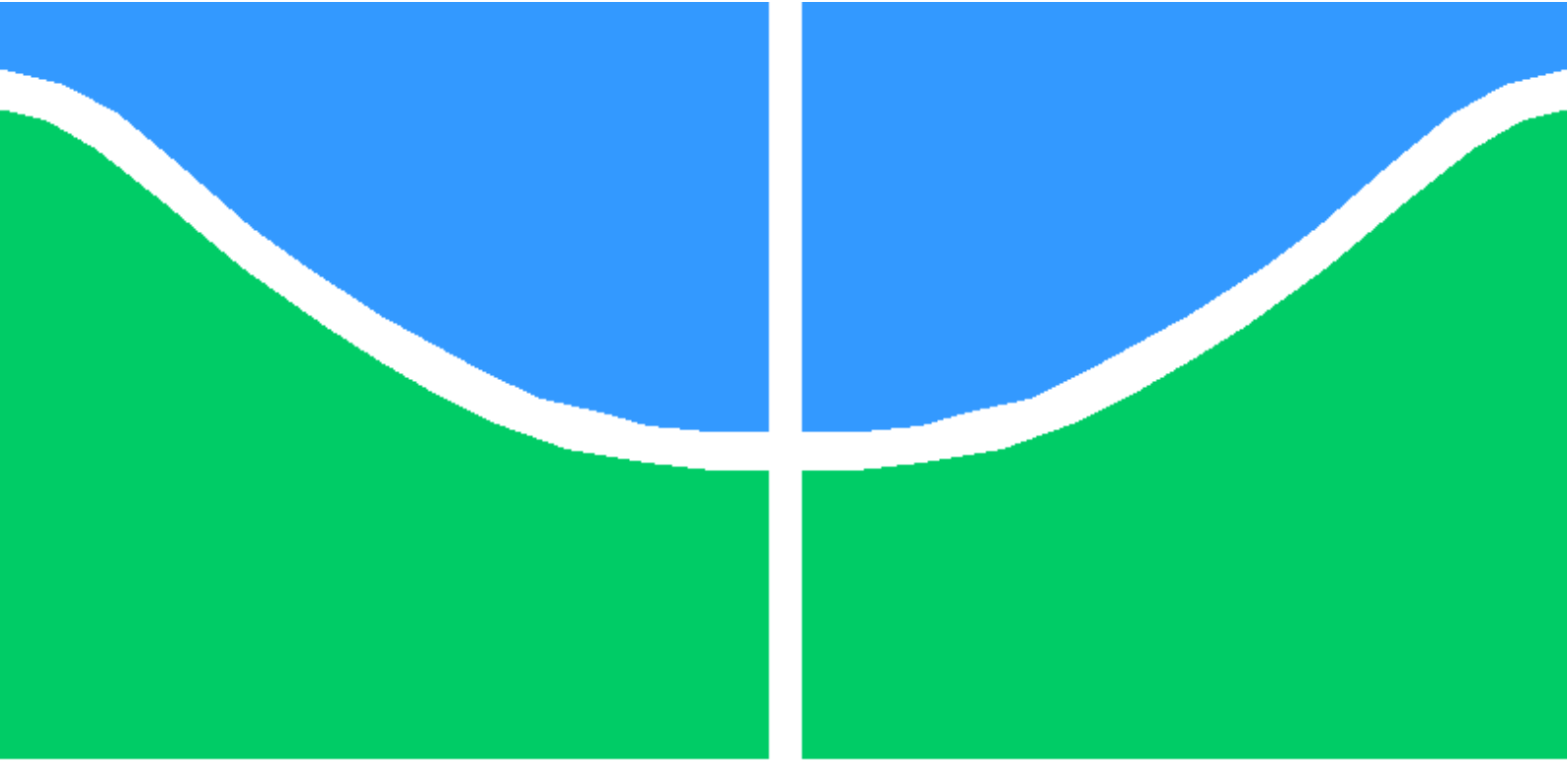


DISSERTAÇÃO DE MESTRADO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ANÁLISE DE TÉCNICAS DE OTIMIZAÇÃO MULTIOBJETIVO PARA O POSICIONAMENTO DE CONTROLADORES EM REDES SDN

ANA CAROLINA DE OLIVEIRA CHRISTÓFARO

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

Christófaro, Ana Carolina de Oliveira
CC556a Análise de Técnicas de Otimização Multiobjetivo para o posicionamento de controladores em redes SDN / Ana Carolina de Oliveira Christófaro; orientador Daniel Guerreiro e Silva; co-orientador Marcelo Menezes de Carvalho. -- Brasília, 2018.
116 p.

Dissertação (Mestrado - Mestrado em Engenharia Elétrica) - Universidade de Brasília, 2018.

1. redes. 2. SDN. 3. NSGA-II. 4. PSA. I. Guerreiro e Silva, Daniel, orient. II. Menezes de Carvalho, Marcelo, co orient. III. Título.

ANA CAROLINA DE OLIVEIRA CHRISTÓFARO

ANÁLISE DE TÉCNICAS DE OTIMIZAÇÃO MULTIOBJETIVO PARA O POSICIONAMENTO DE CONTROLADORES EM REDES SDN

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica.

Orientador: Prof. Daniel Guerreiro e Silva
Coorientador: Prof. Marcelo Menezes de Carvalho

Brasília, Março de 2018

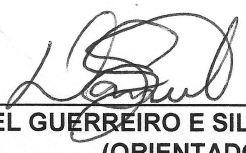
UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ANÁLISE DE TÉCNICAS DE OTIMIZAÇÃO MULTIOBJETIVO PARA O POSICIONAMENTO DE CONTROLADORES EM REDES SDN

ANA CAROLINA DE OLIVEIRA CHRISTOFARO

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:



DANIEL GUERREIRO E SILVA, Dr., ENE/UNB
(ORIENTADOR)

ANDRÉ COSTA DRUMMOND, Dr., CIC/UNB
(EXAMINADOR INTERNO)

GUILHERME PALERMO COELHO, DR. FT/UNICAMP
(EXAMINADOR EXTERNO)

Brasília, 07 de março de 2018.

Dedicatória

Dedico este trabalho aos meus pais, Marília e Ângelo.

Ana Carolina de Oliveira Christófaro

Agradecimentos

Agradeço aos Professores Daniel e Marcelo pelo apoio, dedicação, incentivo e compreensão.

Ana Carolina de Oliveira Christófaro

RESUMO

O conceito de redes definidas por *software* introduz uma solução que, por meio de uma arquitetura logicamente centralizada, apresenta potencial para aperfeiçoar o uso de recursos de rede. O crescente interesse na indústria vem levantando questões associadas, por exemplo, ao impacto do posicionamento de um ou mais controladores no desempenho da rede. Dentro deste contexto, este trabalho avalia o posicionamento do controlador a partir de medidas de desempenho individuais como latência, balanceamento de carga e taxa de transmissão. Considerando que o posicionamento de um conjunto de controladores dentro de uma dada topologia, de forma a satisfazer certas medidas de desempenho, constitui-se em um problema de otimização multiobjetivo, este trabalho também apresenta um estudo comparativo sobre a utilização das metaheurísticas *Pareto Simulated Annealing* (PSA) e *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) como uma alternativa para prover uma boa relação entre precisão e tempo de processamento, resultando em menor utilização de recursos computacionais, com vantagem para o NSGA-II em termos de convergência e grau de exploração do espaço de busca. Tais características tornam viável a solução de problemas reais de posicionamento de controladores em redes SDN.

Palavras-chave: redes, SDN, NSGA-II, PSA.

ABSTRACT

Software defined networking introduces a solution that, through a logically centralized architecture, has potential to improve the use of network resources. The growing interest of the industry has raised issues associated, for example, with the impact of one or more controllers' position on network performance. In this context, this work presents an evaluation of the controller positioning from individual performance measures such as latency, load balancing and transmission rate. Considering that the positioning of a set of controllers within a given topology, in order to satisfy certain performance measures, constitutes a multiobjective optimization problem, this work also presents a solution based on Pareto Simulated Annealing (PSA) and Non-dominated Sorting Genetic Algorithm II (NSGA-II) metaheuristics for the SDN controllers positioning problem, including a comparative analysis between PSA and NSGA-II, with different network performance measures. The results yield good results for both techniques, in terms of accuracy and processing time, resulting in a lower utilization of computational resources, with an advantage to NSGA-II algorithm in terms of convergence and exploration degree of the search space. Such features make feasible the solution of real controller positioning problems in SDN networks.

Keywords: network, SDN, NSGA-II, PSA.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

AARS

E	Conjunto de conexões disponíveis
$F1$	Fronteira de Pareto
G	Grafo da rede
J	Número total de objetivos
k	Número de controladores
n	Número total de nós na rede
S	Conjunto de soluções candidatas
V	Conjunto de nós

PSA

E	Conjunto de conexões disponíveis
F	Fronteira
f_j	Função objetivo j
G	Grafo da rede
i	Número total de níveis de temperatura percorridos
j	Identificador do objetivo
J	Número total de objetivos
k	Número de controladores
m	Número de iterações por nível de temperatura
n	Número total de nós na rede
N	Tamanho do conjunto de soluções
P	Probabilidade de aceitação
q	Solução vizinha $\in Q$
Q	Conjunto de soluções vizinhas
s	Solução candidata $\in S$
S	Conjunto de soluções candidatas
T	Temperatura
V	Conjunto de nós
α	Fator de multiplicação de pesos
λ_j	Peso da função objetivo j da solução s

Λ	Vetor de pesos da solução s
ρ	Taxa de resfriamento

NSGA-II

E	Conjunto de conexões disponíveis
F	Fronteira
G	Grafo da rede
i	Indicador de iteração
k	Número de controladores
m	Número total de iterações
n	Número total de nós na rede
N	Tamanho do conjunto de soluções
n_c	Número total de recombinações por iteração
n_m	Número total de mutações por iteração
p	Solução obtida por torneio
q	Solução gerada $\in Q$
Q	Conjunto de soluções geradas
R	Conjunto de soluções geradoras e geradas
s	Solução geradora $\in S$
S	Conjunto de soluções geradoras
SC	Conjunto de soluções obtidas por recombinação
SM	Conjunto de soluções obtidas por mutação
t	Identificador de iteração
t_c	Taxa de recombinação
t_m	Taxa de mutação
V	Conjunto de nós
$\prec_n$	Operador de comparação de aglomeração

Fast Non-Dominated Sorting

C_s	Conjunto de soluções dominadas por s
F	Fronteira
i	Indicador de iteração
n_s	Número de soluções que dominam a solução s
n_q	Número de soluções que dominam a solução q
q	Solução $\in S$ do conjunto C_s
Q	Fronteira iterativa
R	Conjunto de soluções geradoras e geradas
s	Solução $\in S$
S	Conjunto de soluções

Distância de Aglomeração

F	Fronteira
f_j^{max}	Valor máximo da função objetivo j
f_j^{min}	Valor mínimo da função objetivo j
j	Identificador do objetivo
J	Número total de objetivos
l	Tamanho da fronteira de Pareto F
s	Solução avaliada

Medidas de desempenho de rede

B	Matriz de taxa de transmissão
b	Maior taxa de transmissão entre o nó x e o nó y
c_p	Custo dos nós associados ao controlador p
D	Matriz de latências
$d_{x,y}$	Menor latência entre o nó x e o nó y
E	Conjunto de conexões disponíveis
f_j	Função objetivo j
G	Grafo da rede
j	Identificador do objetivo
J	Número total de objetivos
k	Número de controladores
n	Número total de nós na rede
n_p	Número de nós associados ao controlador p
p	Controlador examinado
v	Nó examinado
V	Conjunto de nós
w_p	Peso do nó v
π_{balanc}	Diferença em valores de n_p entre os controladores com menor e maior número de nós associados
π_{avg}^{BwC2C}	Média sobre as taxas de transmissão de todos os pares de controladores
π_{avg}^{BwN2C}	Média sobre as taxas de transmissão de todos os nós para o controlador mais próximo
π_{max}^{BwC2C}	Mínima taxa de transmissão entre todos os pares de controladores
π_{max}^{BwN2C}	Mínima taxa de transmissão entre todos os nós para o controlador mais próximo
π_{avg}^{LatC2C}	Média sobre as latências de todos os pares de controladores
π_{avg}^{LatN2C}	Média sobre as latências de todos os nós para o controlador mais próximo
π_{max}^{LatC2C}	Máxima latência entre todos os pares de controladores
π_{max}^{LatN2C}	Máxima latência entre todos os nós para o controlador mais próximo
$\mathcal{P}$	Conjunto de posições para conectar os k controladores

Siglas e Acrônimos

AARS	Automatic Accumulated Ranking Strategy
API	Application Programming Interface
BID	Bridge ID
GD	Generational Distance
GLM	Graphical Modelling Language
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IRTF	Internet Research Task Force
ITZ	Internet Topology Zoo
LLDP	Link-Layer Discovery Protocol
MAC	Media Access Control
MOCO	Multi-Objective Combinatorial Optimization
NSGA	Non-dominated Sorting Genetic Algorithm
ONRC	Open Networking Research Center
OVS	Open vSwitch
PCKM	Pareto Capacitated k-Medoids
PLC	Programmable Logic Controller
POCO	Pareto-based Optimal Controller Placement
PSA	Pareto Simulated Annealing
QoS	Quality of Service
REST	Representational State Transfer
RTT	Round-Trip Time
SDN	Software Defined Network
SL	Scalar Linear SNMP
SNMP	Simple Network Management Protocol
STP	Spanning Tree Protocol

Capítulo 1

Introdução

O paradigma de redes definidas por *software* (SDN, do inglês *Software Defined Networking*) introduz uma solução que, por meio de uma visão global da rede, apresenta potencial para aperfeiçoar o uso de recursos de rede, através da otimização do encaminhamento de pacotes. Para tanto, as redes SDN se baseiam em uma arquitetura logicamente centralizada, que separa o plano de controle do plano de dados.

Diferentemente das redes convencionais onde, para cada dispositivo de rede, protocolos e conjuntos de regras são implementados para controlar e monitorar o fluxo de dados, nas redes SDN as funções de controle dos dispositivos de rede são movidas para um controlador externo dedicado, implementado em *software* (NUNES et al., 2014a). Desta forma, no plano de controle, um controlador logicamente centralizado otimiza as decisões de encaminhamento e separação de tráfego. No plano de dados, o encaminhamento de pacotes é realizado pelos dispositivos de rede de acordo com os comportamentos para manipulação de pacotes ditados pelo plano de controle. Essa separação permite a construção dinâmica e eficiente de redes mais simples através dos controladores.

A comunicação entre o plano de controle e o plano de dados é realizada por interfaces de programação de aplicações (API, do inglês *Application Programming Interfaces*) (JARSCHEL et al., 2014) implementadas por protocolos como o OpenFlow (MCKEOWN et al., 2008). O OpenFlow é um protocolo aberto que permite que um controlador gerencie *switches* e dite o comportamento dos mesmos.

Para minimizar o tempo de resposta do plano de controle para o plano de dados e prover aspectos como escabilidade e resiliência, ao manter o controle de rede logicamente centralizado, Tootoonchian e Ganjali (2010) propõem que a tomada de decisão seja realizada em controladores locais com base no conceito de HyperFlow. O HyperFlow introduz um plano de controle distribuído capaz de particionar a rede em vários domínios, cada domínio com um controlador dedicado. Um grande desafio desse novo paradigma envolve a avaliação do impacto da posição de cada controlador nas medidas de desempenho da rede, e a definição da quantidade de controladores requeridos para operação confiável e resiliente da rede, no que se refere a capacidade de se adequar às mudanças nas suas condições de operação e de superar crises.

O problema de posicionamento dos controladores em redes SDN, que pode considerar a existên-

cia de um ou mais controladores, foi introduzido pela primeira vez por Heller, Sherwood e McKeown (2012), que defendem a sua relevância em um estudo quantitativo do impacto do posicionamento dos controladores na latência de propagação entre o plano de controle e o plano de dados. Para isso, os autores propõem uma análise exaustiva de todas as posições possíveis, a fim de analisar as características das soluções ótimas. Este trabalho foi reforçado por Phemius e Bouet (2013) através de uma análise do impacto da latência no consumo de largura de banda e na capacidade reativa da rede. Para redes habilitadas com múltiplos controladores, Yao et al. (2014) investigam ainda o impacto do posicionamento na carga de cada controlador.

No entanto, em ambientes reais, as medidas de desempenho são concorrentes, de forma que não há uma solução que satisfaça simultaneamente objetivos de otimização como latência e balanceamento. Por este motivo, propõe-se que o problema de posicionamento dos controladores seja tratado como um problema de otimização combinatória multiobjetivo (MOCO, do inglês *Multi-Objective Combinatorial Optimization*), onde a solução mais apropriada seja selecionada pelo tomador de decisão de acordo com políticas por ele definidas a partir de um conjunto de soluções ótimas, correspondentes a compromissos entre os diferentes objetivos. Este conjunto é conhecido como conjunto ótimo de Pareto. Movendo-se de uma solução para outra no conjunto de Pareto, um objetivo necessariamente é sacrificado para que se obtenha ganho em outro. Neste contexto, Hock et al. (2013) investigam uma versão estendida do problema de posicionamento dos controladores que lida com a otimização de múltiplos objetivos, como considerações de resiliência e latência entre controladores, através da avaliação exaustiva de todo o espaço de variáveis.

Em problemas combinatórios pequenos e médios, avaliar de maneira exaustiva todo o espaço de soluções e encontrar o conjunto ótimo de Pareto pode ser possível com um esforço computacional razoável. Entretanto, para redes de larga escala, a avaliação exaustiva necessita de uma quantidade considerável de esforço computacional e uso de memória. É neste contexto que se torna necessária a busca por uma solução computacional rápida.

Lange et al. (2015b) mostram que o problema de posicionamento dos controladores para redes de larga escala pode ser otimizado via soluções heurísticas capazes de apresentar soluções com boa relação entre precisão e tempo de processamento. Para tanto, eles implementam a metaheurística *Pareto Simulated Annealing* (PSA) (CZYZAK; JASZKIEWICZ, 1998), que produz resultados dentro de dezenas de segundos para instâncias de problemas cuja solução exaustiva leva dezenas de minutos, com uma média de erro de 2%. O estudo proposto ainda disponibiliza um ambiente em MATLAB que permite calcular o posicionamento ótimo de um ou mais controladores por meio de técnicas exaustivas e heurísticas, com respeito a diferentes medidas de desempenho. Este ambiente, conhecido como POCO, apresenta uma interface gráfica com uma clara ilustração dos compromissos entre as medidas concorrentes, além de permitir a extensão de funcionalidades como a implementação de novos mecanismos para o cálculo e a análise do posicionamento dos controladores.

Em um segundo trabalho, Lange et al. (2015a) propõem a utilização do algoritmo Pareto Capacitated k-Medoids (PCKM), que combina algoritmos de teoria dos grafos para construir uma aproximação da fronteira de Pareto com respeito a duas funções objetivo. Já a utilização de

algoritmos genéticos foi introduzida por Jalili et al. (2015) através da implementação e avaliação do *Nondominated Sorting Genetic Algorithm II* (NSGA-II) (DEB et al., 2002), onde concluiu-se que o NSGA-II é capaz de prover uma boa aproximação da solução ideal para o problema de posicionamento dos controladores.

Com a crescente popularidade das redes SDN, um número considerável de trabalhos já foram realizados no que diz respeito ao desenvolvimento de funcionalidades do plano de controle para o tratamento dos pacotes no plano de dados. Contudo, o estudo de redes habilitadas com OpenFlow com o objetivo de explorar o impacto do posicionamento de um ou mais controladores no desempenho de redes reais ou emuladas ainda merece mais investigação. Até o momento, a maioria dos trabalhos utilizam o Mininet como emulador de redes SDN e se baseiam na utilização de um controlador remoto, cuja operação independe da infraestrutura da rede emulada. Sendo assim, existe ainda a necessidade de se investigar a operação do controlador como parte da infraestrutura emulada pelo Mininet, de forma que o controlador seja submetido às condições da rede, modo este conhecido como *in-band*.

Motivado por este fato, este trabalho apresenta um estudo do desempenho de rede em condições estáticas para diferentes posições do controlador a partir da emulação no Mininet de uma rede SDN habilitada com OpenFlow e um controlador *in-band*.

Adicionalmente aos trabalhos já desenvolvidos para o POCO, onde o problema de posicionamento dos controladores lida com a otimização de objetivos relacionados à consideração de resiliência e latência, e tendo como base os trabalhos desenvolvidos por Yao et al. (2014) e Phemius e Bouet (2013), este trabalho propõe a implementação de novos objetivos relacionados à consideração de taxa de transmissão.

Dentro deste contexto, este trabalho também explora o algoritmo genético multiobjetivo NSGA-II como uma heurística para o problema de posicionamento, similarmente ao realizado por Jalili et al. (2015), mas adicionalmente discute os impactos na precisão e no esforço computacional necessário, e realiza uma comparação de resultados entre esta reconhecida metaheurística evolutiva e a metaheurística PSA dentro da ferramenta POCO.

1.1 Objetivos

1.1.1 Objetivo Geral

Este trabalho tem como objetivo geral explorar o algoritmo genético multiobjetivo NSGA-II como a abordagem heurística para o problema de posicionamento dos controladores em redes SDN, ao mesmo tempo que realiza uma comparação de resultados entre esta reconhecida metaheurística evolutiva e os resultados obtidos com a utilização do algoritmo PSA na ferramenta POCO.

1.1.2 Objetivos Específicos

Os objetivos específicos a serem atingidos são os seguintes:

- Analisar o problema de posicionamento ótimo do controlador em redes SDN por meio de uma rede estática habilitada com OpenFlow emulada em laboratório;
- Desenvolver e implementar uma nova medida de desempenho de rede relacionada à taxa de transmissão como função objetivo para análise do posicionamento ótimo dos controladores;
- Desenvolver e implementar o algoritmo genético multiobjetivo NSGA-II como uma metaheurística para o problema de posicionamento na ferramenta POCO;
- Contribuir para a extensão das funcionalidades disponíveis no POCO;
- Contribuir para o desenvolvimento e o estudo de redes SDN.

1.2 Apresentação do manuscrito

Para concretização deste trabalho, neste primeiro Capítulo foi realizada uma rápida contextualização e foram determinados objetivos a serem alcançados.

O restante deste trabalho é estruturado de forma que:

O Capítulo 2 apresenta a fundamentação teórica para as redes SDN, detalhando a sua arquitetura e os seus componentes, que incluem o OpenFlow e o HyperFlow.

O Capítulo 3 apresenta a fundamentação teórica para o problema de otimização multiobjetivo, incluindo a definição do problema e abordando-se conhecimentos relacionados à estratégia de solução: métodos exaustivo e metaheurísticas PSA e NSGA-II.

O Capítulo 4 define a forma como o trabalho foi desenvolvido. Em um primeiro momento, apresentam-se detalhes sobre a implementação das medidas de desempenho de rede a serem utilizadas como funções objetivo para análise do posicionamento ótimo dos controladores e sobre a implementação dos métodos exaustivos e das metaheurísticas PSA e NSGA-II na ferramenta POCO. Em um segundo momento, o capítulo apresenta o ambiente de emulação de rede e os mecanismos utilizados para demonstrar o problema de posicionamento do controlador em redes SDN por meio de uma rede habilitada com OpenFlow emulada em laboratório.

No Capítulo 5, são apresentados e discutidos os resultados experimentais obtidos, de forma comparativa entre os métodos exaustivo e metaheurísticas PSA e NSGA-II.

Por fim, o Capítulo 6 tece considerações finais sobre o trabalho, e os Anexos apresentam os códigos e os dados utilizados nas etapas de desenvolvimento e experimentação.

Capítulo 2

Redes Definidas por Software

2.1 Introdução

As redes de comunicação compreendem um conjunto de dispositivos que, por meio de protocolos e regras comuns, são capazes de compartilhar dados e recursos através de uma infraestrutura de conexão, que pode ser por fio de cobre, fibra ótica, ondas de rádio e também via satélite. Suas funcionalidades podem ser separadas em três planos:

- *Plano de dados*: compreende os dispositivos de rede responsáveis pelo encaminhamento efetivo dos dados.
- *Plano de controle*: compreende os protocolos e conjuntos de regras implementados para popular as tabelas de encaminhamento dos dispositivos que compõem o plano de dados.
- *Plano de gerenciamento*: inclui ferramentas baseadas em serviços como SNMP (do inglês, *Simple Network Management Protocol*) utilizadas para monitorar e configurar remotamente funcionalidades de controle.

Em redes convencionais, o plano de dados e plano de controle são embutidos em um mesmo dispositivo de rede, o que permite uma arquitetura altamente descentralizada capaz de garantir a resiliência da rede, no que se refere a capacidade de se adequar a mudanças nas suas condições de operação e de superar crises, além de ser bastante efetiva em termos de desempenho. Apesar disso, as redes convencionais são rígidas e complexas de se gerenciar e controlar. Um erro de configuração simples é capaz de comprometer a operação de toda a Internet por horas (BARRETT; HAAR; WHITESTONE, 1997; FEAMSTER; BALAKRISHNAN, 2005).

Para contornar tais limitações, os administradores de rede buscam por diferentes soluções de gerenciamento, em sua maioria proprietárias, e são obrigados a manter equipes especializadas, o que eleva os custos operacional e de capital para construir e manter a infraestrutura de uma rede convencional.

Nesse contexto, surgiu o conceito de redes definidas por *software* (SDN, do inglês *Software Defined Networking*), que propõe uma arquitetura de rede que separa o plano de controle do plano

de dados. Diferente das redes convencionais onde, para cada dispositivo de rede, protocolos e conjuntos de regras são implementados para controlar e monitorar o fluxo de dados, nas redes SDN as funções de controle dos dispositivos de rede são logicamente centralizadas em um ou mais controladores externos dedicados, implementados em *software* (NUNES et al., 2014a).

Os controladores compreendem o plano de controle e, por meio de uma visão global da rede, são capazes de otimizar as decisões de encaminhamento e separação de tráfego. No plano de dados, o encaminhamento de pacotes é realizado pelos dispositivos de rede de acordo com os comportamentos para manipulação de pacotes ditados pelo plano de controle. Essa separação permite a construção dinâmica e eficiente de redes mais simples, e a centralização da lógica do controle em controladores externos com conhecimento global da rede simplifica o desenvolvimento de funções mais sofisticadas de redes, serviços e aplicações.

Com SDN, o gerenciamento se torna mais simples e serviços de *middleboxes*, como roteamento, balanceamento de carga, *multicasting*, segurança, controle de acesso, gerenciamento de largura de banda, qualidade de serviço (QoS, do inglês *Quality of Service*), entre outros, podem ser entregues de maneira otimizada, como aplicações do controlador SDN.

A Fig. 2.1 ilustra a diferença entre as redes convencionais e as redes SDN. Em redes SDN, a separação do *hardware* de encaminhamento do controle lógico da rede permite o desenvolvimento de novos protocolos e aplicações, a virtualização e o gerenciamento da rede, além de consolidar vários serviços de *middleboxes* através de um *software* de controle. Em vez de aplicar políticas e executar protocolos sobre uma variedade de dispositivos, a rede é reduzida ao simples encaminhamento em *hardware* e um ou mais controladores de rede realizam a tomada de decisão (NUNES et al., 2014b).

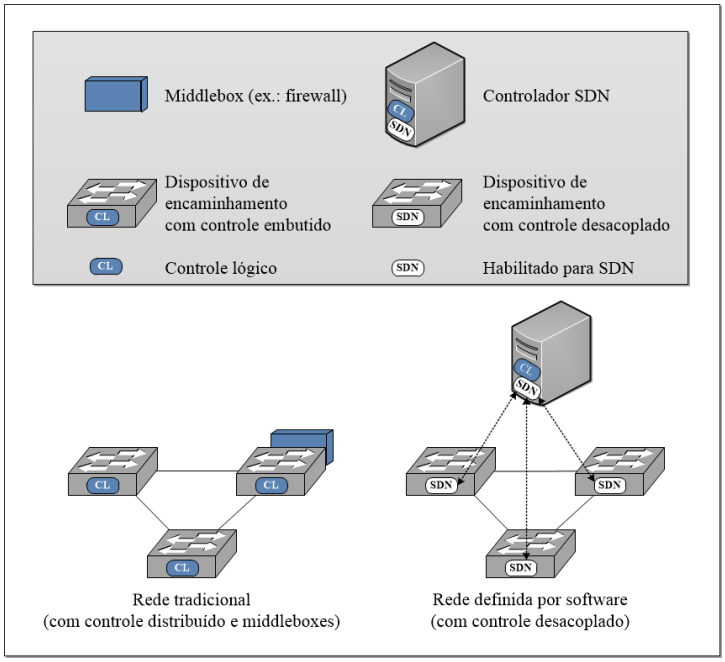


Figura 2.1: A arquitetura SDN separa o controle lógico do *hardware* de encaminhamento e habilita a consolidação de *middleboxes*, políticas mais simples de gestão e novas funcionalidades. As linhas sólidas definem os enlaces do plano de dados e as linhas pontilhadas definem os enlaces do plano de controle. Fonte: Nunes et al. (2014b).

