em outros DTSAs

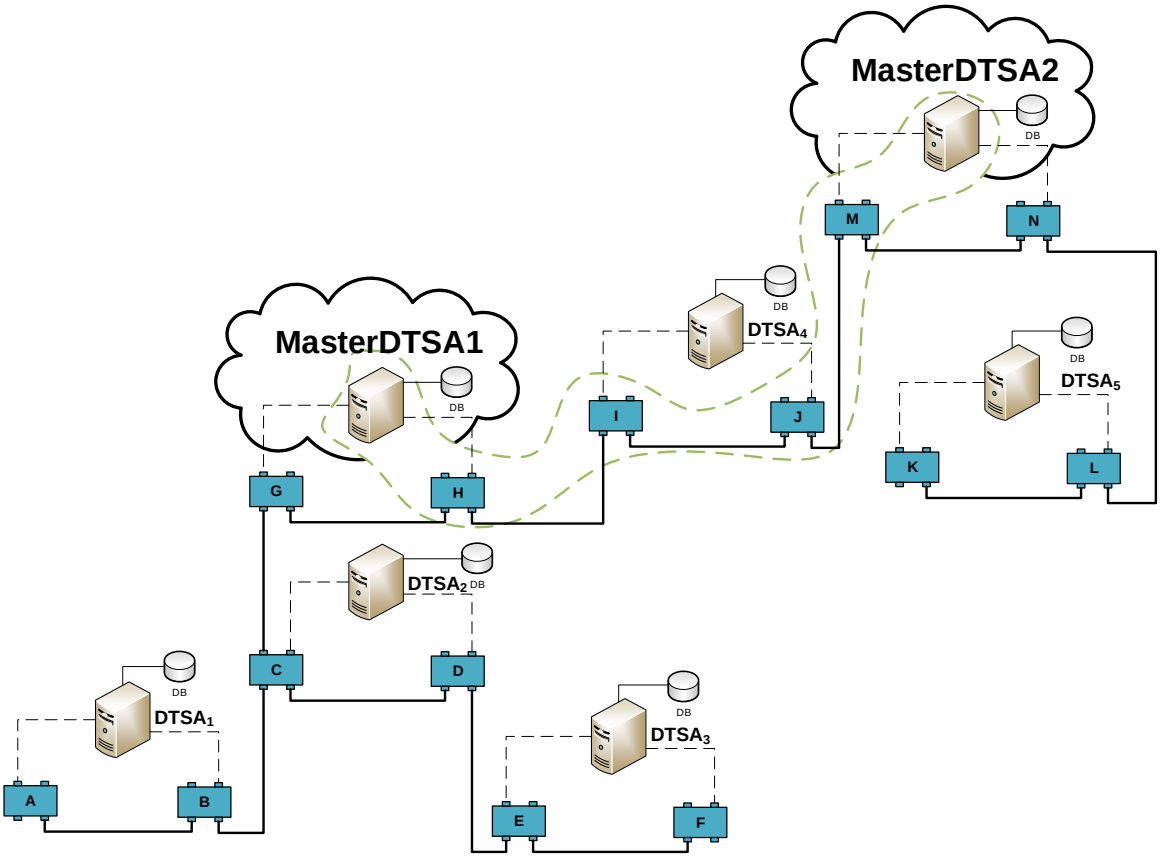

Figura 17 – Controle na topologia Inter-*Master*-DTSA

são os vizinhos (*neighbors*) do $DTSA_1$ da Figura 16. Todos os DTSAs que entram na rede precisam ter uma configuração similar, para saberem como chegar até seu MDTSA. Além disso, é necessário especificar o NE de borda e a porta desse NE, pois é através desse NE de borda que as primitivas de controle inter domínio serão enviadas.

Os principais elementos da Figura 18 são: a identificação do DTSA em questão, através da *tag* $dtsa-title$, que determina o título daquele DTSA; o tipo do *workspace* de controle ao qual aquele DTSA faz parte, na *tag workspace*; a identificação se é um *Master*-DTSA, na *tag master*; e o nível na *tag level*.

A Figura 18 contém ainda informações sobre o vizinho do $DTSA_1$. A *tag neighbor* contém: o Título do vizinho, na *tag title*; o nome do NE de borda que conecta ao vizinho, na *tag* $ne-title$; e a porta de saída para o vizinho, na *tag* $phisical-link$. Essas informações sobre o NE e porta de saída do vizinho são importantes para que o DTSA saiba para qual de seus NEs, e qual porta de saída no NE, enviar primitivas de controle.

A comunicação entre diferentes DTSAs passa pelos próprios NEs da topologia e essa comunicação é mostrada na Figura 19. Utilizando *switches* OpenFlow é possível injetar primitivas destinadas a outros DTSAs. O procedimento começa com o DTSA enviando para seu NE da borda uma primitiva PACKET_OUT (do protocolo OpenFlow), com a

```
<dts-resolver>
    <dtsa-title>DTSA1</dtsa-title>
    <workspace>DTSAPRIVATE</workspace>
    <master>false</master>
    <level>1</level>

    <address>
        <ne-title>B</ne-title>
        <phisical-link>2</phisical-link>
    </address>

    <neighbors>
        <neighbor>
            <title>DTSA2</title>
            <address>
                <ne-title>B</ne-title>
                <phisical-link>3</phisical-link>
            </address>
        </neighbor>
    </neighbors>
</dts-resolver>
```

Figura 18 – Configuração de DTSA vizinho

porta de saída do enlace que conecta esse NE ao NE de borda do outro DTSA (próximo vizinho). A primitiva PACKET_OUT encapsula uma das primitivas ETArch, que pode ser um WORKSPACE_LOOKUP, por exemplo.

Uma primitiva PACKET_OUT, ao chegar a um NE de borda, é encaminhada para a porta de saída com destino ao próximo vizinho no caminho para o MDTSA. É necessário esse procedimento, pois, para se formar o *Workspace* de Controle, cada NE possui regras determinando o envio de primitivas de controle ao MDTSA.

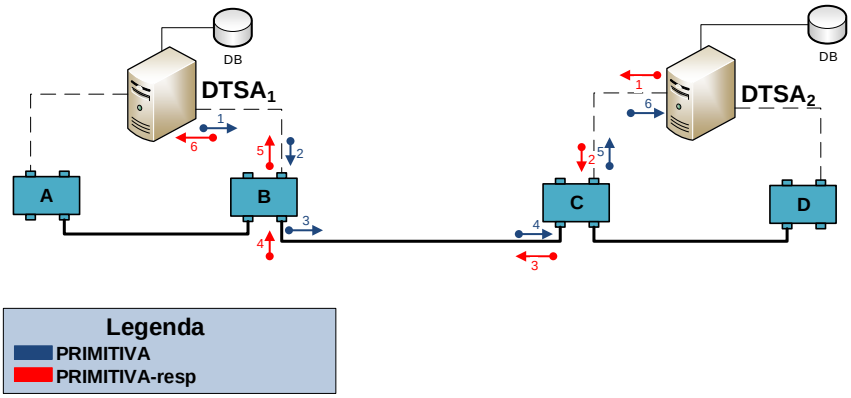

Figura 19 – Sequência de comunicação entre dois DTSAs

A arquitetura ETArch estabelece que um DTSA é um tipo de entidade, então, todo DTSA precisa se registrar em seu MDTSA. Observe-se que na inicialização da rede os

Workspaces de Controle devem ser configurados de forma que as primitivas de controle do protocolo DTSCP sejam conduzidas entre diferentes DTSAs até seu MDTSA.

A Figura 19 mostra a sequência de comunicação entre $DTSA_1$ e $DTSA_2$, quando o $DTSA_1$ envia uma primitiva de controle para o $DTSA_2$. A sequência é análoga para qualquer primitiva de controle (do protocolo DTSCP), exceto WORKSPACE_LOOKUP, pois essa deverá ser enviada até o MDTSA.

Na Figura 19, as setas em azul mostram o envio da primitiva, inicialmente saindo do $DTSA_1$ e entrando no NE B, com destino a porta de saída para o NE C. Uma vez que os NEs são *switches* OpenFlow (neste trabalho), eles não possuem informação da primitiva. No passo 4, a primitiva chega no NE C, que não contém regras para aquele tipo de informação. Portanto, o NE C criará uma primitiva PACKET_IN, com as informações que chegaram, e enviará ao seu controlador (DTSA), que tratará a primitiva.

As setas vermelhas, na mesma Figura 19, mostram o retorno de uma primitiva no plano de controle. Análogo ao envio, as primitivas de retorno passam pelos NEs de borda até chegarem ao $DTSA_1$ que enviou a mensagem original.

4.1.4 Organização do DTS

Uma das principais funcionalidades do DTS é resolver Títulos para localizar seu endereço corrente. Considerando que o DTS pretende ser um sistema distribuído em nível mundial, então é necessário definir sua organização para que o tempo de resolução seja o menor possível.

O DTS é composto de um tipo de entidade, que se comporta como um agente, denominada DTSA (DTS *Agent* – Seção 4.1). Em princípio, um DTSA controla um domínio local (uma pequena empresa, uma residência, por exemplo), sendo que os MDTSAs controlam domínios mais amplos (um campus, uma cidade, por exemplo). Considerando que há uma hierarquia envolvida nestas relações e que há necessidade de resoluções que podem envolver nível mundial, então há que se pensar em uma organização em níveis (*tiers*) para diminuir o tráfego em toda a rede e também para tornar o tempo de resposta aceitável.

Por exemplo, poder-se-ia pensar em um nível de granularidade mais fina envolvendo organizações ou residências, em seguida um nível de campus ou cidade, então um nível de estado e, assim sucessivamente, até chegar a um nível de países ou continentes.

É uma abordagem semelhante àquela adotada pela Internet no que se refere aos *Tiers*. De fato, não existe no mundo Internet uma autoridade que defina o termo *Tier*. Entretanto, a definição mais comum de uma rede "*Tier 1* é uma rede que pode chegar a qualquer outra rede na Internet sem a aquisição de tráfego IP ou sem pagamento pelo uso". Por esta definição, uma rede *Tier 1* é uma rede de tráfego livre com outras redes *Tier 1*.

A arquitetura ETArch organizou o DTS em quatro níveis sendo: Nível Local – com o escopo para empresas, campi, residências etc; Nível Regional – que abrange diversos

Níveis Locais – poderia ser visto como empresas provedoras de serviços de acesso (como os ISPs da Internet); Nível Nacional – que abrange diversos Níveis Regionais – com escopo para estados ou regiões metropolitanas; e o Nível Raiz ou Global (a organização do DTS é uma árvore invertida) que teria um aspecto semelhante às redes *Tier 1*.

O Nível Raiz é o último nível ao qual uma requisição para resolução de *Workspaces* de Dados pode chegar. A interface entre dois níveis da organização do DTS é desempenhado por um MDTSA (*Master* DTSA). Se os MDTSAs do nível Raiz não localizarem um *Workspace* de Dados especificado, pode-se afirmar que esse *workspace* não existe. A Figura 20 apresenta uma possibilidade de organização do DTS. Note-se que o roteamento orientado a *workspace* não é feito de forma hierárquica e a disposição dos elementos na figura têm o intuito único de apresentar a formação de *Tiers*.

Na Figura 20, os retângulos representam NEs do plano de dados, responsáveis por encaminhar primitivas contendo dados das aplicações. Os círculos brancos representam DTSAs, que controlam determinadas topologias dos NEs (plano de dados). Os círculos na cor verde representam *Master*-DTSAs, na visão da borda (interface) de cada nível.

Observa-se na Figura 20 que dois MDTSAs estão controlando topologias no Nível Local. O *Master* de título M_{0101} é responsável por um domínio contendo três DTSAs: D_1, D_2 e D_3. A figura mostra na cor azul um *Workspace* de Controle Privado que abrange os três DTSAs e seu *Master* M_{0101}. Todas as entidades conectadas aos NEs D_1, D_2 e D_3 criam e estendem *Workspaces* de Dados no Nível Local, que são conhecidos por M_{0101}.

Quando um MDTSA no Nível Local não possui informações sobre determinado *Workspace* de Dados (solicitado por alguma entidade do seu domínio) significa que esse *Workspace* não está presente naquele domínio. Nesse caso, o MDTSA deverá requisitar o *Workspace* de Dados (procurá-lo) no Nível Regional. Essa procura é feita através da troca de primitivas no *Workspace* de Controle Público (representado na Figura 20 pela cor vermelha).

Os MDTSAs do Nível Local estão ligados logicamente aos MDTSAs do Nível Regional, apesar de fisicamente ser o *Workspace* de Controle Público quem faz o encaminhamento das primitivas de controle entre os MDTSAs. No Nível Regional, os MDTSAs possuem informação do *Workspace* de Controle Público desse Nível. Caso o *Workspace* de Dados especificado (objeto da busca) não esteja no Nível Regional, o MDTSA desse nível deve solicitar pela procura no Nível Nacional.

Análoga à ligação entre os níveis Local e Regional, a comunicação de MDTSAs com o Nível Nacional é feita através de *Workspaces* de Controle Públicos. Da mesma forma, os MDTSAs presentes no Nível Nacional possuem informações referentes à *Workspaces* de Dados publicizados nacionalmente. Caso a informação não exista o nível mais alto é solicitado, o Nível Global.

No Nível Global, há o armazenamento de informações dos *Workspaces* de Dados conhecidos mundialmente. Nesse nível há a comunicação entre seus próprios MDTSAs

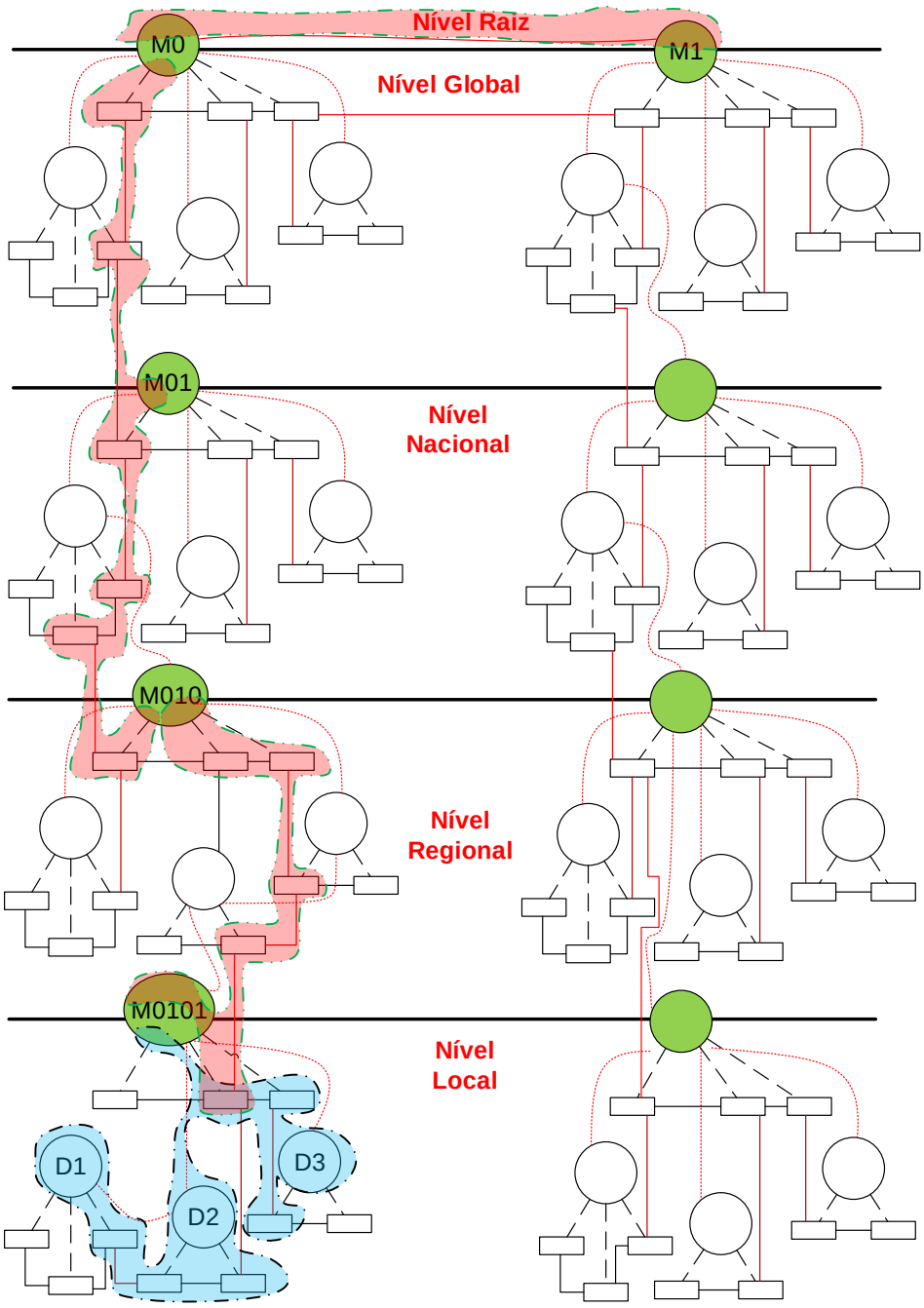

Figura 20 – DTS em níveis e Roteamento em larga escala

(*Peering*) apresentados na Figura 20. Observa-se que a comunicação nesse nível é a última de controle da rede. Se o *Workspace* de Dados não for encontrado nesse nível, então conclui-se que ele não existe.

A representação de cor vermelha na Figura 20 mostra os *Workspaces* de Controle Públicos presentes na ETArch. Além disso, essa representação pode ser considerada como a exposição de níveis (*Tiers*), como dito anteriormente, que é conforme a Internet atual trabalha. No Nível Raiz tem-se o *Tier 1*, considerado como núcleo da rede. No Nível

Nacional tem-se o *Tier* 2, e no Nível Regional o *Tier* 3. O Nível Local pode possuir o *Tier* 4, por exemplo.

Na Figura 20 a ligação do *Tier* 1 é o *Workspace* de Controle Público entre M_0 e M_1. O *Tier* 2 está na ligação entre M_0 e M_{01}. Para o *Tier* 3 tem-se a ligação entre M_{01} e M_{010}, e por fim a ligação entre M_{010} e M_{0101} (*Tier* 4).

O roteamento orientado a *Workspace* (*Workspace-driven Routing*) deve trabalhar com a organização em níveis do DTS. Observe-se que na Figura 20 sempre que um *Workspace* de Dados é criado, ele deve ser publicado no respectivo MDTSA (Local). Após isso, uma publicação no MDTSA regional pode ser feita (para *Workspaces* regionais) e assim sucessivamente. Uma busca deve ser iniciada quando um *Workspace* é requerido por alguma entidade (ela requisita fazer *attach*[6]).

Após a busca ser concluída, o *Workspace* de Dados especificado deve ser estendido até chegar à entidade que o solicitou. O Roteamento é feito, para determinar por quais DTSAs o *Workspace* vai passar, para chegar até a entidade requisitante. O algoritmo proposto no presente capítulo vai detalhar como a escolha do caminho pode ser multiobjetivo, até mesmo para topologias globais.

4.2 Roteamento ETArch: *Status Quo*

Atualmente o roteamento na ETArch já é feito no plano de controle, separadamente do plano de dados, mas funciona com apenas um DTSA. Este único DTSA controla toda a topologia e aplica um algoritmo de menor caminho para os elementos de rede sob seu controle. O roteamento ETArch atual se inspira nas redes ATM, onde os caminhos virtuais (VP – *Virtual Path*) (rotas) são definidos *a priore*, antes que a comunicação de dados comece.

Com a utilização de apenas um DTSA, para o controle de todos os NEs da rede, a ETArch faz o roteamento Intra-DTSA. Nesse caso, as entidades ligadas aos NEs criam novos *workpsaces* e estendem os que foram criados por outras entidades. A comunicação se dá entre entidades controladas pelo mesmo DTSA. Caso uma entidade solicite um *workspace*, do qual o DTSA não tenha informações, o DTSA vai negar a requisição da entidade.

Quando uma entidade solicita um *workspace*, e seu DTSA de controle não o conhece, ocorre o roteamento Inter-DTSA. Nesse caso, o DTSA deverá se comunicar com outros DTSAs para buscar informações sobre o *workspace* especificado. A ETArch deve considerar os requisitos das entidades em ambos os roteamentos Intra e Inter-DTSA.

O objetivo deste capítulo é portanto melhorar o roteamento Intra-DTSA atualmente utilizado na ETArch e apresentar uma proposta para o roteamento Inter-DTSA. A abordagem orientada a *workspace* é aplicada a ambos os roteamentos na ETArch e é devida

[6] Solicitação de uma entidade para se juntar a determinado *Workspace*

ao uso de *workspaces* no plano de controle, para troca de primitivas entre dois ou mais DTSAs, que é necessária a busca de informações no roteamento Inter-DTSA.

As comunicações do plano de controle entre DTSAs serão feitas por meio de '*Workspaces* de Controle', que são criados a *priore* e servem aos propósitos de interações de DTSAs. As comunicações das entidades (aplicações) se darão no plano de dados e serão feitas por meio de '*Workspaces* de Dados', que serão criados sob demanda para atender às comunicações em geral das entidades. Doravante, por simplicidade, '*Workspaces* de Dados' serão referenciados apenas como *Workspaces*.

4.3 Roteamento Intra Domínio

O roteamento Intra Domínio (Intra-DTSA) na ETArch tem como objetivo estabelecer rotas para comunicações entre entidades conectadas a NEs controlados por um único DTSA. A vantagem desse tipo de roteamento é o conhecimento total da topologia de rede que o DTSA possui. Em tese o DTSA deve possuir informações completas sobre os NEs que controla, como por exemplo capacidade dos enlaces, quantidade de portas em cada NE etc.

O conhecimento e controle de toda a topologia de rede permite que diversos algoritmos possam ser implementados no DTSA. Como exemplo, pode-se criar um algoritmo para controlar congestionamento de enlaces ou, até mesmo, um algoritmo para separar enlaces que sejam pouco utilizados e possam ser desligados temporariamente para economia de energia. Outra vantagem é o controle total que o DTSA possui sobre todos os *workspaces* criados por suas entidades.