A comunicação entre o plano de controle e o plano de dados é realizada por interfaces de programação de aplicações (API, do inglês *Application Programming Interfaces*) (JARSCHEL et al., 2014) implementadas por protocolos como o OpenFlow (MCKEOWN et al., 2008). O OpenFlow é um protocolo aberto que permite que um controlador gerencie dispositivos de rede e dite o comportamento dos mesmos. A implementação de padrões abertos simplifica o projeto e operação da rede, dado que as instruções são fornecidas pelos controladores, ao invés de protocolos e dispositivos proprietários.

Através das redes SDN, os administradores de rede são capazes de controlar o fluxo de dados e alterar características dos dispositivos de encaminhamento através de uma aplicação de controle implementada em *software*, sem a necessidade de lidar individualmente com cada dispositivo de rede. Desta forma, políticas de rede podem ser atualizadas para reagir às características de fluxo em tempo real. É mais simples e menos suscetível a erros modificar as políticas de rede através de linguagens de alto nível e componentes de *software*, quando comparado às configurações de baixo nível específicas de cada equipamento. Além disso, a aplicação de controle pode automaticamente reagir à mudanças inesperadas na rede, enquanto mantém as políticas de alto nível intactas (KREUTZ et al., 2015).

2.2 Arquitetura e componentes

Kreutz et al. (2015) definem que a arquitetura de redes SDN é composta por quatro pilares:

- O plano de controle e o plano de dados são desacoplados, de forma que as funcionalidades de controle são removidas dos dispositivos de rede que passam a atuar como simples dispositivos de encaminhamento de pacotes;
- A lógica de controle é centralizada para uma entidade externa, o controlador;
- As decisões de encaminhamento são baseadas em fluxo, ou seja, uma sequência de pacotes entre uma mesma origem e um mesmo destino;
- A rede é programável através de aplicações de *software* rodando nos controladores que interagem com os dispositivos do plano de dados.

Para atender a esses requisitos, a arquitetura de redes SDN é composta por camadas que exercem funções específicas em cada plano, conforme apresentado na Fig. 2.2.

O plano de dados compreende as camadas de infraestrutura de redes e interface *southbound*. A camada de infraestrutura de rede é responsável pela interconexão entre os dispositivos de encaminhamento (dispositivos de *hardware* ou dispositivos do plano de dados baseado em *software*) que desempenham um conjunto de instruções bem definidas, tais como regras de fluxo usadas para aplicar ações nos pacotes recebidos como, por exemplo, encaminhar o pacote por uma porta específica, descartar o pacote, encaminhar o pacote para o controlador e reescrever o cabeçalho. Este conjunto de instruções é definido por APIs que fazem parte da camada interface *southbound*. Desta

forma, a camada de interface *southbound* define o protocolo de comunicação entre os dispositivos de encaminhamento e os componentes do plano de controle, sendo responsável por formalizar como os componentes do plano de controle e de dados interagem. Um protocolo bastante conhecido e aceito é o OpenFlow (MCKEOWN et al., 2008).

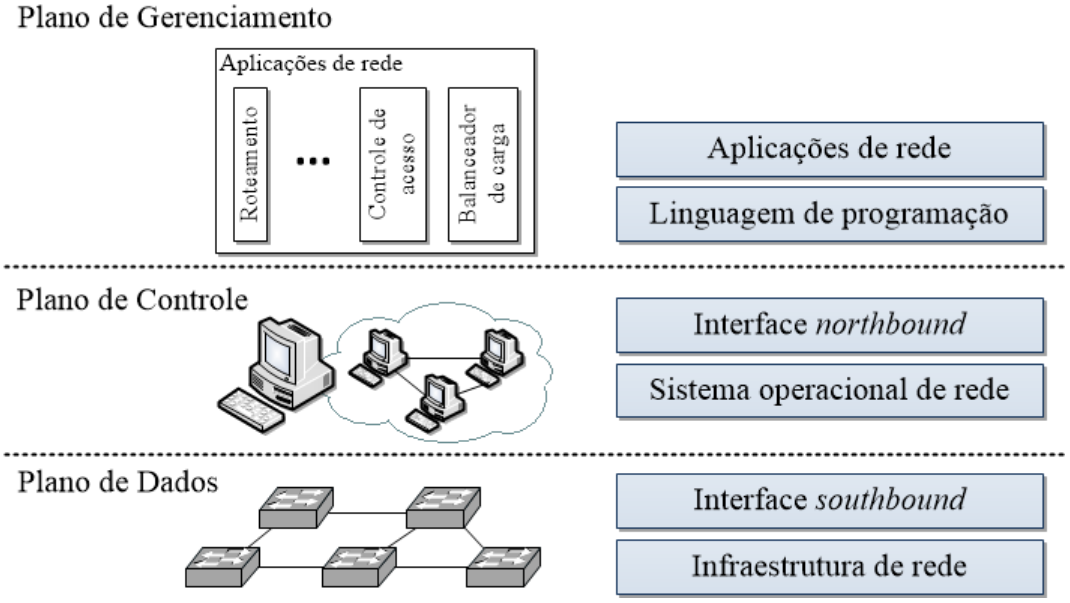


Figura 2.2: Arquitetura de redes SDN em camadas. Fonte: Adaptado de Kreutz et al. (2015).

O plano de controle, por sua vez, é composto basicamente pelas camadas de sistema operacional de rede e interface *northbound*. A camada de sistema operacional de rede compreende os controladores que realizam o controle lógico da rede com base nas políticas definidas pelo administrador. Similar aos sistemas operacionais tradicionais, a plataforma de controle abstrai detalhes de conexão e interação com os dispositivos de encaminhamento, gerencia o acesso simultâneo aos recursos subjacentes e provê mecanismos de proteção e segurança. A camada de interface *northbound* compreende as APIs oferecidas pelos controladores para desenvolvedores de aplicação. Tipicamente, a interface *northbound* abstrai o conjunto de instruções de baixo nível usadas pelas interfaces *southbound* para programar os dispositivos de encaminhamento. Apesar da interface *southbound* já possuir uma proposta bastante aceita, a interface *northbound* ainda é um problema em aberto (KREUTZ et al., 2015).

Por fim, o plano de gerenciamento envolve o conjunto de aplicações que alavancam as funções oferecidas pela interface *northbound* para implementar controle de rede e lógica de operação. Para tanto, ele é composto pelas camadas de aplicação de redes e linguagens de programação. As aplicações de rede, tais como roteamento, balanceamento de carga, *multicasting*, segurança, controle de acesso, gerenciamente de largura de banda, QoS, entre outros, definem as políticas que serão traduzidas por meio de linguagens de programação específicas para programar o comportamento dos dispositivos de encaminhamento.

Dentre as arquiteturas de redes SDN propostas, a mais amplamente aceita é a arquitetura OpenFlow (MCKEOWN et al., 2008). Para prover aspectos como escalabilidade e resiliência, enquanto se mantém os benefícios do controle centralizado, é possível distribuir fisicamente o plano

de controle entre vários controladores com base no conceito de HyperFlow (TOOTOONCHIAN; GANJALI, 2010). Detalhes sobre o OpenFlow e o HyperFlow são apresentados a seguir.

2.2.1 OpenFlow

O OpenFlow, proposto inicialmente pela Universidade de Stanford em 2008 (MCKEOWN et al., 2008), é a arquitetura mais conhecida e aceita para redes SDN. Apesar de ser relativamente novo, o Openflow já possui um conjunto de especificações bem definidas.

Para promover o conceito de SDN e padronizar o protocolo OpenFlow, um grupo formado por operadores de rede, provedores de serviços e fornecedores deu origem à Open Networking Foundation (2011), uma organização orientada para o setor industrial. No campo da educação, também foi criado o Open Networking Research Center - ONRC (2014), além de outros esforços desempenhados pelo IETF (do inglês, *Internet Engineering Task Force*) e IRTF (do inglês, *Internet Research Task Force*).

A arquitetura do OpenFlow, ilustrada na Fig. 2.3, é composta por dispositivos de encaminhamento, denominados *switches* OpenFlow, que se comunicam com o controlador via protocolo OpenFlow, responsável por definir um conjunto de mensagens que podem ser trocadas através de um canal seguro. As decisões de controle são realizadas pelo controlador para o fluxo de dados com base nas informações contidas no cabeçalho do pacote e/ou na porta de entrada a partir da qual o pacote foi recebido.

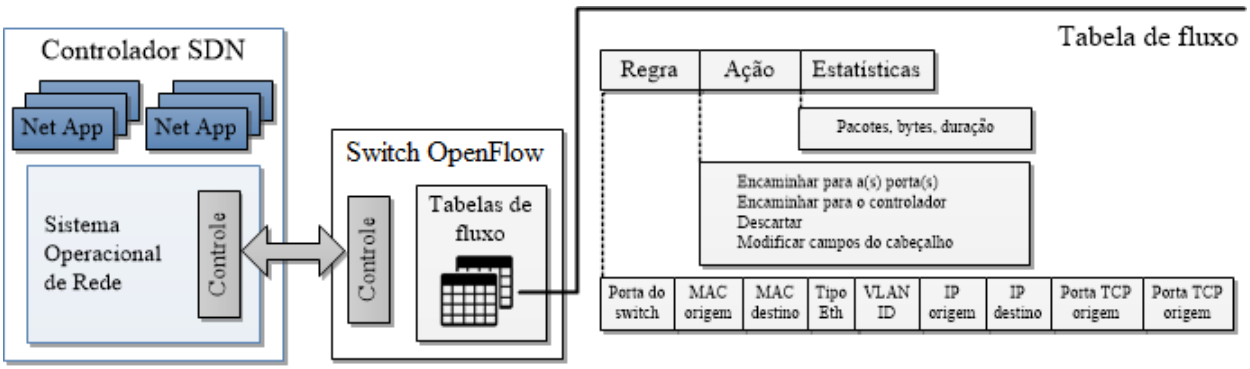


Figura 2.3: Arquitetura OpenFlow. Fonte: Adaptado de Kreutz et al. (2015).

Para determinar como os pacotes pertencentes a um determinado fluxo devem ser processados e encaminhados, o *switch* OpenFlow mantém uma ou mais tabelas de fluxo, cujas entradas são compostas por (NUNES et al., 2014a):

- *Campos ou regras de correspondência*: usadas para identificar pacotes de entrada, podem conter informações encontradas no cabeçalho do pacote, porta de ingresso e metadados.
- *Estatísticas*: contadores usados para coletar estatísticas de um fluxo específico, tais como o número de pacotes recebidos, o número de *bytes* e a duração do fluxo.

- *Conjunto de instruções ou ações*: a serem executadas em pacotes que atendem às regras de correspondência, podem ditar como manipular estes pacotes.

Através do protocolo OpenFlow, o controlador remoto é capaz de adicionar, atualizar ou remover entradas da tabela de fluxo dos *switches*, de maneira reativa e proativa, por meio de mensagens do tipo OFPT_FLOW_MOD.

Quando um determinado pacote é recebido por um *switch* OpenFlow, parte das informações do seu cabeçalho é comparada às entradas da tabela de fluxo. Se for encontrada uma regra de correspondência, o *switch* aplica o conjunto de instruções ou ações associadas a ela. Caso contrário, a ação a ser tomada pelo *switch* irá depender das intruções definidas pelas entradas da tabela de perda (conhecida como *miss-table*), tais como descartar o pacote, continuar o processo de verificação na próxima tabela de fluxo ou encaminhar o pacote para o controlador através de um canal OpenFlow (NUNES et al., 2014a) por meio de uma mensagem do tipo OFPT_FLOW_IN. Neste último caso, a decisão de controle é realizada analisando-se apenas o primeiro pacote do fluxo e o controlador então comunica a ação a ser tomada pelo *switch* por meio de uma mensagem do tipo OFPT_FLOW_OUT. A mesma ação pode ser reutilizada para todos os pacotes subsequentes. Assim, o controle baseado no fluxo reduz significativamente o tráfego entre o controlador e o *switch* (JAIN; PAUL, 2013).

2.2.2 HyperFlow

O HyperFlow (TOOTOONCHIAN; GANJALI, 2010) introduz um plano de controle distribuído baseado em eventos capaz de particionar a rede em vários domínios, cada domínio com um controlador dedicado. Desta forma, ele localiza a tomada de decisão em controladores locais para minimizar o tempo de resposta do plano de controle para solicitações do plano de dados, e fornece escalabilidade ao manter o controle de rede logicamente centralizado (AKYILDIZ et al., 2014).

Para tanto, o HyperFlow define a comunicação necessária para distribuição e redundância do plano de controle entre os controladores, e usa o protocolo OpenFlow para realizar a comunicação entre o controlador e o *switch*. O OpenFlow também permite estabelecer a conexão de um *switch* com vários controladores, de forma que controladores de contingência possam se tornar ativos em um evento de falha.

Através dos esquemas de sincronização, o Hyperflow permite que todos os controladores compartilhem a mesma visão consistente de toda a rede. A rede baseada em HyperFlow é composta por *switches* OpenFlow como elementos de encaminhamento, controladores como elementos de decisão, cada um executando uma instância da aplicação do controlador HyperFlow, e um sistema de propagação de eventos para a comunicação entre os controladores. Cada *switch* é conectado ao melhor controlador em sua proximidade, de acordo com as medidas de desempenho consideradas.

Tootoonchian e Ganjali (2010) sugerem a implementação do HyperFlow como uma aplicação para o NOX (GUDE et al., 2008). O NOX define uma sistema operacional de rede, tal que a rede é composta por *switches* e um ou mais servidores que executam o *software* NOX e as aplicações de gerenciamento nele implementadas, como ilustrado na Fig. 2.4. O NOX pode ser pensado como

envolvendo vários processos do controlador (normalmente um em cada servidor conectado à rede) e uma única visão da rede mantida em um banco de dados executado em um dos servidores. A visão única da rede permite que as aplicações, como o HyperFlow, usem este estado para tomar decisões de gerenciamento.

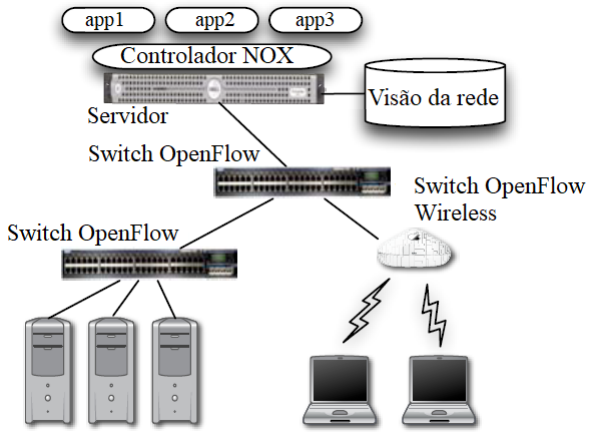


Figura 2.4: Componentes de uma rede com NOX: *switches* OpenFlow, servidor executando um processo do controlador NOX e um banco de dados que contém um banco de dados com a visão única da rede. Fonte: Gude et al. (2008).

Para alcançar uma visão consistente de toda a rede entre os controladores e minimizar o tráfego necessário para propagar os eventos, cada instância da aplicação do controlador HyperFlow publica seletivamente os eventos que alteram o estado da rede através de um sistema de *publish/subscribe*, de forma que os controladores em um domínio devem obter a maioria das atualizações de outros domínios de controladores próximos para evitar congestionar a rede entre regiões (AKYILDIZ et al., 2014). Para o armazenamento persistente dos eventos publicados, Tootoonchian e Ganjali (2010) usam o WheelFS (STRIBLING et al., 2009), um sistema de arquivos projetado para aplicações distribuídas. Detalhes sobre a operação do sistema de *publish/subscribe* e a utilização do WheelFS são descritos em (TOOTOONCHIAN; GANJALI, 2010) .

2.2.3 Desempenho de rede

De acordo com Curtis et al. (2011), o caminho utilizado por um *switch* Openflow para se comunicar com o seu controlador é de primordial importância para o dimensionamento e desempenho de uma rede. Neste contexto, Phemius e Bouet (2013) avaliaram como a largura de banda e a latência entre o *switch* Openflow e o seu controlador afetam o desempenho de toda a rede.

O controlador consiste em um dispositivo responsável por controlar centenas e milhares de *switches* e, de fato, em muitos casos ele está distante, em termos de latência, dos elementos que ele controla.

De acordo com Phemius e Bouet (2013), quando a comunicação entre o *switch* OpenFlow e o controlador é submetida a uma largura de banda baixa, o *switch* não pode enviar tantas mensagens de OFPT_PACKET_IN para o controlador que, em contrapartida, responde com

poucos OFPT_PACKET_OUT. Isso ocorre uma vez que o controlador não pode tratar mensagens mais rapidamente do que as recebe. Além disso, uma baixa largura de banda resulta em algumas perdas de pacotes à medida que as filas de saída são preenchidas. Aumentar a largura de banda permite uma rede mais reativa, de forma que a capacidade de cada controlador é limitada pela largura de banda dos seus acessos, o que resulta em um número limitado de dispositivos associados ao controlador em um determinado momento (YAO et al., 2014).

A latência, por sua vez, tem um efeito diferente no desempenho de rede. Antes de iniciar sua operação, cada *switch* estabelece conexão com o controlador, o que resulta em uma janela de tempo durante a qual nenhuma operação é possível. À medida em que a latência aumenta, o tempo necessário para o estabelecimento da conexão também aumenta.

Phemius e Bouet (2013) apresentaram em seu estudo a relação entre o consumo de largura de banda e o tempo para diferentes valores de latência em uma rede contendo 32 *switches*. O resultado do estudo é apresentado na Fig. 2.5, onde verificou-se que à medida que a latência aumenta, o tempo para completar o estabelecimento da comunicação aumenta, tal que alcança 7s para uma latência de 100ms, 20s para uma latência de 300ms e 3/4 do tempo alocado para ativar a normalidade quando a latência é de 500ms.

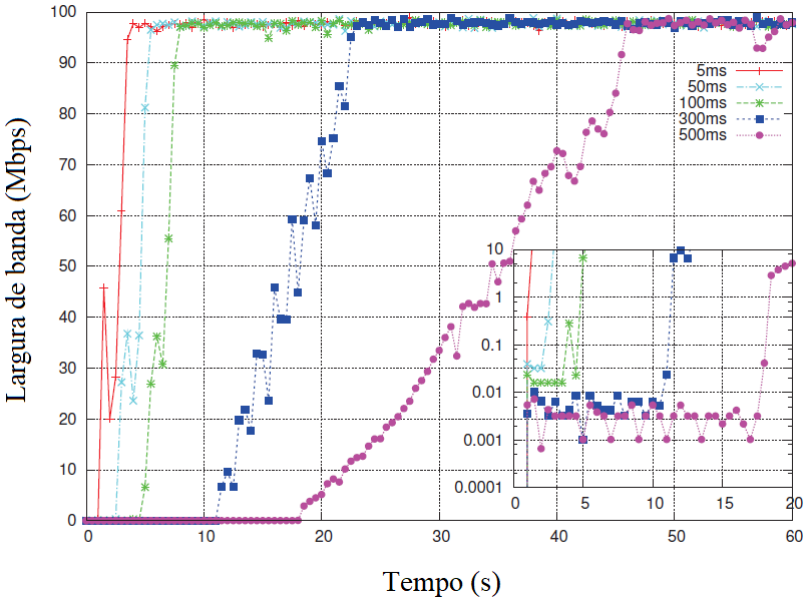


Figura 2.5: Utilização da largura de banda para várias latências. Largura de banda: 100Mbps 32 *switches* e 100 *hosts* por *switch*. Fonte: Phemius e Bouet (2013).

A latência na comunicação entre o *switch* e o controlador também tem efeito sobre os fluxos de dados entre os *switches*. Como já foi apresentado anteriormente, quando um determinado fluxo é recebido por um *switch* OpenFlow, parte das informações do seu cabeçalho é comparada às entradas da tabela de fluxo. Se não for encontrada uma regra de correspondência, o primeiro pacote do fluxo é encaminhado para o controlador, a quem cabe definir ações para o encaminhamento do fluxo e adicionar, atualizar ou deletar entradas da tabela de fluxo do *switch*. Desta forma, à medida que a latência na comunicação entre o *switch* e o controlador aumenta, o tempo necessário para a atualização da tabela de fluxo e início efetivo da transmissão do fluxo entre os *switches* também

aumenta.

Para minimizar o tempo de resposta do plano de controle para solicitações do plano de dados, o HyperFlow permite que a tomada de decisão seja distribuída entre controladores locais. Entretanto, o HyperFlow também lida com os efeitos da latência e da largura de banda durante a sincronização dos controladores, além disso, cada controlador também é limitado pela sua capacidade de processamento.

Neste contexto, verifica-se a importância de se avaliar diferentes medidas de desempenho de rede para o posicionamento adequado de controladores em uma rede SDN, de forma que essas medidas são concorrentes entre si.

2.3 Considerações finais

Este capítulo apresentou o paradigma de redes SDN que, diferentemente das redes convencionais, propõe a separação do plano de controle do plano de dados a fim de tornar o gerenciamento de rede mais simples e entregar serviços de *middleboxes*, como roteamento, balanceamento de carga, *multicasting*, segurança, controle de acesso, entre outros, de maneira otimizada.

Dentre detalhes de arquitetura e componentes, foram apresentados o OpenFlow e o HyperFlow. O OpenFlow é um padrão/protocolo que define um conjunto de mensagens para a comunicação entre os *switches* e o controlador, responsável por realizar decisões de controle de fluxo de maneira centralizada. O HyperFlow, por sua vez, introduz um plano de controle distribuído capaz de particionar a rede em vários domínios, cada domínio com um controlador dedicado. Por fim, este capítulo apresentou medidas de desempenho a serem consideradas em redes SDN.

Considerando o efeito do posicionamento do controlador nas diferentes medidas de desempenho de rede apresentadas, o Capítulo 3 trata da definição dos problemas de otimização multiobjetivo, caracterizados pela existência de mais de um critério a ser otimizado, e apresenta métodos capazes de obter soluções com menor custo computacional.

Capítulo 3

Otimização Multiobjetivo

3.1 Introdução

Os problemas de otimização combinatória multiobjetivo (MOCO, do inglês *Multi-Objective Combinatorial Optimization*) são caracterizados pela existência de mais de um critério a ser otimizado. Esses critérios são, na maioria das vezes, concorrentes entre si.

Diferente da otimização mono-objetivo, onde o ótimo corresponde às soluções extremas (mínimas ou máximas) da função-objetivo do problema em questão, na otimização multiobjetivo é impossível adotar a solução extrema de um dos objetivos pois os demais objetivos também são relevantes ao problema, e as soluções em extremos de um único objetivo exigem um compromisso nos demais.

Neste contexto, o ótimo de um problema multiobjetivo é definido pelo conjunto de soluções que correspondem a compromissos (*trade-offs*) diferentes entre os objetivos. Essas soluções, geralmente conhecidas como soluções Pareto-ótimas, também podem ser denominadas como soluções eficientes, ou conjunto admissível do problema.

Coello, Lamont e Veldhuizen (2007) definem que a resolução de problemas multiobjetivo pode ser tratada por três abordagens:

- *Inserção de preferência a posteriori*: busca-se encontrar o maior número de soluções possível, para só depois selecionar a mais adequada ao problema (busca $\rightarrow$ decisão);
- *Inserção de preferência a priori*: já se tem de antemão alguma informação sobre o tipo de solução mais adequada ao problema, então a busca é direcionada para encontrar este tipo de solução (decisão $\rightarrow$ busca);
- *Inserção progressiva de preferências*: é feito um direcionamento da busca, durante sua execução, para regiões que contenham soluções mais adequadas (decisão $\leftrightarrow$ busca).

3.2 Definição

O problema de otimização multiobjetivo (KUMAR; SUMAN, 2006; COELLO; CORTES, 2005; COELLO, 2005) consiste em encontrar um vetor de variáveis de decisão $\mathbf{x} = [x_1, x_2, ..., x_N]^T$, capaz de otimizar o vetor de funções objetivo:

$$f(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_J(\mathbf{x})]^T, \tag{3.1}$$

de forma que uma dada função objetivo seja minimizada ou maximizada, vistas as restrições dadas pelas funções:

$$g_i(\mathbf{x}) \geq 0, i = 1, 2, 3, ..., m \tag{3.2}$$

$$h_i(\mathbf{x}) = 0, i = 1, 2, 3, ..., l. \tag{3.3}$$

Um dado vetor $\mathbf{x}$ é dito pertencer ao conjunto Pareto-ótimo se não existir nenhum outro vetor $\mathbf{x}^*$ factível, capaz de melhorar um dos objetivos do problema (em relação a $\mathbf{x}$) sem simultaneamente piorar pelo menos um dos demais. Todas as soluções pertencentes ao conjunto Pareto-ótimo são ditas não dominantes. Se todas as funções objetivo são para minimização, pelo conceito de dominância, entende-se que uma dada solução $\mathbf{x}^*$ domina uma solução $\mathbf{x}$ se, e somente se, $\forall\ j$ $f_j(\mathbf{x}^*) \leq f_j(\mathbf{x})$ e $\exists\ i$ $f_i(\mathbf{x}^*) < f_i(\mathbf{x})$ para pelo menos um i (KONAK; COIT; SMITH, 2006). Para o conjunto de soluções Pareto-ótimas, os valores correspondentes das funções objetivo no espaço de objetivos é denominado Fronteira de Pareto, como ilustrado na 3.1.

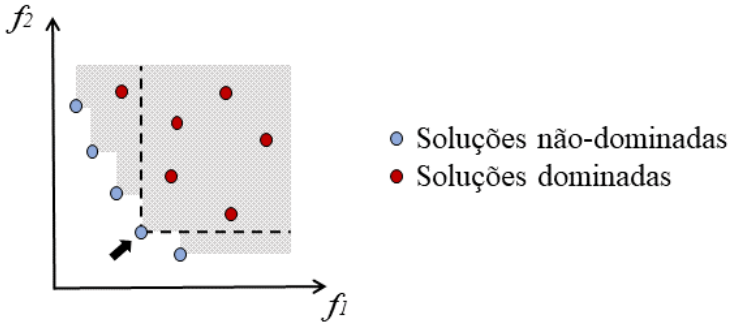


Figura 3.1: Fronteira de Pareto.

Goldberg (1989) propõe o método conhecido como *Automatic Accumulated Ranking Strategy* (AARS) para identificar o conjunto de soluções não-dominadas. Este método, representado pelo Algoritmo 3.1, consiste em classificar as soluções em níveis de não-dominância, tal que as soluções que formam a fronteira de Pareto são aquelas cuja classificação de não-dominância seja igual a 1. Neste trabalho, propomos a implementação do AARS como um método exaustivo do problema de otimização multiobjetivo para o posicionamento de controladores em redes SDN, dada a sua facilidade de implementação.

Algoritmo 3.1: Automatic Accumulated Ranking Strategy (AARS)

```
Marca cada solução no conjunto de soluções S como não avaliada;
Define o nível atual igual a 1;
enquanto existir alguma solução não avaliada faça
    para cada solução não avaliada faça
        Verifica se a solução é dominada em relação às demais soluções não avaliadas ou com
         o nível igual ao nível atual;
        se for não dominada então
            Define o nível da solução igual ao nível atual;
            Marca a solução como avaliada;
    Depois de verificar todas as soluções, incrementa o nível atual por 1;
```

3.3 Metaheurísticas

O desafio ao resolver problemas combinatórios multiobjetivos é função não apenas da complexidade combinatorial, mas também pela dificuldade em alcançar soluções eficientes, que crescem com o número de objetivos do problema.

A resolução dos problemas de otimização multiobjetivo deve ainda envolver algoritmos capazes de preservar as soluções não-dominadas encontradas, progredir continuamente em direção à Fronteira de Pareto, manter a diversidade de soluções e retornar ao usuário uma quantidade suficiente (mas ao mesmo tempo limitada) de soluções. Neste contexto, os métodos baseados em metaheurísticas se apresentam como uma boa alternativa devido às suas características de flexibilidade, eficiência e fácil implementação.

Por heurística entende-se uma classe de algoritmos que permite obter soluções aproximadas em problemas de otimização, sem ter que varrer todo o espaço de soluções, apresentando-se como métodos mais flexíveis e de fácil implementação. Tais algoritmos compõem uma gama de métodos aplicáveis à busca de soluções denominadas, neste contexto, de metaheurísticas.

De acordo com Perez e Raupp (2012), o prefixo "meta" é utilizado para descrever uma heurística que está sobreposta a uma outra heurística, constituindo um outro "nível heurístico". Em geral, uma metaheurística constitui uma estrutura mais genérica baseada em princípios ou conceitos, sobreposta a uma heurística específica do problema em estudo.