No roteamento Intra-DTSA, no qual não há necessidade de trocar primitivas entre diversos DTSAs, o uso do *workspace* de controle não é necessário. Note-se que esse roteamento está limitado apenas ao(s) caso(s) em que o DTSA conhece o(s) *workspace*(s). O DTSA pode conhecer um *workspace* por dois motivos: (i) uma das entidades que ele controla criou o *workspace*; ou, (ii) uma das entidades que ele controla se ligou (estendeu) um *workspace* existente.

Caso uma entidade, controlada por um DTSA, estenda um *workspace* existente em outro DTSA, o DTSA da entidade requisitante armazena informações sobre o *workspace* requisitado, para diminuir (ou evitar) buscas Inter-DTSA. Isso significa que se uma entidade qualquer requisitar um *workspace* já estendido, o DTSA não precisará se comunicar com outros, pois as informações do *workspace* já foram armazenadas em seu banco de dados local.

O roteamento Intra-DTSA pode ser realizado através da classe estado de enlace. Inicialmente não é necessário que os NEs guardem nenhuma informação, visto que informações de definição da rota são de responsabilidade do plano de controle. A ideia é que o DTSA guarde a topologia, em forma de grafo, dos seus NEs. Quando um novo NE é inserido,

para ser controlado pelo DTSA, a topologia armazenada deve ser atualizada.

Inicialmente o DTSA não possui informações sobre as entidades, apenas sobre os NEs que controla. Porém, a ETArch define que uma entidade deve fazer seu registro junto ao DTSA para poder participar de qualquer comunicação (criar ou estender *workspaces*). Nesse registo, o DTSA recebe da entidade seus requisitos e capacidades e deve armazená-las para uso posterior.

Uma entidade pode criar um *workspace* após se registrar. A ideia é que a entidade informe ao DTSA quais requisitos o *workspace* deve satisfazer. Nesse momento o *workspace* é armazenado no DTSA e nenhuma rotina de roteamento é realizada, pois a única regra de encaminhamento criada é no NE ao qual a entidade está fisicamente conectada. Essa regra é um fluxo na tabela de fluxos do NE determinando que todas as primitivas destinadas àquele *workspace* devem ser entregues na porta de ligação entre entidade e NE.

O procedimento de roteamento deve ser realizado quando uma entidade deseja se ligar a um *workspace* existente. A entidade requisita a ligação a um *workspace* e o DTSA resgata informações no seu *storage* local. O DTSA, que conhece as capacidades da entidade, decide se a extensão do *workspace* pode ser realizada. Além disso, o DTSA deve analisar a topologia da rede para verificar a melhor rota.

Como exposto na Seção 4.2, a ETArch considera o menor caminho entre os elementos de rede para definir a melhor rota. O propósito deste trabalho é incrementar os objetivos de roteamento com as informações que o DTSA possui da topologia e dos NEs para determinar a rota. O DTSA define a melhor rota baseando-se nos requisitos necessários informados na criação do *workspace*, ou seja, por onde o *workspace* será estendido ao longo da topologia (entre NEs) para chegar até o NE da entidade que solicitou a extensão (*attach*).

Como os requisitos foram inicialmente informados na criação do *workspace*, o DTSA tem ciência da comunicação em curso no *workspace*. Uma aplicação com características de telefonia, por exemplo, difere de uma aplicação que transfira arquivos de texto e assim por diante, para diferentes tipos de aplicações.

A Figura 21 apresenta uma topologia onde há necessidade de roteamento Intra-DTSA. Na figura, um servidor de vídeo cria um *workspace* com o título Wks-FinalCopaAmerica2015-live. Esse servidor pretende realizar o *streaming* ao vivo para um jogo de futebol, utilizando um *codec* MPEG-4 para transmitir com resolução de 1080p, e como requisitos para a comunicação exige uma *bandwidth* de 1800 kbps, além de eficiência energética.

Observa-se na Figura 21 que o todos os elementos de rede estão ligados fisicamente ao $DTSA_1$, que conhece a topologia da rede. Os enlaces entre elementos de rede estão em cores diferentes e o $DTSA_1$ já determinou um valor para cada enlace, considerando o tempo médio de resposta entre cada NE (o enlace verde, por exemplo, possui menor tempo de resposta que o amarelo). Observa-se ainda um notebook conectado ao NE_1.

Na Figura 21, o roteamento Intra-DTSA vai ocorrer caso uma entidade residente no

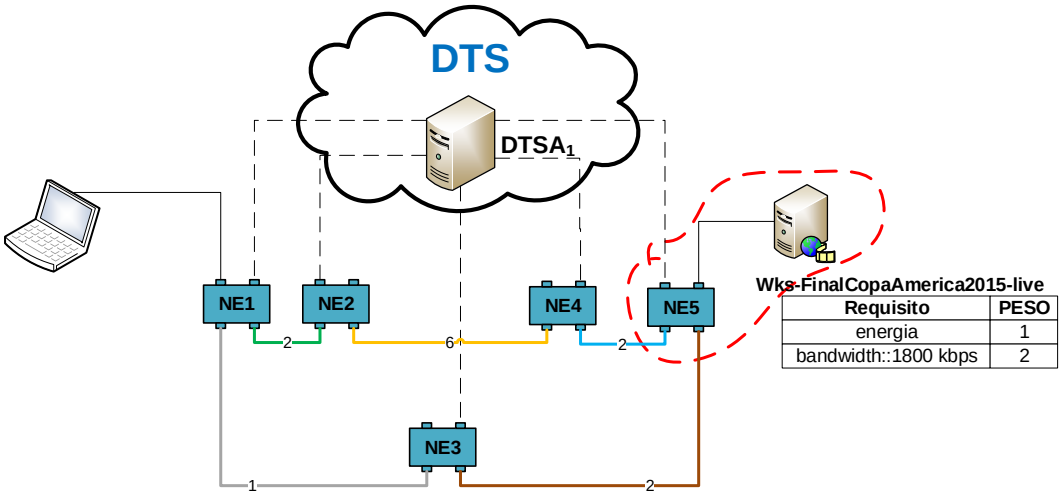

Figura 21 – Roteamento Intra-DTSA

notebook conectado ao NE_1 deseje se ligar ao *workspace* Wks-FinalCopaAmerica2015-live. O roteamento vai ser necessário, pois o *workspace* pode ser estendido até o notebook por dois caminhos de NEs diferentes: NE_5–NE_4–NE_2–NE_1 ou NE_5–NE_3–NE_1.

O primeiro passo do algoritmo de roteamento Intra-DTSA é verificar os requisitos necessários para realizar a comunicação. A Figura 21 mostra que o servidor de vídeo requisitou da rede garantia de 1800 kbps com peso maior, ou seja, deverá ter ponderação maior para ser atendido. O outro requisito solicitado, com peso menor, é eficiência energética. Os requisitos são aplicados pelo $DTSA_1$ sempre que uma primitiva WORKSPACE_ATTACH é requisitada ao $DTSA_1$.

O notebook da Figura 21 requisita um WORKSPACE_ATTACH. Nesse momento o $DTSA_1$ calcula as possíveis rotas para estender o *workspace*, obtendo como resultado duas sequências de NEs: $R(NE_1, NE_2, NE_4, NE_5)$ e $R(NE_1, NE_3, NE_5)$, onde $R(x)$ é o conjunto de NEs que formam uma rota R. Pela figura observa-se que $R(NE_1, NE_3, NE_5)$ é uma rota menor, tanto pelo tempo nos enlaces quanto pelo número de *hops*. Porém, o $DTSA_1$ deve observar a banda disponível nos enlaces e também o consumo de energia em cada rota.

O $DTSA_1$ seleciona a melhor rota após verificar quais rotas atendem os requisitos do *workspace*. Na Figura 21 pode-se supor que as duas rotas possíveis para estender Wks-FinalCopaAmerica2015-live até o notebook atendam 1800 kbps em largura de banda. Porém, pode-se supor que o consumo de energia em $R(NE_1, NE_3, NE_5)$ é maior que em $R(NE_1, NE_2, NE_4, NE_5)$. Dessa forma, o $DTSA_1$ deve optar pela rota $R(NE_1, NE_2, NE_4, NE_5)$. Em outra suposição as duas rotas atendem todos os requisitos, nesse caso o $DTSA_1$ deverá decidir pelo menor caminho.

Na Figura 21 os pesos dos requisitos do *workspace* são definidos pela entidade que

solicita a criação do *workspace*. Contudo deve-se ressaltar que o DTSA, conhecedor de todos os *workspaces* em seu domínio, pode alterar os pesos de forma a tentar atender o máximo de requisitos de todos os *workspaces* diferentes trabalhando simultaneamente. O DTSA toma essa decisão pois a tentativa de atender os pesos solicitados pelas entidades pode interferir no funcionamento de outros *workspaces*.

O diagrama de atividades da Figura 22 apresenta o algoritmo de roteamento Intra-DTSA orientado a *workspace*. Sempre que um WORKSPACE_ATTACH chega a um DTSA, que contém informações sobre o *workspace* especificado, o algoritmo deve ser executado. Caso o DTSA não tenha informações sobre o *workspace*, o serviço de *lookup* deve ser invocado, sendo esse um procedimento do roteamento Inter-DTSA (descrito na seção 4.5).

O diagrama representado na Figura 22 se inicia quando o DTSA recebe a primitiva WORKSPACE_ATTACH. O primeiro passo é buscar a topologia da rede (grafo de NEs) controlada pelo DTSA. O segundo passo é determinar quais NEs estenderam o *workspace*, que pode ter sido estendido por diversos NEs da rede (caso entidades desses NEs já tenham feito WORKSPACE_ATTACH outrora).

Na Figura 22 há dois cálculos de rota, as ações "Calcular rotas para o NE mais próximo" e "Calcular rotas até o destino". A primeira significa que o DTSA deverá analisar se existe um NE que estende o *workspace* mais próximo que o NE da entidade que o criou. Caso possua e atenda os requisitos o DTSA deve estender o *workspace* a partir desse NE. Caso contrário, o DTSA deverá calcular a rota desde o NE que criou o *workspace*.

Sempre que houver várias rotas que atendam aos requisitos de um *workspace*, o DTSA deve executar o algoritmo de Dijkstra (DIJKSTRA, 1959) para definir a melhor rota. O algoritmo proposto na Figura 22 finaliza quando nenhuma rota foi encontrada ou sempre que uma rota é definida.

Após o algoritmo da Figura 22 finalizar, o DTSA deverá criar e enviar aos NEs as regras de encaminhamento necessárias para estabelecer a rota que o algoritmo definiu. Como o DTSA possui informações completas da topologia, é possível que o mesmo defina as novas configurações, ou seja, quais NEs deverão ser configurados e quais as portas de saída em cada NE.

Pode-se considerar que o algoritmo de roteamento Intra-DTSA proposto nesta seção pode ser adequado para um algoritmo multiobjetivo. Isso significa que mais de um objetivo é considerado para definição da rota e esses objetivos são avaliados como consequência dos requisitos que os *workspaces* e as entidades solicitam na comunicação. O *workspace* na Figura 21 requisitou *bandwidth* e eficiência energética, portanto, as decisões de rotas para essa topologia devem ter como objetivo atender esses dois requisitos.

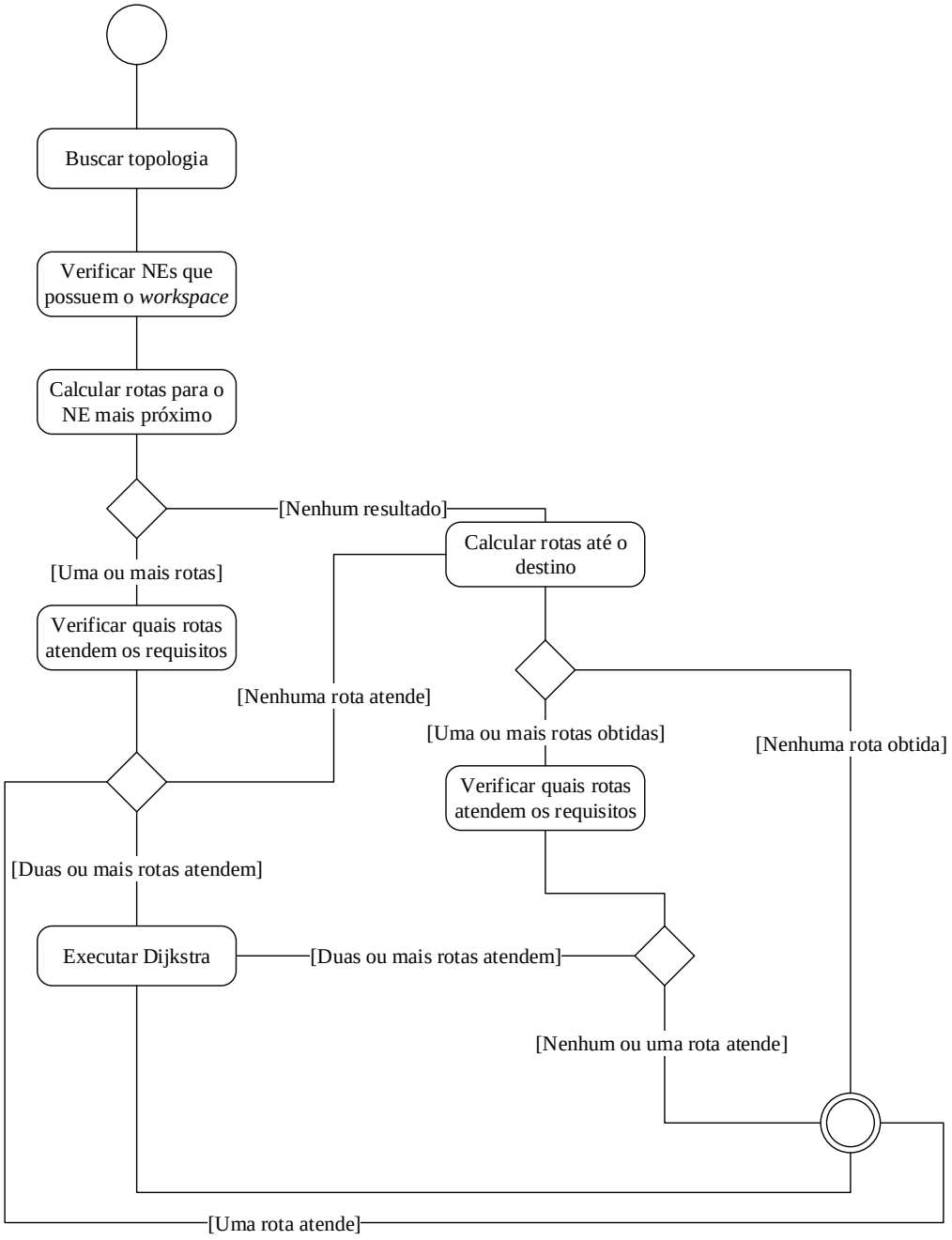

Figura 22 – Algoritmo de roteamento Intra-DTSA

4.4 Serviços de Roteamento Orientado a *Workspaces*

O protocolo DTSCP (DTS *Control Protocol*) oferece serviços de controle, que servem às comunicações entre dois ou mais DTSAs, para suporte ao roteamento Inter-DTSA. Considerando a situação atual (Seção 4.2), novos serviços devem ser disponibilizados para serviços específico, a serem descritos nesta seção.

A Tabela 2 resume os serviços utilizados para o roteamento, sendo que o serviço

WORKSPACE_CONFIGURATION não é original da arquitetura ETArch. Este serviço é introduzido, pois, quando o MDTSA encontra um *Workspace* e o DTSA solicitante escolhe uma das rotas, uma primitiva deve ser enviada aos DTSAs selecionados para que possam configurar seus NEs.

Tabela 2 – Primitivas relacionadas ao roteamento.

Primitiva	**Descrição**
WORKSPACE_LOOKUP	O roteamento utiliza essa primitiva quando um DTSA não possui informações sobre um *workspace*. A primitiva é encaminhada pelo DTSA até o seu *Master*-DTSA, podendo passar por outros pelo caminho.
DTS_NOTIFY	Quando um *workspace* é criado ou estendido por um NE o DTSA que controla o NE deve informar a criação/extensão ao seu *Master*-DTSA.
WORKSPACE_CONFIGURATION	Ao encontrar um *workspace* o *Master*-DTSA determina uma lista de DTSAs que vão estender o *workspace* solicitado até o DTSA da entidade que requisitou. Em seguida o DTSA deve informar a todos os DTSAs na rota para que configurarem seus NEs de forma a estender o *workspace*.

Todas as primitivas utilizadas para roteamento Inter-Domínio (Inter-DTSA) devem fazer parte do protocolo DTSCP, pois a comunicação (associada a controle) nesse caso é apenas entre vários DTSAs. Os serviços do protocolo ETCP (*Entity Title Control Protocol*) não fazem parte do roteamento em si, uma vez que este protocolo é destinado às comunicações entre entidades e seu DTSA (local). Contudo, o serviço WORKSPACE_ATTACH é o único mencionado uma vez que é ele quem dá início ao roteamento orientado a *workspace*.

4.5 Roteamento Inter-Domínio

O roteamento Inter-Domínio (Inter-DTSA) ocorre quando um *workspace* deve ser estendido através de dois ou mais DTSAs. Pode-se considerar esse roteamento similar ao Inter-AS utilizado na Internet através do protocolo BGP. O desafio desse tipo de roteamento é que um DTSA não conhece a topologia das redes sob controle de outros DTSAs. O autômato para esse roteamento possui dois diagramas, sendo: i) para o roteamento no âmbito de *Workspaces* de Controle Privados, i.e., entre DTSAs; e, ii) para o roteamento no âmbito de *Workspaces* de Controle Públicos, i.e., entre *Masters*-DTSA.

O roteamento se inicia com a chegada da primitiva WORKSPACE_ATTACH requerendo a extensão de um *Workspace* especificado que o DTSA receptor não conheça. O DTSA receptor deverá enviar a seu MDTSA, via *Workspace* de Controle Privado, primitiva requerendo informações sobre o *workspace* especificado. O caminho entre o DTSA receptor e seu MDTSA pode conter um ou mais DTSAs. Se receber respostas com infor-

mações sobre o *Workspace* especificado, primitivas de configurações devem ser trocadas entre os DTSAs.

No MDTSA, o serviço de roteamento deve resgatar informações do *Workspace* especificado. Uma vez que a rota seja escolhida por um DTSA, ele deverá informar aos demais DTSAs no caminho (rota) que novas configurações são necessárias para estender o referido *workspace*.

4.5.1 Algoritmo de Roteamento

O algoritmo de roteamento é invocado em função da chegada das primitivas WORKSPACE_ATTACH ou WORKSPACE_CONFIGURATION. A primeira ocorre quando uma entidade, controlada pelo DTSA receptor, requer se ligar a um determinado *workspace*. A segunda acontece quando um DTSA, selecionado pelo roteamento, é avisado que um *workspace* será estendido e que um ou mais de seus NEs devem ser configurados.

Quando chega a primitiva WORKSPACE_ATTACH, o DTSA busca informações sobre o *workspace* especificado, em seu banco de dados local. Caso o encontre, é executado o roteamento Intra-DTSA. Caso contrário, o procedimento Inter-DTSA é iniciado e é enviada a primitiva WORKSPACE_LOOKUP por meio do *workspace* de controle privado, informando o título do *workspace* especificado.

Considerando que WORKSPACE_LOOKUP é um serviço confirmado, então um temporizador é configurado para definir o *time out* de espera por resposta de seu MDTSA. Caso ocorra o *time out*, o DTSA receptor enviará uma resposta negativa à entidade que requereu o serviço de busca.

Ressalte-se que ao requerer o *lookup*, a ETArch deixa em aberto a especificação dos requisitos Inter-DTSA. Acontece que a arquitetura prevê a especificação dos requisitos quando um *workspace* é criado, mas não apresenta uma definição a respeito do *attachment*. Este aspecto será levado em conta pelo roteamento, pois um *workspace* é criado com determinados requisitos e uma entidade pode querer participar mesmo não atendendo totalmente. Por exemplo, suponha um *workspace* criado com qualidade para vídeo em 1080p, sendo que uma entidade com 720p queira se ligar (*attach*) para receber a *stream*.

Para a primeira versão do roteamento aqui proposto será considerado que as capacidades da entidade devem ser suficientes para que ela participe de um *workspace*. No exemplo dado, a entidade que fará a transmissão de vídeo deverá criar vários *workspaces*, uma para cada *codec* e resolução a ser transmitido.

A melhor rota (que estenda o *workspace* e atenda os requisitos) é escolhida pelo MDTSA. Cada rota contém uma lista de DTSAs e os requisitos que essa rota pode atender. Para uma primeira versão do algoritmo de roteamento, entende-se que todas as rotas atendem aos requisitos do *workspace*.

Após o DTSA receber a resposta do serviço WORKSPACE_LOOKUP, contendo a rota escolhida pelo MDTSA, ele deve configurar os NEs sob seu controle, selecionados

para fazer parte da rota, e enviar a primitiva WORKSPACE_CONFIGURATION.

Como o DTSA conhece sua topologia e possui informações sobre os NEs de borda para a rota escolhida, ele configura o *workspace* desde o NE da entidade solicitante até o NE de borda para o primeiro DTSA da rota escolhida. Note-se que diversas rotas podem ser possíveis na topologia do DTSA, portanto, o procedimento de roteamento Intra-DTSA deve ser chamado para essas configurações (cada DTSA decide como estender o *workspace* através de seus NEs).

A Figura 23 representa um exemplo do *Workspace* $WKS\text{-}APP_1$ criado no $DTSA_3$. A entidade $User$ no $DTSA_1$ solicita se juntar ao $WKS\text{-}APP_1$. Nesse caso, o $DTSA_1$ vai configurar um *workspace* partindo de NE_1 até NE_2, que é a borda pela qual $DTSA_1$ consegue chegar até o $DTSA_3$. Esse *workspace* está representado em vermelho tracejado na figura.

O DTSA também deve enviar a primitiva WORKSPACE_CONFIGURATION para o *workspace* de controle, informando a rota escolhida. A primitiva chega em cada DTSA da rota escolhida e cada DTSA vai configurar seus NEs de forma a estender o *workspace*. Na Figura 23, o $DTSA_1$ envia a primitiva WORKSPACE_CONFIGURATION em direção ao $DTSA_2$ e a primitiva vai caminhar na topologia até $DTSA_3$.