As metaheurísticas se caracterizam pela facilidade de incorporar dispersão entre as soluções e de explorar regiões do conjunto de soluções factíveis na tentativa de superar ótimos locais. Procurando fazer uso dos benefícios apresentados por esses métodos, Lange et al. (2015b) implementaram a metaheurística *Pareto Simulated Annealing* (PSA) para o problema de posicionamento de controladores em redes SDN. No entanto, dada a potencial variedade de abordagens para problemas de otimização multiobjetivo, este trabalho propõe ainda a implementação do *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) (DEB et al., 2002), vista a sua capacidade de preservação de diversidade, enquanto mantém uma solução ótima encontrada, e a sua baixa complexidade.

3.3.1 PSA - Pareto Simulated Annealing

O *Pareto Simulated Anneling* (PSA) (CZYZAK; JASZKIEWICZ, 1998) é um mecanismo derivado da conhecida técnica de otimização *Simulated Annealing* e permite explorar apenas um subconjunto do espaço de busca no âmbito da otimização combinatória multiobjetivo para encontrar soluções Pareto-ótimas.

O *Simulated Anneling* (METROPOLIS et al., 1953) original é um método de Monte Carlo capaz de explorar novas áreas no espaço de busca de problemas mono-objetivo através de movimentos para soluções vizinhas potencialmente piores que a atual, com o intuito de evitar ótimos locais.

Para determinar a probabilidade de se aceitar tal movimento, o *Simulated Anneling* se baseia em um parâmetro de controle conhecido como "temperatura", que incorpora níveis de dispersão durante a análise dos subconjuntos que constituem a vizinhança da solução atual. Ao longo das iterações, a probabilidade de movimento para uma solução pior reduz gradualmente com o decaimento da temperatura. Aceitando soluções bastante ruins no início, devido à temperatura elevada, têm-se uma cobertura mais ampla do espaço de busca, enquanto a menor probabilidade de aceitação no final, devido à redução da temperatura, leva à convergência.

O PSA foi proposto por Czyzak e Jaszkiewicz (1998) como uma modificação do *Simulated Anneling* para problemas multiobjetivos de forma que, ao invés de buscar por uma solução única, o PSA busca por uma boa representação do conjunto de soluções não-dominadas. Assim como no método original, o PSA se baseia nos conceitos de vizinhança, temperatura e probabilidade de aceitação, tal que uma solução s pode ser modificada pela aceitação de uma solução vizinha com uma probabilidade P, podendo ocorrer as seguintes situações quando uma nova solução q é comparada com à s:

1. q domina ou é igual a s;
2. s domina q;
3. q é não-dominado com relação à s.

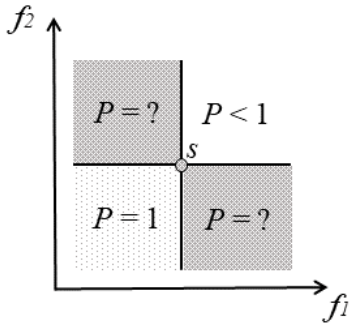


Figura 3.2: Ideia geral das regras de aceitação multiobjetivo para a probabilidade de aceitação P. Fonte: Czyzak e Jaszkiewicz (1998).

A Fig. 3.2 ilustra essas três situações no caso de dois objetivos. Na primeira situação, a nova

solução pode ser considerada não pior que a solução atual e aceita com uma probabilidade igual a 1, enquanto na segunda situação, a nova solução pode ser considerada pior que a solução atual e aceita com probabilidade menor que 1. Já a terceira situação é tratada por meio de regras de aceitação, como por exemplo a regra SL (*Scalar Linear*) apresentada por Serafini (1993), que pode ser interpretada como uma agregação local de todos os objetivos com uma função de escala linear, tal que:

$$P(s, q, T, \mathbf{\Lambda}) = min\left\{1, e^{\sum_{j=1}^{J} \lambda_j (f_j(s) - f_j(q))/T}\right\}, \tag{3.4}$$

onde s é a solução atual e q é a solução vizinha a s, $f_j(s) - f_j(q)$ é a diferença normalizada no valor da função objetivo j entre as soluções s e q, T é a temperatura, $\mathbf{\Lambda} = [\lambda_1, \lambda_2, ..., \lambda_J]$ é o vetor de pesos da solução s e J é o número de objetivos.

O vetor de pesos $\mathbf{\Lambda}$ usado nas regras de aceitação permite influenciar a direção da busca no espaço multiobjetivo para aumentar a probabilidade de mover uma dada solução s na direção do vizinho mais próximo s'. Inicialmente, o vetor de pesos é gerado aleatoriamente e depois é modificado a cada iteração pelo incremento por um fator α ($\alpha > 1$ e constante próximo de 1, por exemplo $\alpha = 1.05$) dos pesos dos objetivos no qual s é melhor que s' e pelo decremento dos pesos dos objetivos no qual s é pior que s' por um fator $1/\alpha$, ou seja:

$$\lambda_j = \begin{cases} \alpha\lambda_j^s, & \text{se } f_j(s) \leq f_j(s') \\ \lambda_j^s/\alpha, & \text{se } f_j(s) > f_j(s') \end{cases} \tag{3.5}$$

A operação do PSA é apresentada pelo Algoritmo 3.2. O procedimento se inicia com um conjunto de soluções inicial $S = S_0$, de tamanho N. Para cada solução $s \in S$ avaliada, um conjunto de soluções Q é atualizado com as soluções Pareto-ótimas encontradas.

Inicialmente, T assume um valor elevado T_0 e após m iterações, seu valor decai gradativamente em razão de uma taxa de resfriamento ρ, tal que $T_n = \rho \cdot T_{n-1}$, sendo $0 < \rho < 1$. Uma condição de parada comumente utilizada faz com que algoritmo finalize quando $T \leq 1$, tal que o número total de níveis de temperatura percorridos durante a sua execução é definido por (LANGE et al., 2015b):

$$\begin{gathered} T_0\rho^i \leq 1, \\ \rho^i \leq \tfrac{1}{T_0}, \\ i \log \rho \leq -\log T_0, \\ i \leq -\tfrac{\log T_0}{log\rho}. \end{gathered} \tag{3.6}$$

A cada iteração, é gerada uma solução q vizinha a s. Se q for não-dominada por s, o conjunto de soluções potencialmente Pareto-ótimas Q é atualizado com q. Em seguida, seleciona-se a solução $s' \in S$ mais próxima a s e não dominada em relação a s, que é utilizada no cálculo do vetor de pesos $\mathbf{\Lambda}$. O vetor de pesos $\mathbf{\Lambda}$ influenciará a probabilidade da solução s ser modificada pela solução

Algoritmo 3.2: Pseudo-código do algoritmo Pareto Simulated Anneling

Gera aleatoriamente um conjunto de soluções S_0 de tamanho N;
$S = S_0$;
para *cada solução* $s \in S$ **faça**
  Atualiza o conjunto de soluções potencialmente Pareto-ótimas Q com s;
$T := T_0$;
enquanto *condições de parada não são cumpridas* **faça**
  para *cada solução* $s \in S$ **faça**
    Gera uma solução viável q;
    se q *for não dominada por* s **então**
      Atualiza o conjunto de soluções potencialmente Pareto-ótimas Q com q;
    Seleciona a solução $s' \in S$ mais próxima à s e não dominada em relação à s;
    se s' *não existir ou for a primeira iteração com* s **então**
      Define um peso aleatório tal que $\forall j, \lambda_j \geq 0$ e $\sum_j \lambda_j = 1$;
    senão
      para *cada objetivo* f_j **faça**
        $$\lambda_j = \begin{cases} \alpha\lambda_j^s, & \text{se } f_j(s) \leq f_j(s') \\ \lambda_j^s/\alpha, & \text{se } f_j(s) > f_j(s') \end{cases}$$
      Normaliza os pesos tal que $\sum_j \lambda_j = 1$;
    $s := q$ (aceita q) com probabilidade $P(s, q, T, \mathbf{\Lambda})$;
  se *as condições de mudança de temperatura forem cumpridas* **então**
    Decrementa T em razão de ρ;

vizinha q em S. O processo continua até que a condição de parada seja atendida e tenha-se o conjunto estimado de soluções Pareto-ótimas Q.

3.3.2 NSGA II - Non-dominated Sorting Genetic Algorithm II

O *Non-dominated Sorting Genetic Algorithm* II (NSGA II) (DEB et al., 2002) é um algoritmo evolutivo elitista que permite explorar apenas um subconjunto do espaço de busca no âmbito da otimização combinatória multiobjetivo, a fim de encontrar uma estimativa da fronteira de Pareto mais próxima da real, sendo amplamente utilizado em problemas mais complexos.

Para tanto, o NSGA II se baseia nos conceitos de população e dominância, onde a população define um conjunto de soluções, e utiliza os operadores de seleção, cruzamento e mutação para criar novas populações capazes de gerar soluções melhores a cada iteração. O funcionamento do NSGA II se destaca ainda por possuir mecanismos importantes no processo de seleção, são eles: *Fast Non-Dominated Sorting* e distância de aglomeração (ou *crowding distance*), apresentados com mais detalhes nas Seções 3.3.2.1 e 3.3.2.2, respectivamente.

Através do critério de dominância, o *Fast Non-Dominated Sorting* agrega o conceito de elistismo, de forma que a população gerada em cada iteração é classificada em diferentes níveis de não dominância, tal que as soluções localizadas no primeiro nível são consideradas as melhores daquela geração, enquanto que o último nível concentra as piores soluções. Para manter uma boa dispersão entre as soluções geradas em cada iteração, o critério de distância de aglomeração permite avaliar

o espalhamento das soluções classificadas em um mesmo nível de não dominância.

A seleção das soluções da população a serem utilizadas nos processos de cruzamento e mutação é realizada através de torneio com base na aptidão de cada solução, tal que a aptidão de uma determinada solução s em relação a uma solução q é avaliada de acordo com o nível de não dominância no qual foi classificada pelo método *Fast Non-Dominated Sorting* ($s^{classificacão}$) e com a distância de aglomeração ($s^{distância}$), por meio do operador de comparação de aglomeração ($\prec_n$):

$$s \prec_n q \begin{cases} \text{se} \quad s^{classificacão} < q^{classificacão} \\ \text{ou} \quad ((s^{classificacão} = q^{classificacão}) \quad \text{e} \quad (s^{distância} > q^{distância})) \end{cases} \tag{3.7}$$

Desta forma, o NSGA II opta por soluções mais bem classificadas com relação à não-dominância para garantir que as melhores soluções continuarão a existir na próxima geração e, se a classificação for a mesma, prefere-se aquela localizada em uma região com menor aglomeração, para se manter dispersão na população gerada.

Bastante similar à união cromossômica para formar novas combinações de genes, o operador de cruzamento é responsável por trocar sequências aleatoriamente escolhidas entre as soluções selecionadas, gerando novas soluções que herdam características das soluções anteriores. O operador de mutação, por sua vez, não agrega novas soluções à população, ele apenas modifica soluções que já existem, transformando-nas em soluções diferentes por meio da modificação de sequências distintas aleatoriamente escolhidas. Isso faz com que o NSGA II consiga uma maior dispersão que a função de cruzamento sozinha não seria capaz de fornecer, evitando que o algoritmo fique estacionado em ótimos locais.

A operação do NSGA-II é apresentada pelo Algoritmo 3.3 (DEB et al., 2002). Inicialmente, é criada uma população geradora S_0, de tamanho N. S_0 é ordenada em fronteiras de acordo com o nível de não dominância no qual cada solução é classificada, e para cada solução é calculada a distância de aglomeração.

Depois da inicialização, o passo a passo do algoritmo é simples e direto. A cada iteração t, as populações geradora (S_t) e gerada (Q_t) são combinadas em $R_t = S_t \cup Q_t$, de tamanho $2N$, a fim de formar uma nova população geradora S_{t+1}, de tamanho N.

A população R_t é ordenada de acordo com o nível de não dominância em uma lista de fronteiras estimadas de Pareto F, onde F_1 define o primeiro nível de não dominância. As soluções pertencentes ao conjunto F_1 são considerados as melhores soluções e devem ser mais enfatizadas que as demais soluções de R_t. Se o tamanho de F_1 for menor que N, todos os membros de F_1 são selecionados para a nova população S_{t+1}. Os demais membros de S_{t+1} são selecionados a partir das fronteiras seguintes pela ordem de classificação de não dominância. Assim, as soluções do conjunto F_2 são as próximas a serem selecionadas, seguidas pelas soluções do conjunto F_3, e assim por diante, até que mais nenhum conjunto possa ser acomodado em S_{t+1}. Se a quantidade de membros do último conjunto avaliado F_l for maior que a capacidade de acomodação em S_{t+1}, a distância de aglomeração é utilizada em ordem decrescente como critério de desempate.

Algoritmo 3.3: Pseudo-código do algoritmo Non-dominated Sorting Genetic Algorithm

Gera aleatoriamente um conjunto de soluções S_0 de tamanho N;
$Q_0 = \emptyset$;
$t = 0$;
enquanto *critério de parada não for atingido* **faça**
  $R_t = S_t \cup Q_t$;
  F = fast-non-dominated-sorting(R_t);
  $S_{t+1} = \emptyset$;
  $i = 1$;
  enquanto $|S_{t+1}| + |F_i| \leq N$ **faça**
    crowding-distance-assignment(F_i);
    $S_{t+1} = S_{t+1} \cup F_i$;
    $i = i + 1$;
  Ordena F_i em ordem descrescente com base na aptidão de cada solução ($\prec_n$);
  $S_{t+1} = S_{t+1} \cup F_i[1 : (N - |S_{t+1}|)]$;
  Q_{t+1} = Gera nova população à partir de S_{t+1} usando os operadores de seleção, cruzamento e mutação;
  $t = t + 1$;

Ao final de cada iteração, a nova população S_{t+1} utiliza os operadores de seleção, cruzamento e mutação para gerar uma nova população Q_{t+1}, de tamanho N. Uma vez que esses operadores precisam do classificação de não dominância e da distância de aglomeração, esses valores são calculados durante a geração da população S_{t+1}.

O NSGA II implementa dois métodos de convergência, o primeiro e mais simples utiliza a quantidade de iterações como condição de parada. Assim, quando o algoritmo alcança determinado número de iterações m, ele finaliza a sua execução. No segundo método de convergência, verifica-se se houve uma alteração significativa entre as soluções da população na iteração atual em relação às iterações passadas. Se não for detectada uma variação de qualidade significativa, então existe a indicação de convergência e o algoritmo é finalizado. Caso contrário, o algoritmo segue sua execução normal até que o algoritmo forneça uma indicação de convergência.

3.3.2.1 Fast Non-Dominated Sorting

O *Fast Non-Dominated Sorting* foi proposto por Deb et al. (2002) e define um algoritmo capaz de verificar a relação de dominância entre os elementos de um dado conjunto de soluções e organizá-los de acordo com o nível de classificação de não-dominância em diferentes fronteiras, a fim de encontrar a fronteira de Pareto, cuja classificação de não-dominância será igual a 0. Sua operação é apresentada pelo Algoritmo 3.4, que tem como entrada o conjunto de soluções possíveis S.

Para cada solução $s \in S$ são calculadas duas variáveis: (i) n_s, o número de soluções que dominam a solução s, e (ii) C_s, o conjunto de soluções dominadas por s. A fronteira de Pareto F_1 é definida pelas soluções s classificadas no primeiro nível de não-dominância, onde $n_s = 0$. Para cada solução s com $n_s = 0$, todos os membros $q \in S$ do conjunto C_s têm seu contador n_q

decrementado em um. Se o contador de qualquer membro q se tornar zero, q é classificado em uma lista separada Q, que define o segundo nível de não-dominância. O mesmo procedimento é realizado com cada membro de Q para encontrar o terceiro nível, e continua até que todas as fronteiras ou níveis de classificação sejam definidos.

Algoritmo 3.4: fast-non-dominated-sorting

```
para cada s ∈ S faça
    para cada q ∈ S faça
        se s ≺ q então
            C_s = C_s ∪ {q}
        se q ≺ s então
            n_s = n_s + 1
    se n_s = 0 então
        F_1 = F_1 ∪ {s}
i = 1;
enquanto Fi ≠ ∅ faça
    Q = ∅;
    para cada s ∈ F_i faça
        para cada q ∈ C_s faça
            n_q = n_q − 1;
            se n_q = 0 então
                Q = Q ∪ {q}
    i = i + 1;
    F_i = Q;
```

3.3.2.2 Distância de aglomeração

Para manter uma boa dispersão entre as soluções geradas em cada iteração, o critério de distância de aglomeração permite avaliar o espalhamento das soluções classificadas em um mesmo nível de não dominância: ela é medida para cada objetivo j, de forma que as soluções situadas nos extremos (soluções com maior e menor valor) têm a distância de aglomeração igual a ∞, enquanto que para as soluções intermediárias este valor é definido pela diferença normalizada absoluta dos valores do objetivo de duas soluções adjacentes, descrita por:

$$\frac{f_j(s+1) - f_j(s-1)}{f_j^{max} - f_j^{min}}, \tag{3.8}$$

$f_j(s+1) - f_j(s-1)$ sendo a diferença no valor da função objetivo j entre as soluções adjacentes $(s+1)$ e $(s-1)$ e os parâmetros f_j^{max} e f_j^{min} são os valores máximo e mínimo da função objetivo f_j.

Desta forma, a distância de aglomeração é obtida pela distância média entre uma solução central s selecionada dentro da população e duas soluções adjacentes à solução central $(s-1)$ e $(s+1)$. A ideia é que a partir de uma solução central seja possível encontrar soluções extremas e priorizar soluções mais distantes durante o processo de seleção. Uma vez que a disposição das soluções extremas forma um cuboíde em relação à solução central, o cálculo da distância de aglomeração

é definido pelo perímetro normalizado do cuboide envolvendo a solução s, representado pela Fig. 3.3, cujos vértices identificam as soluções adjacentes $(s-1)$ e $(s+1)$, tal que as soluções localizadas ao extremo do nível de não dominância terão sua distância igual a ∞.

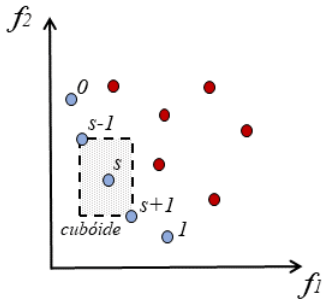


Figura 3.3: Cálculo da distância de aglomeração. Os círculos em azul representam membros de um mesmo nível de não dominância, onde a distância da s-ésima solução é o perímeto do cubóide mostrado na caixa tracejada.

Por fim, o valor geral da distância de aglomeração de uma solução é a soma dos valores de distâncias individuais correspondentes a cada objetivo, tal que um valor geral pequeno indica que uma dada solução se encontra em uma região menos densa.

A distância de aglomeração entre as soluções contidas em uma mesma fronteira F deve ser calculada para cada solução conforme o Algoritmo 3.5, onde $F[s] \cdot j$ define o valor da j-ésima função objetivo de uma solução s no conjunto F e os parâmetros f_j^{max} e f_j^{min} são os valores máximo e mínimo da j-ésima função objetivo (DEB et al., 2002; FORTIN; PARIZEAU, 2013). As soluções são ordenadas em ordem crescente de acordo com os valores de cada função objetivo. Para cada função objetivo, a distância das soluções de borda (soluções com menor e maior valor) será igual a infinito e das soluções intermediárias será igual à diferença absoluta normalizada entre as soluções adjacentes. Ao final do cálculo de todas as funções objetivo, a distância de aglomeração da solução será a soma das distâncias individuais que correspondem à cada objetivo.

Algoritmo 3.5: crowding-distance-assignment

Entrada: $l = |F|$
para *cada* s **faça**
  $F[s]_{distância} = 0$
para *cada objetivo* j **faça**
  $F = \text{ordena}(F,j)$;
  $F[1]_{distância} = F[l]_{distância} = \infty$;
  para $s = 2$ *até* $(l-1)$ **faça**
    $F[s]_{distância} = F[s]_{distância} + (F[s+1].j - F[s-1].j)/(f_j^{max} - f_j^{min})$;

3.4 Considerações finais

Este capítulo apresentou a definição para problemas de otimização multiobjetivo, caracterizados pela existência de mais de um critério a ser otimizado.

Dado que desafio de resolver problemas combinatórios multiobjetivos é função não apenas da complexidade combinatorial, mas também da dificuldade em alcançar soluções eficientes, propõe-se a utilização de metaheurísticas como uma alternativa para obter soluções aproximadas com menor custo computacional. Neste contexto, foram apresentados o PSA e o NSGA-II.

No Capítulo 4 serão discutidos os detalhes de implementação do PSA e do NSGA-II para computar o posicionamento ótimo de controladores, tal como as medidas e ferramentas utilizadas.

Capítulo 4

Desenvolvimento

4.1 Introdução

Este trabalho tem como base a utilização do *framework* POCO para computar o posicionamento ótimo de controladores e do Mininet como emulador de redes SDN para análise do impacto do posicionamento do controlador na operação de redes habilitadas com OpenFlow.

Na Seção 4.2, são apresentadas as funcionalidades disponíveis no POCO e propõe-se a extensão dessas funcionalidades por meio de novas medidas e implementação de novos mecanismos para o cálculo e a análise do posicionamento ótimo de controladores. Na Seção 4.3, é feita uma breve introdução da operação do Mininet, exemplificando a operação do protocolo OpenFlow com o controlador Floodlight.

4.2 POCO

O POCO é um ambiente em MATLAB desenvolvido pela Universidade de Würzburg (HOCK et al., 2013, 2014; HARTMANN et al., 2014; LANGE et al., 2015a, 2015b) que permite calcular o posicionamento ótimo de controladores por meio de técnicas exaustiva e heurísticas, com respeito a diferentes medidas de desempenho de rede. Este ambiente apresenta uma interface gráfica que disponibiliza uma clara ilustração dos compromissos entre medidas concorrentes, como ilustrado na Fig. 4.1, além de permitir a extensão de funcionalidades como, por exemplo, a implementação de novas medidas e de novos mecanismos para o cálculo e a análise do posicionamento ótimo de controladores.

Em sua versão original (HOCK et al., 2013, 2014), o POCO implementa um método exaustivo que permite o cálculo da fronteira de Pareto para até quatro medidas de desempenho de rede, conforme opções selecionadas pelo usuário na interface gráfica, permitindo também considerar fatores de resiliência.

Para tanto, considera-se uma rede composta por n nós que correspondem às posições habilitadas com *switches* OpenFlow conectados entre eles. Desta forma, a rede é representada pelo grafo

$G = (V, E)$, onde $V = \{v_1, v_2, ..., v_n\}$ denota o conjunto de nós (vértices) e E denota o conjunto de conexões disponíveis (arestas). Dado um número de controladores k, a tarefa de otimização está em determinar um conjunto de posições $\mathcal{P}$ para conectar os k controladores, tal que $\mathcal{P}_k = \{\mathcal{P} \in 2^V || \mathcal{P}| = k\}$.

Todo o cálculo necessário é realizado uma única vez e os resultados são armazenados localmente para tornar a experiência de avaliação dos diferentes compromissos entre as diferentes medidas mais rápida para o usuário.

Na busca pela redução do tempo de execução e menor margem de erro, em versões posteriores, o POCO recebeu funcionalidades que permitem o cálculo da fronteira de Pareto para várias medidas de desempenho de rede através da metaheurística Pareto Simulated Annealing (PSA) (LANGE et al., 2015b) e do algoritmo Pareto Capacitated k-Medoids (PCKM) (LANGE et al., 2015a). Uma das extensões do POCO, conhecida como POCO-PLC (HARTMANN et al., 2014), permite ainda explorar condições de redes dinâmicas.

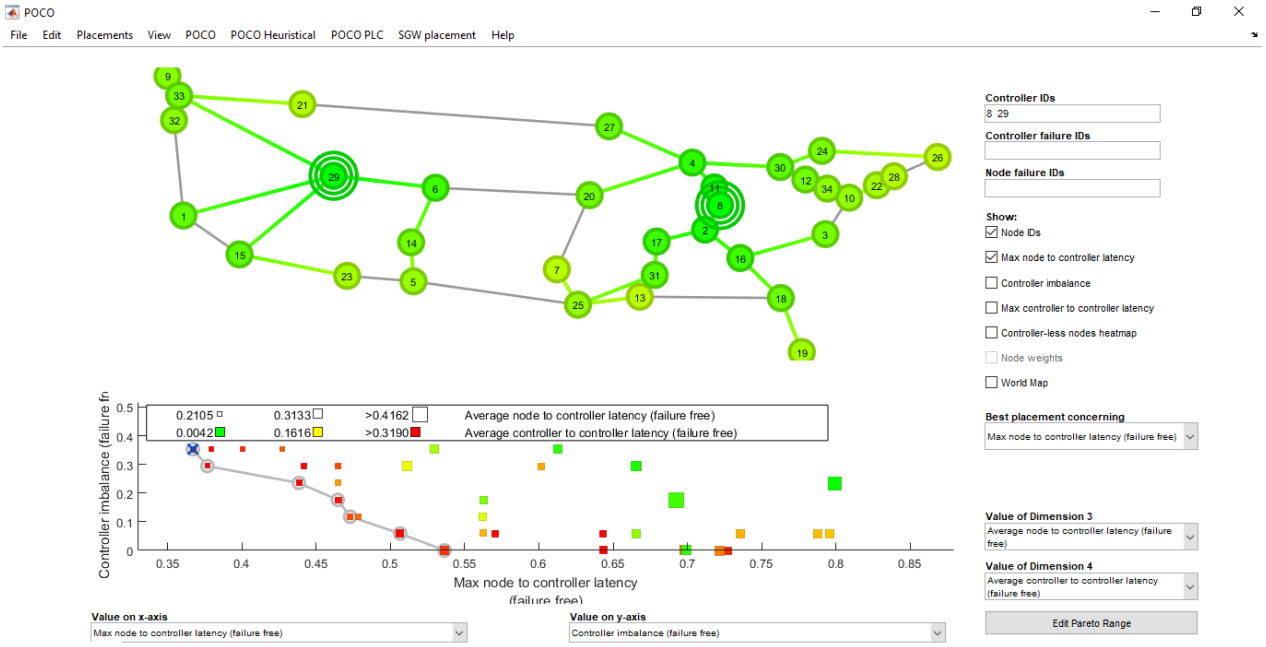


Figura 4.1: Interface gráfica do POCO.

4.2.1 Medidas de desempenho de rede

Dada uma rede representada pelo grafo $G = (V, E)$, conforme descrito anteriormente, a tarefa de otimização está em determinar um conjunto de posições $\mathcal{P}$ para conectar os k controladores, de forma que as funções objetivo $f_j (j \in \{1, ..., J\})$ sejam minimizadas ou maximizadas. As funções objetivo mapeiam medidas de desempenho individuais, como as apresentadas na Seção 2.2.3: latência, balanceamento de carga e taxa de transmissão em valores numéricos.

Neste trabalho, as funções objetivo são consideradas para o caso livre de falhas, ou seja, não se considera a hipótese de um dos controladores parar de funcionar.

4.2.1.1 Latência

As funções objetivo definidas para avaliação da latência são definidas por Lange et al. (2015b) e têm a finalidade de prover informação sobre a conectividade (i) entre cada *switch* e o controlador ao qual está associado e (ii) entre os controladores. Havendo mais de um controlador, considera-se para fins de cálculo que cada *switch* deverá associar-se ao controlador mais próximo em termos de latência.

Neste caso, a tarefa de otimização está em determinar um conjunto de posições $\mathcal{P}$ para conectar os k controladores de forma que a latência seja minimizada.

A menor latência entre cada par de nós compõe a matriz de latências D, onde d_{v_x,v_y} representa a menor latência entre o nó v_x e o nó v_y, medida em um determinado momento estático, e está associada única e exclusivamente ao atraso de propagação, diretamente relacionado à distância entre os dois nós. Os valores em D são normalizados pelo diâmetro do grafo.

Dada a matriz de latência D, a latência entre os nós e o(s) controlador(es) pode ser mensurada pelo cálculo da média π_{avg}^{LatN2C} sobre as latências de todos os nós para o controlador mais próximo p, na posição examinada, ou selecionando o valor máximo π_{max}^{LatN2C}, conforme as Equações (4.1) e (4.2).

$$\pi_{avg}^{LatN2C}(\mathcal{P}) = \frac{1}{|V|} \sum_{v \in V} \big(\min_{p \in \mathcal{P}} d_{v,p} \big), \tag{4.1}$$

$$\pi_{max}^{LatN2C}(\mathcal{P}) = \max_{v \in V} \min_{p \in \mathcal{P}} d_{v,p}. \tag{4.2}$$

A latência entre os controladores, por sua vez, pode ser mensurada calculando-se a média das latências entre todos os pares p_1, p_2 de controladores π_{avg}^{LatC2C} ou selecionando o valor máximo π_{max}^{LatC2C}, conforme as Equações (4.3) e (4.4).

$$\pi_{avg}^{LatC2C}(\mathcal{P}) = \frac{1}{\binom{|\mathcal{P}|}{2}} \sum_{p_1,p_2 \in \mathcal{P}} d_{p_1,p_2}, \tag{4.3}$$

$$\pi_{max}^{LatC2C}(\mathcal{P}) = \max_{p_1,p_2 \in \mathcal{P}} d_{p_1,p_2}. \tag{4.4}$$

4.2.1.2 Balanceamento de Carga

De acordo com Yao et al. (2014), a carga de um controlador SDN é formada por quatro componentes: processamento de eventos OFPT_PACKET_IN e entrega desses eventos para as aplicações, manutenção da visão do domínio local da rede, comunicação com os outros controladores para formar a visão global da rede e instalação das entradas de fluxo geradas pelas aplicações.