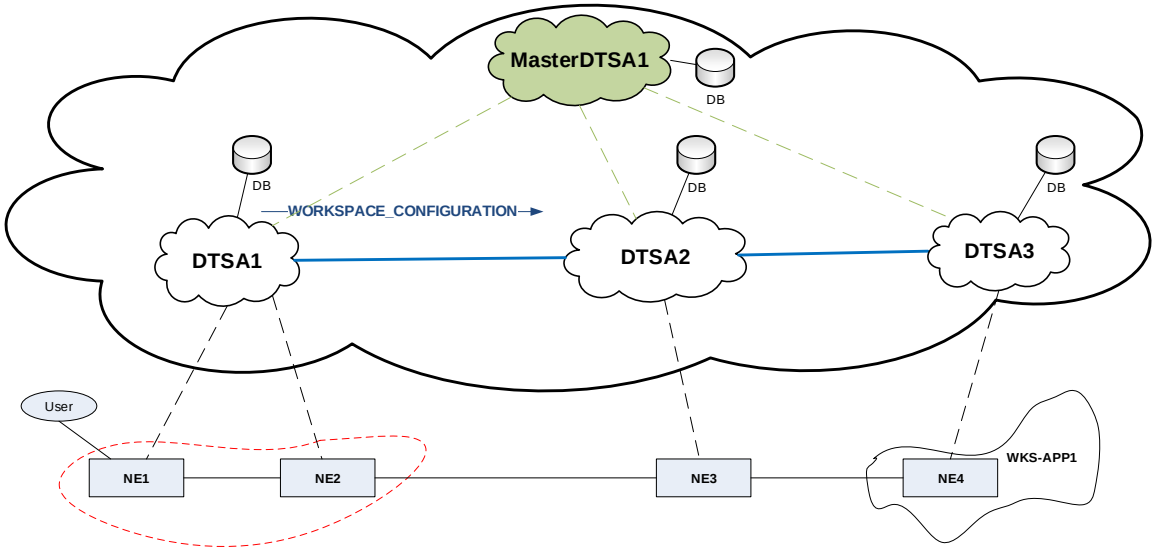

Figura 23 – Roteamento Inter-DTSA

Cada DTSA no caminho da rota deve inicialmente configurar o *workspace* em seu próprio banco de dados local, além de configurar regras nos seus NEs. Pela ETArch, um *workspace* sempre possui uma entidade proprietária, portanto, o DTSA deve armazenar qual é o DTSA proprietário do *workspace* sendo estendido. Isso é necessário, pois um DTSA não possui informações a respeito de entidades controladas por outros, o único título que ele conhece é dos seus DTSAs vizinhos.

Note-se que para um DTSA estender um *workspace*, que não seja de sua propriedade, diversos caminhos são possíveis. Portanto, o DTSA tem autonomia para decidir como vai configurar seus NEs. Ele só precisa considerar que o *workspace* deve ser estendido do NE da borda do DTSA solicitante até outro. Na Figura 23, caso o $DTSA_2$ possuísse mais de um NE, ele deveria escolher a melhor rota entre esses NEs.

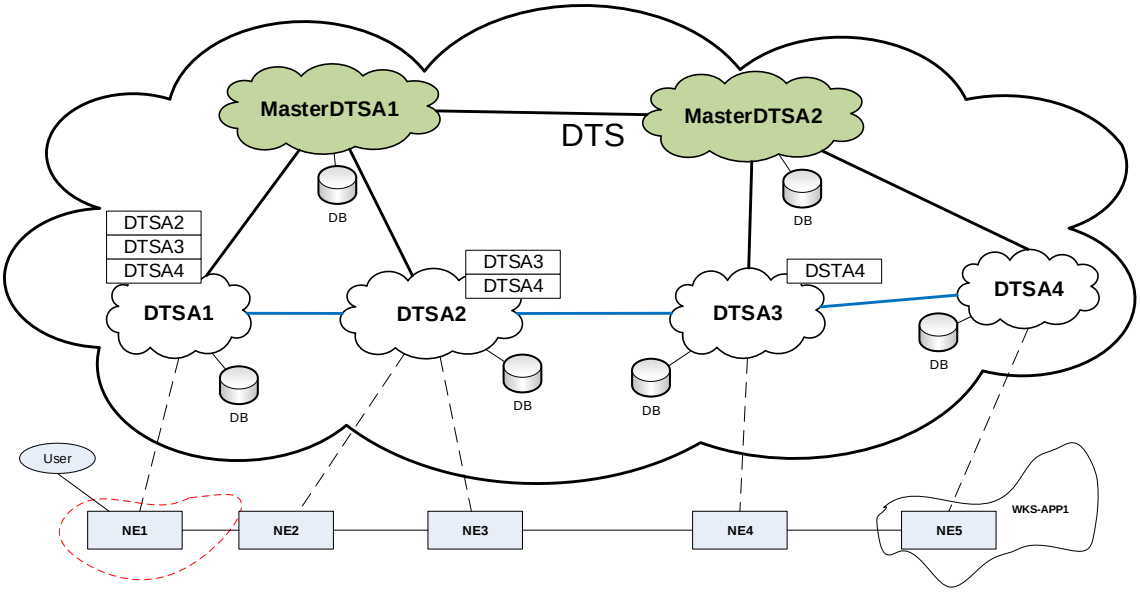

Figura 24 – Lista de DTSAs em uma rota

Observa-se que a primitiva WORKSPACE_CONFIGURATION contém inicialmente todos os DTSAs da rota escolhida. Porém, essa lista diminui em cada DTSA, visto que cada DTSA se retira da rota, pois as configurações para trás já foram executadas. Interessa ao DTSA posterior apenas os DTSAs entre seu vizinho anterior e os demais à frente. A Figura 24 mostra o desempilhamento feito em cada DTSA.

Na Figura 24 observa-se que o *workspace* $WKS\text{-}APP_1$ originalmente do $DTSA_4$ será estendido até $DTSA_1$. A melhor rota definida pelo $DTSA_1$ é representada em azul na figura (na topologia entre DTSAs, interna na nuvem maior do DTS). Dessa forma, $DTSA_1$ envia a primitiva WORKSPACE_CONFIGURATION contendo a lista de DTSAs na rota. Quando a primitiva chega em $DTSA_2$, ele se remove da lista e encaminha para o $DTSA_3$ apenas os próximos DTSAs na rota, e assim sucessivamente.

Ainda na Figura 24, em relação ao $DTSA_3$, observa-se que a rota contém apenas $DTSA_4$, pois é o último DTSA da rota. O $DTSA_4$, uma vez sendo o último da rota, deve configurar seus NEs para que o *workspace* seja estendido desde a entidade que o criou (ou alguma mais próxima que estende o *workspace*) para a borda que vai para NE_4.

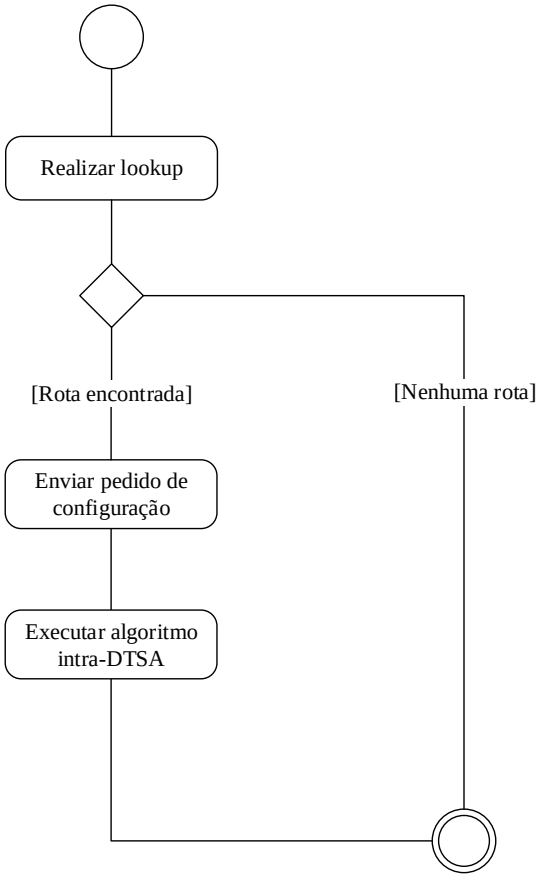

Figura 25 – Algoritmo de roteamento Inter-DTSA - *Lookup*

A Figura 25 apresenta a primeira parte do algoritmo de roteamento Inter-DTSA. Nela se observa que o procedimento se inicia quando é requerido um *workspace* que o DTSA não conhece, então realiza um *lookup* e aguarda a resposta. Caso uma rota seja encontrada para o *workspace*, o DTSA deve enviar o pedido de configuração para aquela rota e logo após executar o algoritmo de roteamento Intra-DTSA para configurar seus próprios NEs.

Enviar um pedido de configuração, como mostra uma das ações da Figura 25, significa enviar a primitiva WORKSPACE_CONFIGURATION contendo a rota (lista de DTSAs) para o primeiro DTSA dessa rota (que é vizinho do DTSA em questão). A última ação da figura institui que o algoritmo Intra-DTSA (Figura 22) deve ser executado.

A Figura 26 apresenta a segunda parte do algoritmo de roteamento Inter-DTSA. Nela se observa que a primeira função é um laço para verificar se existem mais DTSAs (posteriores ao corrente) que devam receber o pedido de configuração. Caso existam, o DTSA corrente deve configurar seus NEs, de modo a estender o *workspace* de um DTSA vizinho (do qual recebeu o pedido de configuração), até um DTSA vizinho (próximo na rota recebida no pedido de configuração) e por fim encaminhar o pedido de configuração ao próximo DTSA na rota.

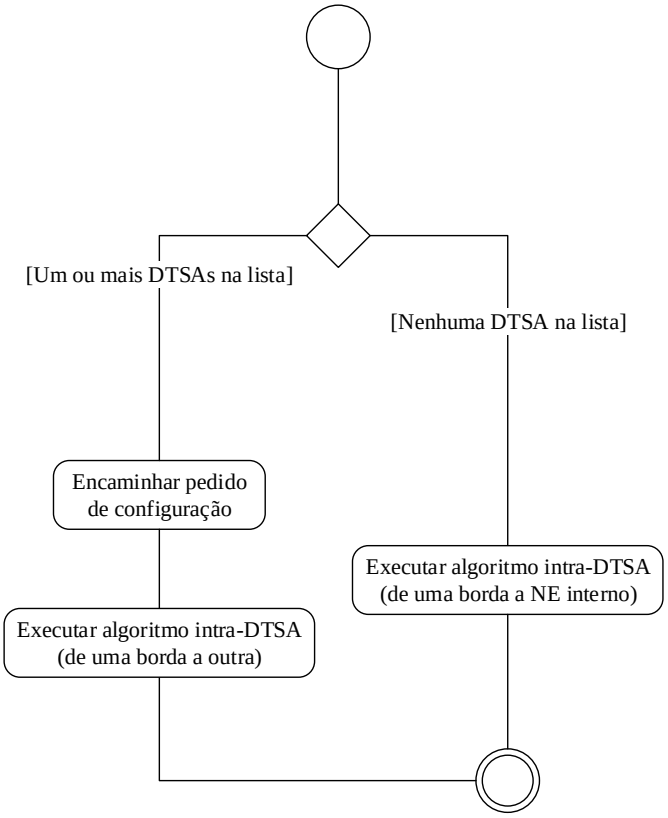

Figura 26 – Algoritmo de roteamento Inter-DTSA - Configuração

Na Figura 26, caso não existam mais DTSAs posteriores na rota, este DTSA (corrente) é quem possui o *workspace* outrora solicitado. Portanto, a única ação desse DTSA é configurar seus NEs de forma a estender o *workspace* para que chegue ao DTSA vizinho que enviou a solicitação de configuração. Nesse momento o algoritmo de roteamento Intra-DTSA deverá ser executado, considerando que a extensão vai do NE mais próximo que possua o *workspace* estendido até a borda (NE de ligação) com o DTSA vizinho.

4.5.2 Autômato de roteamento no DTSA

Inicialmente, um DTSA está no estado em que espera uma primitiva WORKSPACE_ATTACH (do ETCP) ou WORKSPACE_CONFIGURATION (do DTSCP). Na primeira, o DTSA recebe uma requisição indicando que alguma de suas entidades quer se ligar a um *workspace*. Na segunda, o DTSA recebe uma mensagem de configuração de outro DTSA.

A Figura 27 mostra a máquina de estado (FSM – *Finite State Machine*) que roda em um DTSA. Quando um WORKSPACE_ATTACH chega ao DTSA, há duas possibilidades: (i) quando o DTSA possui informações do *workspace* especificado; ou (ii) caso contrário, desconhece o *Workspace* especificado. No caso (i) o DTSA deve configurar seus NEs para estender o *workspace* e finalizar o roteamento. No caso (ii), será requerido o

serviço WORKSPACE_LOOKUP.

Ao requisitar o serviço WORKSPACE_LOOKUP, o DTSA aguarda a resposta do seu MDTSA. O autômato finaliza se ocorrer o *time out*. Se houver resposta, o DTSA a avalia e, em caso positivo (encontrou o *workspace*), configura seus NEs e responde positivamente à entidade que requereu o *attach*.

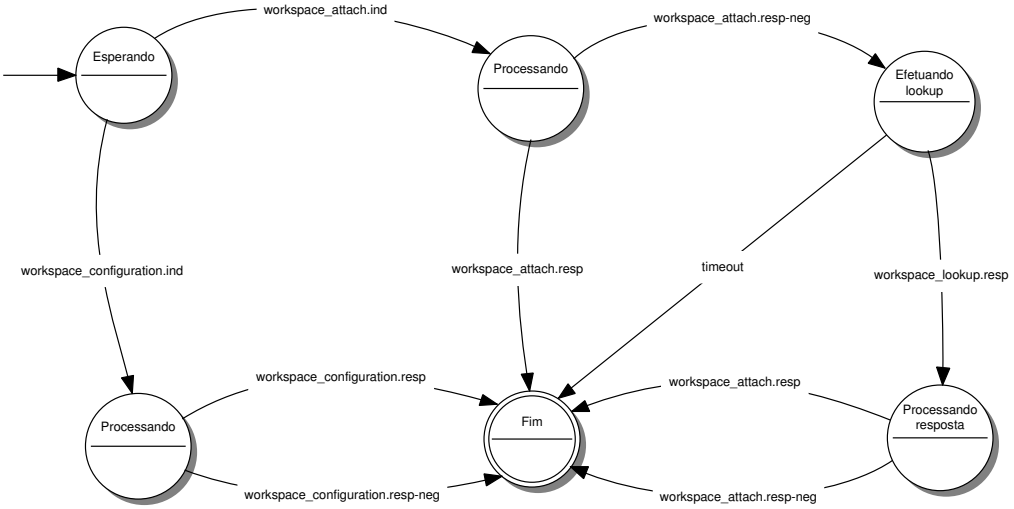

Figura 27 – FSM para o DTSA

Na FSM da Figura 27, observa-se também a execução do autômato, no caso do DTSA receber a primitiva WORKSPACE_CONFIGURATION. Nesse caso, o DTSA procederá de forma a configurar seus NEs, para estender o *workspace* especificado, e, logo após, finalizar de modo positivo (seus NEs aceitaram a configuração) ou negativo (seus NEs não aceitaram a configuração).

4.5.3 Autômato de roteamento no *Master*-DTSA

O *Master*-DTSA também pode receber um pedido de *lookup* ou *attach*, os quais diferem no procedimento executado.

O MDTSA consulta seu banco de dados local, quando recebe um WORKSPACE_LOOKUP, para buscar informações sobre o *workspace* solicitado. Se for encontrado, o próprio MDTSA define as possíveis rotas para estender o *workspace* até o DTSA solicitante. Em caso negativo, o MDTSA envia o *lookup* através do *workspace* de controle público.

Nesse caso, em que o WORKSPACE_LOOKUP foi enviado por meio do *workspace* de controle público, poderão haver várias respostas (se o *workspace* solicitado existir) ou nenhuma resposta em caso negativo. Considerando a temporização descrita para o serviço WORKSPACE_LOOKUP, as respostas recebidas antes do *time out* serão consideradas e aquelas havidas após o *time out* serão descartadas.

Caso haja mais de uma resposta, o MDTSA escolhe a melhor rota recebida e acrescenta a rota interna entre seus próprios DTSAs, para então montar a resposta a ser enviada ao DTSA requisitante (que deu origem à requisição do WORKSPACE_LOOKUP).

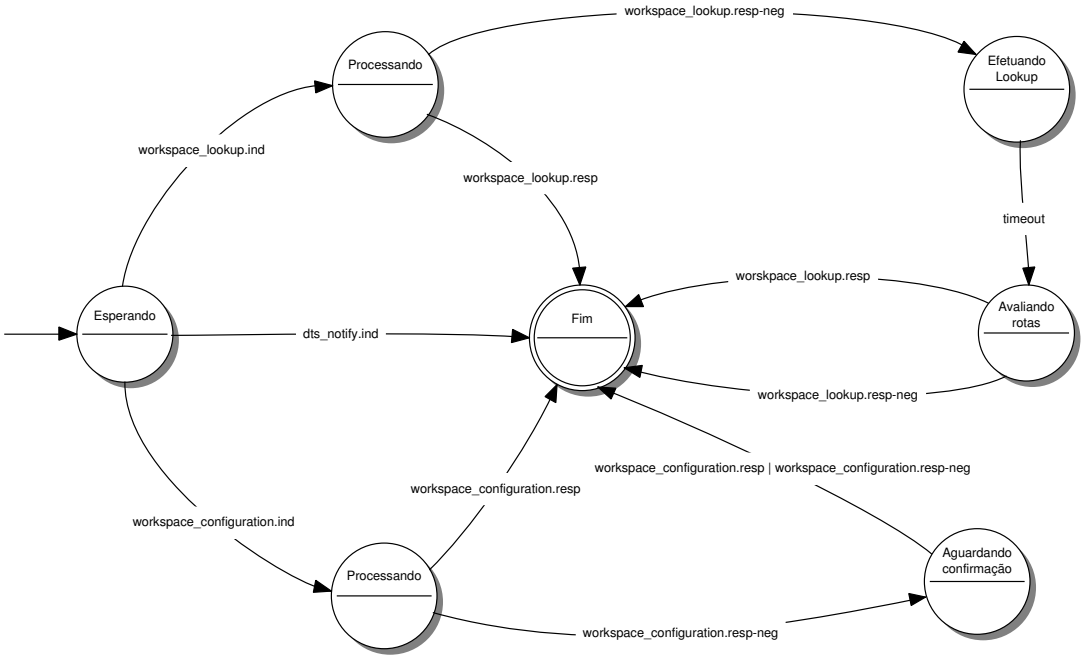

Figura 28 – Autômato para o *Master*-DTSA

A Figura 28 mostra a FSM para o MDTSA. Note-se que o algoritmo inicia com a chegada de uma das três primitivas relacionadas ao roteamento: WORKSPACE_LOOKUP; WORKSPACE_CONFIGURATION; ou DTS_NOTIFY.

No caso do WORKSPACE_LOOKUP, a FSM entra no estado em que processa a requisição e apresenta duas possibilidades: 1) ela conhece as informações do *workspace* especificado e tem, portanto, as informações para responder ao requisitante, finalizando o autômato; e 2) o *workspace* especificado não se encontra em seu banco local e, então, é enviada uma requisição do WORKSPACE_LOOKUP e a FSM vai para o estado em que recebe rotas de outros MDTSA, i.e., o MDTSA receptor vai se comportar como o corrente neste mesmo estado, e só sai desse estado quando ocorre o *time out*.

Após ocorrer o *time out*, a FSM analisa as respostas e determina, considerando os requisitos, a melhor rota. Nessa avaliação, ela também determina o caminho entre o *Master* e o DTSA requisitante.

No caso do DTS_NOTIFY, ocorre apenas a atulização do banco de dados local do MDTSA. Esse serviço é requisitado por um DTSA quando um *workspace* é criado por uma de suas entidades ou quando uma entidade estende esse *workspace* a partir de outro MDTSA. Após a atuliazação do banco, o MDTSA envia uma resposta ao requisitante.

No caso do WORKSPACE_CONFIGURATION, o MDTSA vai para o estado Processando, no qual haverá configurações na rede. Se as configurações se restringirem ao âmbito do MDTSA corrente, então, após as configurações, é enviada uma resposta ao requisitante. Se houver necessidade de configurações em outros MDTSAs, então será enviada uma requisição de WORKSPACE_CONFIGURATION àqueles. Após as configurações, a resposta ao requisitante indicará que o serviço foi encerrado.

4.6 Roteamento em Larga Escala

O roteamento orientado a *workspace* pode ser expandido em larga escala, podendo chegar ao nível mundial, por meio da interconexão de domínios diferentes, numa escala global. Nesse caso, as comunicações das primitivas de controle de roteamento serão trocadas entre MDTSAs por meio de *Workspaces* de Controle Públicos.

Um MDTSA pode receber quaisquer das três primitivas de controle de roteamento (WORKSPACE_LOOKUP, WORKSPACE_CONFIGURATION e DTS_NOTIFY). Quando recebe o serviço de notificação, o MDTSA atualiza seu *storage* com informações sobre o novo *workspace* criado no seu domínio. Os demais serviços podem envolver comunicações com outros MDTSAs e foram abordados na Subseção 4.5.3.

WORKSPACE_CONFIGURATION merece uma completação em relação ao mencionado. Ao receber um *configuration*, o MDTSA deve configurar os NEs controlados diretamente por ele (de modo a estender corretamente o *workspace*) e, caso não seja o último da rota, deverá enviar a primitiva WORKSPACE_CONFIGURATION para o próximo agente (DTSA/MDTSA) da rota. Isso é necessário pois pode ser que a configuração requisitada estenda um *workspace* entre diferentes MDTSAs.

A Figura 20, além a organização do DTS, apresenta também a estrutura de suporte ao roteamento em larga escala. No Nível Local, no lado esquerdo da Figura 20, o MDTSA (M_{0101}) possui um domínio com três DTSAs (D_1, D_2 e D_3) e é representado o *Workspace* de Controle Privado na cor azul. As primitivas de roteamento são transmitidas por meio dess plano de controle. Portanto, serviços de *lookup* e *notify* são recebidos, através desse *Workspace* por M_{0101}, e serviços de *configuration* são trocados pelos DTSAs também pelo mesmo meio.