A avaliação do balanceamento de carga entre os controladores é definida por Lange et al. (2015b) de forma a minimizar a diferença entre o número de nós associados a cada controlador,

tal que n_p indica o número total de nós associados ao controlador p em uma dada posição em $\mathcal{P}$. Para tanto, π_{balanc} indica a diferença em valores de n_p entre os controladores com menor e maior número de nós associados, ou seja,

$$\pi_{balanc}(\mathcal{P}) = \max_{p \in \mathcal{P}} n_p - \min_{p \in \mathcal{P}} n_p \tag{4.5}$$

A avaliação do balanceamento de carga entre os controladores pode ainda considerar que cada nó v possui um peso w_v capaz de representar a importância do nó ou a quantidade de tráfego gerado. O custo total dos nós associados a p, definido por c_p, é obtido pela Equação (4.6).

$$c_p = \sum_{v \in V} w_v \tag{4.6}$$

Desta forma,

$$\pi_{balanc}(\mathcal{P}) = \max_{p \in \mathcal{P}} c_p - \min_{p \in \mathcal{P}} c_p. \tag{4.7}$$

4.2.1.3 Taxa de transmissão

As funções objetivo definidas para avaliação da taxa de transmissão são propostas neste trabalho e têm a finalidade de prover informação sobre a capacidade de transmissão (i) entre cada *switch* e o controlador ao qual está associado e (ii) entre os controladores. Havendo mais de um controlador, considera-se para fins de cálculo que cada *switch* deverá associar-se ao controlador que apresentar maior taxa de transmissão no caminho da comunicação.

Diferentemente das funções objetivo apresentadas anteriormente, neste caso a tarefa de otimização está em determinar um conjunto de posições $\mathcal{P}$ para conectar os k controladores de forma que a taxa de transmissão seja maximizada.

A taxa de transmissão entre cada par de nós compõe a matriz de taxa de transmissão B, onde b_{v_x,v_y} representa a maior taxa entre nó v_x e o nó v_y, medida em um determinado momento estático, considerando todos os caminhos possíveis entre eles. É importante ressaltar que, quando o nó v_x e o nó v_y não são diretamente conectados, a taxa de transmissão entre eles é definida pelo menor valor entre dois nós diretamente conectados no caminho da transmissão. Os valores em B são normalizados pelo diâmetro do grafo.

Dada a matriz de taxa de transmissão B, a taxa de transmissão entre os nós e o(s) controlador(es) pode ser mensurada pelo cálculo da média π_{avg}^{BwN2C} através da taxa de transmissão de todos nós para o controlador mais próximo p na posição examinada ou selecionando o valor mínimo π_{min}^{BwN2C}, conforme as Equações (4.8) e (4.9).

$$\pi_{avg}^{BwN2C}(\mathcal{P}) = \frac{1}{|V|} \sum_{v \in V} (\max_{p \in \mathcal{P}} b_{v,p}), \tag{4.8}$$

$$\pi_{min}^{BwN2C}(\mathcal{P}) = \min_{v \in V} \max_{p \in \mathcal{P}} b_{v,p}. \tag{4.9}$$

A taxa de transmissão entre os controladores, por sua vez, pode ser mensurada calculando-se a média das taxas entre todos os pares p_1, p_2 de controladores π_{avg}^{BwC2C} ou selecionando o valor mínimo π_{min}^{BwC2C}, conforme as Equações 4.10 e 4.11.

$$\pi_{avg}^{BwC2C}(\mathcal{P}) = \frac{1}{\binom{|\mathcal{P}|}{2}} \sum_{p_1,p_2 \in \mathcal{P}} b_{p_1,p_2}, \tag{4.10}$$

$$\pi_{min}^{BwC2C}(\mathcal{P}) = \min_{p_1,p_2 \in \mathcal{P}} b_{p_1,p_2}. \tag{4.11}$$

4.2.2 Implementação do Método Exaustivo

A implementação da análise exaustiva desenvolvida no POCO consiste em calcular os valores das funções objetivo de cada conjunto de posições $\mathcal{P}$ para conectar os k controladores em uma rede contendo n nós, de forma que cada conjunto de posição define uma solução possível do problema, gerando assim $\binom{n}{k}$ soluções possíveis. As soluções são então comparadas a fim de se encontrar a fronteira de Pareto, conforme Algoritmo 4.1. Definem-se como entradas deste algoritmo o grafo da rede G e o número de controladores k.

Algoritmo 4.1: Exaustivo

Entrada: $G = (V, E)$, k
$n = |V|$;
S = combinaPosições(n,k);
$F1$ = obtemFronteiraPareto(S);
Resultado: $F1$ e as posições correspondentes

Por padrão, o método exaustivo implementado no POCO permite obter a fronteira de Pareto para até quatro funções objetivo diferentes, conforme opções selecionadas pelo usuário. Neste trabalho, essa análise foi estendida para J funções objetivo implementando-se o método AARS, desenvolvido por Goldberg (1989) para identificar o conjunto de soluções não-dominadas. Este método é descrito pelo Algoritmo 3.1.

4.2.3 Implementação do PSA

A implementação do PSA no POCO para o problema do posicionamento dos controladores foi proposta por Lange et al. (2015b) com base no trabalho desenvolvido por Czyzak e Jaszkiewicz (1998). A fundamentação teórica do PSA é apresentada na Seção 3.3.1 e sua operação para o problema do posicionamento dos controladores é apresentada no Algoritmo 4.2.

Definem-se como entradas o grafo da rede G, o número de controladores k e os parâmetros do PSA, que incluem a temperatura inicial T_0 , a taxa de redução da temperatura ρ, o número

de iterações por nível de temperatura m e o número de soluções a serem avaliadas durante cada iteração N. Cada solução corresponde a um conjunto de posições $\mathcal{P}$ para conectar os k controladores. Define-se também um conjunto de soluções S contendo N conjuntos de posições escolhidos aleatoriamente.

Para cada solução contida em S, são calculados os valores das funções objetivo e é associado um vetor de pesos $\mathbf{\Lambda} = [\lambda_1, \lambda_2, ..., \lambda_J]$ aleatório e independente, onde J indica o número de objetivos. O vetor de pesos é responsável por ativar dispersão sobre todas as regiões do conjunto não-dominado através da troca de informações sobre as soluções. A fronteira de Pareto atualizada a cada iteração é definida por F, sendo inicialmente composta por S.

O processamento do algoritmo inicia a uma temperatura $T = T_0$, e ao final de m iterações a temperatura T é decrementada por uma taxa de resfriamento ρ, até que seja menor ou igual a 1. A cada iteração, são geradas novas soluções vizinhas às contidas em S, que são armazenadas em Q. Para que as soluções sejam consideradas vizinhas, elas podem diferenciar em até $\lceil \frac{k}{2} \rceil$ posições. Após cada geração, as soluções vizinhas são integradas a F e os vetores de pesos $\mathbf{\Lambda}$ são recalculados, de forma que o vetor de pesos associado à solução s seja modificado para aumentar a probabilidade de afastar-se da solução vizinha mais próxima q, aumentando-se os pesos dos objetivos para os quais s é melhor que q e diminuindo-se os pesos para os objetivos para os quais s é pior que q.

Uma solução de S é substituída pela solução vizinha em Q com probabilidade $P(S, Q, T, \mathbf{\Lambda})$, tal que o resultado desta probabilidade é igual a 1 quando a solução vizinha constitui uma melhoria e decrementado para indicar uma piora em função da quantidade de deterioração (determinada pela diferença entre S e Q), da temperatura T e da matriz de pesos $\mathbf{\Lambda}$ em cada iteração.

Algoritmo 4.2: Pareto Simulated Anneling

Entrada: $G = (V, E)$, k, N, m, T_0, ρ
$n = |V|$;
S = geraPosiçõesRandomicamente(n,N,k);
$\mathbf{\Lambda}$ = geraPesosRandomicamente(S);
F = fronteiraPareto(avalia(S));
$T = T_0$;
enquanto $T > 1$ **faça**
  Q = defineVizinhos$\left(S,\ N, \left\lceil \frac{kT}{2T_0} \right\rceil \right)$;
  atualizaFronteiraPareto (F,Q);
  $\mathbf{\Lambda}$ = atualizaPesos(S);
  S:= aceita $q \in Q$ com probabilidade $P(S, Q, T, \mathbf{\Lambda})$;
  se *m iterações forem efetuadas em T* **então**
    $T = \rho \cdot T$
Resultado: F e as posições correspondentes

4.2.4 Implementação do NSGA II

A implementação do NSGA-II no POCO para o problema do posicionamento dos controladores é proposta neste trabalho com base no algoritmo desenvolvido por Deb et al. (2002) e na implementação apresentada pelos desenvolvedores do YARPIZ (2018). A fundamentação teórica

do NSGA-II é apresentada na Seção 3.3.2 e sua operação para o problema do posicionamento dos controladores é apresentada no Algoritmo 4.3.

Definem-se como entradas do algoritmo o grafo da rede G, número de controladores k, o tamanho da população N, o número máximo de iterações m, a taxa de cruzamento t_c e a taxa de mutação t_m. Cada solução, neste contexto denominada por indivíduo, corresponde a um conjunto de posições $\mathcal{P}$ para conectar os k controladores. O conjunto de soluções é conhecido por população S. Por fim, F define as fronteiras de Pareto obtidas a cada iteração.

Para cada indivíduo, são calculados os valores das funções objetivo. A partir da população inicial S_0, composta por N indivíduos escolhidos aleatoriamente, cada indivíduo é comparado aos demais a fim de verificar a relação de dominância entre eles e assim ordená-los em níveis de classificação de não-dominância, ou fronteiras. Este procedimento constitui no método *Fast Non-Dominated Sorting*, descrito pelo Algoritmo 3.4.

A dispersão entre os indivíduos contidos em uma mesma fronteira F é mantida pela distância de aglomeração, calculada para cada um de seus indíviduos conforme o Algoritmo 3.5.

Algoritmo 4.3: NSGA-II

Entrada: $G = (V, E)$, k, N, t_c, t_m,m
$n = |V|$;
$n_c = t_c.N$;
$n_m = t_m.N$;
S = geraPosiçõesRandomicamente(n,N,k);
[S,F] = fast-non-dominated-sorting(S);
S = crowding-distance-assignment(S,F);
[S,F] = ordenaPopulação(S);
para t = *1:m* **faça**
  para i = *1:*n_c **faça**
    p_1 = selecionaTorneio(S);
    p_2 = selecionaTorneio(S);
    [SC_i,SC_{i+1}] = recombina(p_1,p_2);
  para i = *1:*n_m **faça**
    p_3 = selecionaTorneio(S);
    SM_i = muta(p_3);
  S = [S,SC,SM];
  [S,F] = fast-non-dominated-sorting(S);
  S = crowding-distance-assignment(S,F);
  S = ordenaPopulação(S);
  $S = S(1 : N)$;
  [S,F] = fast-non-dominated-sorting(S) ;
  S = crowding-distance-assignment(S,F);
  [S,F] = ordenaPopulação(S);
  $F1 = S(F\{1\})$;
Resultado: $F1$ e as posições correspondentes

A seleção dos indivíduos da população a serem utilizados nos processos de cruzamento e mutação é realizada através da técnica de torneio com base na aptidão de cada indivíduo, tal que

a aptidão de um determinado indivíduo $s_1 \in S$ em relação a um indivíduo $s_2 \in S$ é avaliada de acordo com o nível de não dominância no qual o indivíduo foi classificado pelo método *Fast Non-Dominated Sorting* e com a distância de aglomeração.

Após selecionados os indivíduos, tal que cada indivíduo corresponde a um conjunto de posições $\mathcal{P}$ para conectar os k controladores, o cruzamento é realizado mantendo-se as posições unânimes entre os indivíduos selecionados e escolhendo aleatoriamente as demais posições dentre as restantes contidas nestes. A mutação, por sua vez, é realizada substituindo-se aleatoriamente posições de um indivíduo selecionado.

A cada iteração, $t_c.N$ cruzamentos e $t_m.N$ mutações são geradas. Os indivíduos gerados são concatenados à população atual, a população resultante é reclassificada em níveis de não-dominância e a distância de aglomeração é recalculada. De acordo com a ordem de classificação das melhores soluções, os N primeiros indivíduos são selecionados para serem a população da geração (iteração) seguinte e são organizados em fronteiras de acordo com o nível de classificação de não-dominância. A fronteira de Pareto será determinada pela fronteira cuja classificação de não-dominância for igual a 0, ao final de todas as iterações.

4.3 Ambiente de emulação de rede

4.3.1 Mininet

O Mininet é um emulador de redes que permite criar redes personalizadas de *hosts*, *switches*, roteadores e enlaces em um único GNU/Linux Kernel. Para tanto, ele se baseia em virtualização do tipo "*lightweight*" que faz com que um único sistema seja visto como uma rede completa, executando o mesmo Kernel, sistema e código de usuário.

Os *hosts*, *switches*, enlaces e controladores virtuais são criados usando apenas o *software* ao invés do *hardware*, em sua maior parte, seu comportamento é semelhante aos elementos de *hardware* discretos. Um *host* virtual se comporta como uma máquina real e os programas nele executados podem enviar pacotes através do que parece ser uma *interface* Ethernet real, dada a velocidade e a latência do enlace. Os pacotes são processados pelo que parece ser um *switch* Ethernet, roteador ou *middlebox* real, dada a capacidade de enfileiramento.

Em geral, é possível criar uma rede Mininet que se assemelhe a uma rede composta por *hardwares* ou a uma rede composta por *hardwares* que se assemelhe a uma rede Mininet, e executar o mesmo código binário e aplicativos em ambas plataformas.

O Mininet é um projeto de código aberto em desenvolvimento que promete ser fácil e rápido para executar, editar e depurar. Além de permitir a programação dos *switches* por meio do protocolo OpenFlow, ele é capaz de executar programas reais, desde servidores *web* até ferramentas de monitoração. O comportamento do encaminhamento dos pacotes é definido pelo controlador com as funcionalidades desejadas.

4.3.1.1 Topologias no Mininet

Atualmente, a *Internet Topology Zoo* (ITZ) (KNIGHT et al., 2011) é a coleção mais conhecida de topologias de redes reais disponibilizadas publicamente pelos seus operadores. Todas as topologias são definidas em formato gráfico GML (do inglês, *Graphical Modelling Language*) e GraphML, e fornecem informações suficientes para criar redes de teste tendo como base topologias do mundo real (GROßMANN; SCHUBERTH, 2013). Dentre as informações fornecidas têm-se rótulos, conexão entre os nós, posições de longitude e latitude e descrições de localização.

No Mininet, por sua vez, as topologias de rede são definidas em Python. Por este motivo, para construir uma rede fornecida pelo ITZ no Mininet, é necessário converter um arquivo de entrada GML ou GraphML em uma topologia definida em Python.

Neste contexto, Großmann e Schuberth (2013) definem um mecanismo conhecido como "Auto-Mininet" que permite construir uma rede de acordo com a sintaxe do Mininet baseada no formato gráfico GraphML do ITZ.

Na Fig. 4.2 Großmann e Schuberth (2013) apresentam um exemplo da conversão do formato GraphML para a sintaxe em Python utilizada pelo Mininet. O arquivo GraphML, à esquerda, mostra os elementos importantes para construir a rede. Cada elemento é identificado como "*key*", e contém no campo "*attr.name*" informações relevantes a seu respeito. Para construir a rede no Mininet, é essencial que cada "*node*" contenha entre os seus elementos o rótulo para o nó, a longitude e a latitude. A conexão entre os nós é definida por "*edge*", e pode conter entre seus elementos informações sobre a taxa de transmissão, latência e taxa de perda de pacote.

```
<?xml version="1.0" encoding="utf-8"?>
<graphml ...>
  <key attr.name="key" attr.type="int" for
     ="edge" id="d36" />
  ...
  <graph edgedefault="undirected">
    <data key="d0">2/01/11</data>
    <data key="d1">Germany</data>
    <data key="d2">Country</data>
    <data key="d3">DFN</data>
    ...
    <data key="d28">1</data>
    <node id="0">
      <data key="d29">1</data>
      <data key="d30">50.83333</data>
      <data key="d31">Germany</data>
      <data key="d32">0</data>
      <data key="d33">12.91667</data>
      <data key="d34">CHE</data>
    </node>
    ...
    <edge source="0" target="1">
      <data key="d35">e52</data>
      <data key="d36">0</data>
    </edge>
    ...
  </graph>
</graphml>
```

```
#!/usr/bin/python

from mininet.topo import Topo
...

class GeneratedTopo( Topo ):
    def __init__( self, **opts ):
        # Initialize Topology
        Topo.__init__( self, **opts )
        # switches first
        CHE = self.addSwitch( 's0' )
        ...
        # and now hosts
        CHE_host = self.addHost( 'h0' )
        ...
        # add edges between switch and
            corresponding host
        self.addLink( CHE , CHE_host )
        ...
        # add edges between switches
        self.addLink( CHE , LEI, bw=10,
            delay='0.348009502ms')
        ...
topos = { 'generated': ( lambda:
   GeneratedTopo() ) }
...
if __name__ == '__main__':
    sshd( setupNetwork() )
```

Figura 4.2: À esquerda, são apresentados os valores importantes do formato GraphML. À direita, é apresentada a transformação do arquivo em Python para o Mininet. Fonte: Großmann et al. (2013)

Durante a conversão, alguns valores podem ser omitidos. No caso da taxa de transmissão,

quando omitida, é inicializada automaticamente como 10Mbps. No caso do IP do controlador remoto, ele é inicializado como "10.0.2.2". A latência para o Mininet é calculada a partir das coordenadas geográficas fornecidas pela topologia em formato gráfico. Desta forma, o atraso de propagação estimado t_L é calculado por:

$$t_L = \frac{dist}{v_L}, \tag{4.12}$$

onde *dist* define a distância geográfica entre dois nós e v_L define a velocidade do sinal (EITTENBERGER; GROßMANN; KRIEGER, 2012).

A distância geográfica entre dois nós é calculada pela Lei Esférica dos Cosenos com base nas coordenadas geográficas fornecidas, desconsiderando-se desvios no caminho da infraestrutura, tal que:

$$dist = acos\{sin(\varphi_1) \cdot sin(\varphi_2) + cos(\varphi_1) \cdot cos(\varphi_2) \cdot cos(\Delta\psi)\}.r, \tag{4.13}$$

onde φ define a latitude em radianos, ψ define a longitute em radianos e r é o raio da terra, tal que $r = 6371 \cdot 10^3 m$.

Para uma topologia composta por enlaces de fibra ótica, a velocidade do sinal é aproximada à velocidade da luz por um fator reflexivo de 1.52 inerente à fibra ótica. Nessas condições, $v_L = \frac{3.10^8}{1.52} = 1.97 \cdot 10^8$ m/s.

4.3.2 Floodlight

O Floodlight é um controlador SDN implementado em código aberto como parte do Projeto OpenDaylight, escrito em Java e licenciado pela Apache. Ele oferece recursos e abstrações necessárias para o controle de uma rede OpenFlow, por meio de uma arquitetura modular e um conjunto de aplicações básicas.

Através do módulo *Link-discovery*, tem-se um mecanismo de detecção automática de topologia baseado no uso de pacotes LLDP (*Link-Layer Discovery Protocol*) e *broadcast*, que permitem construir o mapa topológico da rede na forma de um grafo. O protocolo LLDP (802.1AB) opera na camada de enlace do modelo OSI e permite que informações básicas como *hostname*, versão do Sistema Operacional, endereço da interface, entre outras, sejam aprendidas dinamicamente entre equipamentos de rede de diversos fabricantes diretamente conectados, como servidores, *switches* e roteadores.

Desta forma, o controlador mantém todas as informações relevantes a respeito da interconectividade entre os *switches*, que podem ser usadas por outras aplicações. Para tanto, o controlador Floodlight utiliza interfaces de programação de aplicação de transferência de estado representativo (REST APIs) que permite a sua iteração com diferentes plataformas.

Testado com *switches* físicos e virtuais compatíveis com OpenFlow, o Floodlight Controller pode

funcionar em uma variedade de ambientes e suportar redes onde grupos de *switches* compatíveis com OpenFlow são conectados através de *switches* tradicionais.

4.3.3 Operação de rede

Para avaliar a operação básica de uma rede SDN implementada no Mininet, foi utilizada a topologia apresentada na Fig. 4.3, composta por um controlador Floodlight remoto (192.168.10.200) e duas estações de trabalho - 192.168.10.101 e 192.168.10.102, conectadas a um *switch* OVS (Open vSwitch, *switch* virtual implementado em código aberto independente de *hardware*) operando como um *switch* OpenFlow.

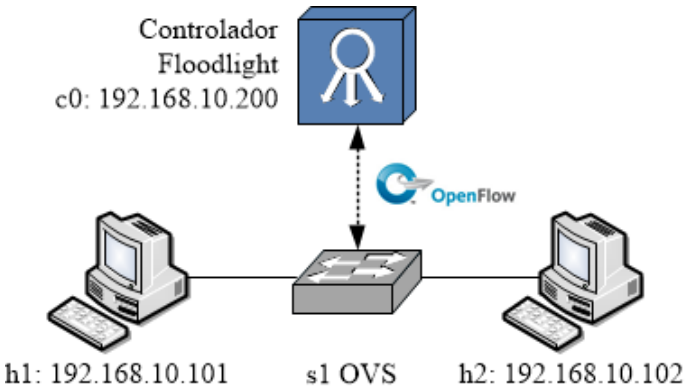


Figura 4.3: Topologia utilizada para avaliar a operação básica de uma rede SDN implementada no Mininet.

No Mininet é possível validar a topologia implementada através dos comandos:

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

```
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
```

```
mininet> dump
<Host h1: h1-eth0:192.168.10.101 pid=17737>
<Host h2: h2-eth0:192.168.10.102 pid=17739>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=17732>
<RemoteController c0: 192.168.10.200:6653 pid=17724>
```

A análise do tráfego de rede foi realizada através do Wireshark. O Wireshark é o analisador de protocolo de rede mais utilizado no mundo, de forma que permite capturar e analisar interativamente os pacotes transmitidos pelos dispositivos de uma rede em tempo de execução, incluindo

os pacotes OpenFlow na versão 1.0 (OF 1.0) transmitidos entre o *switch* OVS e o controlador, conforme Figura 4.4.

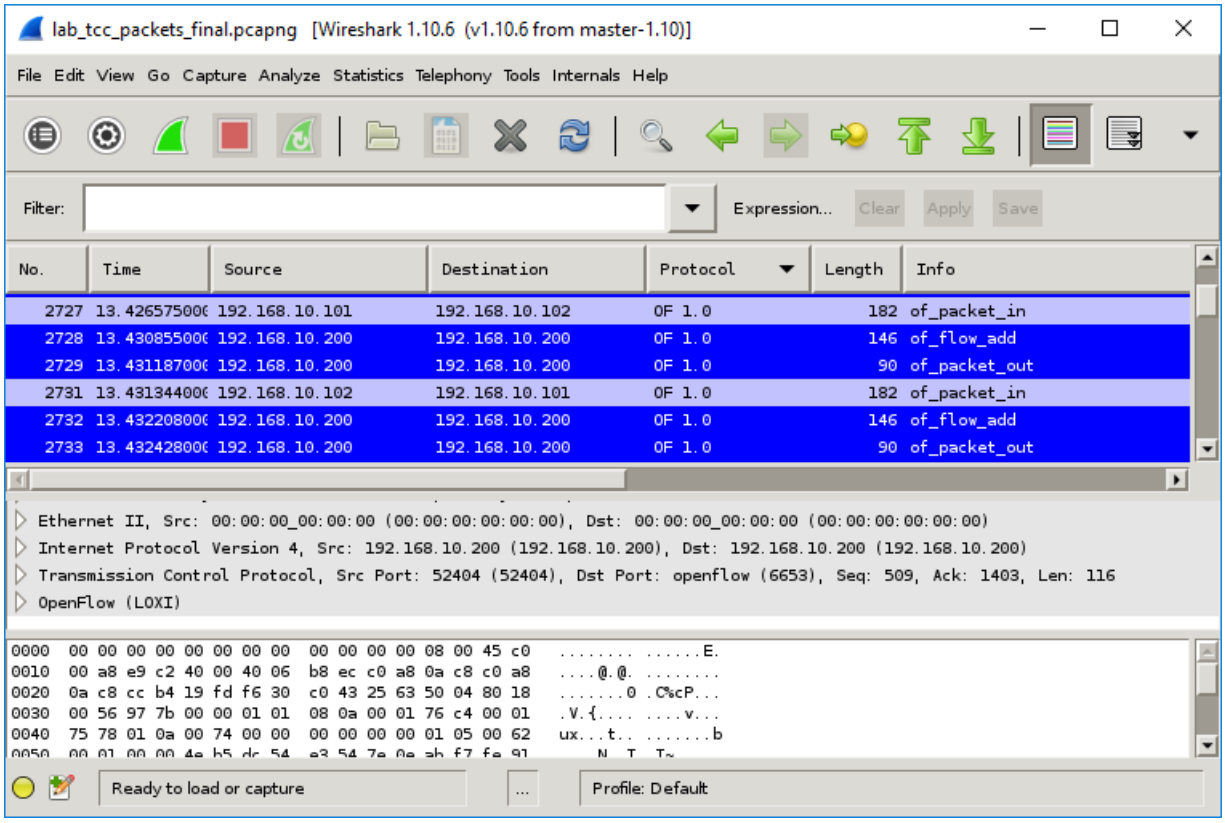


Figura 4.4: Wireshark - Pacotes OF 1.0 transmitidos entre o *switch* OVS e o controlador.

Através da análise dos pacotes capturados, é possível identificar que, ao iniciar um tráfego ICMP entre as estações h1 e h2, o pacote ICMP recebido pelo *switch* OVS é encaminhado para o controlador por meio de uma mensagem OFPT_PACKET_IN, apresentada na Fig. 4.5, contendo um identificador de *buffer* único igual a 258. Esta ação ocorre para todo primeiro pacote de um fluxo recebido por um *switch* OpenFlow cuja entrada correspondente à origem e ao destino do fluxo não seja encontrada na sua tabela de fluxo.

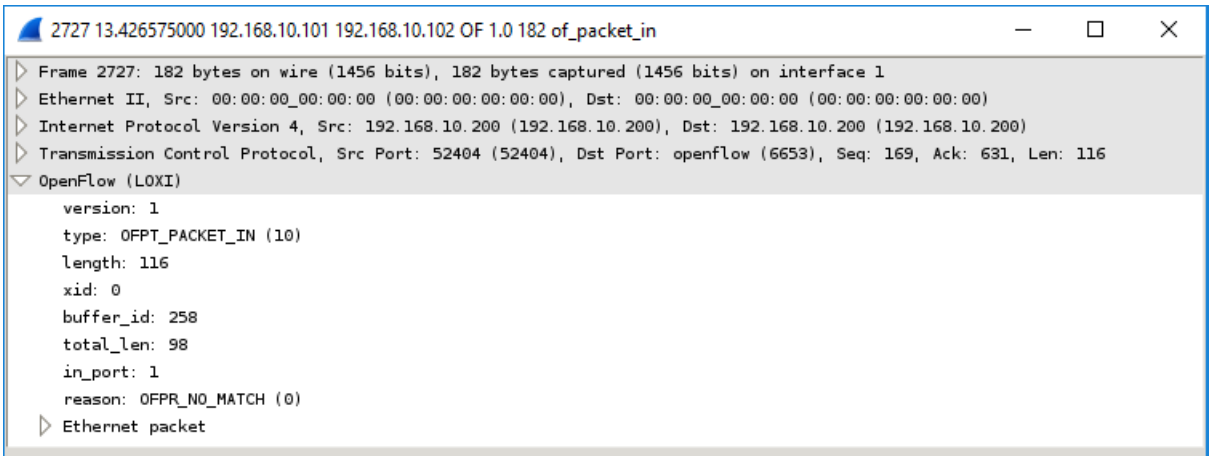


Figura 4.5: Frame 1: OFPT_PACKET_IN.

Após receber o pacote OFPT_PACKET_IN, o controlador utiliza uma mensagem OFPT_FLOW_MOD, apresentada na Fig. 4.6, para modificar a tabela de fluxo do *switch*. A primitiva OFPT_FLOW_MOD permite adicionar, atualizar ou remover uma entrada da tabela. Neste caso, é adicionada uma entrada relativa ao tráfego de 192.168.10.101 para 192.168.10.102 da porta 1 para a porta 2. Desta forma, pacotes similares serão enviados automaticamente pelo *switch* até que um *idle timeout* de 5 segundos seja excedido. Em seguida, o processo deve se repetir.

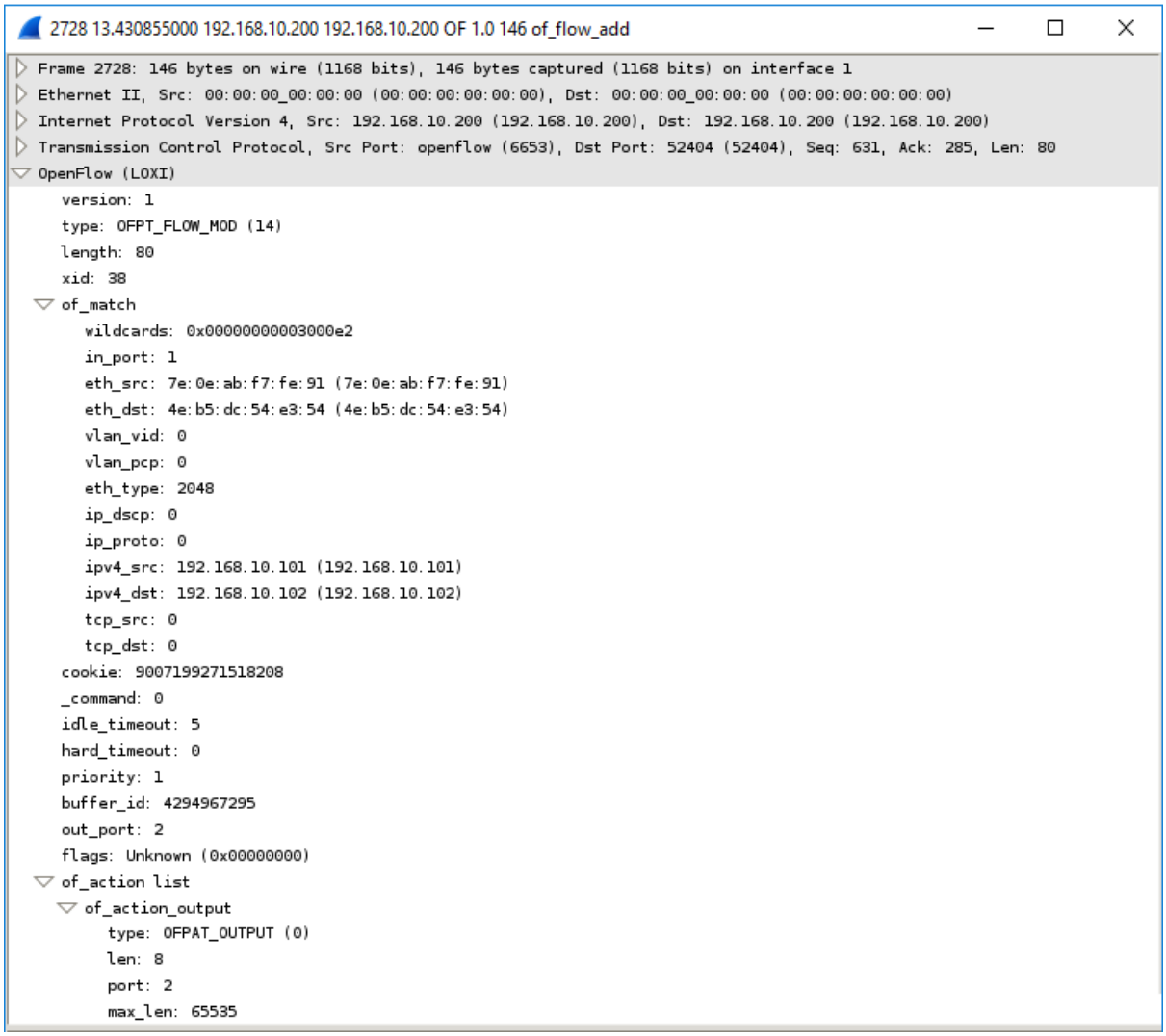


Figura 4.6: Pacote 2: OFPT_FLOW_MOD.

De maneira complementar, o controlador envia uma mensagem do tipo OFPT_PACKET_OUT, apresentada na Fig. 4.7, para o *switch* com as regras de fluxo. Neste caso, é possível verificar o identificador de buffer 258 com a decisão do tipo "*Output to switch port*" (OFPAT_OUTPUT) na porta 2.

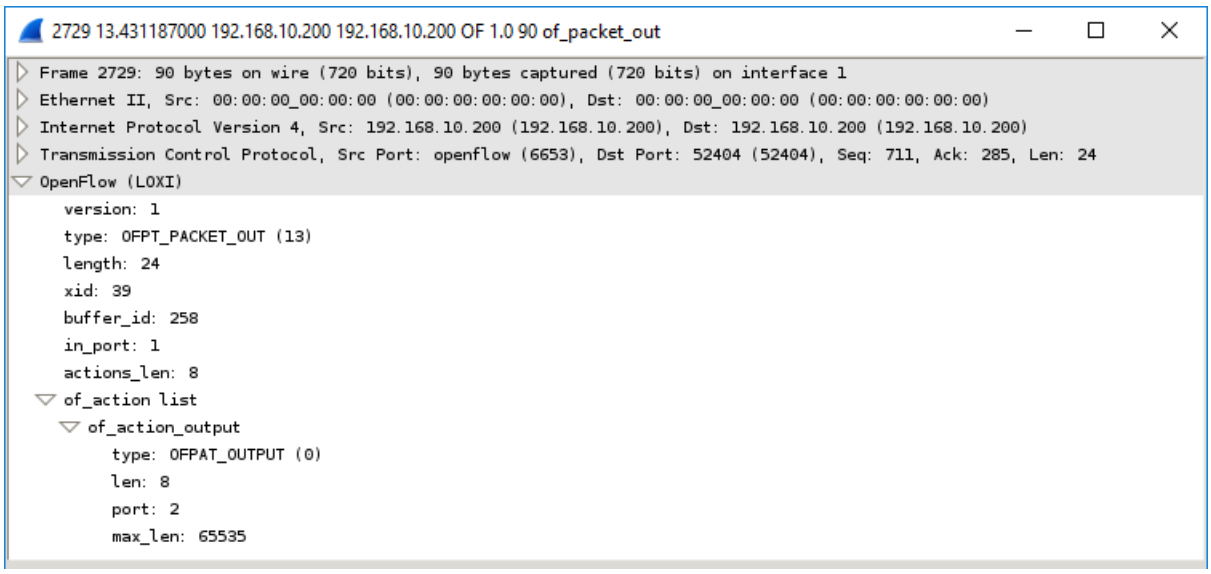


Figura 4.7: Pacote 3: OFPT_PACKET_OUT.

A Fig. 4.8 ilustra o fluxo de mensagens OpenFlow descrito.

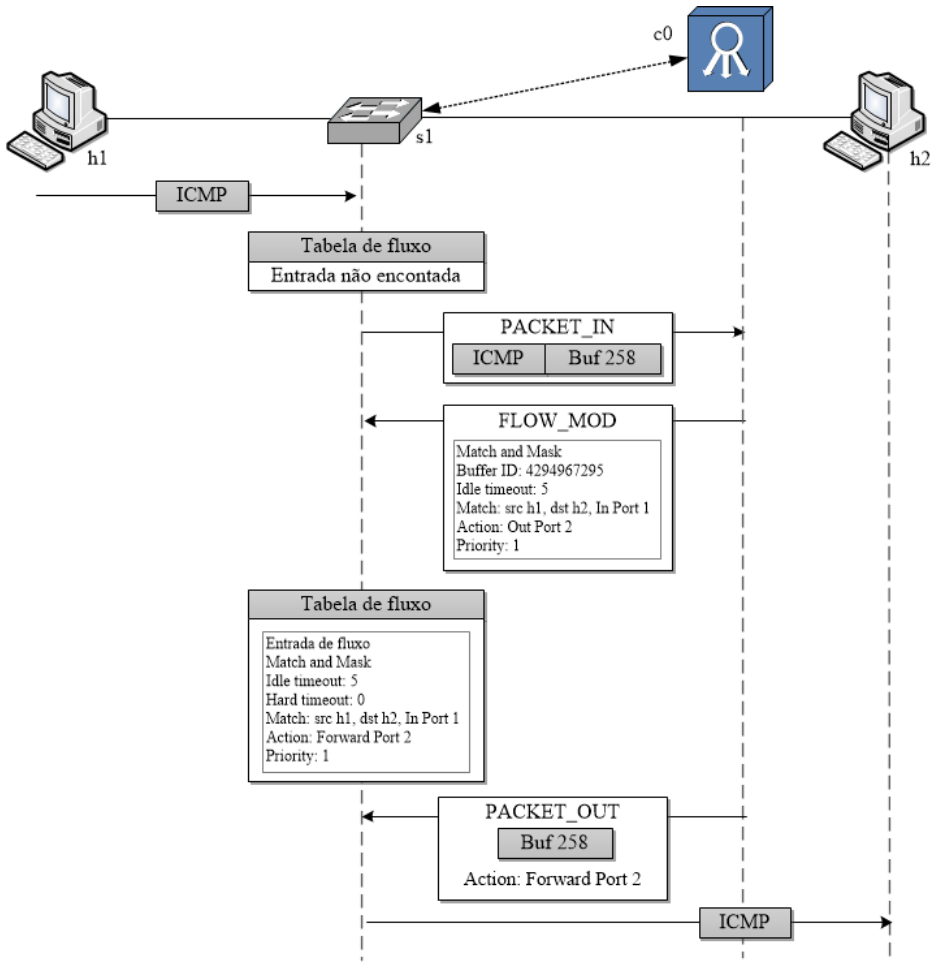


Figura 4.8: Fluxo de mensagens OpenFlow.

4.4 Considerações finais

Este capítulo apresentou o framework POCO como ferramenta para calcular o posicionamento ótimo de controladores no contexto da otimização combinatória multiobjetivo, tal que os objetivos de otimização mapeiam medidas de desempenho de rede a serem consideradas em redes SDN.

Para tanto, foi apresentada a formulação do problema e das medidas de desempenho relacionadas à latência, balanceamento de carga e taxa de transmissão. Este capítulo também descreveu os algoritmos implementados no POCO: método exaustivo, PSA e NSGA-II.

Para demonstrar o efeito do posicionamento ótimo calculado pelo POCO em um rede emulada em laboratório, propõe-se a utilização do Mininet com o controlador Floodlight. Os resultados de desempenho de rede obtidos para a rede emulada serão apresentados no Capítulo 5.

No Capítulo 5 também serão apresentados os resultados experimentais obtidos para avaliação do desempenho dos algoritmos PSA e NSGA-II.

Capítulo 5

Resultados Experimentais

5.1 Introdução

Neste Capítulo, demonstra-se o problema de posicionamento do controlador em uma rede SDN habilitada com OpenFlow emulada em laboratório, com um único controlador. Em seguida, são apresentados os resultados obtidos pelos métodos metaheurísticos PSA e NSGA-II a fim de determinar experimentalmente se esses são capazes de convergir para a fronteira de Pareto, previamente calculada pelo método exaustivo. Adicionalmente, é realizada uma comparação da precisão e do esforço computacional, no que diz respeito ao tempo de processamento.

5.2 Demonstração do problema

A fim de demonstrar o problema de posicionamento do controlador em redes SDN, foi construída uma rede habilitada com OpenFlow no Mininet, com base na topologia da rede Internet2 OS3E (KNIGHT et al., 2011), apresentada na Fig. 5.1.

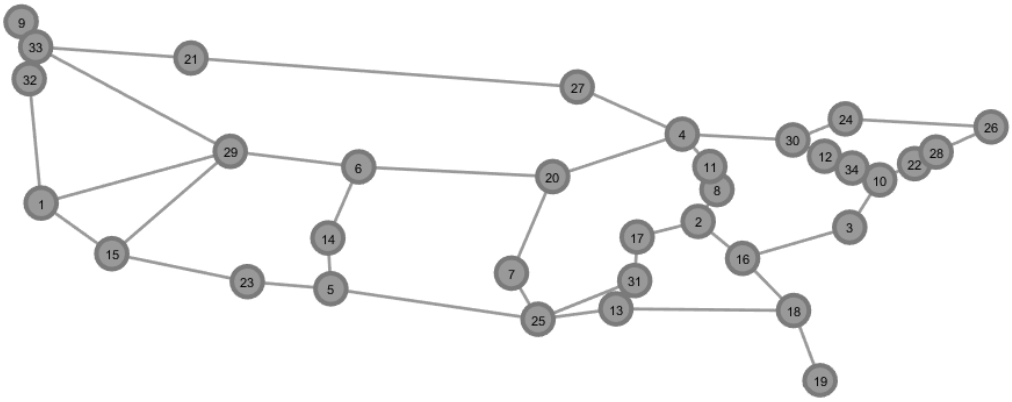


Figura 5.1: Rede Internet2 OS3E.

A topologia de rede, contendo 34 nós, foi convertida para uma topologia definida em Python conforme o mecanismo Auto-Mininet, apresentado na Seção 4.3.1.1. Assim como a maioria das

topologias disponíveis no ITZ (KNIGHT et al., 2011), a topologia utilizada não continha informações de taxa de transmissão das conexões. Neste caso, o Auto-Mininet define todas as conexões com um valor igual a 10Mbps. Entretanto, para imprimir maior realidade nas emulações, a taxa de transmissão entre os nós diretamente conectados foi alterada para valores no intervalo de 1Mbps a 10Mbps, de forma a impactar na taxa de transmissão de toda a rede, conforme a topologia definida em Python apresentada no Anexo I. A cada *switch* foi conectado um *host*.

A rede construída no Mininet foi então habilitada com um controlador Floodlight, operando *in-band*, conectado a um *switch*. No contexto do Mininet, quando um controlador é colocado operando *in-band* significa que ele está sendo emulado como parte da infraestrutura da rede construída, sem ter qualquer relação com o tipo de acesso utilizado para o tráfego de controle. Até o momento, a maioria dos trabalhos publicados em Mininet - Wiki (2018) se baseiam na utilização de um controlador remoto, externo ao controle do Mininet, de forma que sua operação é independente da infraestrutura da rede emulada e não permite avaliar o impacto do posicionamento do controlador.

Para utilizar o controlador *in-band*, é necessário configurar um mecanismo para evitar *loops* no plano de controle (IRAWATI; NURUZZAMANIRRIDHA, 2015; FAQ MININET, 2017). Neste sentido, foi habilitado o protocolo 802.1d Spanning Tree (STP, do inglês *Spanning Tree Protocol*) (IEEE, 1998) em todos os *switches*. O STP é um protocolo da camada de enlace que monitora e identifica eventuais conexões redundantes e, caso caminhos redundantes sejam identificados, elege um caminho como primário e desativa caminhos alternativos por meio do bloqueio de portas. Desta forma, o protocolo resulta em uma topologia em árvore. A topologia da rede Internet2 OS3E habilitada com o STP é vista pelo controlador Floodlight conforme apresentado na Fig. 5.2.

Em uma rede operando com o STP, um *switch* é definido como *root bridge* com base em uma prioridade configurável. Caso todos os equipamentos tenham a mesma prioridade, o *root bridge* é eleito com base no menor endereço MAC (do inglês, *Media Access Control*). A prioridade e o endereço MAC formam um identificador BID (do inglês, *Bridge ID*). A topologia lógica da rede comutada é então construída a partir do *root bridge*, responsável por identificar os caminhos redundantes. Os demais *switches* da rede são denominados *non-root bridges*, e a porta com menor custo até o *root bridge* determinará um enlace ativo na rede. Para o cálculo do STP, o custo do enlace é inversamente proporcional a sua taxa de transmissão, de forma que um menor custo indica uma taxa de transmissão mais elevada. Em caso de empate, opta-se pelo caminho com menor BID e, em caso de novo empate, pelo caminho pela cuja porta tenha menor numeração.

Dada a latência e a taxa de transmissão entre todos os nós da topologia construída em laboratório, essas informações foram utilizadas pelo POCO para avaliar o posicionamento ótimo do controlador. Para fins de otimização, todas as funções objetivo são consideradas. Entretanto, quando se trata de apenas um controlador, as funções objetivo responsáveis pelas medidas de desempenho entre controladores são nulas, ou seja, $\pi_{avg}^{LatC2C} = \pi_{max}^{LatC2C} = \pi_{balanc} = \pi_{avg}^{BwC2C} = \pi_{min}^{BwC2C} = 0$. Neste caso, a tarefa de otimização está em determinar uma posição para conectar um controlador de forma que a latência entre os nós e o controlador seja minimizada e a taxa de transmissão entre os nós e o controlador seja maximizada, medidas essas definidas pelas funções objetivo π_{avg}^{LatN2C}, π_{max}^{LatN2C}, π_{avg}^{BwN2C} e π_{min}^{BwN2C}.

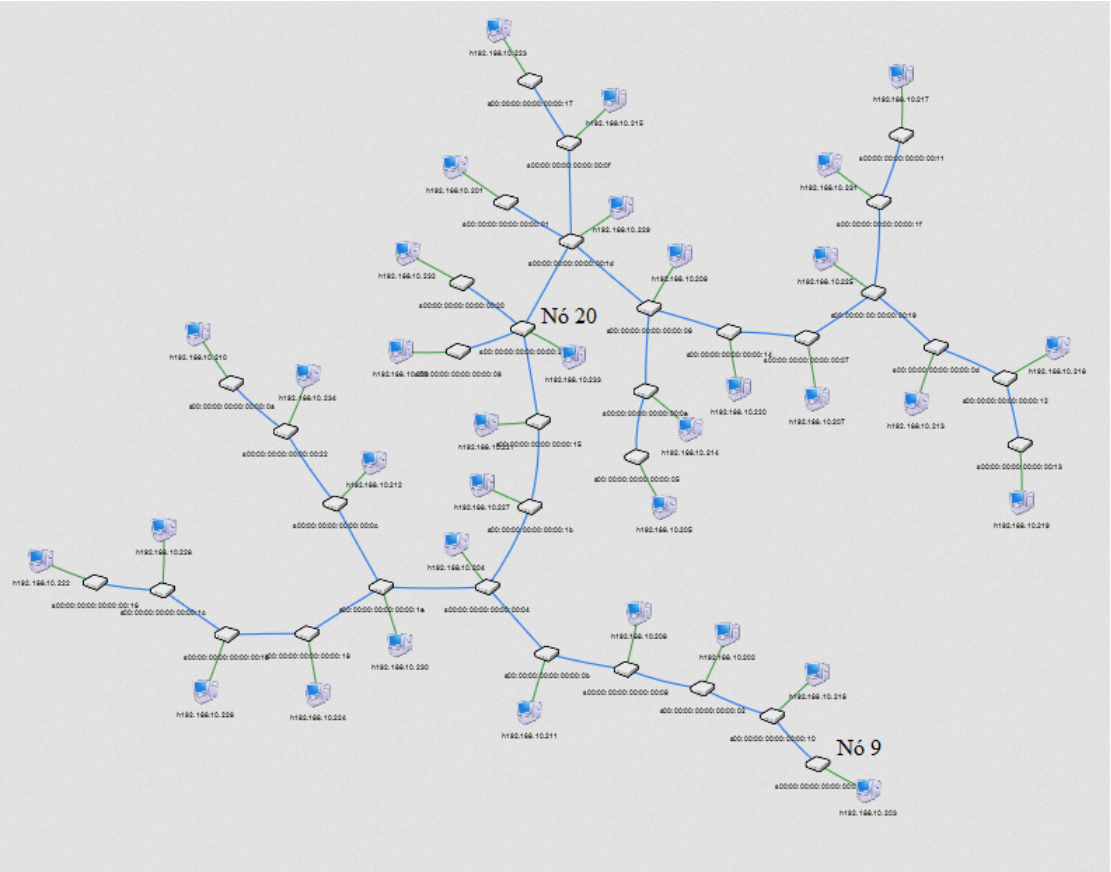


Figura 5.2: Topologia habilitada com o STP vista pelo controlador Floodlight.

A avaliação no POCO através do método exaustivo resultou na fronteira de Pareto apresentada na Fig. 5.3.

Para demonstrar o impacto do posicionamento do controlador na operação da rede, a partir da otimização das medidas de desempenho de rede definidas pelas funções objetivo π_{avg}^{LatN2C}, π_{max}^{LatN2C}, π_{avg}^{BwN2C} e π_{min}^{BwN2C}, foram avaliados dois cenários: (a) posição ótima e (b) pior posição. Neste caso, por posição ótima entende-se o posicionamento no qual a latência entre os nós e o controlador é minimizada e a taxa de transmissão entre os nós e o contrador é maximizada. Por pior posição, entende-se o posicionamento para o qual tem-se maior latência e menor taxa de transmissão entre os nós e o controlador. Em ambos os cenários foram considerados valores estáticos medidos para a rede emulada em laboratório. Os Anexos II e III apresentam os valores estáticos de taxa de transmissão e latência entre todos os nós, respectivamente.

A partir da fronteira de Pareto apresentada pela avaliação no POCO através do método exaustivo, optou-se pelo nó 20 como posição ótima e pelo nó 9 como pior posição, tal que a escolha de uma solução a partir da fronteira de Pareto deve ser realizada pelo administrador de rede, conforme uma política por ele definida. Os valores das funções objetivo nos cenários (a) e (b) são apresentados na Tabela 5.1, tal que os valores de π_{avg}^{LatN2C} e π_{max}^{LatN2C} são menores e os valores de π_{avg}^{BwN2C} e π_{min}^{BwN2C} são maiores no cenário (a), em que o controlador é conectado a um *switch* em uma posição ótima.

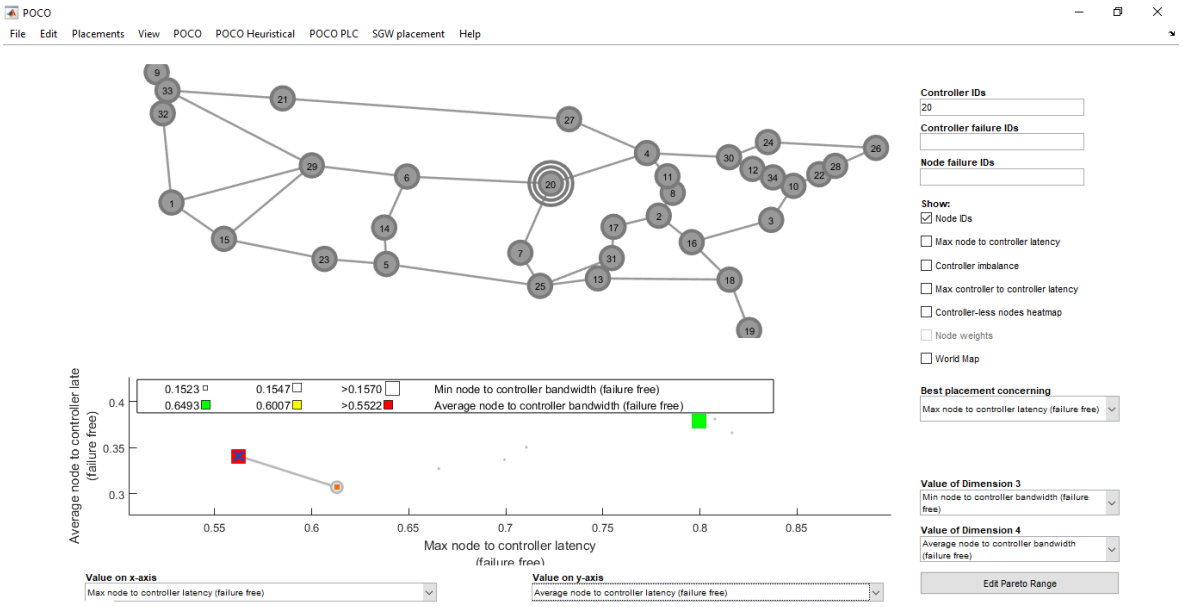


Figura 5.3: Tela contendo o resultado apresentado pelo POCO.

Tabela 5.1: Valores das funções objetivo para os cenários: (a) posição ótima e (b) pior posição

Funções objetivo	**(a) Posição ótima: Nó 20**	**(b) Pior posição: Nó 9**
π_{avg}^{LatN2C}	0.340886	0.6069
π_{max}^{LatN2C}	0.562358	0.9919
π_{avg}^{BwN2C}	0.662589	0.1935
π_{min}^{BwN2C}	0.157009	0.1561

A fim de avaliar o impacto da latência no desempenho da rede com o controlador em diferentes posições, foi analisado o tempo para enviar os pacotes de requisição de eco e o tempo para tê-lo de volta, métrica conhecida como *round-trip time* (RTT). Para tanto, foram gerados fluxos de dados entre todos os pares de *hosts*, tal que cada fluxo é composto por 5 pacotes do tipo ICMP (do inglês, *Internet Control Message Protocol*). O RTT dos pacotes dos fluxos gerados nos cenários (a) e (b) são apresentados nas Figs. 5.4 e 5.5, respectivamente.

Como apresentado na Seção 4.3.3, quando um fluxo é recebido por um *switch* OpenFlow cuja entrada correspondente à origem e ao destino não é encontrada na sua tabela de fluxo, o *switch* encaminha o primeiro pacote do fluxo para o controlador por meio de uma mensagem do tipo OFPT_PACKET_IN. Após receber o pacote, o controlador define a regra de encaminhamento a ser utilizada para o fluxo e encaminha uma mensagem do tipo OFPT_PACKET_OUT para o *switch*. Desta forma, espera-se que o primeiro pacote de cada fluxo apresente um RTT maior quanto mais distante, em termos de latência, estiver do controlador.

As Figs. 5.4 e 5.5 permitem verificar que, como esperado, em ambos os cenários o primeiro pacote de cada fluxo gerado apresenta um RTT maior em comparação aos pacotes subsequentes. A Tabela 5.2 resume os valores mínimo, médio e máximo do RTT dos primeiros pacotes nos cenários avaliados, tal que o valor médio e mínimo no cenário (a) foi aproximadamente 15% menor quando

comparado ao cenário (b), indicando uma vantagem em termos de desempenho de rede para o cenário (a). Se tratando do valor máximo, essa vantagem foi de 50%.

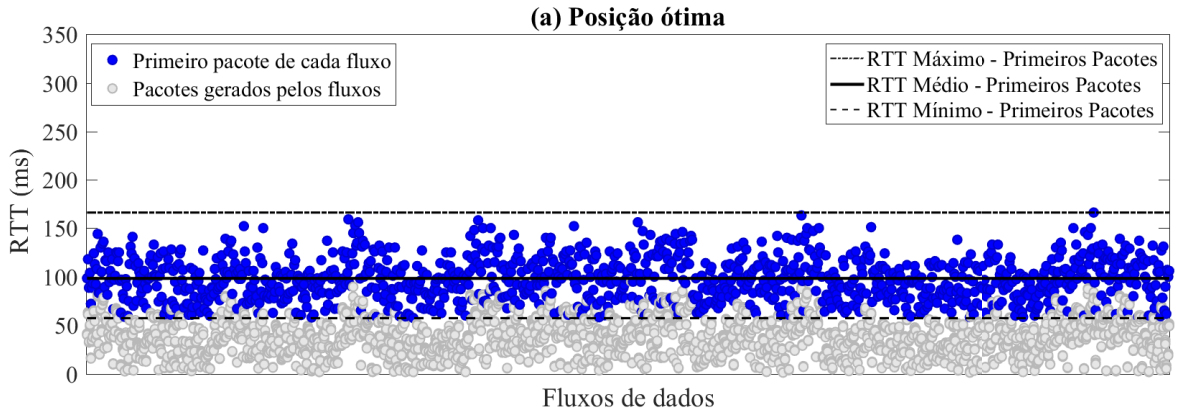


Figura 5.4: Fluxos de dados gerados entre todos os pares de *hosts* na topologia Internet2 OS3E para o cenário (a).