Sempre que um *Workspace* é criado, as primitivas de controle levam suas informações, que 'sobem na' organização (entre níveis), até chegar ao Nível Raiz. Portanto, o controle de uma comunicação entre entidades distribuídas globalmente podem passar pelo nível mais alto do DTS. Isso é análogo ao que acontece na Internet, o serviço de DNS também passa frequentemente pelo nível mais alto (*root*), quando um nome é resolvido pela primeira vez.

Caso uma entidade requeira se ligar a um *Workspace* existente, i.e., uma entidade requer o estendimento de um *workspace*, o que é feito através da requisição do serviço

WORKSPACE_ATTACH, se o DTSA tem as informações, então, as solicitações para *attach* nesse *Workspace* serão resolvidas nesse nível, uma vez que na extensão do *Workspace*, o MDTSA do domínio é avisado e armazena as informações.

Por exemplo, na Figura 20, no nível local uma entidade do DTSA D_3 solicita *attachment* a um *Workspace* que esteja no Nível Raiz no MDTSA M_1. Nesse caso, os DTSAs da rota escolhida serão configurados e o MDTSA M_{0101} armazenará as informações do *Workspace* especificado. Qualquer outra entidade do MDTSA M_{0101} que solicitar o mesmo *Workspace*, será atendida sem que o *Workspace* de Controle Público precise novamente ser utilizado.

Observa-se que o roteamento em larga escala trata de um arranjo de DTSAs e MDTSAs dispostos nos moldes da organização proposta pela arquitetura ETArch e representado na Figura 20, formando uma estrutura de dados do tipo Árvore B. Nela a complexidade de tempo para buscar ou atualizar determinada informação em um nó é logarítmica, justificável para uso neste trabalho. No Nível Raiz, entende-se que a busca por um *Workspace* pode ser feita de forma paralela, onde cada MDTSA espera o *time out* para resposta do *lookup* efetuado para os demais.

Embora escalabilidade seja uma questão importante, mas, assim como ocorreu com a Internet em seus primórdios, não faz parte do escopo deste trabalho dirimir questões relativas a essa característica no plano de controle do DTS.

Capítulo 5

Experimentos e Análise dos Resultados

O presente capítulo descreve os cenários de experimentos realizados com o roteamento orientado a *workspace* proposto neste trabalho e apresenta os resultados obtidos. Como apresentado na Seção 4.2, atualmente a arquitetura ETArch não possui estratégias de roteamento claras relacionadas à definição de rotas para uma rede de determinado domínio, tampouco em domínios diferentes. Assim sendo, implementou-se algoritmos de roteamento para uma prova de conceito das estratégias abordadas a partir da Seção 4.3.

Uma vez que o roteamento orientado a *workspace* abre mão do endereçamento dos *hosts* de origem e destino (para localização), aplicando o endereçamento horizontal, os experimentos demonstram a nova abordagem. Portanto, os objetivos deste capítulo são: demonstrar a utilização de *Workspaces* de Controle para roteamento; e verificar a viabilidade de definição da melhor rota antes que os dados comecem a ser encaminhados através dos *Workspaces* de Dados.

Como definido no Capítulo 4, o DTSA é responsável por iniciar a rotina de roteamento, sendo primeiramente possível rotear entre DTSAs (utilizando o MDTSA) e em seguida entre os NEs de um DTSA. Toda primitiva para controle da rede é destinada de fato ao *Workspace* de Controle, sendo que todo DTSA que dele faça parte a receberá.

A implementação do roteamento propriamente dito foi realizada levando em consideração a definição prévia da rota. Os serviços de roteamento são aplicados sempre que uma entidade deseja se juntar a determinado *Workspace*, tanto para estender o *Workspace* solicitado entre NEs ou DTSAs, quanto para buscá-lo em um domínio diferente (se necessário).

Inicialmente será descrito o método utilizado para validar a hipótese apresentada no Capítulo 4. Na Seção 5.2, serão descritos os experimentos realizados, detalhando minuciosamente o ambiente utilizado, tipos de máquinas, topologias, simuladores etc. A Seção 5.3 apresentará uma breve avaliação sobre os resultados obtidos nos experimentos, mostrando os ganhos, possibilidades e limitações desta proposição.

5.1 Método para a Avaliação

Para avaliar o roteamento orientado a *workspace* aplicou-se os *Workspaces* de Controle em determinadas topologias para coletar resultados. Cada topologia contém um grafo com caminhos diferentes. Ressalta-se que as topologias podem ser do DTSA (grafo de NEs) ou do MDTSA (grafo de DTSAs). O desenvolvimento realizado para a prova de conceito levou em consideração o requisito de menor caminho.

Os algoritmos desenvolvidos calculam o menor caminho entre NEs ou DTSAs considerando o número de nós que serão configurados para que um *Workspace* de Dados seja estendido à entidade solicitante. Na topologia de um DTSA, esse número de nós é a quantidade mínima de NEs necessária para extensão. Na topologia do MDTSA, o número de nós é a quantidade de DTSAs que serão envolvidos para extensão do *Workspace*.

A avaliação dos experimentos foi realizada através da verificação do caminho escolhido em cada extensão de *Workspaces* ocorrida em cada teste. A verificação do caminho escolhido foi realizada coletando-se os fluxos inseridos nos NEs, os quais mostram as regras de encaminhamento que cada DTSA definiu para cada *Workspace*. Na prática uma regra de encaminhamento é o resultado do roteamento realizado pelos DTSAs.

Foi utilizado um método para avaliação de tempo nos experimentos. A análise de tempo foi feita através da coleta nos DTSAs, desde o momento que cada primitiva de controle chega no DTSA, vinda do *Workspace* de Controle. Com essa análise, foi possível avaliar as rotinas de roteamento, além dos próprios conceitos da ETArch utilizados, mostrando determinada vantagem da utilização de *Workspaces*.

O método propõe avaliar um requisito aplicando-o intra e inter-DTSA. O menor caminho foi escolhido como requisito por ser o objetivo principal de roteamento em arquiteturas convencionais. A utilização de outros requisitos de roteamento, como por exemplo economia de energia, é objeto de trabalhos futuros. Por este motivo, resolveu-se que o foco da avaliação deste capítulo está na demonstração de roteamento através da ETArch, inclusive considerando diferentes DTSAs (foco maior por ser uma arquitetura que pretende atuar na Internet).

O método de avaliação também considera a quantidade de DTSAs que podem ser envolvidos em uma comunicação. Deve-se demonstrar que cada DTSA resolva as rotas para extensão de um *Workspace*, independentemente da forma que outro DTSA resolverá. Ainda, um MDTSA deve ser capaz de determinar a rota (caminho entre DTSAs) para diferentes domínios.

Para melhor entendimento dos experimentos, são apresentadas imagens das topologias envolvidas em cada experimento, além de outras imagens mostrando a rota definida para extensão dos *workspaces* solicitados pelas entidades. Para desempenho escolheu-se tabelas para mostrar o tempo gasto para extensão de *Workspace* após solicitado por uma entidade.

5.2 Descrição dos Experimentos

Os experimentos realizados pretendem demonstrar o funcionamento de rotinas do roteamento orientado a *workspace*, focando nas primitivas de controle. A ideia é validar o uso dos *Workspaces* de controle para fins de definição de rota. Serão detalhados os cenários (topologia e ambiente computacional utilizado) e posteriormente as rotas escolhidas pelos algoritmos de roteamento.

Para todos os experimentos realizados foram utilizados *switches* Openflow. Por esse motivo as figuras deste capítulo mostram os elementos de rede (NEs) com a nomenclatura de prefixo 's'.

5.2.1 Experimentos intra-DTSA

Primeiramente justifica-se uma demonstração do algoritmo de roteamento intra-DTSA. Esse procedimento é utilizado para topologias conhecidas por um DTSA, além de ser utilizado no algoritmo inter-DTSA (quando um DTSA recebe uma primitiva para configurar determinada extensão de *Workspace*). Este experimento é ainda justificável, pois com a utilização da ETArch, DTSAs podem ser configurados para controlar desde redes domésticas até grandes topologias corporativas.

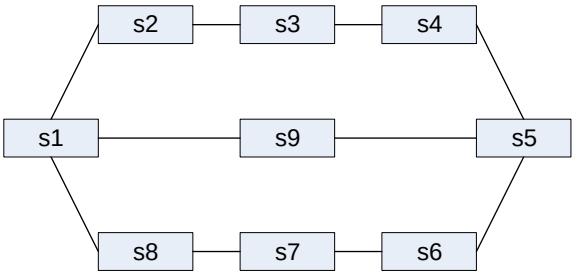

Figura 29 – Topologia do experimento intra-DTSA

A Figura 29 apresenta a topologia de rede utilizada no experimento. Nota-se que o grafo provê diferentes possibilidades de rotas para extensão de *Workspaces* entre entidades de diferentes NEs. Um exemplo é que um *Workspace* criado em uma entidade de s_1 pode ser estendido para s_5 por três caminhos diferentes. Todos os NEs da figura são controlados pelo mesmo DTSA.

Os NEs do experimento foram criados utilizando-se o simulador Mininet, com a topologia sendo customizada para se adequar à Figura 29 (Apêndice A.1). O Mininet foi utilizado em uma máquina virtualizada com processador Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz, 1 GB de memória RAM e adaptador de rede em modo *Bridged*. Utilizou-se sistema operacional Ubuntu 14.04 LTS (Linux Kernel 3.13.0-24). Para o Mininet utilizou-se versão 1.0.0 do software.

O DTSA do experimento foi desenvolvido em linguagem Java utilizando-se a abordagem JAIN SLEE (FERRY, 2014) disponível através da plataforma de comunicação Mobicents (MOBICENTS, 2014). Os códigos de roteamento foram desenvolvidos e implantados no DTSA do projeto EDOBRA. Este DTSA já possuía códigos para controle das entidades (registro, pedidos de *attach* etc). O desenvolvimento focou nas rotinas de roteamento (extensão de *Workspaces* intra e inter-DTSA).

A máquina rodando o Mobicents foi virtualizada utilizando-se processador Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz, 768 MB de memória RAM e adaptador de rede em modo *Bridged*. Utilizou-se sistema operacional Fedora release 20 (Linux Kernel 3.11.10-301). Os códigos do DTSA e Mobicents foram implantados no servidor de aplicação JBoss versão 5.1.0.GA. Ressalta-se que a JVM (*Java Virtual Machine*) do servidor de aplicação foi configurada para alocar 512 MB de memória RAM.

Os experimentos realizados envolvem a criação de um *Workspace* e sua extensão para três entidades diferentes. A entidade criadora do *Workspace* e as três demais foram registradas em NEs diferentes. Inicialmente uma entidade de título $Entity_1$ criou o Workspace no NE s_1 da Figura 29. O resultado foi a configuração apresentada na cor azul claro da Figura 30, na qual o *Workspace* foi configurado entre $Entity_1$ e seu NE.

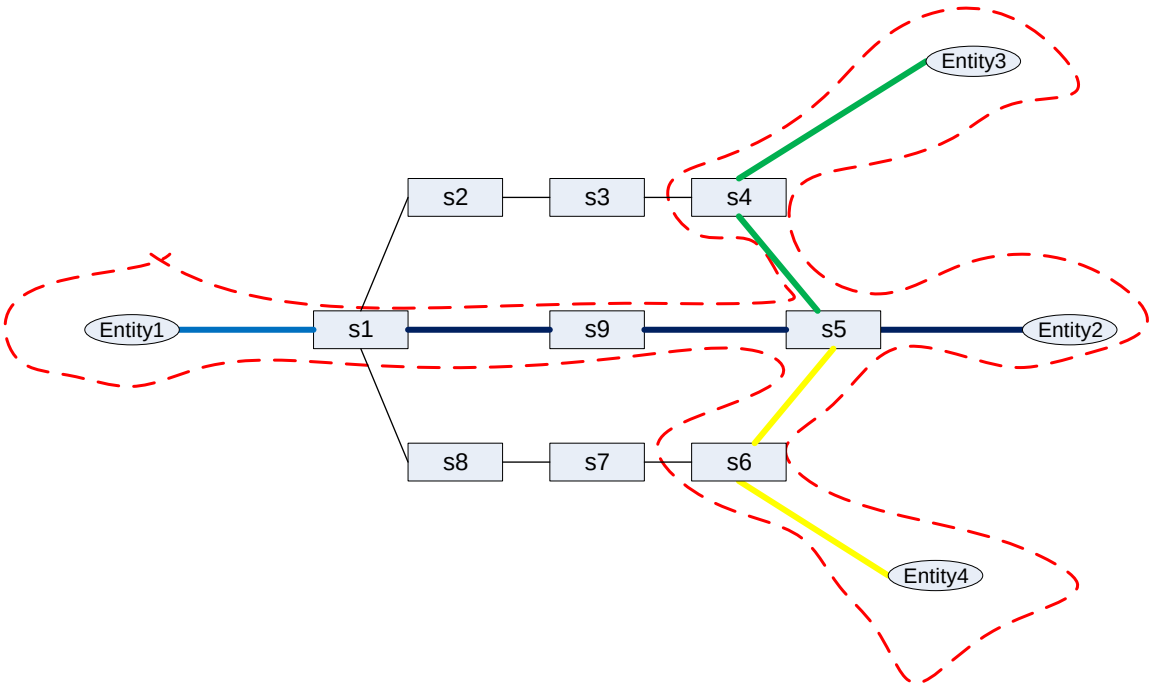

Figura 30 – Rotas escolhidas no experimento intra-DTSA

Após a criação do *Workspace*, submeteu-se o registro da entidade $Entity_2$ no NE s_5 da topologia e essa entidade solicitou se juntar ao *Worksapce* criado por $Entity_1$. O resultado é mostrado na Figura 30 pela cor azul escuro: a rota escolhida (pelo DTSA) para extensão

do *Workspace* de s_1 até s_5 foi passando por s_9. Esse resultado prova que o DTSA optou pela melhor rota, considerando que o requisito dos experimentos deste capítulo é a escolha da rota com menos saltos para extensão de um *Workspace*.

O experimento prosseguiu com o registro das entidades $Entity_3$ e $Entity_4$, nos NEs s_4 e s_6 respectivamente. A Figura 30 mostra o resultado nas cores verde e amarelo. De fato prova-se que o DTSA escolheu a melhor rota, estendendo o *Workspace* para s_4 e s_6 através de s_5.

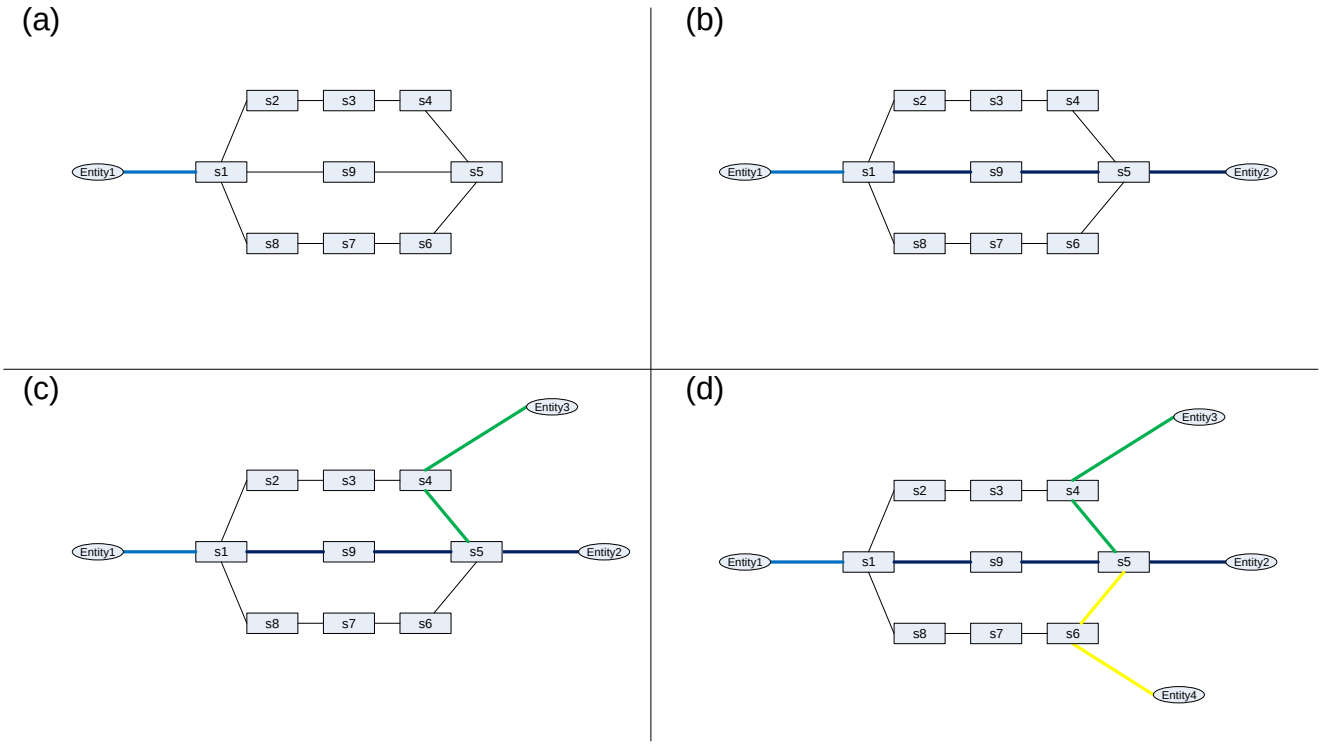

Figura 31 – Etapas do experimento intra-DTSA

A Figura 31 apresenta de forma consolidada a sequência do experimento (Apêndice B.1). Na Figura 31(a) é apresentado o registo e criação do *Workspace* por $Entity_1$. Em (b) apresenta-se o registro e *attach* no *Workspace* feito pela $Entity_2$. Em (c) a $Entity_3$ se registra e estende o *Workspace*. Por fim, a Figura 31(d) mostra as quatro entidades registradas e o *Workspace* estendido ao longo da topologia. As arestas entre NEs na cor preta não foram utilizadas pelo DTSA.

O segundo experimento realizado utiliza topologia de duzentos NEs. Esses NEs foram dispostos de forma linear, onde o NE s_1 é ligado ao s_2, s_2 ao s_3 e assim por diante. O objetivo do experimento é demonstrar o desempenho do roteamento em extensões de *Workspaces* por entidades registradas em NEs distintos. Uma entidade registrada em s_1 criou um *Workspace* e duas entidades na ponta da topologia solicitam *attach*. A Tabela 3 mostra o tempo médio gasto entre registro e *attach* das duas entidades.

Tabela 3 – Tempos do experimento intra-DTSA topologia linear.

Entidade	**NE da entidade**	**Tempo gasto (ms)**
$h_1 s_{199}$	s_{199}	33.2935
$h_1 s_{200}$	s_{200}	2.9783

Os números da Tabela 3 mostram que a entidade registrada na entidade $h_1 s_{199}$ gasta mais tempo para ter a confirmação de *attach* no *Workspace*. Isso demonstra que o tempo gasto para o DTSA configurar os 199 NEs entre s_1 e s_{199} é maior que o tempo para configurar entre s_{199} e s_{200}.

Contudo, considerando que dificilmente haverá esta quantidade de *switches* em *workspaces*, este tempo de espera de configuração da rede é plenamente aceitável se considerarmos os tempos da atual Internet. Além disso, este tempo somente será dispendido na configuração do *workspace*, sendo depois apenas o tempo de chaveamento (*switching*), que é significativamente menor que o tempo de roteamento.

Uma vez que o *Workspace* está passando pelo NE s_{199}, para a última extensão, o NE s_{200} estende de s_{199}, não precisando buscar na origem do *Workspace* (entidade registrada em s_1). Ressalte-se que o 'Tempo gasto' na Tabela 3 é o tempo entre registro da entidade e confirmação de *attach* no *Workspace*.

5.2.2 Experimentos inter-DTSA

É justificável uma demonstração do algoritmo de roteamento inter-DTSA visto que a ETArch não possuía esta capacidade. Além disso, pretende-se utilizar o roteamento orientado a *workspace* para a rede em escala global. Os diferentes domínios da rede referentes às topologias de universidades, corporações, ISPs etc exigem uso de roteamento inter-DTSA para extensão de *Workspaces* presentes nesses domínios.

A implementação para demonstração nesta seção utiliza as rotinas implementadas na Subseção 5.2.1 quando um DTSA recebe a incumbência de configurar uma extensão de *Workspace*. Além disso, implementou-se algoritmos inter-DTSA considerando como requisito o menor caminho. Para cálculo do menor caminho em uma topologia entre DTSAs optou-se pelo número de DTSAs pelos quais passará a extensão de determinado *Workspace*.

A topologia de rede do primeiro experimento é apresentada na Figura 32 (Apêndice A.2). Cada octógono representa um DTSA diferente, sendo M_1 um MDTSA. Em azul tracejado há o *Workspace* de Controle Privado pré-configurado. Retângulos representam os NEs controlados pelos diferentes DTSAs e os links entre NEs estão apresentados nas linhas pretas contínuas. Tracejado na cor preta estão as arestas indicando qual DTSA controla seu(s) respectivo(s) NE(s). Note-se ainda que diferentes rotas são possíveis para extensão de *Workspaces* criados em DTSAs diferentes.