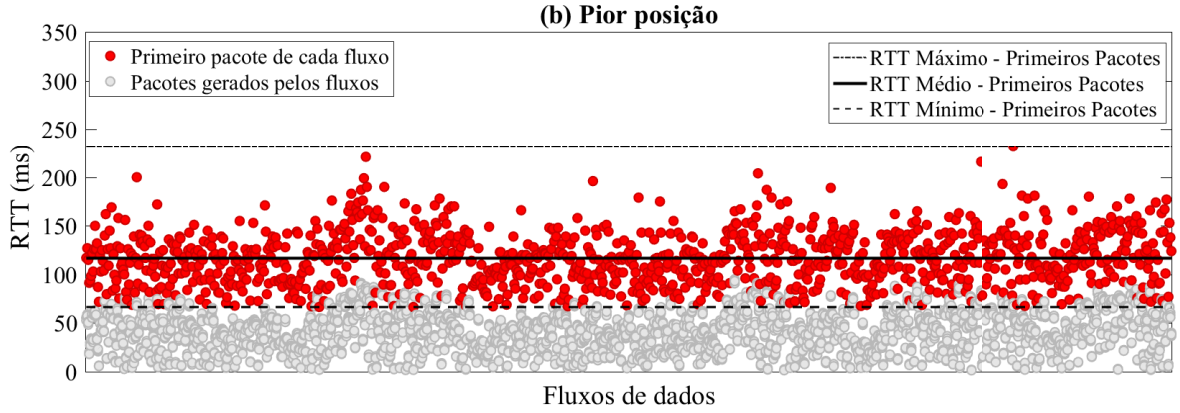


Figura 5.5: Fluxos de dados gerados entre todos os pares de *hosts* na topologia Internet2 OS3E para o cenário (b).

Tabela 5.2: RTT: (a) posição ótima e (b) pior posição

RTT - Primeiros pacotes	**(a) Posição ótima: Nó 20**	**(b) Pior posição: Nó 9**
Mínimo	57.4 ms	66.5
Médio	98.2745 ms	116 ms
Máximo	116 ms	232 ms

O histograma apresentado na Fig. 5.6 permite ainda avaliar a distribuição do RTT dos primeiros pacotes em ambos cenários. No cenário (b), o número de pacotes com RTT superior a 100ms ultrapassa 70% do total de pacotes, enquanto no cenário (a) cerca de 40% dos pacote alcança tal valor de RTT.

A taxa de transmissão entre um determinado nó e o controlador, por sua vez, irá depender da taxa de transmissão entre os nós diretamente conectados no caminho da comunicação, sendo limitada pelo enlace com menor taxa ao longo do caminho. Para medir a taxa de transmissão

entre o *switch* e o controlador em determinada posição é utilizado o iPerf, um gerador de tráfego que permite medir a qualidade da comunicação entre dois dispositivos finais. Para tanto, o iPerf é iniciado como servidor no controlador conectado ao *switch* na posição avaliada, e como cliente nos *hosts* conectados aos *switches* nas demais posições. O cliente envia tráfego TCP para o servidor por 10 segundos, e em seguida o iPerf apresenta a quantidade de dados transferida (MBytes) e a taxa de transmissão atingida (Mbits/s).

Os resultados obtidos para os cenários (a) e (b) são apresentados na Fig. 5.7, e mostram uma grande vantagem para o cenário (a), onde cerca de 45% dos *switches* alcançam o controlador com a taxa de transmissão máxima da rede, 10Mbps, o que permite uma rede mais reativa, como discutido na Seção 2.2.3.

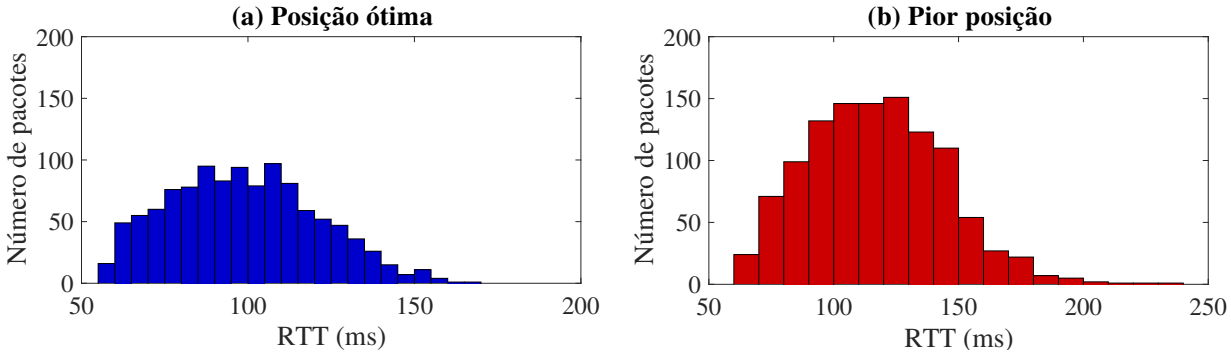


Figura 5.6: Histograma do RTT dos primeiros pacotes dos fluxos gerados nos cenários (a) e (b).

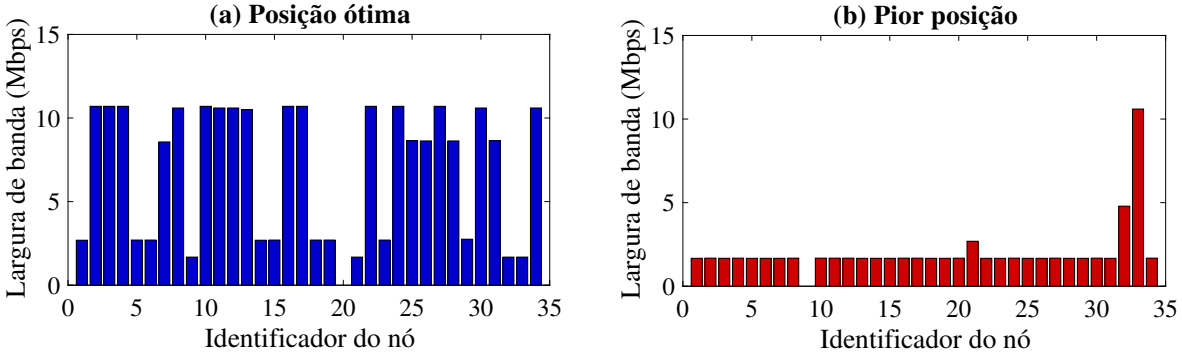


Figura 5.7: Taxa de transmissão entre os nós e o controlador nos cenários (a) e (b).

5.3 Metaheurísticas

Considerando o problema de posicionamento de um controlador na topologia de rede Internet2 OS3E, já apresentada na Fig. 5.1, o tempo de processamento do POCO através da análise exaustiva foi de aproximadamente 0.02 segundos. Entretanto, quando a mesma análise é realizada para mais controladores, o tempo de processamento cresce exponencialmente devido ao aumento do tamanho do espaço de busca, definido por $\binom{n}{k}$, como apresentado na Tabela 5.3.

Neste contexto, a fim de avaliar o desempenho dos algoritmos propostos pelas metaheurísticas

Tabela 5.3: Análise exaustiva: tamanho do espaço de busca e tempo de processamento

Quantidade de controladores	**Tamanho do espaço de busca**	**Tempo de processamento (s)**
1	34	0.02
2	561	2.69
3	5984	281
4	46376	16809

PSA e NSGA-II e determinar experimentalmente se esses são capazes de convergir para o conjunto de soluções Pareto-ótimas, obtido previamente pelo método exaustivo, testes foram realizados tendo como caso de uso a topologia de rede Internet2 OS3E, para 2, 3 e 4 controladores. As avaliações foram realizadas em um computador Dell Inspiron 5557 Intel(R) Core(TM) i7-6500U CPU 2.50GHz e 8GB de RAM.

Ambos os algoritmos foram inicializados com uma amostra de $N = 100$ soluções-candidatas, para permitir avaliar o comportamento de ambos sob a mesma condição inicial. As funções objetivo utilizadas para fins de otimização foram calculadas a partir das informações extraídas da topologia pelo método indicado na Seção 4.3.1.1.

Uma métrica de desempenho comumente utilizada para medir a qualidade de algoritmos multiobjetivo é a distância geracional (GD, do inglês *Generational Distance*). O conceito de distância geracional foi introduzido por Veldhuizen e Lamont (1998, 2000) como uma forma de estimar o quão distante as soluções não dominadas de um conjunto encontrado estão do conjunto ótimo de Pareto. Dada uma fronteira de Pareto candidata F_{atual} e a fronteira de Pareto F_{real}, que pode ser calculada pelo método exaustivo, a distância geracional no espaço dos objetivos é definida por:

$$GD \triangleq \frac{\left(\sum_{i=1}^{n} d_i^J\right)^{\frac{1}{J}}}{n}, \tag{5.1}$$

onde n é o número de soluções na fronteira de Pareto candidata F_{atual}, J é o número de funções objetivo, e d_i é a distância Euclidiana medida no espaço de objetivos entre cada solução de F_{atual} e a solução mais próxima a ela em F_{real}. Desta forma, GD = 0 indica que todas as soluções geradas pertencem à fronteira de Pareto, e qualquer outro valor indica o quão longe se está desta para o problema em questão.

Por meio da distância geracional, nas Seções 5.4 e 5.5, avalia-se experimentalmente a capacidade dos algoritmos PSA e NSGA-II de convergir para a fronteira de Pareto, no caso de uso considerado, e discute-se a importância da escolha adequada dos parâmetros inerentes a cada algoritmo.

Na Seção 5.6, compara-se o desempenho dos algoritmos propostos, em termos de distância geracional, tempo de processamento e quantidade de soluções avaliadas. Nesta mesma Seção, também são apresentados os resultados obtidos para outras topologias de rede, com diferentes dimensões e densidades geográficas.

5.4 Avaliação do PSA

A avaliação de desempenho do algoritmo PSA tem por objetivo determinar se o algoritmo proposto é capaz de convergir para a fronteira de Pareto e analisar como a escolha da temperatura inicial (T_0), da taxa de resfriamento (ρ) e do número de iterações (m), impactam o número de avaliações de soluções candidatas durante o processamento do algoritmo.

A escolha adequada desses parâmetros é essencial para determinar o desempenho de um algoritmo baseado no método *Simulated Annealing*. Uma taxa de resfriamento ρ muito baixa, por exemplo, pode levar a resultados fracos devido a falta de estados representativos, enquanto uma taxa alta requer mais processamento e tempo para chegar a um resultado.

Apesar de não haver um consenso, Kumar e Suman (2006) discutem a escolha desses paramêtros, tal que:

- **Temperatura inicial (T_0)**
  A temperatura inicial deve ser escolhida de forma que tente capturar todo o espaço de soluções. Para tanto, ela deve ser relativamente alta, porém um valor alto pode levar o algoritmo a executar um grande número de iterações, que podem não levar aos melhores resultados. A T_0 pode ser escolhida por experimentação, dependendo da natureza do problema. Entretanto, existe uma forma simples de selecionar a temperatura inicial proposta por Kouvelis e Chiang (1992) pela equação:

  $$P = exp\Big(\frac{-\Delta s}{T}\Big), \tag{5.2}$$

  onde P é a probabilidade de aceitação inicial, definida no intervalo de 0.50 a 0.95, e Δs é a diferença no valor da função objetivo entre duas soluções.

- **Taxa de resfriamento (ρ)**
  A taxa de resfriamento determina uma forma funcional de alterar a temperatura, como requerido pelos métodos *Simulated Annealing*. Para tanto, $T_k = \rho.T_{k-1}$, onde ρ é uma constante que varia de 0.80 a 0.99 de maneira análoga ao resfriamento físico de substâncias.

- **Número de iterações (m)**
  Um número suficiente de iterações devem ser executadas a cada temperatura. Se um número pequeno de iterações forem executadas, poucos estados representativos podem ser pesquisados e a solução possivelmente não representará o ótimo global. O seu valor depende da natureza do problema.

Neste contexto, os experimentos realizados para a avaliação do PSA consideraram uma probabilidade de aceitação inicial $P_0(s, q, T, \Lambda) = 95\%$. A partir da regra SL descrita pela Equação (3.4), a temperatura inicial encontrada para a topologia de rede Internet2 OS3E foi $T_0 \approx 4$.

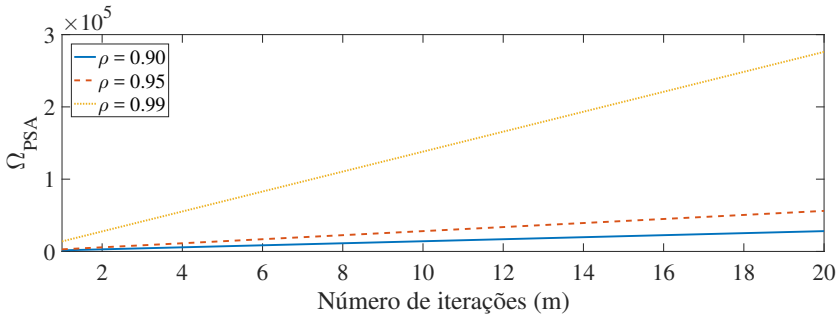


Figura 5.8: Quantidade de avaliações de soluções candidatas para $\rho = 0.99$, $\rho = 0.95$ e $\rho = 0.90$, com $T_0 = 4$.

Fixando-se os valores da temperatura e da taxa de resfriamento, o número de iterações é responsável por definir o número total de avaliações de soluções candidatas pelo PSA (Ω_{PSA}), descrito por:

$$\Omega_{PSA} = N\Big[1 + m\Big\lceil - \frac{\log T_0}{log\rho}\Big\rceil\Big], \tag{5.3}$$

onde N indica o número de avaliações de soluções candidatas em cada iteração, m indica o número de iterações e $\lceil - \frac{\log T_0}{log\rho} \rceil$ indica o número de níveis de temperatura percorridos pelo algoritmo à partir da temperatura inicial T_0 e da taxa de resfriamento ρ. A Fig. 5.8 apresenta a quantidade de avaliações para $\rho = 0.90$, $\rho = 0.95$ e $\rho = 0.99$ em função do número de iterações, com $T_0 = 4$. Observe que a quantidade de avaliações é independente do número de controladores, pois pode-se explorar durante a busca a mesma solução repetidas vezes, dada a natureza probabilística do algoritmo.

Foi então avaliado o impacto do número de iterações na operação do algoritmo. Os resultados experimentais obtidos para $k = 2$, $k = 3$ e $k = 4$ a uma taxa de resfriamento $\rho = 0.99$, considerando $m = 1$, $m = 4$ e $m = 8$, são apresentados nas Figs. 5.9, 5.10 e 5.11, respectivamente. Tais resultados permitem verificar o impacto do número de iterações na inclinação da curva obtida pela probabilidade de aceitação e no tempo de processamento do algoritmo.

A partir dos resultados apresentados para a probabilidade de aceitação, verifica-se que quando o algoritmo é iniciado, qualquer movimento entre as soluções candidatas é aceito devido à alta temperatura. À medida que a temperatura diminui, o movimento se torna mais seletivo com o aumento do número de iteração, que tende a apresentar menor probabilidade de aceitação para uma mesma temperatura. Tal comportamento é indicado pela inclinação da curva da probabilidade de aceitação ao longo do processamento do algoritmo, de forma que um maior número de iterações irá resultar em uma curva mais íngreme. Ao final do algoritmo, espera-se que apenas movimentos de melhoria sejam aceitos, tal que a probabilidade de aceitação seja ≈ 0.

Especialmente quando $m = 1$, é possível verificar que o algoritmo não converge, uma vez que termina seu processamento com uma probabilidade de aceitação alta, na qual ainda são aceitos movimentos para soluções que não apresentem melhoria.

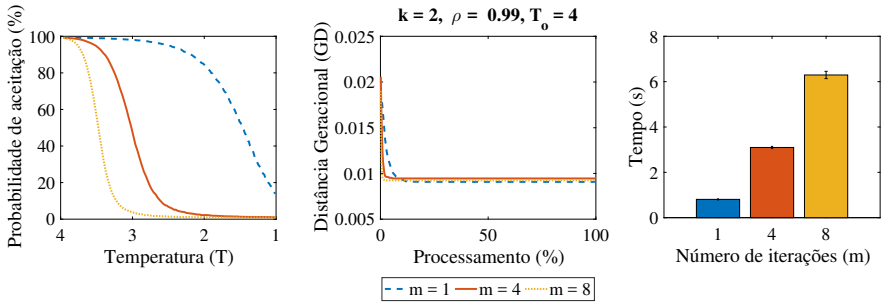


Figura 5.9: Desempenho médio do algoritmo PSA para 10 experimentos - $k = 2$, $\rho = 0.99$, $T_0 = 4$.

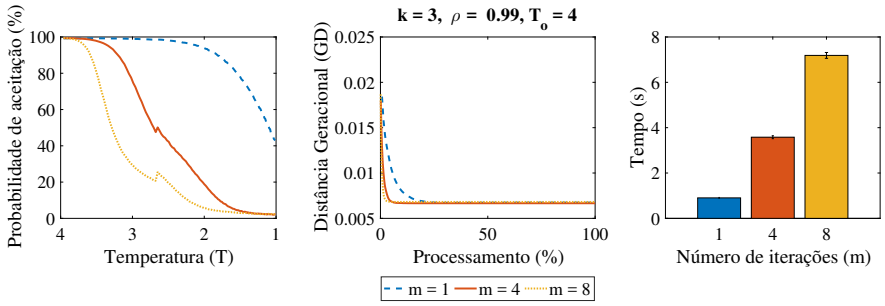


Figura 5.10: Desempenho médio do algoritmo PSA para 10 experimentos - $k = 3$, $\rho = 0.99$, $T_0 = 4$.

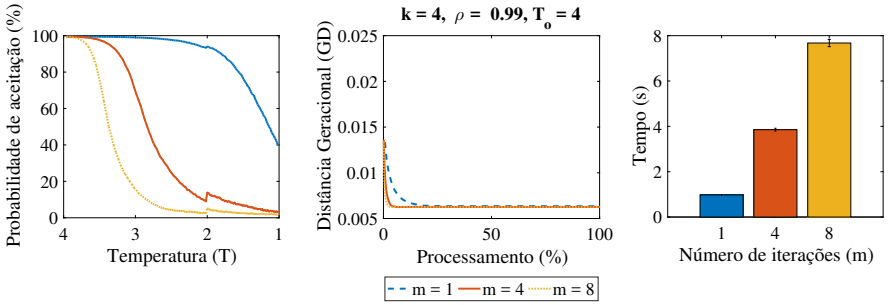


Figura 5.11: Desempenho médio do algoritmo PSA para 10 experimentos - $k = 4$, $\rho = 0.99$, $T_0 = 4$.

Como discutido anteriormente, fixados os demais parâmetros a serem utilizados pelo PSA, o número de iterações m é responsável por definir quantidade de avaliações de soluções candidatas (Ω_{PSA}). Desta forma, quanto maior o número de iterações, maior é a quantidade de avaliações realizadas pelo algoritmo. Assim, também é possível concluir que, quanto maior o número de iterações a serem efetuadas pelo algoritmo, maior também será o tempo de processamento necessário. Nos resultados apresentados pelas Figs. 5.9, 5.10 e 5.11 é possível verificar que, no caso de uso em questão, esse aumento é exponencial.

Os resultados da distância geracional apresentados para os diferentes valores de k e m permitem concluir que, no caso de uso em questão, o PSA não foi capaz de encontrar um conjunto estimado de soluções Pareto-ótimas mais próximo do real ao longo do seu processamento. Mesmo com uma alta probabilidade de aceitação, o algoritmo não incorporou dispersão suficiente para buscar por novas soluções.

Para uma taxa de resfriamento $\rho = 0.95$, é necessário um número maior de iterações para se

obter resultados próximos aos obtidos para $\rho = 0.99$. Os resultados experimentais obtidos para $k = 2$, $k = 3$ e $k = 4$, considerando $m = 2$, $m = 8$ e $m = 16$, são apresentados nas Figs. 5.12, 5.13 e 5.14, respectivamente. A sua análise permite confirmar a análise realizada nos experimentos apresentados anteriormente.

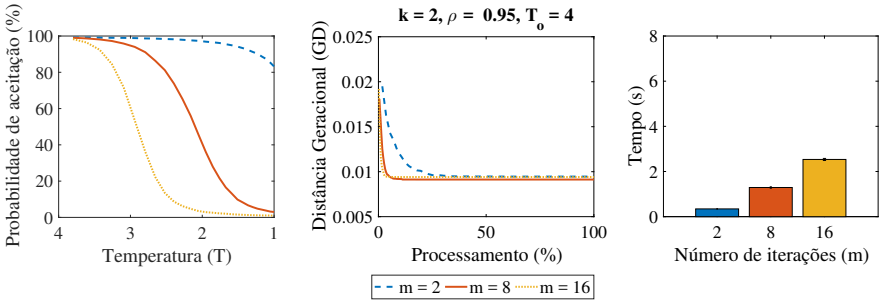


Figura 5.12: Desempenho médio do algoritmo PSA para 10 experimentos - $k = 2$, $\rho = 0.95$, $T_0 = 4$.

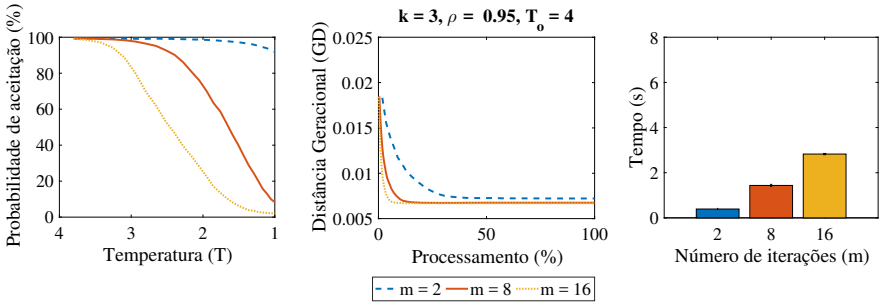


Figura 5.13: Desempenho médio do algoritmo PSA para 10 experimentos - $k = 3$, $\rho = 0.95$, $T_0 = 4$.

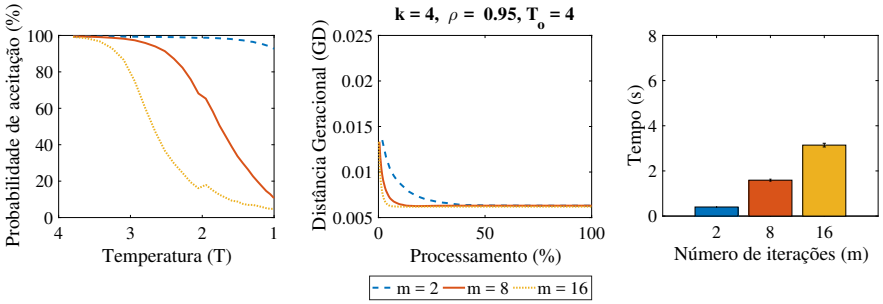


Figura 5.14: Desempenho médio do algoritmo PSA para 10 experimentos - $k = 4$, $\rho = 0.95$, $T_0 = 4$.

No contexto dos resultados apresentados para o PSA, é possível concluir que o algoritmo é capaz de convergir para um conjunto estimado de soluções Pareto-ótimas muito próximo ao conjunto real logo no início de seu processamento.

Como exemplo, a Fig. 5.15 apresenta a Fronteira de Pareto estimada pelo PSA para $k = 2$, tal que $\rho = 0.95$, $T_0 = 4$ e $m = 8$, onde pontos em azul indicam as soluções que formam a Fronteira de Pareto estimada pelo PSA e os pontos em cinza indicam as soluções que formam a Fronteira de Pareto real. Para melhor visualização, as fronteiras estimada e real são apresentadas em gráficos de duas dimensões para as diferentes funções objetivo consideradas.

No caso de uso em questão, o aumento do número de avaliações de soluções, promovido pelo

aumento do número de iterações, resultou no aumento do tempo de processamento do algoritmo, sem apresentar melhoria significativa no que diz respeito à distância geracional.

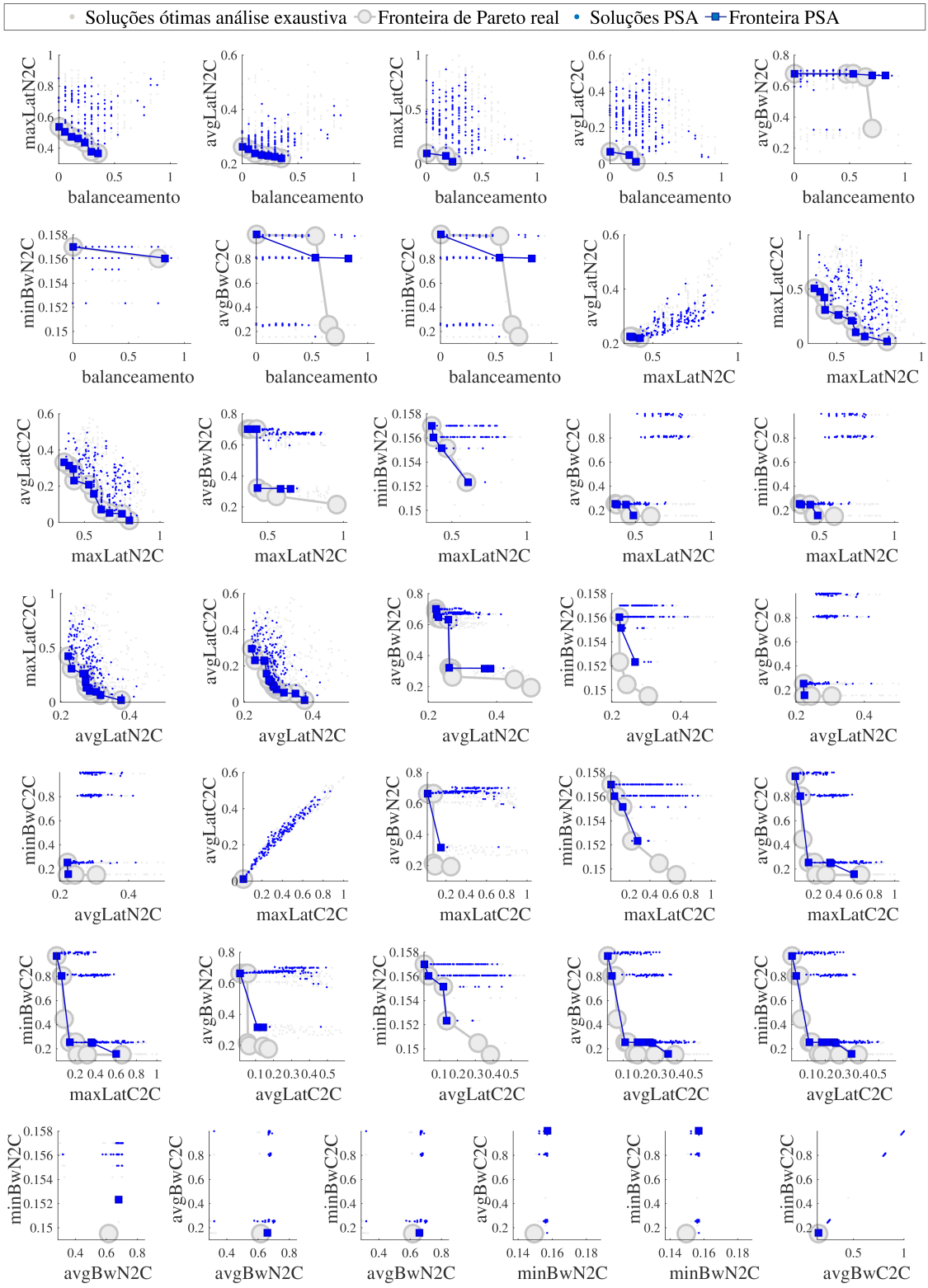


Figura 5.15: Fronteira de Pareto obtida pelo PSA para $k = 2$, $\rho = 0.95$, $T_0 = 4$ e $m = 8$. Perspectivas bidimensionais que demostram a relação entre todas as funções objetivo avaliadas.

5.5 Avaliação do NSGA-II

A avaliação de desempenho do algoritmo NSGA-II tem por objetivo determinar se o algoritmo proposto é capaz de convergir para o conjunto de soluções Pareto-ótimas, no caso de uso em questão.

A definição de valores apropriados para os parâmetros de algoritmos evolutivos é essencial para um bom desempenho, mas encontrar esses valores é um grande desafio que ainda persiste no campo da pesquisa (EIBEN A.E., 2012). Frequentemente, esses valores são escolhidos por convenções, como por exemplo, a de que a taxa de mutação deve ser baixa, e pela comparação experimental em escala limitada, onde são testadas diferentes taxas de cruzamento e mutação. Eiben A.E. (2012) discutem esses parâmetros e definem os valores comumente utilizados, que são apresentados na Tabela 5.4.

Tabela 5.4: Valores da taxa de mutação (t_m), da taxa de cruzamento (t_c), tamanho do conjunto de soluções (N) e tamanho do torneio. Os algoritmos nas colunas EA1 e EA2 são variações nos parâmetros a serem adotadas no NSGA-II.

	Algoritmo Evolutivo 1 (EA1)	**Algoritmo Evolutivo 2 (EA2)**
t_m	0.01	0.1
t_c	0.5	0.7
N	100	100
Torneio	2	4

Neste contexto, faz-se necessário definir o número de iterações para a parada do algoritmo. Fixando-se os valores da taxa de cruzamento e da taxa de mutação, o número de iterações é responsável por definir o número de avaliações de soluções realizadas pelo NSGA-II ($\Omega_{NSGA-II}$), descrito por:

$$\Omega_{NSGA-II} = N[1 + m(t_c + t_m)], \tag{5.4}$$

onde N indica o número de avaliações em cada iteração, m indica o número de iterações, t_c indica a taxa de cruzamento e t_m indica a taxa de mutação. Observe novamente que a quantidade de soluções é independente do número de controladores, visto que, assim como o PSA, o NSGA-II também é uma metaheurística de natureza probabilística e, por isso, não se atrela à dimensão do problema mas sim aos operadores que permitam avaliar um número adequado de soluções candidatas, mesmo que repetidamente.

Considerando os parâmetros definidos na Tabela 5.4 para EA1 e EA2, a quantidade de avaliações de soluções candidatas para a topologia de rede Internet2 OS3E com diferentes valores de m é apresentado na Fig. 5.16. Os resultados experimentais, no que diz respeito à distância geracional e tempo de processamento, obtidos por diferentes valores de m para $k = 2$, $k = 3$ e $k = 4$ são apresentados nas Figs. 5.17, 5.18 e 5.19, respectivamente.

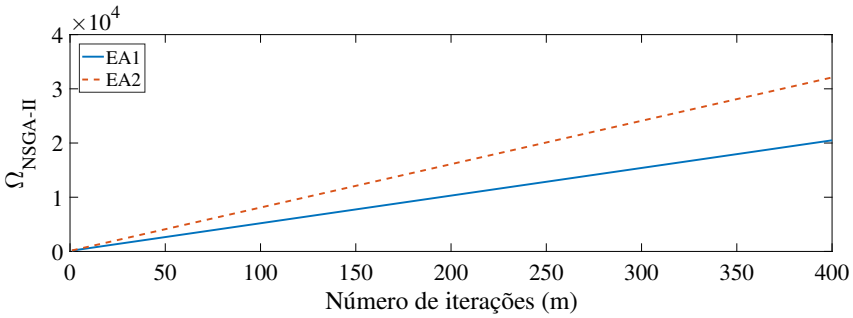


Figura 5.16: Quantidade de soluções avaliadas para EA1 e EA2.

No contexto dos resultados apresentados para o NSGA-II, o decaimento da distância geracional ao longo do processamento indica que o algoritmo é capaz de convergir para uma estimativa do conjunto de soluções Pareto-ótimas muito próximo do conjunto real.

É possível verificar que, no caso de uso em questão, o valor da distância geracional apresentou grande decaimento nas primeiras $m = 20$ iterações, tal que um número maior de iterações resultou em um aumento significativo do tempo de processamento sem ser capaz de apresentar a estimativa do conjunto de soluções Pareto-ótimas significativamente mais próximo do real. Ou seja, apesar de que para $k = 2$ o algoritmo é capaz de alcançar o conjunto de soluções Pareto-ótimas quando $m \geq 300$, isso exige um tempo de processamento maior, que pode não ser desejável.

Os resultados apresentados também permitem verificar que com o aumento do número de controladores, os parâmetros definidos para EA1 e EA2 foram capazes de alcançar valores de distância geracional muito próximos. Entretanto, EA1 apresentou vantagem em termos de tempo de processamento à medida que o número de iterações aumenta.

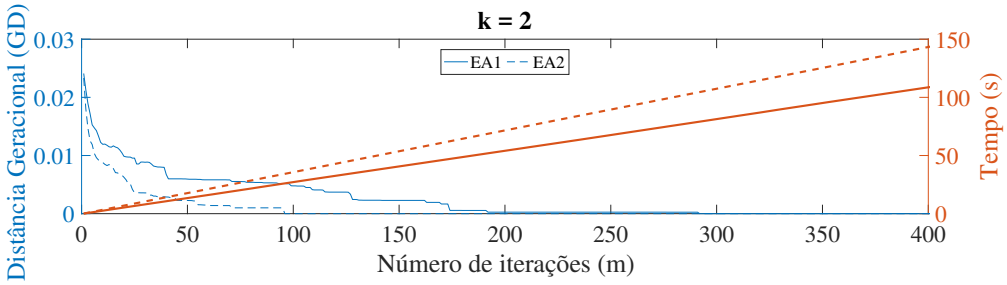


Figura 5.17: Desempenho médio do algoritmo NSGA II para 10 experimentos - $k = 2$.

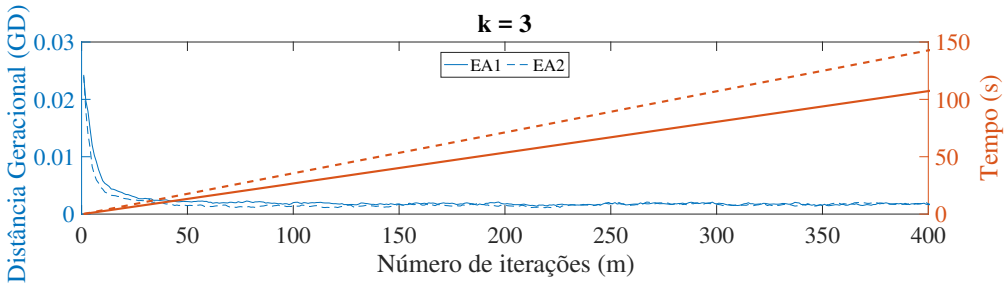


Figura 5.18: Desempenho médio do algoritmo NSGA II para 10 experimentos - $k = 3$.

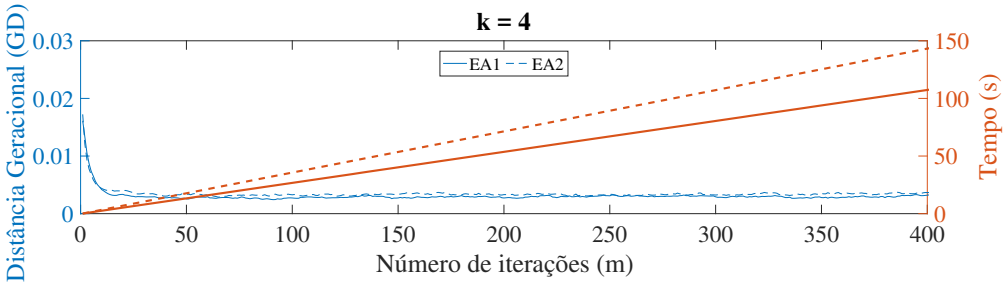


Figura 5.19: Desempenho médio do algoritmo NSGA II para 10 experimentos - $k = 4$.

Como exemplo, a Fig. 5.20 apresenta a Fronteira de Pareto obtida pelo NSGA-II para $k = 2$, tal que $t_c = 0.7$, $t_m = 0.1$ e $m = 20$, onde pontos em vermelho indicam as soluções que formam a Fronteira de Pareto estimada pelo NSGA-II e os pontos em cinza indicam as soluções que formam a Fronteira de Pareto real. Para melhor visualização, as fronteiras estimada e real são apresentadas em gráficos de duas dimensões para as diferentes funções objetivo consideradas.

5.6 Análise Comparativa

A análise comparativa tem por objetivo comparar o desempenho dos algoritmos PSA e NSGA-II em termos de distância geracional, quantidade de avaliações de soluções candidatas e tempo de processamento, considerando a média obtida para 10 experimentos.

Para o caso de uso da topologia de rede Internet2 OS3E, os algoritmos foram inicializados com um conjunto de $N = 100$ (PSA_{100} e $NSGA-II_{100}$) e $N = 200$ (PSA_{200} e $NSGA-II_{200}$) soluções escolhidas aleatoriamente, tal que cada solução corresponde a uma escolha de posições para conectar os k controladores, a fim de avaliar o impacto do aumento do conjunto de soluções inicial no processamento do algoritmo. Os demais parâmetros utilizados são apresentados na Tabela 5.5, e foram escolhidos a partir das avaliações realizadas nas Seções 5.4 e 5.5.

Tabela 5.5: Parâmetros definidos para experimentação

Algoritmo	**Parâmetros**
PSA_{100}	$T_0 = 4, \rho = 0.95, m = 8, N = 100$
PSA_{200}	$T_0 = 4, \rho = 0.95, m = 8, N = 200$
$NSGA-II_{100}$	$t_m = 0.01, t_c = 0.07, m = 20, torneio = 4, N = 100$
$NSGA-II_{200}$	$t_m = 0.01, t_c = 0.07, m = 20, torneio = 4, N = 200$

Os valores de distância geracional obtidos ao longo do processamento dos algoritmos para $k = 2$, $k = 3$ e $k = 4$ são apresentados nas Figs. 5.21, 5.22 e 5.23, respectivamente. A partir dos resultados apresentados, é possível verificar que o NSGA-II é capaz de gerar soluções mais próximas do conjunto de soluções Pareto-ótimas ideal ao longo de todo o seu processamento, enquanto o PSA converge com 10% de processamento.

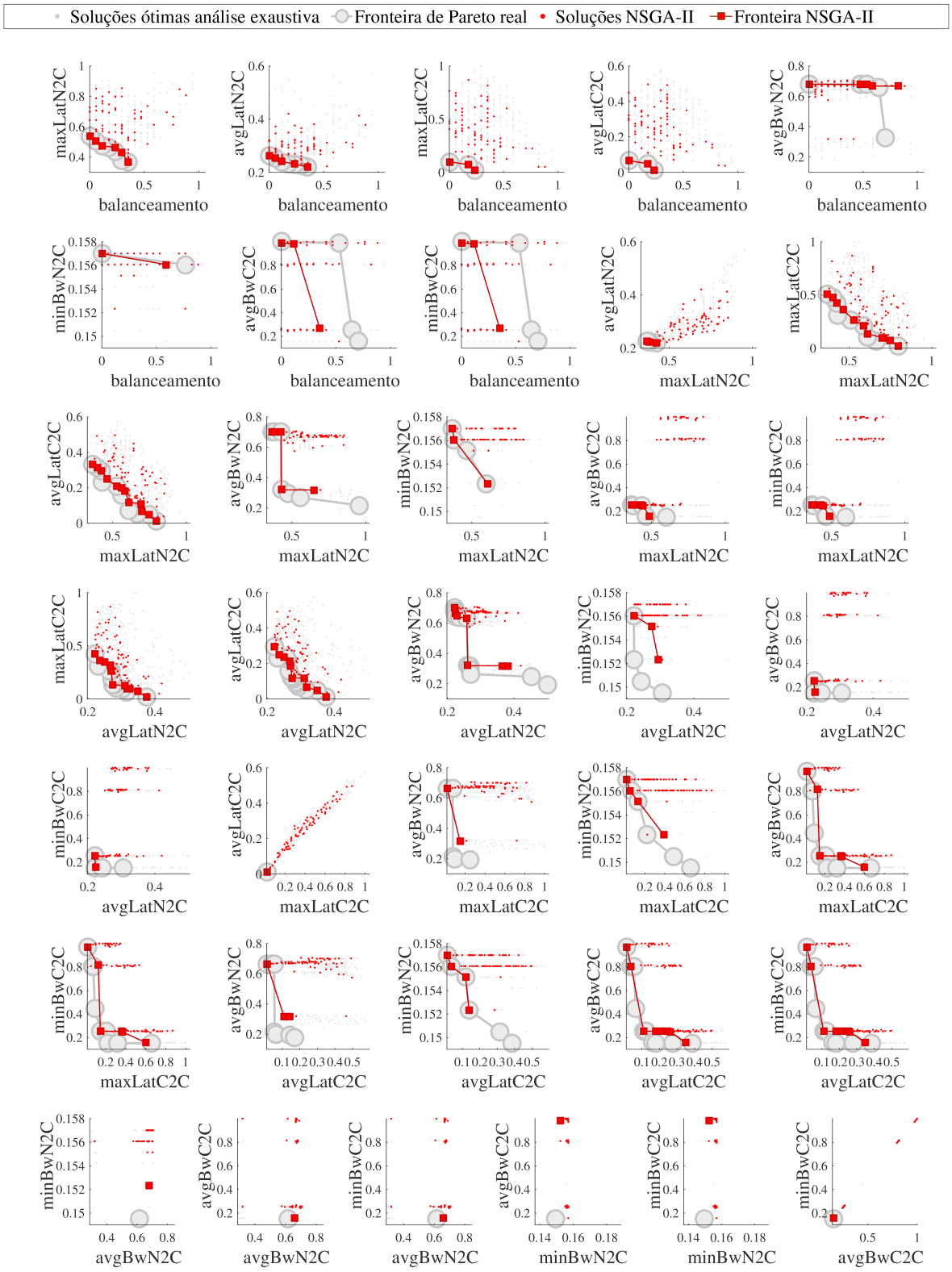


Figura 5.20: Fronteira de Pareto obtida pelo NSGA-II para $k = 2$, $t_c = 0.7$, $t_m = 0.1$ e $m = 20$. Perspectivas bidimensionais que demostram a relação entre todas as funções objetivo avaliadas.

O comportamento apresentado pelo PSA foi discutido na Seção 5.4, onde concluiu-se pela dificuldade do PSA em encontrar uma estimativa de conjunto de soluções Pareto-ótimas mais próxima do real ao longo de seu processamento, mesmo com uma alta probabilidade de aceitação, que deveria incorporar dispersão suficiente para que o algoritmo evitasse ótimos locais.

Em termos gerais, a Fig. 5.24 apresenta a distância geracional média para 10 experimentos.

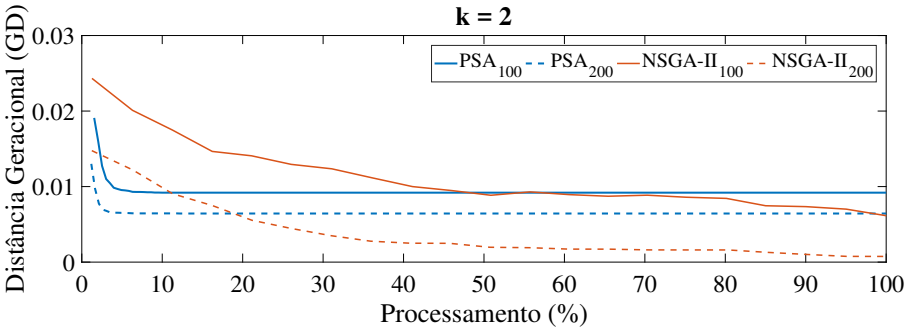


Figura 5.21: Processamento médio dos algoritmos PSA e NSGA-II para 10 experimentos - $k = 2$.

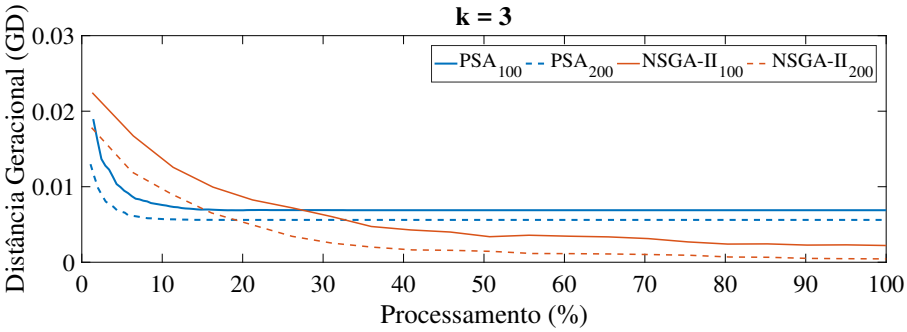


Figura 5.22: Processamento médio dos algoritmos PSA e NSGA-II para 10 experimentos - $k = 3$.

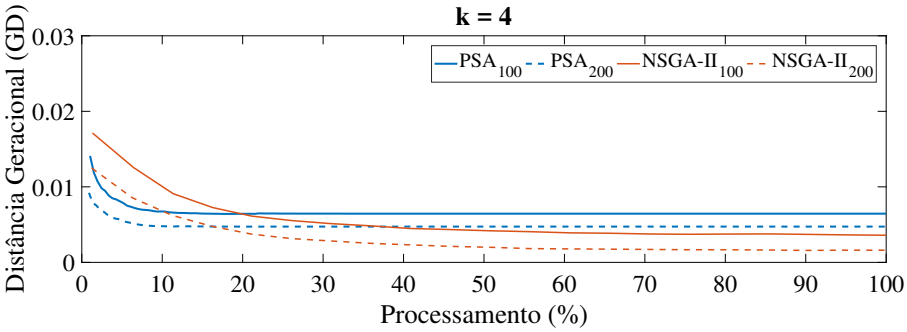


Figura 5.23: Processamento médio dos algoritmos PSA e NSGA-II para 10 experimentos - $k = 4$.

Avaliando-se a distância geracional obtida pelos algoritmos, conclui-se que o NSGA-II é capaz de gerar estimativas do conjunto de soluções Pareto-ótimas mais próximas do conjunto real.

Em todos os casos, o aumento do tamanho do conjunto de soluções de $N = 100$ para $N = 200$ também contribuiu para a redução da distância geracional e consequente melhoria na qualidade do conjunto final gerado. Intuitivamente, isso se deve ao consequente aumento da quantidade de soluções avaliadas por ambos algoritmos.

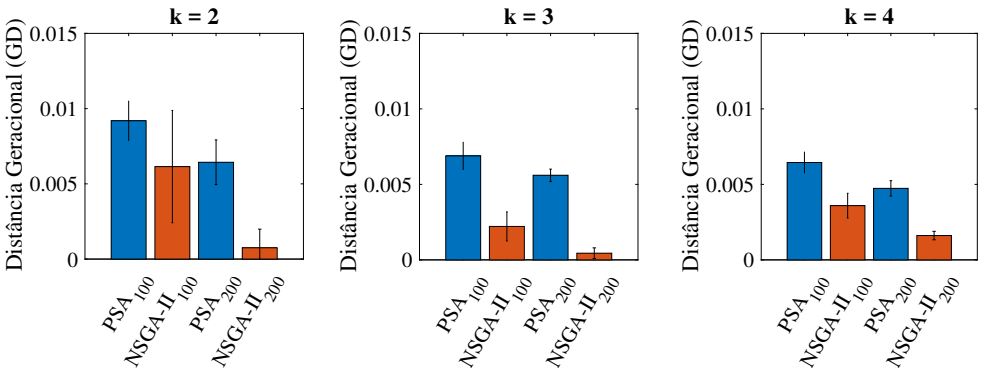


Figura 5.24: Distância geracional média dos algoritmos PSA e NSGA-II para 10 experimentos.

As Figs. 5.25, 5.26 e 5.27 permitem analisar a relação entre a quantidade de avaliações de soluções candidatas e a distância geracional obtida com $N = 100$ para $k = 2$, $k = 3$ e $k = 4$, respectivamente. Através dos resultados apresentados, conclui-se que o NSGA-II é capaz de gerar soluções mais próximas às ideais avaliando apenas 7.3% da quantidade equivalente de avaliações realizadas pelo PSA.

Apesar disso, verifica-se também que o PSA converge após realizar uma quantidade de avaliações igual à quantidade total de avaliações realizadas pelo NSGA-II, ou seja 1600. Tal resultado indica que o PSA potencialmente processa novas análises para soluções já avaliadas em iterações anteriores, tais como ocorrências de ótimos locais.

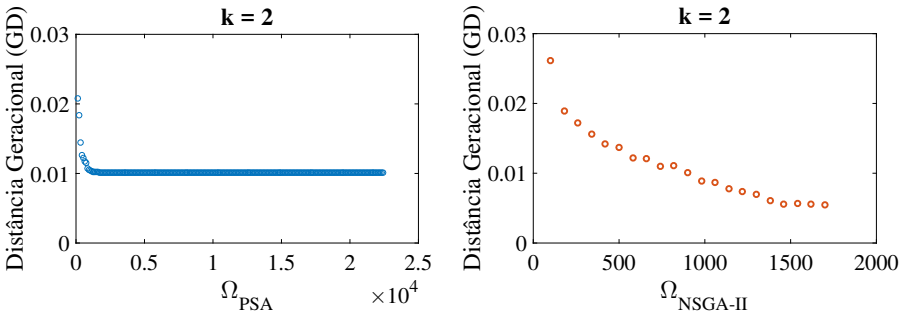


Figura 5.25: Quantidade de soluções avaliadas pelos algoritmos PSA e NSGA-II com $N = 100$ para 10 experimentos - $k = 2$.

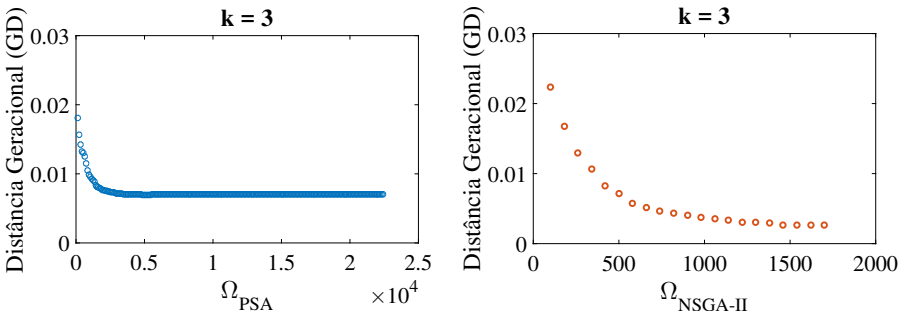


Figura 5.26: Quantidade de soluções avaliadas pelos algoritmos PSA e NSGA-II com $N = 100$ para 10 experimentos - $k = 3$.

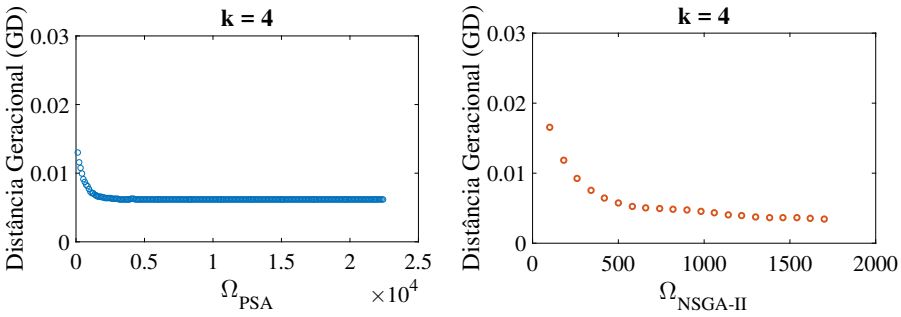


Figura 5.27: Quantidade de soluções avaliadas pelos algoritmos PSA e NSGA-II com $N = 100$ para 10 experimentos com N = 100 - $k = 4$.

É importante reafirmar que o número de avaliações realizadas pelo PSA e pelo NSGA-II possuem um valor fixo para diferentes valores de k, como definido pelas Equações (5.3) e (5.4), respectivamente. O espaço de busca do problema, por sua vez, depende diretamente do número de controladores k, sendo definido por $\binom{n}{k}$. Desta forma, dependendo dos parâmetros de entrada dos algoritmos PSA e NSGA-II, o número de soluções avaliadas pelas metaheurísticas pode ser maior que o tamanho total do espaço de busca do problema, o que significa que uma mesma solução é avaliada mais de uma vez.

Para a topologia da rede Internet2 OS3E, o tamanho do espaço de busca do problema e o número avaliações de soluções pelos algoritmos propostos para diferentes números de controladores k é apresentado na Tabela 5.6. É possível verificar que, para problemas pequenos, ambos os algoritmos avaliam mais soluções que o tamanho total do espaço de busca do problema.

Tabela 5.6: Tamanho do espaço de busca x Quantidade de soluções avaliadas

Número de controladores	**Tamanho do espaço de busca do problema**	Ω_{PSA}	$\Omega_{NSGA-II}$
$k = 2$	561	21700	1600
$k = 3$	5984	21700	1600
$k = 4$	46376	21700	1600
$k = 5$	278256	21700	1600
$k = 6$	1344904	21700	1600

A Fig. 5.28 apresenta o tempo médio de processamento dos algoritmos. As metaheurísticas apresentaram vantagem em relação ao método exaustivo para $k = 3$ e $k = 4$, com vantagem significativa para o PSA. Para $k = 2$, o método exaustivo ainda foi capaz de terminar o seu processamento em tempo inferior, o que indica que a utilização das metaheurísticas da forma como foram propostas neste trabalho se apresentaram eficientes para problemas de maior porte. O aumento do tamanho do conjunto de soluções de $N = 100$ para $N = 200$ também contribuiu para o aumento do tempo de processamento, uma vez que o número de soluções avaliadas por ambos algoritmo aumentou.

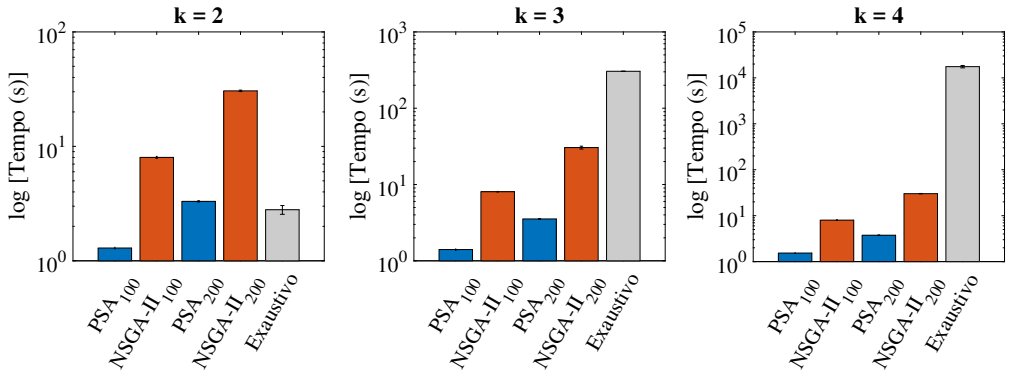


Figura 5.28: Tempo médio de processamento dos algoritmos PSA, NSGA-II e Exaustivo para 10 experimentos.

A mesma avaliação considerando $k = 3$ e $N = 100$ foi realizada para outras topologias de rede, com diferentes dimensões e densidade geográficas. Os resultados são apresentados nas Figs. 5.29 à 5.38, e permitem confirmar a análise realizada para a topologia da rede Internet2 OS3E.

Através dos experimentos apresentados, conclui-se que as metaheurísticas propostas são capazes de convergir ou aproximar-se bem da fronteira de Pareto, pré-calculada pelo método exaustivo, com vantagem em termos de distância geracional para o NSGA-II e com vantagem em termos de tempo de processamento para o PSA. Para topologias menores, como as redes GtsRomania e IBM apresentadas nas Figs. 5.32 e 5.34, a utilização do NSGA-II da forma como foi proposto neste trabalho não apresentou vantagens em termos de tempo de processamento, entretanto é possível sua adequação ajustando-se os valores dos parâmetros utilizados.

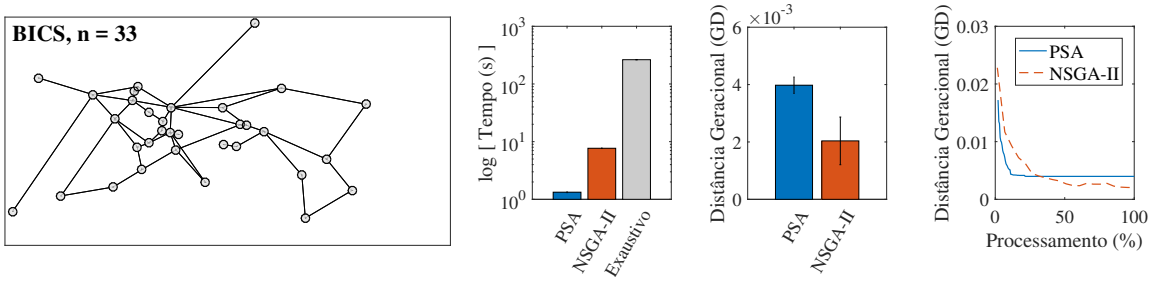


Figura 5.29: Valores médios obtidos com a topologia da rede BICS para 10 experimentos, k = 3.

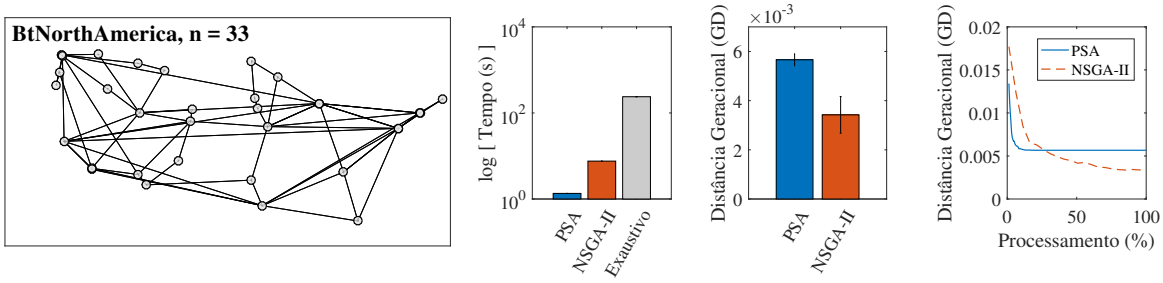


Figura 5.30: Valores médios obtidos com a topologia da rede BtNorthAmerica para 10 experimentos, k = 3.