Para o experimento, a criação de NEs foi feita na mesma configuração do Mininet da

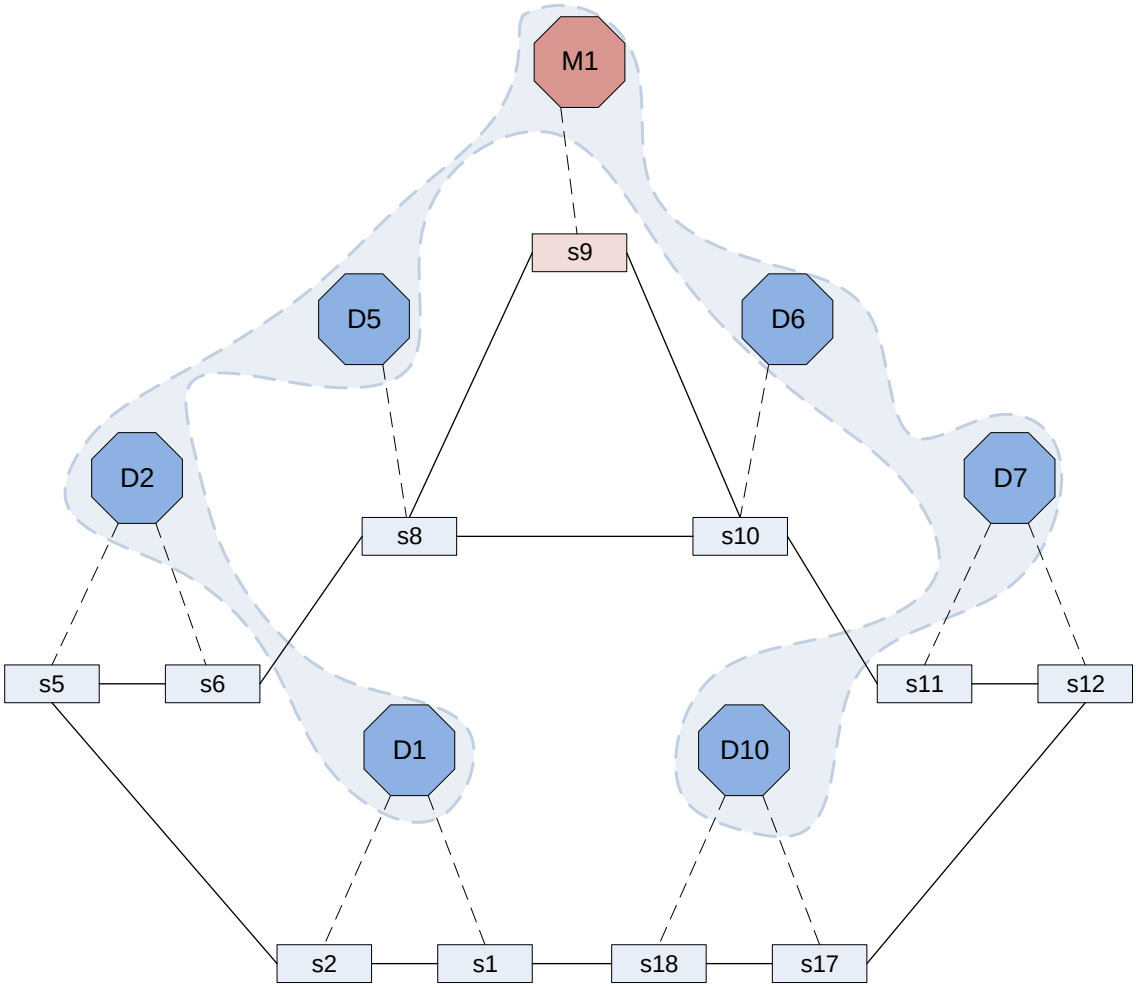

Figura 32 – Topologia do primeiro experimento inter-DTSA

Subseção 5.2.1, alterando-se a topologia. Nessa topologia indicou-se os diferentes DTSAs que controlariam cada NE. Para a criação dos DTSAs utilizou-se a mesma configuração do Mobicents da Subseção 5.2.2, com a diferença que ao invés de uma máquina foram utilizadas sete, uma para cada DTSA.

A intenção do experimento é demonstrar que uma entidade cria um *Workspace* em determinado DTSA e que outra entidade (pertencente a outro DTSA) solicite *attach* nesse *Workspace*. Com isso, as rotinas de roteamento devem ser executadas no MDTSA M_1, visto que uma primitiva de *lookup* chegará até M_1. Após definir a rota, as primitivas para configuração serão enviadas pelo *Workspace* de Controle Privado e cada DTSA que participa da rota escolhida configurará seus NEs.

O experimento foi realizado registrando-se uma entidade (conectada ao switch s_1) no DTSA D_1. A essa entidade deu-se o título h_1s_1. Um *Workspace* foi então criado pela entidade h_1s_1 e, posteriormente, uma outra entidade h_1s_{18}, conectada ao switch s_{18} (registrada no DTSA D_{10}), solicitou fazer parte do *Workspace*. Como resultado do

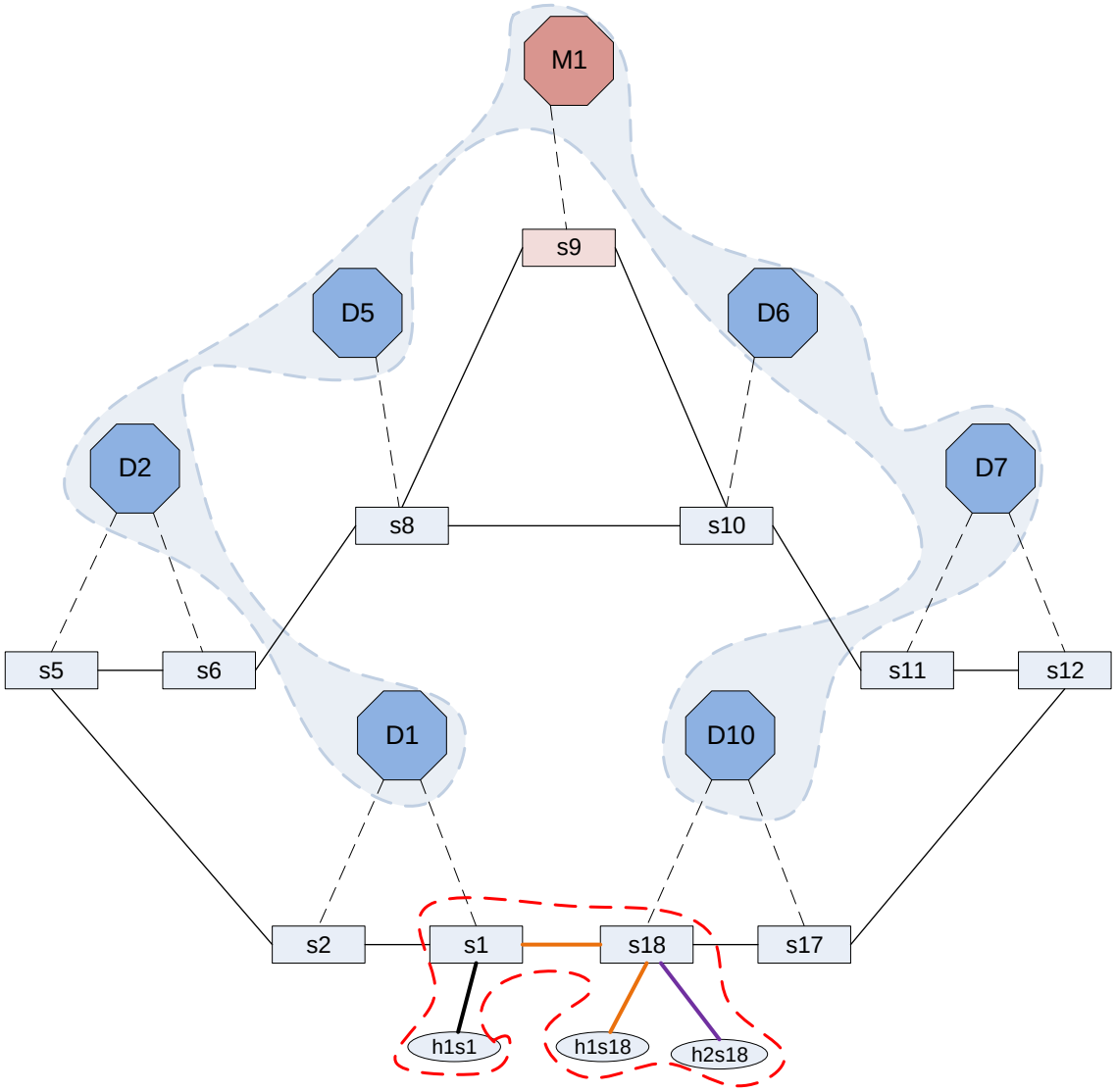

Figura 33 – Rotas escolhidas no primeiro experimento inter-DTSA

experimento, o *Master* DTSA M_1 optou por uma rota entre os DTSAs D_1 e D_{10}. De fato a menor rota para extensão entre as entidades h_1s_1 e h_1s_{18}.

Ainda no mesmo experimento, optou-se por registar outra entidade (h_2s_{18}) conectada em s_{18}. Essa entidade também solicitou *attach* no mesmo *Workspace* criado por h_1s_1. Como resultado D_{10} não solicitou *lookup*, apenas estendeu o *Workspace* (já em seu domínio) para h_2s_{18}.

A Figura 33 mostra a configuração resultante do experimento (Apêndice B.2). As arestas na cor laranja mostram o primeiro *attach* (da entidade h_1s_{18}). A aresta na cor roxa mostra o segundo attach (da entidade h_2s_{18}). Nota-se que o MDTSA optou pela rota D_1-D_{10}. Outras rotas possíveis (porém maiores) seriam: D_1-D_2-D_5-D_6-D_7-D_{10} ou D_1-D_2-D_5-M_1-D_6-D_7-D_{10}. A linha vermelha tracejada mostra os NEs e entidades envolvidos no

Workspace ao final do experimento.

Foi realizado um segundo experimento para demonstração do algoritmo de roteamento orientado a *workspace* atuando inter-DTSA. Utilizou-se os mesmos DTSAs do primeiro experimento, porém com uma diferença na topologia. No primeiro experimento, o algoritmo optou por uma extensão de *Workspace* através de dois DTSAs, sendo a menor rota de fato. Porém pretende-se com o segundo experimento demonstrar uma configuração de extensão através de mais DTSAs.

A Figura 34 apresenta a topologia utilizada no segundo experimento (Apêndice A.3). Observa-se que a única diferença para a topologia do primeiro experimento é que não existe aresta entre s_1 e s_{18}, portanto D_1 e D_{10} não se conhecem. Essa topologia foi escolhida para observar o algoritmo determinar uma rota passando por mais de dois DTSAs. As máquinas utilizadas para os DTSAs e NEs seguem o mesmo padrão utilizado no primeiro experimento, a única diferença foi a criação da topologia no Mininet sem o link s_1-s_{18}.

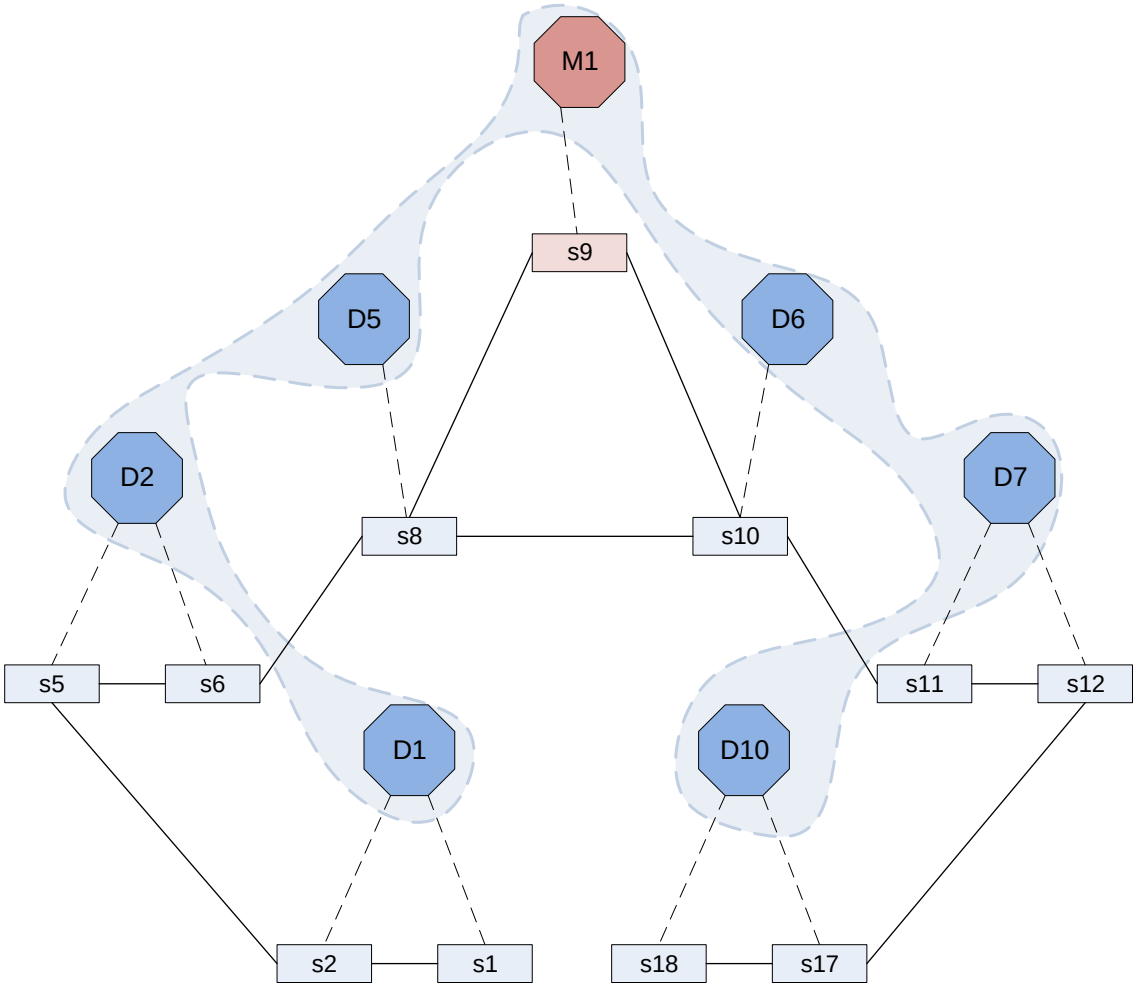

Figura 34 – Topologia do segundo experimento inter-DTSA

Inicialmente uma entidade (de título h_1s_1) registrou-se em D_1 e criou um *Workspace*.

Posteriormente uma entidade (de título h_1s_{18}) registrou-se em D_{10} pelo s_{18} e solicitou o *Workspace* criado por h_1s_1. Como D_{10} não possuía informações a respeito do referido *workspace*, uma primitiva para *lookup* foi enviada ao *Workspace* de Controle Privado. Como resultado a resposta de M_1 foi a rota D_1-D_2-D_5-D_6-D_7-D_{10}, de fato a menor entre os D_1 e D_{10}.

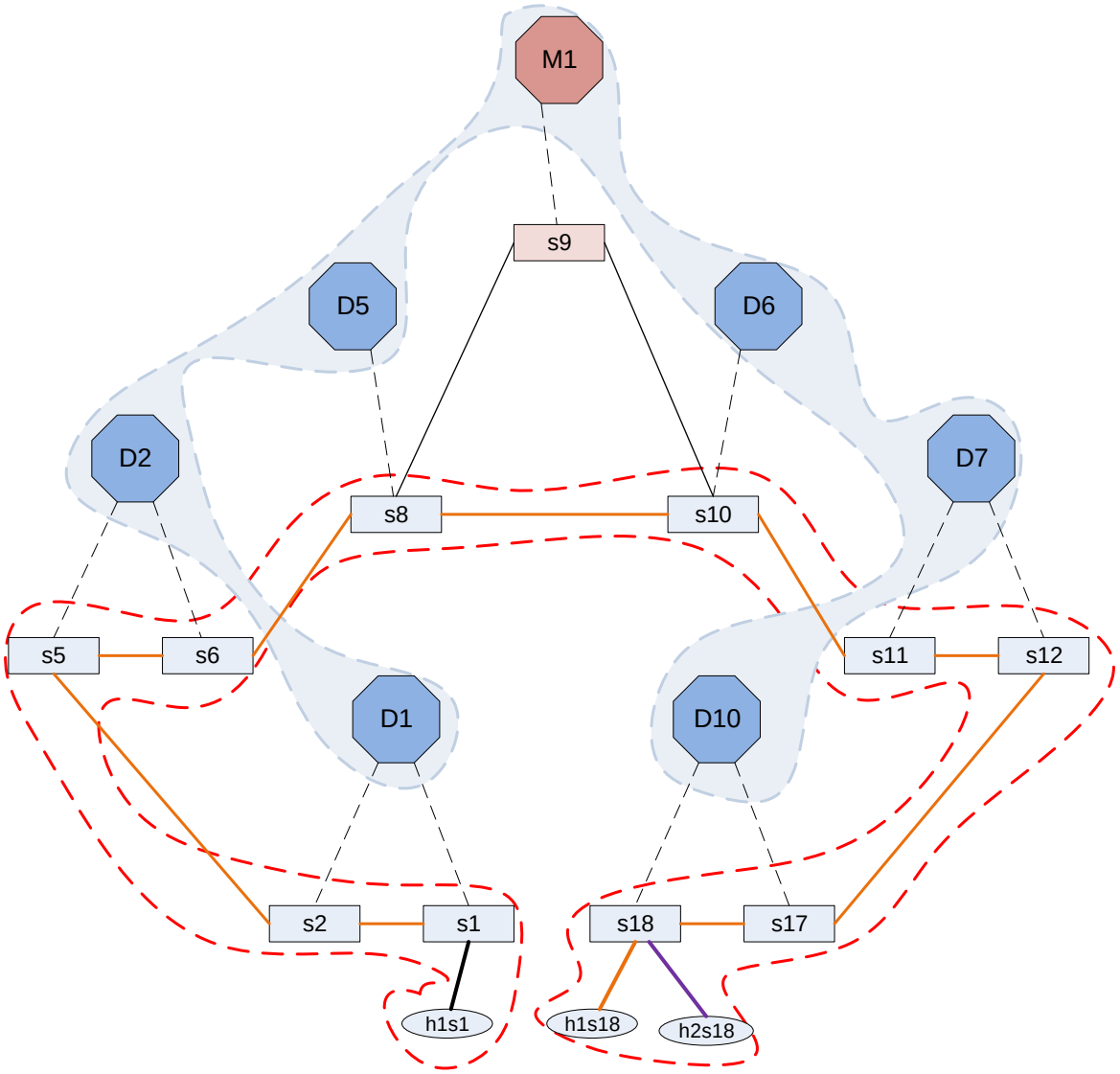

Figura 35 – Rotas escolhidas no segundo experimento inter-DTSA

Após a definição da rota, primitivas de configuração são enviadas ao *Workspace* de Controle Privado e a Figura 35 mostra, na cor laranja, as arestas envolvidas na extensão do *Workspace* de h_1s_1 até h_1s_{18}. Nota-se que uma rota através de M_1 seria possível, mas não foi utilizada.

Logo após a extensão até h_1s_{18}, uma entidade (de título h_2s_{18}) registrou-se em D_{10} e também solicitou o mesmo *Workspace* criado por h_1s_1. O resultado foi que D_{10} apenas

o estendeu para h_2s_{18}, como mostrado na aresta de cor roxa da Figura 35. Nesse passo, nenhuma primitiva foi lançada no *Workspace* de Controle privado, uma vez que D_{10} possuía informações sobre o *Workspace* (pois já o estendera).

A Figura 35 também mostra, em vermelho tracejado, o resultado (Apêndice B.3) da extensão total do *Workspace* ao princípio criado em h_1s_1. Nota-se que s_9 não teve nenhuma configuração em relação ao *Workspace*, uma vez que as rotas definidas para estender até h_1s_{18} e h_2s_{18} não continham M_1.

Tabela 4 – Tempo para extensão de *Workspace* inter-DTSA.

Entidade	**Tempo gasto (ms)**
h_1s_{18}	864
h_2s_{18}	81

O desempenho total para *attach* das entidades h_1s_{18} e h_2s_{18} é apresentado na Tabela 4. Trata-se do tempo gasto por D_{10} para obter informações sobre o *Workspace* e enviar as primitivas de configuração para seus *Workspaces*.

Além dos experimentos relatados acima, para mostrar a definição de rotas dos algoritmos, foram realizados experimentos com vários *Workspaces* sendo criados e diversas entidades realizando *attach*. O DTSA controlador de cada entidade varia em cada cenário. Tais experimentos utilizaram a topologia da Figura 32. Foram considerados quatro cenários diferentes e o resultado da definição das rotas é apresentado na Figura 36.

No primeiro cenário várias entidades foram registradas em D_5 e D_6 e, posteriormente, cada uma criou um *Workspace*. Novas entidades foram registradas em D_5 e solicitaram *attach* nos diversos *Workspaces* criados pelas entidades de D_6. Além disso, novas entidades foram registradas em D_6 e solicitaram *attach* nos diversos *Workspaces* criados pelas entidades de D_5. A Figura 36 apresenta na cor verde a extensão final de todos os *Workspaces*.

No segundo cenário diversas entidades foram registradas em D_6 e criaram diferentes *Workspaces*. Posteriormente outras entidades foram registradas em D_{10} e solicitaram *attach* nos *Workspaces* previamente criados pelas entidades de D_6. Os *switches* envolvidos para extensão dos *Workspaces* (conforme determinado pelos algoritmos de roteamento) são mostrados na Figura 36 pela cor vermelha.

No terceiro cenário diversas entidades foram registradas em D_5 e criaram diferentes *Workspaces*. Outras entidades foram registradas em D_6 e mais entidades registraram-se em D_{10}. As entidades controladas por D_6 e D_{10} solicitaram *attach* nos *Workspaces* criados pelas entidades controladas por D_5. O resultado das extensões é apresentado na Figura 36 pela cor roxa. Observa-se que os *Workspaces* foram estendidos de D_5 até D_6, portanto o algoritmo verificou que a extensão D_6-D_7-D_{10} era menor que D_5-D_2-D_1-D_{10} para estender os *Workspaces* até as entidades de D_{10}.

No quarto cenário uma entidade registrada em D_5 criou um *Workspace*. Logo após várias entidades foram registradas em D_{10} e solicitaram *attach* no Workspace criado em

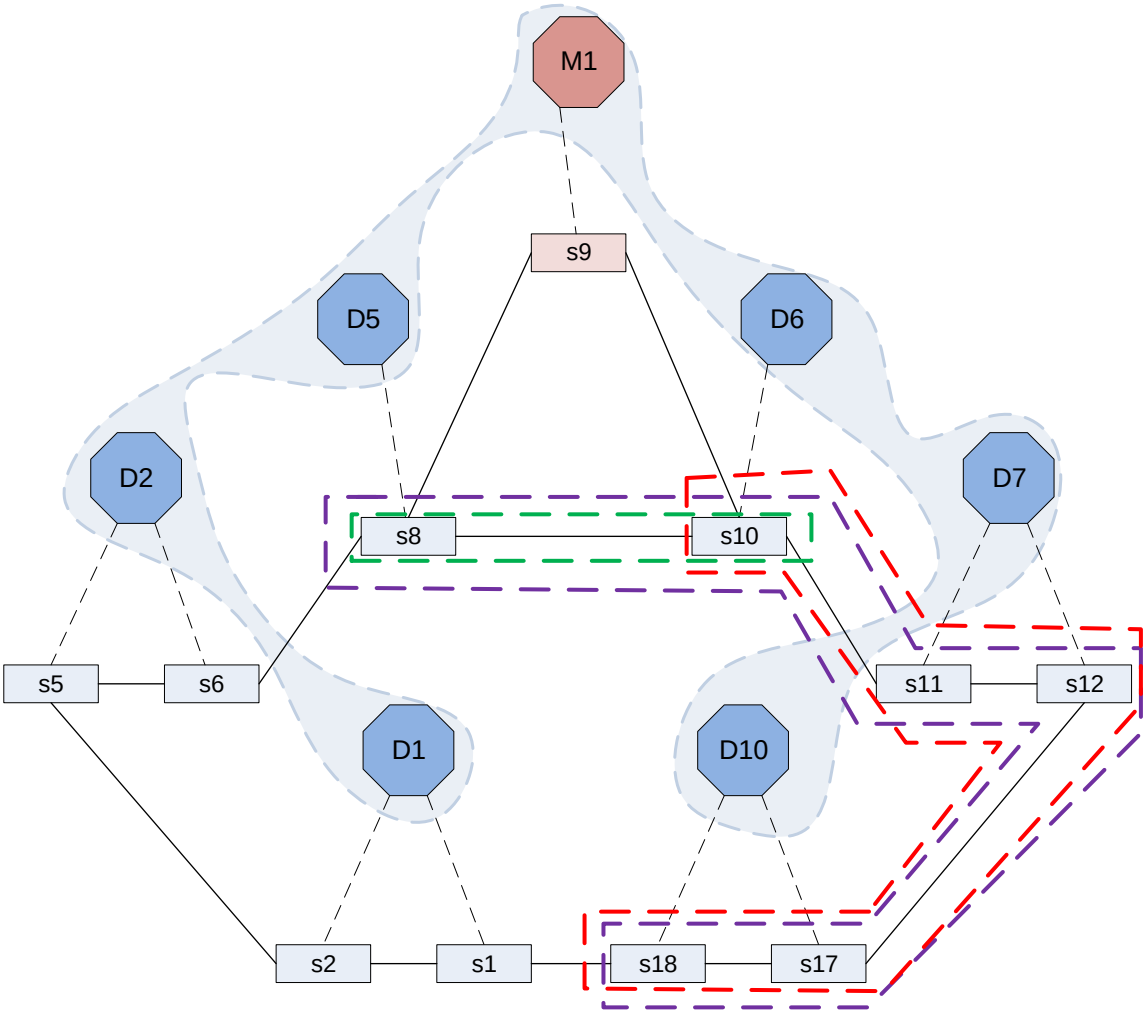

Figura 36 – Rotas escolhidas em cenários com vários *Workspaces*

D_5. A extensão na cor roxa da Figura 36 também representa os *switches* utilizados para estender o *Workspace*. No primeiro *attach* realizado por uma entidade D_{10} passou a conhecer o *Workspace*, portanto algoritmos inter-DTSA não foram mais executados.

A proposta e experimentação dos quatro cenários relatados acima tiveram como intuito verificar o tempo para *attach* das entidades. Esse tempo é apresentado (em *millisegundos*) na Figura 37. Na Figura 37(a) é mostrado o primeiro cenário. O gráfico apresenta o tempo de *attach* das primeiras cem entidades. Observa-se que o mesmo tempo é mantido na maior parte dos casos. De fato, como vários *Workspaces* diferentes foram criados, as entidades (ao solicitarem *attach*) aguardam o tempo dos algoritmos inter-DTSA rodarem e determinarem a rota para extensão.

A Figura 37(b) apresenta os resultados do segundo cenário. A figura mostra o tempo que as cem primeiras entidades aguardaram pela confirmação de *attach* no *Workspace* solicitado. Verifica-se que a amostragem mantém um tempo médio próximo de 400ms, tempo que é gasto para que a entidade solicite o *Workspace*, o DTSA faça *lookup* (pois não

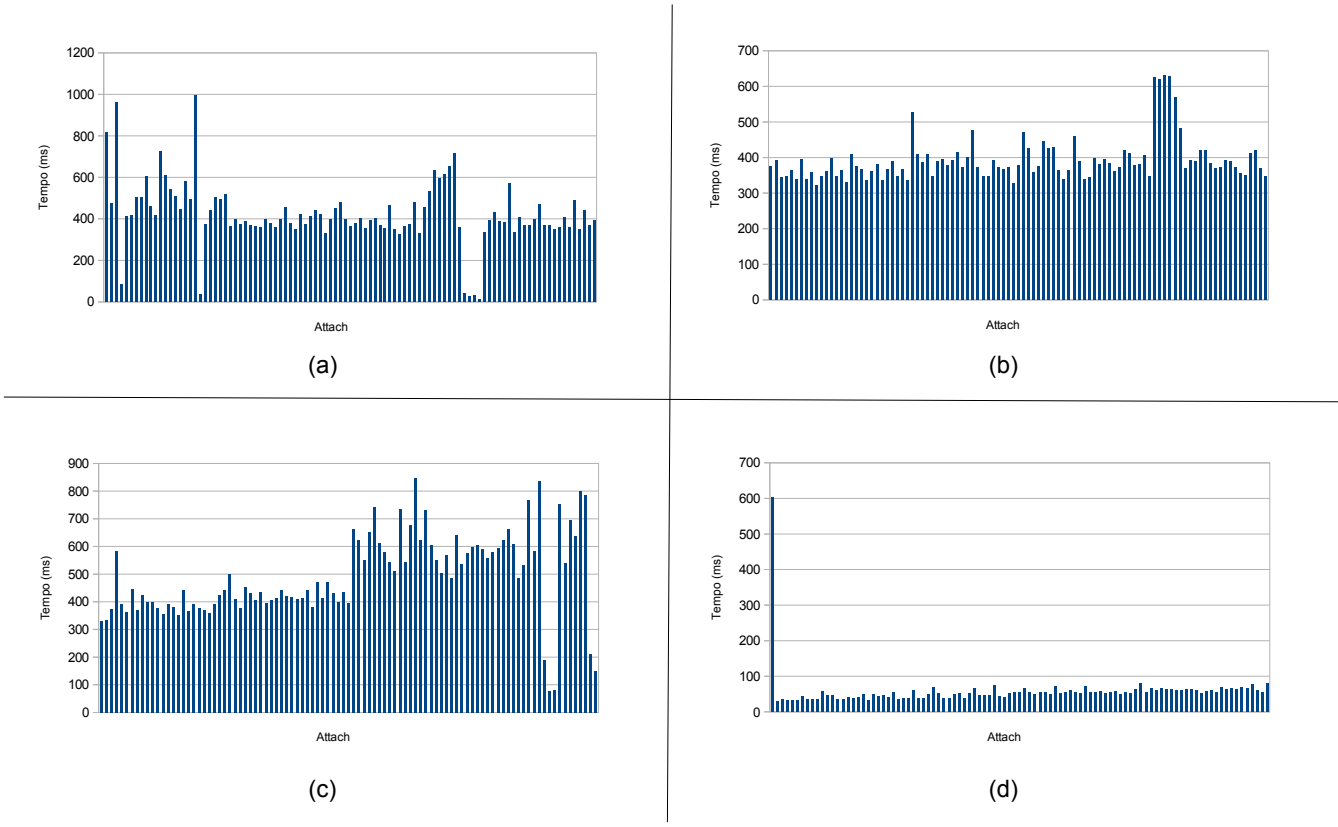

Figura 37 – Tempo de *attach* com vários *Workspaces* simultâneos

possui conhecimento do *Workspace*), receba uma rota e solicite configuração dos demais DTSAs da rota e o DTSA receba ack de cada DTSA da rota.

A Figura 37(c) apresenta o gráfico com os resultados do terceiro cenário. Observa-se que a partir do meio do gráfico os tempos sofrem um acentuado aumento. Isso ocorre pois tratam-se das entidades do DTSA D_{10}, o qual está mais longe do *Master* M_1. Ressalta-se que o *lookup* é encaminhado pelo *Workspace* de Controle e apenas o MDTSA possui lógica para tratá-lo. Assim sendo, DTSAs que estejam mais distantes fisicamente do seu MDTSA terão tempos de respostas de *lookup* maiores que aqueles próximos ao MDTSA.

A Figura 37(d) apresenta o resultado do quarto cenário. A figura mostra o tempo aguardado para cada entidade de D_{10} para receber confirmação de que o *Workspace* solicitado foi estendido até essa entidade. Nota-se que a primeira entidade esperou um tempo próximo de 600ms e as demais mantiveram o tempo abaixo de 100ms. Conclui-se que, uma vez que o DTSA D_{10} passa a conhecer o *Workspace*, o *attach* das demais entidades é mais rápido, visto que apenas algoritmos intra-DTSA serão executados após o primeiro *attach*.

5.3 Avaliação dos Resultados

Os resultados alcançados nos experimentos descritos na Seção 5.2 provaram ser viável utilizar *Workspaces* de Controle para suporte às comunicações dos DTSAs. Foi demonstrada a viabilidade do roteamento orientado a *workspace* tanto para topologias intra-DTSA quanto inter-DTSA. Provou-se que o DTSA é capaz de definir a melhor rota entre NEs previamente e o MDTSA definir a melhor rota entre DTSAs.

Nos experimentos inter-DTSA demonstrou-se que o MDTSA calcula a melhor rota, mesmo alterando-se a topologia de conectividade entre diferentes DTSAs.

Os tempos coletados no experimento intra-DTSA mostraram a eficiência de se estender um *Workspace* já conhecido por um NE vizinho, diferentemente da arquitetura Internet tradicional, onde a comunicação é realizada fim a fim (IP origem até IP destino). Os tempos coletados no experimento inter-DTSA mostraram a vantagem da flexibilidade de extensão dos *Workspaces*, uma vez que um *Workspace* estando já estendido em um DTSA não há necessidade de busca inter-DTSA.

Pode ser visto que o tempo de estabelecimento de um *Workspace* de Dados é plenamente aceitável e, após seu estabelecimento, o desempenho das transmissões dependem apenas de tempos de chaveamento nos elementos de rede.

Nos experimentos cada topologia é configurada antes que o experimento se inicie. O registro dinâmico de NEs em um DTSA e de DTSAs em um MDTSA devem ser propostos futuramente para tratar disponibilidade dos NEs, DTSAs e links em geral. Esta configuração inicial da rede pode ser bastante simplificada, reduzindo-se substancialmente o trabalho de configuração manual, por meio de um sistema de configuração do plano de controle.

Capítulo 6

Conclusão

O presente capítulo apresenta o fechamento deste trabalho, expondo as principais conclusões obtidas ao se propor um novo mecanismo de roteamento pertinente à arquitetura ETArch, em oposição àqueles existentes em arquiteturas de redes tradicionais.

Para se alcançar o objetivo geral do trabalho (propor um mecanismo intra/inter-domínio de roteamento na arquitetura ETArch) optou-se por trabalhar cada objetivo específico de forma a se chegar em contribuições relevantes para a ETArch. Assim sendo, o trabalho trouxe uma hipótese de roteamento que não se dispersa dos aspectos conceituais da ETArch, nem sequer das funções que o roteamento assume na Internet.

O primeiro objetivo introduzido na Seção 1.2 foi resolvido nas Seções 4.3 e 4.5. O mecanismo de roteamento proposto está de acordo com as definições do Modelo de Títulos e da arquitetura ETArch, além de cobrir roteamento intra e inter-domínio. Dessa forma pode-se utilizar o presente trabalho como referência para futuros trabalhos em roteamento e SDN.

Optou-se pela utilização dos *Workspaces* de Controle para tratar primitivas de roteamento (controle). Entende-se como acertada esta escolha, visto que é necessária a comunicação inter-DTSAs. Neste tipo de interação, um DTSA não envia primitivas direcionadas a outros DTSAs. Portanto, primitivas de controle exploram a característica de comunicação *multicast* natural a *Workspaces* de Controle e, portanto, todo DTSA que faça parte de cada *Workspace* de Controle recebe a primitiva (e a trata, se necessário).

Os algoritmos de roteamento intra/inter-DTSA buscam os requisitos nos *Workspaces* criados pelas entidades. Entende-se que esta também foi uma decisão acertada visto que, uma vez criado um *Workspace*, os requisitos passam a ser conhecidos pelo DTSA. Dessa forma, pode-se trabalhar em software quaisquer algoritmos para melhor definição de rotas. Além disso, a hipótese do Capítulo 4 tende a definir rotas antes que as primitivas de dados comecem a trafegar no *Workspace*.

O segundo objetivo específico, conforme introduzido na Seção 1.2, foi implantar o algoritmo proposto em um ambiente ETArch. A utilização dos códigos do projeto EDOBRA tornaram esse objetivo natural. A codificação foi melhorada para realização dos experi-

mentos apresentados no Capítulo 5. Utilizou-se o mesmo servidor de aplicação e versão de *framework* empregados no EDOBRA, criando-se assim rotinas de roteamento em um ambiente ETArch maduro.

O Capítulo 5 apresentou as demonstrações a que se propunha, de forma a alcançar o objetivo de demonstrar o algoritmo proposto. A hipótese do Capítulo 4 foi viabilizada através da implementação em ambiente ETArch do roteamento prévio, utilizando *Workspaces* de Controle. O capítulo demonstrou a utilização do algoritmo em topologia intra-domínio e através de topologias em domínios distintos.

A demonstração dos algoritmos e os números nas tabelas do Capítulo 5 servem como base para viabilizar o roteamento orientado a *workspace*. Na ETArch, diferentemente de arquiteturas convencionais, a comunicação *Multicast* é inata. Com isso, o roteamento consegue trabalhar de forma mais ágil, uma vez que *Workspaces* podem ser estendidos de NEs ou DTSAs mais próximos.

O tempo gasto por roteamento tende a ser mais eficaz, uma vez que as entidades não precisam se comunicar diretamente com a origem dos dados, o que permite ao roteamento apenas estender do NE mais próximo que já estenda o *Workspace* solicitado por uma entidade. Com isso, após estendido um *Workspace*, a parte algorítmica do roteamento tende a ter tempo zero, visto que o trabalho dos NEs será apenas de encaminhar primitivas conforme pré-determinado por cada DTSA.

A principal questão do objetivo geral deste trabalho foi propor um mecanismo para roteamento na ETArch. Entende-se que, como demonstrado nas seções 5.2.1 e 5.2.2, a proposição foi demonstrada, uma vez que nessas seções realizou-se uma prova de conceito do mecanismo proposto no Capítulo 4. Portanto o problema de rotear inter-domínio na ETArch está resolvido, uma vez que a partir deste trabalho diferentes DTSAs podem se comunicar (o que não havia anteriormente na arquitetura).

Um dos problemas encontrados durante o projeto se refere ao registro de *switches* (no DTSA) e de DTSAs (no MDTSA). Sem estes registros, as configurações de topologia são feitas manualmente, antes que os DTSAs sejam iniciados. Além destes problemas, perguntas relacionadas à escalabilidade não foram respondidas e seguem para trabalhos futuros.

Acredita-se que no futuro, os NEs serão apenas *switches*, com responsabilidade de encaminhar os dados das aplicações. Nenhum NE fará roteamento, visto que neste trabalho provou-se que, com o uso de controladores SDN, não é necessário roteadores ao longo do caminho. Acredita-se que na Internet do futuro a função de rotear será responsabilidade dos controladores e que essa função será realizada previamente (antes que as primitivas contendo os dados das aplicações comecem a ser encaminhadas). Na arquitetura ETArch, o plano de controle será constituído pelo DTS, cujos agentes (DTSA) são super conjunto de controladores SDN.

Com os objetivos específicos alcançados e as demonstrações havidas nos experimentos,

entende-se como cumprido o objetivo geral que é desenvolver o roteamento na ETArch. Uma vez atingidos os objetivos geral e específicos, as próximas seções tratam de destacar as principais contribuições deste trabalho e apresentar os trabalhos futuros para melhoria da hipótese atual, além daqueles que podem ser gerados a partir deste. Por fim as contribuições em produção bibliográfica serão listadas.

6.1 Principais Contribuições

O trabalho apresentou uma proposta de roteamento orientado a *workspace* para ser utilizado em ambientes SDN. A contribuição geral foi a definição do mecanismo de roteamento a ser explorado na ETArch, o qual não havia sido discutido em trabalhos anteriores. O mecanismo foi proposto buscando-se atingir domínios externos ao da topologia utilizada, o que viabiliza a utilização da ETArch em grandes redes, como a Internet, para a qual essa arquitetura havia sido projetada.

As primitivas WORKSPACE_LOOKUP e DTS_NOTIFY previstas na ETArch para o DTSCP foram utilizadas para o roteamento orientado a *workspace*, além da definição de uma nova, WORKSPACE_CONFIGURATION. O uso dessa nova primitiva permite que cada DTSA decida a melhor maneira de configurar os NEs sob seu controle para extensão de um *Workspace* entre diversos DTSAs. Com isso, é possível a utilização da ETArch em comunicações inter-domínio.

Entende-se que a utilização da primitiva WORKSPACE_CONFIGURATION é uma escolha válida, visto que na atualidade diferentes ASs se comunicam sem que um AS conheça a topologia dos demais. Conclui-se que esta característica será mantida no futuro, visto que operadoras de telecomunicações e grandes ISPs não têm intenção de divulgar informações sobre sua topologia para seus concorrentes.

Cada DTSA, ao receber um *config* (primitiva WORKSPACE_CONFIGURATION), pode definir em software a melhor rota para extensão de determinado *Workspace* entre seus NEs. Supondo que seja um DTSA intermediário (nenhuma de suas entidades faz parte do *Worskpace*) a configuração de extensão do *Workspace* pode ser realizada utilizando apenas NEs de borda (caso haja uma rota disponível), não necessitando configurações no núcleo da topologia.

A utilização da primitiva DTS_NOTIFY foi demonstrada e viabilizada nos experimentos. No experimento inter-DTSA o MDTSA conseguiu responder corretamente aos pedidos de WORKSPACE_LOOKUP que foram solicitados. A extensão dos *Workspaces* pode ser feita considerando a melhor rota, uma vez que o MDTSA possuía conhecimento do *Workspace* criado em um DTSA diferente daquele que solicitou o serviço de *lookup*.

A utilização da primitiva WORKSPACE_LOOKUP e os serviços que são efetuados pelo MDTSA ao receber essa primitiva foram viabilizados nos experimentos. Cada experimento inter-DTSA provou que um MDTSA define a melhor rota inter-DTSAs, mostrando

que os serviços invocados por esta super entidade são executados corretamente.

Como mencionado nos parágrafos anteriores, as três primitivas idealizadas para definição de rotas e configuração correta para extensão de *Workspaces* foram implementadas em DTSAs, inclusive naqueles do tipo MDTSA. Dessa forma pode-se afirmar que o conceito de MDTSA é viável. Nas demonstrações o MDTSA comunicou-se corretamente com o *Worskpace* de Controle ao qual estava presente, além do que as primitivas lançadas no *Workspace* de Controle chegaram de forma correta ao MDTSA e foram tratadas com sucesso.

A definição de rota previamente por um DTSA mostra tempos de configuração otimizados em relação à arquiteturas tradicionais. No experimento intra-DTSA com duzentos NEs mostrou-se que a confirmação de *attach* para uma entidade é mais rápida uma vez que um NE mais próximo já estende o *Workspace* solicitado. Essa melhora mostra que o algoritmo de roteamento escolhe de fato a melhor forma de estender um *Workspace*, a qual aproveita das vantagens de utilização da ETArch.

Outra contribuição relevante do presente trabalho foi a definição de utilização dos *Workspaces* de Controle para roteamento. A ETArch utilizava os *Workspaces* criados sob demanda para envio e recebimento das primitivas de dados. Com os *Workspaces* de Controle os DTSAs não se comunicam diretamente, cada primitiva de controle inter-DTSA é lançada nos *Workspaces* de Controle e todos os DTSAs participantes receberão e farão as devidas tratativas.

A contribuição do trabalho em relação às topologias inter-domínio foi a definição de um algoritmo para roteamento inter-DTSA. Com isso afirma-se ser viável a utilização da ETArch em ambientes múltiplos, ie, aqueles em domínios separados. Os experimentos mostraram que o algoritmo proposto é capaz de atribuir configurações corretas para extensão de *Workspaces* através da melhor rota disponível.

Como contribuição final do trabalho foram desenvolvidos e incorporados aos códigos do projeto EDOBRA rotinas de roteamento orientado a *workspace*. Os próximos trabalhos na arquitetura podem aproveitar essas rotinas para seus desenvolvimentos. A implementação foi feita em linguagem Java e implantada na plataforma Mobicents, para que futuros trabalhos relacionados com disponibilidade e escalabilidade possam usufruir desta plataforma, não abrindo mão do roteamento idealizado neste trabalho.

Finalmente pode-se concluir pela viabilidade da utilização do roteamento orientado a *workspace*. Provou-se através deste trabalho que este novo mecanismo para roteamento é viável para a ETArch. Afirma-se também que a definição prévia das rotas pode ser implantada em topologias intra e inter-domínio para uso na Internet, trazendo a vantagem que essa abordagem consegue estabelecer: estender *Workspaces* sem a necessidade de se chegar à origem do mesmo em cada nova solicitação de extensão.

Por fim entende-se que a utilização de roteamento orientado a *workspace* conforme demonstrado nos experimentos permite que NEs possuam exclusivamente a função de

encaminhar dados. Todo o trabalho algorítmico do roteamento (definir a rota de fato) passa a ser papel do controlado SDN. Com isso é possível eliminar os roteadores em Redes Definidas por Software.

6.2 Trabalhos Futuros

Com o roteamento orientado a *workspace* tendo sido apresentado e implantado em ambiente ETArch é possível que novos trabalhos possam ser conduzidos. Inicialmente, espera-se continuar as melhorias na proposta deste trabalho. O primeiro passo é a inclusão de novos requisitos para experimentos, conseguindo alcançar um roteamento multiobjetivo através da ETArch.

Para que novos requisitos sejam adicionados para roteamento, um passo necessário é a construção de um *switch* ETArch. Esse *switch* será considerado como uma entidade na rede, portanto fará seu registro a priori. O *switch*, ao se registrar, informará ao DTSA (DTSA que o controla) suas portas, links, capacidade dos links, gasto de energia etc. Qualquer informação que o DTSA possa aproveitar no roteamento deve ser enviada pelo *switch* em seu registro.

Além dos requisitos, determinadas limitações também devem ser incluídas em trabalhos futuros. No presente trabalho a limitação das tabelas de fluxos dos *switches* Openflow não foi considerada. Desta forma, caso algum NE tenha sua tabela lotada a extensão de novos *Workspaces* é negada (visto que não há mais possibilidade de inserir novos registros no NE).

Um trabalho futuro relevante é a inclusão nos DTSAs e MDTSAs de rotinas para que o roteamento analise as limitações dos NEs. No caso da quantidade máxima de registros na tabela de fluxos uma possível solução seria: o DTSA deve escolher uma rota alternativa caso a melhor rota não seja possível (pois algum NE pelo caminho não tem mais suporte). Ainda, sempre que um novo *Workspace* seja incluído ou retirado de um NE o DTSA deve ser avisado, para que este saiba quando aquele NE não está mais disponível para estender *Workspaces*.

A disponibilidade é uma característica que será explorada na ETArch através do roteamento, visto que uma rota estabelecida pode necessitar alterações quando algum link ou DTSA ao longo dessa rota não estiver mais disponível. Neste caso os trabalhos em roteamento deverão propor um mecanismo para controle de links entre NEs e DTSAs, além de rotinas para reconfiguração de NEs.

A questão de sustentabilidade também deverá ser considerada nos trabalhos futuros. Algoritmos para controle dos links deverão ser explorados de forma que as rotas (caminhos por onde passam os *Workspaces*) possam aproveitar de forma eficaz a questão energética. Dessa forma, links podem ser desligados como acontece em outras abordagens ou até mesmo a decisão final de uma rota levar em consideração o requisito de consumo

energético.

Em relação a arquitetura como um todo espera-se que a ETArch evolua de forma que quaisquer novos requisitos que surgirem no futuro possam ser facilmente tratados pela arquitetura. Para isso trabalhos relacionados à características *Carrier Grade* como alta disponibilidade e escalabilidade devem ser explorados. Além disso, trabalhos relacionados especificamente à segurança e desempenho de um futuro *switch* ETArch devem ser analisados.

Para uso em operadores de telecomunicações e ISPs em geral espera-se trabalhos na ETArch envolvendo SDN e NFV. Com isso é possível realizar *testbeds* em redes reais, tornando a ETArch compatível com redes legadas.

Além dos trabalhos esperados em relação a ETArch acima citados, faz-se necessário o desenvolvimento de novas aplicações para testar massivamente o roteamento e a própria arquitetura. Atualmente, os testes realizados são feitos através das aplicações de chat e *streaming* de vídeo implementadas para a ETArch. Ressalva-se a importância em trabalhos futuros de se explorar aplicações de telefonia e também aquelas que transfiram uma carga elevada de dados.

6.3 Contribuições em Produção Bibliográfica

Ao longo do trabalho foram explorados temas relacionados a redes de computadores e telecomunicações. Realizou-se o Projeto de Pesquisa Internacional EDOBRA (ExtenDing Ofelia in BRAzil), concluindo todos os seus *deliverables*.

Diversos artigos foram publicados em congressos qualis classificados no índice restrito, que são:

- *Extending a 3GPP Prepaid Protocol to Improve Credit Pre-reservation Mechanism.* Publicado no The Tenth Advanced International Conference on Telecommunications (AICT), 2014, Paris.

- *UML-based Modeling Entity Title Architecture (ETArch) Protocols.* Publicado no The Tenth Advanced International Conference on Telecommunications (AICT), 2014, Paris.

- *Towards a Carrier Grade SDN Controller: Integrating OpenFlow with Telecom Services.* Publicado no The Tenth Advanced International Conference on Telecommunications (AICT), 2014, Paris. Este artigo ganhou *Best Paper Award*.

- *Control Plane Routing Protocol for the Entity Title Architecture: Design and Specification.* Publicado no The Fourteenth International Conference on Network (ICN^2 2015), 2015, Barcelona.

Todos os artigos acima estão relacionados com pesquisas desenvolvidas ao longo do trabalho. O quarto artigo (*Control Plane Routing Protocol for the Entity Title Architecture: Design and Specification*) é o artigo que especifica o algoritmo de roteamento orientado a *workspace* proposto no presente trabalho.

Além dos acima citados, os artigos *Enabling a Carrier Grade SDN by Using a Top-Down Approach* e *Deployment and Experimentation of the Entity Title Architecture at OFELIA Testbed: Learned Lessons Revealed* foram apresentados respectivamente em 2014 e 2015 no Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF), Workshop que faz parte do SBRC (Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos).

Referências

ATKINSON, R.; BHATTI, S. N. **Identifier-Locator Network Protocol (ILNP) Engineering Considerations**. IETF, 2012. RFC 6741 (Draft Standard). (Request for Comments, 6741). Disponível em: <https://tools.ietf.org/rfc/rfc6741.txt>.

BEHDADFAR, M. et al. Scalar prefix search: A new route lookup algorithm for next generation internet. In: **INFOCOM 2009, IEEE**. [S.l.: s.n.], 2009. p. 2509–2517. ISSN 0743-166X.

BENNESBY, R. et al. An inter-as routing component for software-defined networks. In: **Network Operations and Management Symposium (NOMS), 2012 IEEE**. [S.l.: s.n.], 2012. p. 138–145. ISSN 1542-1201.

Big Switch Networks. **Developing Floodlight Modules. Floodlight OpenFlow Controller**. 2012. Http://floodlight.openflowhub.org/developing-floodlight/. Disponível em: <http://floodlight.openflowhub.org/developing-floodlight/>.

_____. **Floodlight OpenFlow Controller**. 2014. Http://floodlight.openflowhub.org/. Disponível em: <http://floodlight.openflowhub.org/>.

_____. **Project Floodlight**. 2015. Http://www.projectfloodlight.org/documentation/. Disponível em: <http://www.projectfloodlight.org/documentation/>.

CAI, Z. **Maestro**. 2014. Disponível em: <http://code.google.com/p/maestro-platform/>.

CHEN, T.-Y. et al. Eegra: Energy efficient geographic routing algorithms for wireless sensor network. In: **Pervasive Systems, Algorithms and Networks (ISPAN), 2012 12th International Symposium on**. [S.l.: s.n.], 2012. p. 104–113. ISSN 1087-4089.

CHENG, H.; JIA, X. An energy efficient routing algorithm for wireless sensor networks. In: **Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on**. [S.l.: s.n.], 2005. v. 2, p. 905–910.

CIANFRANI, A. et al. An energy saving routing algorithm for a green ospf protocol. In: **INFOCOM IEEE Conference on Computer Communications Workshops , 2010**. [S.l.: s.n.], 2010. p. 1–5.

DAY, J.; ZIMMERMANN, H. The OSI Reference Model. **Proceedings of the IEEE**, v. 71, n. 12, p. 1334–1340, Dec 1983. ISSN 0018-9219.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numer. Math.**, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 1, n. 1, p. 269–271, dez. 1959. ISSN 0029-599X. Disponível em: <http://dx.doi.org/10.1007/BF01386390>.

DU, X. A simulation study of an energy efficient routing protocol for mobile ad hoc networks. In: **Simulation Symposium, 2004. Proceedings. 37th Annual**. [S.l.: s.n.], 2004. p. 125–131. ISSN 1080-241X.

ERICKSON, D. **Beacon**. 2014. Disponível em: <https://openflow.stanford.edu/display/Beacon/Home>.

FARHADY, H.; LEE, H.; NAKAO, A. Software-defined networking. **Comput. Netw.**, Elsevier North-Holland, Inc., New York, NY, USA, v. 81, n. C, p. 79–95, abr. 2015. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2015.02.014>.

FEMMINELLA, M. et al. Implementation and performance analysis of advanced IT services based on open source JAIN SLEE. In: **2011 IEEE 36th Conference on Local Computer Networks (LCN)**. [S.l.: s.n.], 2011. p. 746–753.

FERRY, D. **JAIN SLEE (JSLEE) 1.1 Specification, Final Release**. 2014. Disponível em: <http://www.jcp.org/en/jsr/detail?id=240>.

FIELDING, R. et al. **Hypertext Transfer Protocol – HTTP/1.1**. IETF, 1999. RFC 2616 (Draft Standard). (Request for Comments, 2616). Updated by RFCs 2817, 5785, 6266. Disponível em: <http://www.ietf.org/rfc/rfc2616.txt>.

FOKKINK, W. **Distributed Algorithms: An Intuitive Approach**. [S.l.]: The MIT Press, 2013. ISBN 0262026775, 9780262026772.

GANICHEV, I. A. **Interdomain Multipath Routing**. Tese (Doutorado) — EECS Department, University of California, Berkeley, Dec 2011. Disponível em: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-136.html>.

GAWLICK, R.; KALMANEK, C.; RAMAKRISHNAN, K. G. On-line routing for permanent virtual circuits. **Comput. Commun.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 19, n. 3, p. 235–244, mar. 1996. ISSN 0140-3664. Disponível em: <http://dx.doi.org/10.1016/0140-3664(96)01064-X>.

GONçALVES, M. A. **ETARCH: Projeto e Desenvolvimento de uma Arquitetura para o Modelo de Título com foco na Ageração de Trágego Multicast**. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia, Sep 2014.

GUDE, N. et al. NOX: towards an operating system for networks. **SIGCOMM Comput. Commun. Rev.**, v. 38, n. 3, p. 105–110, jul. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1384609.1384625>.

GUIMARAES, C. et al. Ieee 802.21-enabled entity title architecture for handover optimization. In: IEEE. **Wireless Communications and Networking Conference (WCNC), 2014 IEEE**. [S.l.], 2014. p. 2671–2676.

HAN, B. et al. Network function virtualization: Challenges and opportunities for innovations. **Communications Magazine, IEEE**, v. 53, n. 2, p. 90–97, Feb 2015. ISSN 0163-6804.

HELLER, B. et al. Elastictree: Saving energy in data center networks. In: **Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2010. (NSDI'10), p. 17–17. Disponível em: <http://dl.acm.org/citation.cfm?id=1855711.1855728>.

ITU. **ITU-T Recommendation Q.700 : Introduction to CCITT Signalling System No. 7**. [S.l.], 1994.

_____. **ITU-T Recommendation Q.1200 : General series Intelligent Network Recommendation structure**. [S.l.], 1997.

JANURAIO, G. C. **Método para Avaliação de Sistema de Gerenciamento de Redes Orientado a Eficiência Energética**. Dissertação (Mestrado) — Universidade de São Paulo, São Paulo, Apr 2014.

JIA, W.-K.; WANG, L.-C. A unified unicast and multicast routing and forwarding algorithm for software-defined datacenter networks. **Selected Areas in Communications, IEEE Journal on**, v. 31, n. 12, p. 2646–2657, December 2013. ISSN 0733-8716.

KERNER, S. **OpenFlow Protocol 1.3.0 Approved**. 2012. Http://www.enterprisenetworkingplanet.com/nethub/openflow-protocol-1.3.0-approved.html. Disponível em: <http://www.enterprisenetworkingplanet.com/nethub/openflow-protocol-1.3.0-approved.html>. Acesso em: 17 jun. 2012.

KIM, D.; NOEL, E.; TANG, K. Reliable and efficient routing protocol for graph theory based communication topology. In: **Performance Computing and Communications Conference (IPCCC), 2013 IEEE 32nd International**. [S.l.: s.n.], 2013. p. 1–10.

KONTESIDOU, G.; ZARIFIS, K. **Openflow Virtual Networking: A Flow-Based Network Virtualization Architecture**. Dissertação (Mestrado) — Royal Institute of Technology, Stockholm, Nov 2009.

KOPONEN, T. et al. Onix: a distributed control platform for large-scale production networks. In: **Proceedings of the 9th USENIX conference on Operating systems design and implementation**. Berkeley, CA, USA: USENIX Association, 2010. (OSDI'10), p. 1–6. Disponível em: <http://dl.acm.org/citation.cfm?id=1924943.1924968>.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219.

KUROSE, J.; ROSS, K. **Computer Networking: A Top-down Approach**. Addison-Wesley, 2010. ISBN 9780136079675. Disponível em: <https://books.google.com.br/books?id=gxLePQAACAAJ>.

LEE, M.-W. et al. Robust multipath multicast routing algorithms for videos in software-defined networks. In: **Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of**. [S.l.: s.n.], 2014. p. 218–227.

LEMA, J. C. **Evolving Future Internet Clean-Slate Entity Title Architecture with Quality-Oriented Control-Plane Extensions**. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, Natal, Jul 2014.

LI, D.; SHANG, Y.; CHEN, C. Software defined green data center network with exclusive routing. In: **INFOCOM, 2014 Proceedings IEEE**. [S.l.: s.n.], 2014. p. 1743–1751.

LIBERAL, F.; FAJARDO, J. O.; KOUMARAS, H. Qoe and *-awareness in the future internet. In: **Towards the Future Internet - A European Research Perspective**. [s.n.], 2009. p. 293–302. Disponível em: <http://dx.doi.org/10.3233/978-1-60750-007-0-293>.

LIN, F.-S.; CHENG, K.-T. Virtual path assignment and virtual circuit routing in atm networks. In: **Global Telecommunications Conference, 1993, including a Communications Theory Mini-Conference. Technical Program Conference Record, IEEE in Houston. GLOBECOM '93., IEEE**. [S.l.: s.n.], 1993. p. 436–441 vol.1.

LIN, Z. et al. A routing framework in software defined network environment. In: **Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 2014 Tenth International Conference on**. [S.l.: s.n.], 2014. p. 907–911.

MATSUMOTO, C. blog, **What OpenDaylight Really Wants to Do**. 2014. Disponível em: <http://www.lightreading.com/blog/software-defined-networking/what-opendaylight-really-wants-to-do/240152993>.

MCKENZIE, S. Asynchronous transfer mode (atm). In: **Encyclopedia of Computer Science**. Chichester, UK: John Wiley and Sons Ltd., 2003. p. 107–108. ISBN 0-470-86412-5. Disponível em: <http://dl.acm.org/citation.cfm?id=1074100.1074144>.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>.

MOBICENTS. **Mobicents JAIN SLEE**. 2014. Http://www.mobicents.org/slee/intro.html. Disponível em: <http://www.mobicents.org/slee/intro.html>.

MOY, J. **OSPF specification**. IETF, 1989. RFC 1131 (Proposed Standard). (Request for Comments, 1131). Obsoleted by RFC 1247. Disponível em: <http://www.ietf.org/rfc/rfc1131.txt>.

_____. **OSPF Version 2**. IETF, 1998. RFC 2328 (Standard). (Request for Comments, 2328). Updated by RFC 5709. Disponível em: <http://www.ietf.org/rfc/rfc2328.txt>.

NASCIMENTO, M. R. et al. Virtual routers as a service: The routeflow approach leveraging software-defined networks. In: **Proceedings of the 6th International Conference on Future Internet Technologies**. New York, NY, USA: ACM, 2011. (CFI '11), p. 34–37. ISBN 978-1-4503-0821-2. Disponível em: <http://doi.acm.org/10.1145/2002396.2002405>.

NETWORKS, B. S. **Big Network Controller**. 2014. Disponível em: <http://www.bigswitch.com/products/SDN-Controller>.

NOXREPO. **About POX**. 2013. Disponível em: <http://www.noxrepo.org/pox/about-pox/>.

_____. **About NOX**. 2014. Disponível em: <http://www.noxrepo.org/nox/about-nox/>.

ON.LAB. **ONOS - Open Network Operating System**. 2014. Disponível em: <http://tools.onlab.us/onos.html>.

_____. **ONOS at ONS 2014**. 2014. Disponível em: <http://www.slideshare.net/ON_LAB/onos-at-ons-2014>.

Open vSwitch. **Open vSwitch - An Open Virtual Switch**. 2012. Http://openvswitch.org/. Disponível em: <http://openvswitch.org/>.

OpenDaylight. **OpenDaylight Controller - Architectural Framework**. 2014. Disponível em: <https://wiki.opendaylight.org/view/OpenDaylight_Controller:Architectural_Framework>.

_____. **OpenDaylight Technical Overview**. 2014. Disponível em: <http://www.opendaylight.org/project/technical-overview>.

OpenFlow. **OpenFlow**. 2015. Http://www.openflow.org/. Disponível em: <http://www.openflow.org/>.

OpenStack. **OpenStack: The Open Source Cloud Operating System**. 2015. Https://www.openstack.org/. Disponível em: <https://www.openstack.org/>.

OWENS, H.; DURRESI, A. Explicit routing in software-defined networking (ersdn): Addressing controller scalability. In: **Network-Based Information Systems (NBiS), 2014 17th International Conference on**. [S.l.: s.n.], 2014. p. 128–134.

PARK, S.-S.; SHIRATORI, N. Distributed Systems Management Based On OSI Environment: Problems, Solutions, and Their Evaluation. In: **IEEE 13th Annual International Phoenix Conference on Computers and Communications**. [S.l.: s.n.], 1994. p. 384–.

PEREIRA, J. de S. et al. Title model ontology for future internet networks. In: DOMINGUE, J. et al. (Ed.). **The Future Internet**. [S.l.]: Springer Berlin Heidelberg, 2011, (Lecture Notes in Computer Science, v. 6656). p. 103–114. ISBN 978-3-642-20897-3.

PEREIRA, J. H. d. S. **Modelo de título para a próxima geração de Internet.** Tese (text) — Universidade de São Paulo, fev. 2012. Tese de Doutorado. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3142/tde-13062013-113304/>.

POSTEL, J. **Internet Protocol**. IETF, 1981. RFC 791 (Standard). (Request for Comments, 791). Updated by RFC 1349. Disponível em: <http://www.ietf.org/rfc/rfc791.txt>.

REKHTER, Y.; LI, T.; HARES, S. **A Border Gateway Protocol 4 (BGP-4)**. IETF, 2006. RFC 4271 (Draft Standard). (Request for Comments, 4271). Disponível em: <http://www.ietf.org/rfc/rfc4271.txt>.

RINA. **Recursive InterNetwork Architecture**. 2015. Http://csr.bu.edu/rina/. Disponível em: <http://csr.bu.edu/rina/>.

RODRIGUE, D. et al. Routing protocols: When to use it in terms of energy? In: **Wireless Communications and Networking Conference (WCNC), 2012 IEEE**. [S.l.: s.n.], 2012. p. 2763–2768. ISSN 1525-3511.

ROSEN, E.; VISWANATHAN, A.; CALLON, R. **Multiprotocol Label Switching Architecture**. IETF, 2001. RFC 3031 (Proposed Standard). (Request for Comments, 3031). Updated by RFC 6178. Disponível em: <http://www.ietf.org/rfc/rfc3031.txt>.

ROSENBERG, J. et al. **SIP: Session Initiation Protocol**. IETF, 2002. RFC 3261 (Proposed Standard). (Request for Comments, 3261). Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141. Disponível em: <http://www.ietf.org/rfc/rfc3261.txt>.

RouteFlow. **RouteFlow**. 2015. Https://sites.google.com/site/routeflow/home. Disponível em: <https://sites.google.com/site/routeflow/home>.

SHERWOOD, R. et al. Flowvisor: A network virtualization layer. **OpenFlow Switch Consortium, Tech. Rep**, 2009.

SILVA, F. et al. Entity title architecture extensions towards advanced quality-oriented mobility control capabilities. In: **Computers and Communication (ISCC), 2014 IEEE Symposium on**. [S.l.: s.n.], 2014. p. 1–6.

SILVA, F. de O. **Endereçamento por título: Uma forma de encaminhamento multicast para a próxima geração de redes de computadores**. Tese (Doutorado) — Universidade de Sao Paulo, Sao Paulo, Dec 2013.

SLUIJS, N. et al. Caching strategy for scalable lookup of personal content. In: **Advances in P2P Systems, 2009. AP2PS '09. First International Conference on**. [S.l.: s.n.], 2009. p. 19–26.

SONIYA, M. M. S.; KUMAR, K. A survey on named data networking. In: **Electronics and Communication Systems (ICECS), 2015 2nd International Conference on**. [S.l.: s.n.], 2015. p. 1515–1519.

SWIATEK, P. et al. Supporting content, context and user awareness in future internet applications. In: **The Future Internet - Future Internet Assembly 2012: From Promises to Reality**. [s.n.], 2012. p. 154–165. Disponível em: <http://dx.doi.org/10.1007/978-3-642-30241-1_14>.

THALER, D.; HOPPS, C. **Multipath Issues in Unicast and Multicast Next-Hop Selection**. United States: RFC Editor, 2000.

_____. **Multipath Issues in Unicast and Multicast Next-Hop Selection**. IETF, 2000. RFC 2991 (Informational). (Request for Comments, 2991). Disponível em: <http://www.ietf.org/rfc/rfc2991.txt>.

VENKATARAMANI, A. et al. Mobilityfirst: A mobility-centric and trustworthy internet architecture. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 3, p. 74–80, jul. 2014. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/2656877.2656888>.

VRIJDERS, S. et al. Experimental evaluation of a recursive internetwork architecture prototype. In: **Global Communications Conference (GLOBECOM), 2014 IEEE**. [S.l.: s.n.], 2014. p. 2017–2022.

WANG, R. et al. Energy-aware routing algorithms in software-defined networks. In: **A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on**. [S.l.: s.n.], 2014. p. 1–6.

WETHERALL, D. J.; TANENBAUM, A. S. **Computer Networks**. [S.l.]: Pearson Education, 2013. ISBN 1292024224, 9781292024226.

WOOD, T. et al. Toward a software-based network: integrating software defined networking and network function virtualization. **Network, IEEE**, v. 29, n. 3, p. 36–41, May 2015. ISSN 0890-8044.

XU, Z.; DAI, S.; GARCIA-LUNA, J. J. **A MORE EFFICIENT DISTANCE VECTOR ROUTING ALGORITHM**. Santa Cruz, CA, USA, 1997.

XYLOMENOS, G. et al. A survey of information-centric networking research. **IEEE Communications Surveys Tutorials**, v. 16, n. 2, p. 1024–1049, Second 2014. ISSN 1553-877X.

YANG, X.; CLARK, D.; BERGER, A. Nira: A new inter-domain routing architecture. **Networking, IEEE/ACM Transactions on**, v. 15, n. 4, p. 775–788, Aug 2007. ISSN 1063-6692.

Apêndices

APÊNDICE **A**

Codificação das topologias utilizadas

As topologias utilizadas para os experimentos no Capítulo 5 foram criadas através de scripts em linguagem Python. Esses scripts foram passados para o Mininet através do parâmetro custom. *Dessa forma, o Mininet inicia-se considerando a topologia presente em cada script.*

Os scripts utilizados são apresentados no presente Apêndice.

A.1 Script Python do experimento intra-DTSA

```
#!/usr/bin/python

"""
Create a network where switches are connected to
a remote controller.
"""

import re
from mininet.net import Mininet
from mininet.node import OVSSwitch, Controller, RemoteController
from mininet.topolib import TreeTopo
from mininet.topo import LinearTopo
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.topo import Topo

setLogLevel( 'info' )

# One "external" controller (which is actually d1)
d1 = RemoteController( 'd1', ip='192.168.56.102' )
```

```
cmap = { 's1': d1, 's2': d1, 's3': d1, 's4': d1, 's5': d1,
's6': d1, 's7': d1,'s8': d1,'s9': d1 }

class MultiSwitch( OVSSwitch ):
  def start( self, controllers ):
    return OVSSwitch.start( self, [ cmap[ self.name ] ] )

def toID(mac):
  return '0000' + re.sub('[:]', '', mac)

class MyTopo( Topo ):
  "Simple topology example."

  def __init__( self ):
    "Create custom topo."

    # Initialize topology
    Topo.__init__( self )

    # Add hosts and switches
    h1s1 = self.addHost('h1s1')
    h1s4 = self.addHost('h1s4')
    h1s5 = self.addHost('h1s5')
    h1s6 = self.addHost('h1s6')

    s1 = self.addSwitch('s1', dpid=toID('00:00:00:00:00:01'))
    s2 = self.addSwitch('s2', dpid=toID('00:00:00:00:00:02'))
    s3 = self.addSwitch('s3', dpid=toID('00:00:00:00:00:03'))
    s4 = self.addSwitch('s4', dpid=toID('00:00:00:00:00:04'))
    s5 = self.addSwitch('s5', dpid=toID('00:00:00:00:00:05'))
    s6 = self.addSwitch('s6', dpid=toID('00:00:00:00:00:06'))
    s7 = self.addSwitch('s7', dpid=toID('00:00:00:00:00:07'))
    s8 = self.addSwitch('s8', dpid=toID('00:00:00:00:00:08'))
    s9 = self.addSwitch('s9', dpid=toID('00:00:00:00:00:09'))

    # Add links
    self.addLink(s1, h1s1, 1, 1)
    self.addLink(s1, s2, 2, 1)
    self.addLink(s2, s3, 2, 1)
```

```
        self.addLink(s3, s4, 2, 1)
        self.addLink(s4, s5, 2, 1)
        self.addLink(s5, s6, 2, 1)
        self.addLink(s6, s7, 2, 1)
        self.addLink(s7, s8, 2, 1)
        self.addLink(s8, s1, 2, 3)
        self.addLink(s1, s9, 4, 1)
        self.addLink(s9, s5, 2, 3)
        self.addLink(s4, h1s4, 3, 1)
        self.addLink(s5, h1s5, 4, 1)
        self.addLink(s6, h1s6, 3, 1)

topo = MyTopo()
net = Mininet( topo=topo, switch=MultiSwitch, build=False )
for c in [ d1 ]:
    net.addController(c)
net.build()
net.start()
CLI( net )
net.stop()
```

A.2 Script Python do primeiro experimento inter-DTSA

```
#!/usr/bin/python

"""
Create a network where different switches are connected to
different controllers.
"""

import re
from mininet.net import Mininet
from mininet.node import OVSSwitch, Controller, RemoteController
from mininet.topolib import TreeTopo
from mininet.topo import LinearTopo
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.topo import Topo
```

```
setLogLevel( 'info' )

# Seven "external" controllers
d1 = RemoteController( 'd1', ip='192.168.0.193' )
d2 = RemoteController( 'd2', ip='192.168.0.189' )
m1 = RemoteController( 'm1', ip='192.168.0.192' )
d5 = RemoteController( 'd5', ip='192.168.0.158' )
d6 = RemoteController( 'd6', ip='192.168.0.137' )
d7 = RemoteController( 'd7', ip='192.168.0.128' )
d10 = RemoteController( 'd10', ip='192.168.0.170' )

cmap = { 's1': d1, 's2': d1, 's5': d2, 's6': d2, 's8': d5,
 's9': m1, 's10': d6, 's11': d7, 's12': d7, 's17': d10, 's18': d10 }

class MultiSwitch( OVSSwitch ):
  def start( self, controllers ):
    return OVSSwitch.start( self, [ cmap[ self.name ] ] )

def toID(mac):
  return '0000' + re.sub('[:]', '', mac)

class MyTopo( Topo ):
  "Simple topology example."

  def __init__( self ):
    "Create custom topo."

    # Initialize topology
    Topo.__init__( self )

    # Add hosts and switches
    h1s1 = self.addHost('h1s1')
    h1s18 = self.addHost('h1s18')
    h2s18 = self.addHost('h2s18')

    s1 = self.addSwitch('s1', dpid=toID('00:00:00:00:00:01'))
    s2 = self.addSwitch('s2', dpid=toID('00:00:00:00:00:02'))
    s5 = self.addSwitch('s5', dpid=toID('00:00:00:00:00:05'))
```

```
        s6 = self.addSwitch('s6', dpid=toID('00:00:00:00:00:06'))
        s8 = self.addSwitch('s8', dpid=toID('00:00:00:00:00:08'))
        s9 = self.addSwitch('s9', dpid=toID('00:00:00:00:00:09'))
        s10 = self.addSwitch('s10', dpid=toID('00:00:00:00:00:10'))
        s11 = self.addSwitch('s11', dpid=toID('00:00:00:00:00:11'))
        s12 = self.addSwitch('s12', dpid=toID('00:00:00:00:00:12'))
        s17 = self.addSwitch('s17', dpid=toID('00:00:00:00:00:17'))
        s18 = self.addSwitch('s18', dpid=toID('00:00:00:00:00:18'))

        # Add links
        self.addLink(s1, h1s1, 1, 1)
        self.addLink(s1, s2, 2, 1)
        self.addLink(s2, s5, 2, 1)
        self.addLink(s5, s6, 2, 1)
        self.addLink(s6, s8, 2, 1)
        self.addLink(s8, s9, 2, 1)
        self.addLink(s9, s10, 2, 1)
        self.addLink(s10, s11, 2, 1)
        self.addLink(s11, s12, 2, 1)
        self.addLink(s12, s17, 2, 1)
        self.addLink(s17, s18, 2, 1)
        self.addLink(s18, h1s18, 2, 1)
        self.addLink(s18, h2s18, 3, 1)
        self.addLink(s8, s10, 3, 3)
        self.addLink(s1, s18, 3, 4)

topo = MyTopo()
net = Mininet( topo=topo, switch=MultiSwitch, build=False )
for c in [ d1,d2,m1,d5,d6,d7,d10 ]:
    net.addController(c)
net.build()
net.start()
CLI( net )
net.stop()
```

A.3 Script Python do segundo experimento inter-DTSA

```
#!/usr/bin/python
```

```
"""
Create a network where different switches are connected to
different controllers.
"""

import re
from mininet.net import Mininet
from mininet.node import OVSSwitch, Controller, RemoteController
from mininet.topolib import TreeTopo
from mininet.topo import LinearTopo
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.topo import Topo

setLogLevel( 'info' )

# Seven "external" controllers
d1 = RemoteController( 'd1', ip='192.168.0.193' )
d2 = RemoteController( 'd2', ip='192.168.0.189' )
m1 = RemoteController( 'm1', ip='192.168.0.192' )
d5 = RemoteController( 'd5', ip='192.168.0.158' )
d6 = RemoteController( 'd6', ip='192.168.0.137' )
d7 = RemoteController( 'd7', ip='192.168.0.128' )
d10 = RemoteController( 'd10', ip='192.168.0.170' )

cmap = { 's1': d1, 's2': d1, 's5': d2, 's6': d2, 's8': d5,
 's9': m1, 's10': d6, 's11': d7, 's12': d7, 's17': d10, 's18': d10 }

class MultiSwitch( OVSSwitch ):
  def start( self, controllers ):
    return OVSSwitch.start( self, [ cmap[ self.name ] ] )

def toID(mac):
  return '0000' + re.sub('[:]', '', mac)

class MyTopo( Topo ):
  "Simple topology example."
```

```
def __init__( self ):
  "Create custom topo."

  # Initialize topology
  Topo.__init__( self )

  # Add hosts and switches
  h1s1 = self.addHost('h1s1')
  h1s18 = self.addHost('h1s18')
  h2s18 = self.addHost('h2s18')

  s1 = self.addSwitch('s1', dpid=toID('00:00:00:00:00:01'))
  s2 = self.addSwitch('s2', dpid=toID('00:00:00:00:00:02'))
  s5 = self.addSwitch('s5', dpid=toID('00:00:00:00:00:05'))
  s6 = self.addSwitch('s6', dpid=toID('00:00:00:00:00:06'))
  s8 = self.addSwitch('s8', dpid=toID('00:00:00:00:00:08'))
  s9 = self.addSwitch('s9', dpid=toID('00:00:00:00:00:09'))
  s10 = self.addSwitch('s10', dpid=toID('00:00:00:00:00:10'))
  s11 = self.addSwitch('s11', dpid=toID('00:00:00:00:00:11'))
  s12 = self.addSwitch('s12', dpid=toID('00:00:00:00:00:12'))
  s17 = self.addSwitch('s17', dpid=toID('00:00:00:00:00:17'))
  s18 = self.addSwitch('s18', dpid=toID('00:00:00:00:00:18'))

  # Add links
  self.addLink(s1, h1s1, 1, 1)
  self.addLink(s1, s2, 2, 1)
  self.addLink(s2, s5, 2, 1)
  self.addLink(s5, s6, 2, 1)
  self.addLink(s6, s8, 2, 1)
  self.addLink(s8, s9, 2, 1)
  self.addLink(s9, s10, 2, 1)
  self.addLink(s10, s11, 2, 1)
  self.addLink(s11, s12, 2, 1)
  self.addLink(s12, s17, 2, 1)
  self.addLink(s17, s18, 2, 1)
  self.addLink(s18, h1s18, 2, 1)
  self.addLink(s18, h2s18, 3, 1)
  self.addLink(s8, s10, 3, 3)
```

```
topo = MyTopo()
net = Mininet( topo=topo, switch=MultiSwitch, build=False )
for c in [ d1,d2,m1,d5,d6,d7,d10 ]:
    net.addController(c)
net.build()
net.start()
CLI( net )
net.stop()
```

APÊNDICE **B**

Fluxos configurados no Mininet pelos experimentos

O presente apêndice apresenta os fluxos configurados nos switches *do Mininet após o estabelecimento dos* Workspaces *em determinados passos dos experimentos. Os fluxos foram coletados para demonstrar as regras de encaminhamento presentes no Mininet após a criação dos Workspaces. O* hash *do título de um* Workspace *é inserido no campo* dl_src *das regras de encaminhamento dos* switches *Openflow.*

B.1 Fluxos no experimento intra-DTSA

Fluxos após extensão do *workspace* entre $Entity_1$ e $Entity_2$.

```
*** s1 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=3.913s, table=0, n_packets=1,
  n_bytes=17, idle_age=2, priority=0,
  dl_src=60:c5:59:0f:72:ee,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:4
 cookie=0x0, duration=3.903s, table=0, n_packets=0,
  n_bytes=0, idle_age=3, priority=0,dl_dst=60:c5:59:0f:72:ee
 actions=mod_dl_src:60:c5:59:0f:72:ee,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:4
*** s2 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s3 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s4 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s5 ----------------------------------------------------------------
```

```
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=3.972s, table=0, n_packets=0,
  n_bytes=0, idle_age=3, priority=0,dl_src=60:c5:59:0f:72:ee,
  dl_dst=ff:ff:ff:ff:ff:ff actions=output:3,output:4
 cookie=0x0, duration=3.961s, table=0, n_packets=1,
  n_bytes=17, idle_age=2, priority=0,dl_dst=60:c5:59:0f:72:ee
 actions=mod_dl_src:60:c5:59:0f:72:ee,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:3,output:4
*** s6 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s7 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s8 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s9 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4.019s, table=0, n_packets=1,
  n_bytes=17, idle_age=2, priority=0,
  dl_src=60:c5:59:0f:72:ee,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
```

Fluxos após extensão do *workspace* até $Entity_3$ e $Entity_4$.

```
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=191.013s, table=0, n_packets=6,
  n_bytes=115, idle_age=8, priority=0,
  dl_src=60:c5:59:0f:72:ee,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:4
 cookie=0x0, duration=191.003s, table=0, n_packets=2,
  n_bytes=52, idle_age=4, priority=0,dl_dst=60:c5:59:0f:72:ee
 actions=mod_dl_src:60:c5:59:0f:72:ee,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:4
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s3 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s4 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=95.981s, table=0, n_packets=5,
  n_bytes=114, idle_age=4, priority=0,
```

```
  dl_src=60:c5:59:0f:72:ee,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:2,output:3
 cookie=0x0, duration=95.963s, table=0, n_packets=2,
  n_bytes=36, idle_age=10, priority=0,
  dl_dst=60:c5:59:0f:72:ee
  actions=mod_dl_src:60:c5:59:0f:72:ee,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:2,output:3
*** s5 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=18.3s, table=0, n_packets=3,
  n_bytes=78, idle_age=4, priority=0,
  dl_src=60:c5:59:0f:72:ee,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2,output:3,output:4
 cookie=0x0, duration=18.3s, table=0, n_packets=1,
  n_bytes=18, idle_age=8, priority=0,
  dl_dst=60:c5:59:0f:72:ee
 actions=mod_dl_src:60:c5:59:0f:72:ee,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2,output:3,output:4
*** s6 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=18.329s, table=0, n_packets=3,
  n_bytes=72, idle_age=4, priority=0,
  dl_src=60:c5:59:0f:72:ee,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:3
 cookie=0x0, duration=18.32s, table=0, n_packets=1,
  n_bytes=24, idle_age=14, priority=0,
  dl_dst=60:c5:59:0f:72:ee
 actions=mod_dl_src:60:c5:59:0f:72:ee,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:3
*** s7 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s8 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s9 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=191.18s, table=0, n_packets=8,
  n_bytes=167, idle_age=4, priority=0,
  dl_src=60:c5:59:0f:72:ee,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
```

B.2 Fluxos no primeiro experimento inter-DTSA

Fluxos após extensão do *Workspace* entre h_1s_1 e h_1s_{18}.

```
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4.48s, table=0, n_packets=1,
  n_bytes=17, idle_age=2, priority=0,
  dl_src=2e:39:3d:fc:9c:96,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:3
 cookie=0x0, duration=4.486s, table=0, n_packets=0,
  n_bytes=0, idle_age=4, priority=0,
  dl_dst=2e:39:3d:fc:9c:96
 actions=mod_dl_src:2e:39:3d:fc:9c:96,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:3
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s5 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s6 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s8 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s9 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s10 -----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s11 -----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s12 -----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s17 -----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s18 -----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4.968s, table=0, n_packets=0,
  n_bytes=0, idle_age=4, priority=0,
  dl_src=2e:39:3d:fc:9c:96,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:2,output:4
```

```
 cookie=0x0, duration=4.968s, table=0, n_packets=1,
  n_bytes=17, idle_age=2, priority=0,
  dl_dst=2e:39:3d:fc:9c:96
 actions=mod_dl_src:2e:39:3d:fc:9c:96,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:2,output:4
```

Fluxos após extensão do *Workspace* até $h_2 s_{18}$.

```
*** s1 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=84.454s, table=0, n_packets=2,
  n_bytes=37, idle_age=4, priority=0,
  dl_src=2e:39:3d:fc:9c:96,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:3
 cookie=0x0, duration=84.46s, table=0, n_packets=0,
  n_bytes=0, idle_age=84, priority=0,
  dl_dst=2e:39:3d:fc:9c:96
 actions=mod_dl_src:2e:39:3d:fc:9c:96,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:3
*** s2 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s5 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s6 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s8 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s9 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s10 ---------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s11 ---------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s12 ---------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s17 ---------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s18 ---------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.69s, table=0, n_packets=0,
  n_bytes=0, idle_age=6, priority=0,
```

```
  dl_src=2e:39:3d:fc:9c:96,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:2,output:3,output:4
 cookie=0x0, duration=6.69s, table=0, n_packets=1,
  n_bytes=20, idle_age=4, priority=0,
  dl_dst=2e:39:3d:fc:9c:96
 actions=mod_dl_src:2e:39:3d:fc:9c:96,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:2,output:3,output:4
```

B.3 Fluxos no segundo experimento inter-DTSA

Fluxos após extensão do *Workspace* entre h_1s_1 e h_1s_{18}.

```
*** s1 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.03s, table=0, n_packets=1,
  n_bytes=20, idle_age=3, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=6.028s, table=0, n_packets=0,
  n_bytes=0, idle_age=6, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s2 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.044s, table=0, n_packets=1,
  n_bytes=20, idle_age=3, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=6.038s, table=0, n_packets=0,
  n_bytes=0, idle_age=6, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s5 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=5.936s, table=0, n_packets=1,
  n_bytes=20, idle_age=3, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
```

```
 cookie=0x0, duration=5.934s, table=0, n_packets=0,
  n_bytes=0, idle_age=5, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s6 ------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=5.945s, table=0, n_packets=1,
  n_bytes=20, idle_age=3, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=5.939s, table=0, n_packets=0,
  n_bytes=0, idle_age=5, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s8 ------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.04s, table=0, n_packets=1,
  n_bytes=20, idle_age=3, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:3
 cookie=0x0, duration=6.045s, table=0, n_packets=0,
  n_bytes=0, idle_age=6, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:3
*** s9 ------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s10 -----------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.159s, table=0, n_packets=1,
  n_bytes=20, idle_age=3, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:2,output:3
 cookie=0x0, duration=6.165s, table=0, n_packets=0,
  n_bytes=0, idle_age=6, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
```

```
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:2,output:3
*** s11 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.209s, table=0, n_packets=1,
  n_bytes=20, idle_age=3, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=6.207s, table=0, n_packets=0,
  n_bytes=0, idle_age=6, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s12 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.224s, table=0, n_packets=1,
  n_bytes=20, idle_age=3, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=6.217s, table=0, n_packets=0,
  n_bytes=0, idle_age=6, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s17 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.298s, table=0, n_packets=1,
  n_bytes=20, idle_age=3, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=6.292s, table=0, n_packets=0,
  n_bytes=0, idle_age=6, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s18 ----------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.326s, table=0, n_packets=0,
  n_bytes=0, idle_age=6, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
```

```
 actions=output:1,output:2
 cookie=0x0, duration=6.303s, table=0, n_packets=1,
  n_bytes=20, idle_age=3, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
```

Fluxos após extensão do *Workspace* até $h_1 s_{18}$.

```
*** s1 ------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=493.405s, table=0, n_packets=3,
  n_bytes=54, idle_age=186, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=493.403s, table=0, n_packets=1,
  n_bytes=16, idle_age=345, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s2 ------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=493.414s, table=0, n_packets=4,
  n_bytes=70, idle_age=186, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=493.408s, table=0, n_packets=0,
  n_bytes=0, idle_age=493, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s5 ------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=493.306s, table=0, n_packets=4,
  n_bytes=70, idle_age=186, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=493.304s, table=0, n_packets=0,
  n_bytes=0, idle_age=493, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
```

```
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s6 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=493.313s, table=0, n_packets=4,
  n_bytes=70, idle_age=186, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=493.307s, table=0, n_packets=0,
  n_bytes=0, idle_age=493, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s8 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=493.408s, table=0, n_packets=4,
  n_bytes=70, idle_age=186, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:3
 cookie=0x0, duration=493.413s, table=0, n_packets=0,
  n_bytes=0, idle_age=493, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:3
*** s9 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s10 -----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=493.528s, table=0, n_packets=4,
  n_bytes=70, idle_age=186, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:2,output:3
 cookie=0x0, duration=493.534s, table=0, n_packets=0,
  n_bytes=0, idle_age=493, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:2,output:3
*** s11 -----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=493.574s, table=0, n_packets=4,
```

```
  n_bytes=70, idle_age=186, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=493.572s, table=0, n_packets=0,
  n_bytes=0, idle_age=493, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s12 ------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=493.583s, table=0, n_packets=4,
  n_bytes=70, idle_age=186, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=493.576s, table=0, n_packets=0,
  n_bytes=0, idle_age=493, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s17 ------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=192.164s, table=0, n_packets=1,
  n_bytes=17, idle_age=186, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2
 cookie=0x0, duration=192.161s, table=0, n_packets=0,
  n_bytes=0, idle_age=192, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
 actions=mod_dl_src:7c:2e:cd:07:f1:55,
  mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2
*** s18 ------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=192.172s, table=0, n_packets=0,
  n_bytes=0, idle_age=192, priority=0,
  dl_src=7c:2e:cd:07:f1:55,dl_dst=ff:ff:ff:ff:ff:ff
 actions=output:1,output:2,output:3
 cookie=0x0, duration=192.164s, table=0, n_packets=1,
  n_bytes=17, idle_age=186, priority=0,
  dl_dst=7c:2e:cd:07:f1:55
```

```
actions=mod_dl_src:7c:2e:cd:07:f1:55,
 mod_dl_dst:ff:ff:ff:ff:ff:ff,output:1,output:2,output:3
```