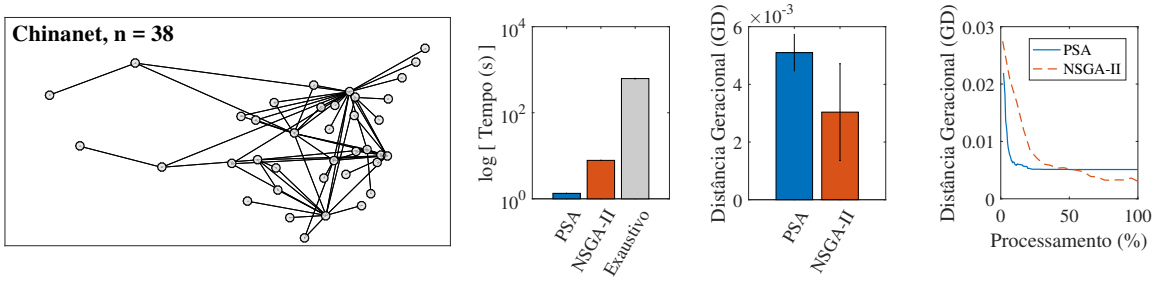


Figura 5.31: Valores médios obtidos com a topologia da rede Chinanet para 10 experimentos, k = 3.

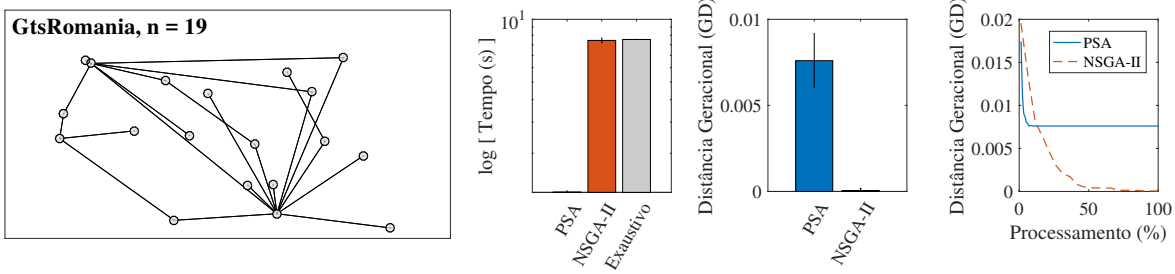


Figura 5.32: Valores médios obtidos com a topologia da rede GtsRomania para 10 experimentos, k = 3.

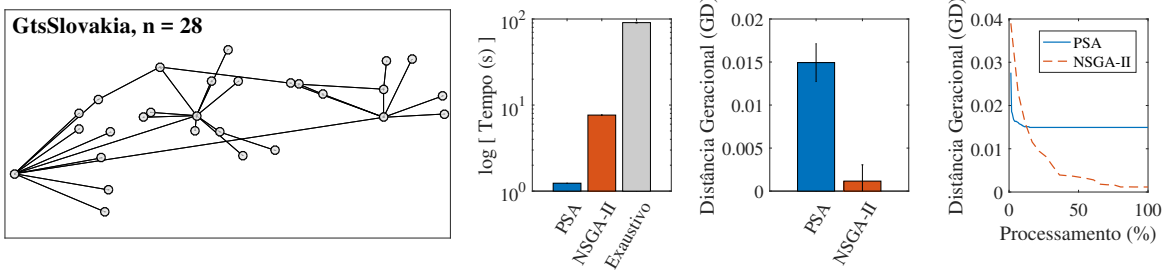


Figura 5.33: Valores médios obtidos com a topologia da rede GtsSlavakia para 10 experimentos, k = 3.

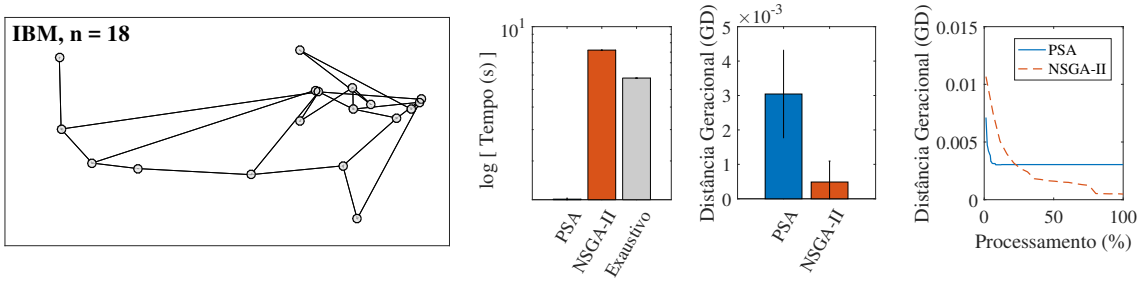


Figura 5.34: Valores médios obtidos com a topologia da rede IBM para 10 experimentos, k = 3.

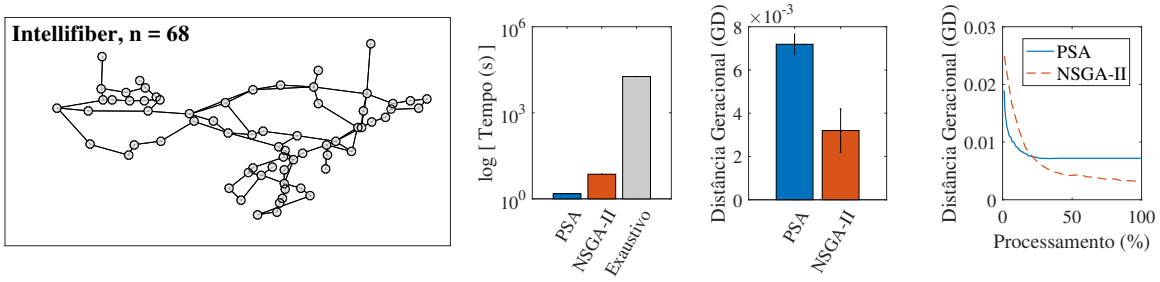


Figura 5.35: Valores médios obtidos com a topologia da rede Intelifiber para 10 experimentos, k = 3.

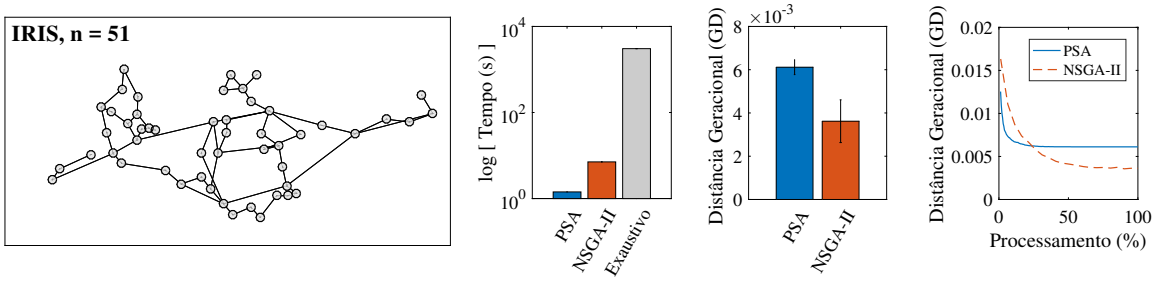


Figura 5.36: Valores médios obtidos com a topologia da rede Iris para 10 experimentos, k = 3.

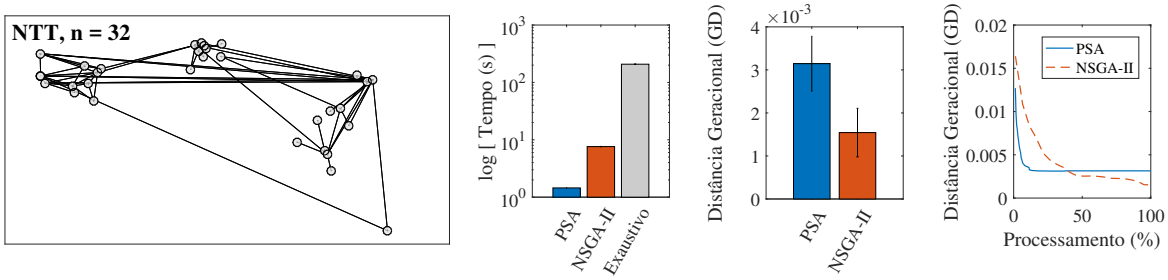


Figura 5.37: Valores médios obtidos com a topologia da rede NTT para 10 experimentos, k = 3.

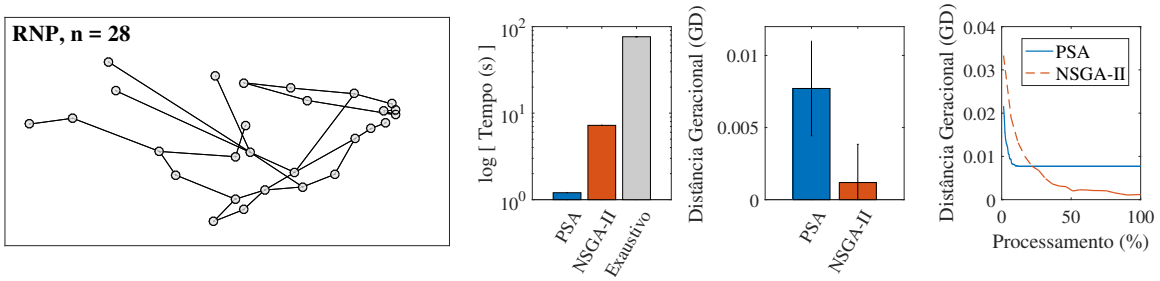


Figura 5.38: Valores médios obtidos com a topologia da rede RNP para 10 experimentos, k = 3.

5.7 Considerações finais

Este capítulo apresentou os resultados de desempenho de uma rede emulada em laboratório, onde demonstrou-se o efeito do posicionamento do controlador calculado pelo POCO. Também foram apresentados resultados experimentais realizados a fim de verificar o desempenho do algoritmo PSA, implementado no POCO por Lange et al. (2015b), e do algoritmo NSGA-II, cuja implementação foi proposta neste trabalho.

As conclusões obtidas para os resultados dos experimentos são apresentadas no Capítulo 6.

Capítulo 6

Conclusões

Neste trabalho apresentamos um estudo sobre o problema de posicionamento de controladores em redes SDN. Descrevemos a implementação realizada no *framework* POCO, que permite calcular o posicionamento ótimo de controladores por meio da busca exaustiva e das metaheurísticas PSA e NSGA-II. Então, avaliamos as funções objetivo que mapeiam métricas de desempenho de rede: latência, balanceamento de carga e taxa de transmissão. Em ambientes reais, as métricas de desempenho são concorrentes, de forma que não há uma solução que satisfaça simultaneamente objetivos de otimização como latência e balanceamento de carga. Para demonstrar tal fato, construímos uma rede habilitada com OpenFlow e um controlador Floodlight no Mininet, e avaliamos o desempenho da rede sob condições estáticas quando o controlador é colocado em diferentes posições. Finalmente, avaliamos o desempenho das metaheurísticas PSA e NSGA-II em relação à técnica exaustiva, e discutimos o compromisso que é assumido pelos diferentes parâmetros de entrada dos algoritmos.

Nos resultados apresentados para o desempenho da rede emulada, considerando as medidas estáticas relacionadas a latência e taxa de transmissão, os ganhos obtidos com o controlador em uma posição ótima mostraram uma rede mais reativa e um menor tempo necessário para o início dos fluxos de dados entre os nós.

Considerando ainda redes reais, onde o tráfego do plano de controle pode apresentar desempenho igual ou inferior ao tráfego do plano de dados, este trabalho chama a atenção para a necessidade de se desenvolver ferramentas que permitam avaliar a adequação de uma dada rede para implantação de SDN com base em informações de desempenho reais da mesma.

As metaheurísticas, por sua vez, mostraram que podem ser utilizadas na busca de soluções ótimas ou aproximações para prover um ganho no tempo de processamento. Para o caso de uso apresentado, o NSGA-II foi capaz de gerar soluções mais próximas do conjunto de soluções Pareto-ótimas ao longo de todo o seu processamento realizando apenas 7,3% da quantidade equivalente de avaliações feitas pelo PSA.

Os resultados apresentados também permitiram verificar que, dependendo dos parâmetros de entrada dos algoritmos PSA e NSGA-II, o número de avaliações de soluções candidatas pelas metaheurísticas pode ser maior que o tamanho do espaço de busca do problema, o que implica que

houve soluções sendo avaliadas repetidamente e, no entanto, não garante que todas as soluções distintas do espaço foram testadas. Dependendo do grau de dispersão das soluções candidatas do algoritmo, ele estará explorando repetidamente somente um sub-espaço do problema. Afinal, tanto o PSA quanto o NSGA-II são algoritmos probabilísticos, isto é, eles testam novas soluções através de modificações aleatórias nas soluções atuais. Este fato reforça a necessidade de se aprofundar o estudo sobre os parâmetros de entrada a serem utilizados pelos algoritmos PSA e NSGA-II de acordo com o tamanho do problema.

A mesma análise foi realizada para outras topologias de rede, que se diferenciavam em termos de tamanho e densidade, e permitiu concluir pela capacidade das metaheurísticas de apresentar uma boa aproximação da solução ideal para o problema de posicionamento dos controladores. Em geral, as metaheurísticas apresentaram vantagem significativa em termo de tempo de processamento em relação à análise exaustiva. Entretanto, os resultados não satisfatórios obtidos pelo NSGA-II para topologias de redes menores indicaram para a importância da calibração adequada dos seus parâmetros de entrada. Apesar disso, em todos os casos avaliados, o NSGA-II apresentou vantagem em relação ao PSA em termos de qualidade das soluções geradas e capacidade de encontrar um conjunto estimado de soluções Pareto-ótimas mais próximo do real ao longo de todo o seu processamento.

Dentre os objetivos propostos, este trabalho contribuiu para a extensão de funcionalidades disponíveis no POCO por meio da implementação da avaliação de novas medidas de desempenho de rede, relacionadas à taxa de transmissão, e do algoritmo genético NSGA-II, além de ampliar a análise exaustiva para n funções objetivo, uma vez que na versão original a análise era realizada para até quatro funções objetivo.

Como trabalho futuro, espera-se ampliar o estudo para o desenvolvimento de técnicas de alocação dinâmica dos *switches* aos controladores, com base em medidas de desempenho de rede dinâmicas capturadas em tempo real pelo controlador. Pretende-se ainda estudar a implantação e resolução do problema de posicionamento de controladores na presença de tráfego de dados. Neste caso, faz necessária a implementação de novas medidas de desempenho de rede relacionadas, por exemplo, a vazão efetiva da rede e medidas de atrasos incorridos por tempo de fila, processamento, propagação e transmissão.

Para tanto, propõe-se também o aprimoramento de ferramentas que permitam medir o desempenho da rede habilitada com OpenFlow no Mininet, uma vez que as ferramentas atuais focam na avaliação de desempenho do controlador, e não da rede como um todo.

Outro desafio que ainda persiste nesse campo de pesquisa é a limitação do Mininet para o tratamento de *loops* quando o controlador é utilizado no modo *in-band*, ou seja, como parte da infraestrutura da rede emulada. Para tanto, sugere-se a utilização de protocolos como o STP. Entretanto, apesar de ainda prover mecanismos de redundância, que aumenta a confiabilidade da rede, o STP remove qualquer melhoria de desempenho possível em redes habilitadas com múltiplos caminhos. Nexte contexto, novos estudos deverão ser desempenhados em prol da alta disponibilidade e da utilização eficiente de todos os recursos das redes habilitadas com múltiplos caminhos.

No contexto das metaheurísticas, propõe-se a possibilidade de estudá-las sob a ótica de outras

medidas além da distância geracional e o estudo de um método híbrido capaz de herdar os benefícios apresentados pelos algoritmos PSA e NSGA-II, como a rápida convergência apresentada pelo PSA e a capacidade do NSGA-II de incorporar dispersão ao longo de todo processamento.

REFERÊNCIAS BIBLIOGRÁFICAS

AKYILDIZ, I.; LEE, A.; WANG, P.; LUO, M.; CHOU, W. A roadmap for traffic engineering in software defined networks. *Computer Networks*, v. 71, p. 1–30, 2014.

BARRETT, R.; HAAR, S.; WHITESTONE, R. Routing snafu causes internet outage. *Interactive Week*, v. 25, 1997.

COELLO, C. A. C. *Recent trends in evolutionary multiobjective optimization.* [S.l.]: Springer London, 2005. 7-32 p.

COELLO, C. A. C.; CORTES, N. C. Solving multiobjective optimization problems using an artificial immune system. In: *Genetic Programming and Evolvable Machines.* [S.l.]: Springer, 2005. v. 6, n. 2, p. 163–190.

COELLO, C. C.; LAMONT, G. B.; VELDHUIZEN, D. A. van. *Evolutionary Algorithms for Solving Multi-Objective Problems.* 2nd. ed.. ed. [S.l.]: Springer, 2007.

CURTIS, A. R.; MOGUL, J. C.; TOURRILHE, J.; YALAGANDULA, P.; SHARMA, P.; BANERJEE, S. DevoFlow: scaling flow management for highperformance networks. *ACM SIGCOMM Computer Communication*, v. 41, n. 4, p. 254–265, 2011.

CZYZAK, P.; JASZKIEWICZ, A. Pareto simulated annealing — a metaheuristic technique for multiple-objective combinatorial optimization. *J. Multi-Criteria Decision Anal.*, v. 7, p. 34–47, 1998.

DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, v. 6, n. 2, p. 182–197, 2002.

EIBEN A.E., S. S. Evolutionary algorithm parameters and methods to tune them. In: *Artificial intelligence, Network Institute, Computational Intelligence.* [S.l.]: Springer, 2012. p. 15–36.

EITTENBERGER, P.; GROßMANN, M.; KRIEGER, U. Doubtless in seattle: Exploring the internet delay space. *8th EURO-NGI Conference on Next Generation Internet (NGI)*, p. 149–155, 2012.

FAQ MININET. *FAQ Mininet - Why does my controller, which implements an Ethernet bridge or learning switch, not work with my network which has loops in it? I can't ping anything!* 2017. Disponível em: <https://github.com/mininet/mininet/wiki/FAQ>. Acesso em: 1 de Fevereiro de 2018.

FEAMSTER, N.; BALAKRISHNAN, H. Detecting BGP configuration faults with static analysis. In: *Proc. 2nd Conf. Symp. Netw. Syst. Design Implement.* [S.l.: s.n.], 2005. v. 2, p. 43–56.

FLOODLIGHT. *Floodlight Documentation.* Disponível em: <http://docs.projectfloodlight.org/display/floodlightcontroller/Floodlight+Documentation/>. Acesso em: 1 de Fevereiro de 2018.

FORTIN, F.-A.; PARIZEAU, M. Revisiting the NSGA-II crowding-distance computation. In: *GECCO '13 Proceedings of the 15th annual conference on Genetic and evolutionary computation*. [S.l.: s.n.], 2013. p. 623–630.

GOLDBERG, D. E. Genetic algorithms in search, optimization, and machine learning. *Addison-Wesley*, 1989.

GROßMANN, M.; SCHUBERTH, S. J. A. Auto-mininet: Assessing the internet topology zoo in a software-defined network emulator. *7ter Workshop Leitungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und verteilten Systemen 2013 (MMBnet 2013)*, v. 7, 2013.

GUDE, N.; KOPONEN, T.; PETTIT, J.; PFAFF, B.; CASADO, M.; MCKEOWN, N.; SHENKER, S. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, v. 38, n. 3, p. 105–110, 2008.

HARTMANN, M.; HOCK, D.; GEBERT, S.; ZINNER, T.; TRAN-GIA, P. POCO-PLC: Enabling dynamic pareto-optimal resilient controller placement in SDN networks. In: *Computer Communications Workshops (INFOCOM WKSHPS)*. [S.l.: s.n.], 2014.

HELLER, B.; SHERWOOD, R.; MCKEOWN, N. The controller placement problem. In: *Proc. HotSDN*. [S.l.: s.n.], 2012. p. 7–12.

HOCK, D.; GEBERT, S.; HARTMANN, M.; ZINNER, T.; TRAN-GIA, P. POCO: A framework for the pareto-optimal resilient controller placement in SDN-based core networks. In: *Network Operations and Management Symposium (NOMS)*. [S.l.: s.n.], 2014.

HOCK, D.; HARTMANN, M.; GEBERT, S.; JARSCHEL, M.; ZINNER, T.; TRAN-GIA, P. Pareto-optimal resilient controller placement in SDN-based core networks. In: *Teletraffic Congress (ITC), 2013 25th International*. [S.l.: s.n.], 2013. p. 1–9.

IEEE. ANSI/IEEE std 802.1D. *LAN/MAN Standards Committee of the IEEE Computer Society, 1998 Edition, Part 3: Media Access Control (MAC) Bridges*, 1998.

IPERF. *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*. Disponível em: <https://iperf.fr/>. Acesso em: 1 de Fevereiro de 2018.

IRAWATI, I. D.; NURUZZAMANIRRIDHA, M. Spanning Tree Protocol simulation based on Software Defined Network using Mininet emulator. In: *International Conference on Soft Computing, Intelligence Systems, and Information Technology ICSIIT 2015: Intelligence in the Era of Big Data*. [S.l.]: Springer, 2015. p. 395–403.

JAIN, R.; PAUL, S. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, v. 51, n. 11, p. 24–31, 2013.

JALILI, A.; AHMADI, V.; KESHTGARI, M.; KAZEMI, M. Controller placement in software-defined wan using multi objective genetic algorithm. *Knowledge-Based Engineering and Innovation*, 2015.

JARSCHEL, M.; ZINNER, T.; HOßFELD, T.; TRAN-GIA, P.; KELLERER, W. Interfaces attributes and use cases: A compass for SDN. *IEEE Communications Magazine*, v. 52, n. 6, p. 210–217, 2014.

KNIGHT, S.; NGUYEN, H.; FALKNER, N.; BOWDEN, R.; ROUGHAN, M. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, v. 29, n. 9, p. 1765–1775, 2011.

KONAK, A.; COIT, D.; SMITH, A. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering System Safety*, v. 91, n. 9, p. 992–1007, 2006.

KOUVELIS, P.; CHIANG, W. A simulated annealing procedure for single row layout problems in flexible manufacturing systems. *Int J Product Res 30*, p. 717–732, 1992.

KREUTZ, D.; RAMOS, F. M. V.; VERíSSIMO, P. E.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, v. 103, n. 1, p. 14–76, 2015.

KUMAR, P.; SUMAN, B. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, v. 57, p. 1143–1160, 2006.

LANGE, S.; GEBERT, S.; SPOERHASE, J.; RYGIELSKI, P.; ZINNER, T.; KOUNEV, S.; TRAN-GIA, P. Specialized heuristics for the controller placement problem in large scale SDN networks. *Teletraffic Congress (ITC 27), 2015 27th International*, p. 210–218, 2015.

LANGE, S.; GEBERT, S.; ZINNER, T.; TRAN-GIA, P.; HOCK, D.; JARSCHEL, M.; HOFFMANN, M. Heuristic approaches to the controller placement problem in large scale SDN networks. *IEEE Transactions on Network and Service Management*, v. 12, n. 1, p. 4–17, 2015.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, v. 38, n. 2, p. 69–74, 2008.

METROPOLIS, N.; ROSENBLUTH, A. W.; ROSENBLUTH, M. N.; TELLER, A. H.; TELLER, E. Equations of state calculations by fast computing machines. *J. Chem. Phys.*, v. 21, p. 1087–1092, 1953.

MININET. *An Instant Virtual Network on your Laptop (or other PC)*. Disponível em: <http://mininet.org/>. Acesso em: 1 de Fevereiro de 2018.

Mininet - Wiki. *Publications*. 2018. Disponível em: <https://github.com/mininet/mininet/wiki/Publications>. Acesso em: 1 de Fevereiro de 2018.

NUNES, B.; MENDONCA, M.; NGUYEN, X.; OBRACZKA, K.; TURLETTI, T. A survey of software-defined networking: Past present and future of programmable networks. *IEEE Communications Surveys Tutorials*, v. 16, n. 3, p. 1617–1634, 2014.

NUNES, B. A.; SANTOS, M. A.; OLIVEIRA, B. T. de; MARGI, C. B.; OBRACZKA, K.; TURLETTI, T. Software-defined-networking-enabled capacity sharing in user-centric networks. *IEEE Communications Magazine*, v. 52, n. 9, p. 28–36, 2014.

OPEN NETWORKING FOUNDATION. 2011. Disponível em: <https://www.opennetworking.org/about>. Acesso em: 1 de Fevereiro de 2018.

OPEN NETWORKING RESEARCH CENTER - ONRC. 2014. Disponível em: <http://onrc.net>. Acesso em: 1 de Fevereiro de 2018.

PEREZ, M. A. F.; RAUPP, F. M. P. *A heuristic method for multiobjective scheduling problem in various machine environments*. [S.l.]: Pontíficia Universidade Católica do Rio de Janeiro, 2012.

PHEMIUS, K.; BOUET, M. Openflow: Why latency does matter. *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, 2013.

SERAFINI, P. Simulated annealing for multi objective optimization problem. In: *Multiple Criteria Decision Making*. [S.l.]: Springer, 1993. p. 283–292.

STRIBLING, J.; SOVRAN, Y.; ZHANG, I.; PRETZER, X.; LI, J.; KAASHOEK, M. F.; MORRIS, R. Flexible, wide-area storage for distributed systems with WheelFS. In: *In Proceedings of the 6th USENIX symposium on Networked systems design and implementation, NSDI'09*. [S.l.: s.n.], 2009. v. 9, p. 43–58.

TOOTOONCHIAN, A.; GANJALI, Y. Hyperflow: A distributed control plane for openflow. In: *Proc. INM/WREN*. [S.l.: s.n.], 2010. p. 1–6.

VELDHUIZEN, D. A. V.; LAMONT, G. B. Evolutionary computation and convergence to a pareto front. In: *In Koza, J. R., editor, Late Breaking Papers at the Genetic Programming 1998 Conference*. [S.l.: s.n.], 1998. p. 221–228.

VELDHUIZEN, D. A. V.; LAMONT, G. B. On measuring multiobjective evolutionary algorithm performance. In: *Evolutionary Computation. Proceedings of the 2000 Congress on*. [S.l.: s.n.], 2000. v. 1, p. 204–211.

WIRESHARK. Disponível em: <https://www.wireshark.org/>. Acesso em: 1 de Fevereiro de 2018.

YAO, G.; BI, J.; LI, Y.; GUO, L. On the capacitated controller placement problem in software defined networks. *IEEE Communications Letters*, v. 18, n. 8, p. 1339–1342, 2014.

YARPIZ. *YARPIZ*. 2018. Disponível em: <http://www.yarpiz.com/>. Acesso em: 1 de Fevereiro de 2018.

ANEXOS

I. TOPOLOGIA DA REDE INTERNET2 OS3E PARA O MININET

internet2OS3E.py

```
#!/usr/bin/python

"""
Custom topology for Mininet, generated by GraphML-Topo-to-Mininet-Network-
    ↪ Generator.
"""
from mininet.net import Mininet
from mininet.node import RemoteController, OVSKernelSwitch, Host, Node
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from time import sleep


def emptyNet():

        "Create␣a␣network."

        # Initialize Topology
        net = Mininet( topo=None, build=False, link=TCLink, ipBase='10.0.0.0/8')

        # add controller
        info( '***␣Adding␣controller\n' )
        c0= net.addController(name='c0', controller=RemoteController, ip='
            ↪ 10.0.0.200') #started in srv with xterm
        server = net.addHost('srv', ip='10.0.0.200')

        # add nodes, switches first...
        info ( '***␣Creating␣switches\n' )
        SunnyvaleCA = net.addSwitch( 's1', cls=OVSKernelSwitch, inband=True)
        Nashville = net.addSwitch( 's2', cls=OVSKernelSwitch, inband=True)
        RaleighNC = net.addSwitch( 's3', cls=OVSKernelSwitch, inband=True)
        Chicago = net.addSwitch( 's4', cls=OVSKernelSwitch, inband=True)
```

```
ElPasoTX = net.addSwitch( 's5', cls=OVSKernelSwitch, inband=True)
Denver = net.addSwitch( 's6', cls=OVSKernelSwitch, inband=True)
Dallas = net.addSwitch( 's7', cls=OVSKernelSwitch, inband=True)
Louisville = net.addSwitch( 's8', cls=OVSKernelSwitch, inband=True)
Vancouver = net.addSwitch( 's9', cls=OVSKernelSwitch, inband=True)
WashingtonDC = net.addSwitch( 's10', cls=OVSKernelSwitch, inband=True)
Indianapolis = net.addSwitch( 's11', cls=OVSKernelSwitch, inband=True)
Pittsburgh = net.addSwitch( 's12', cls=OVSKernelSwitch, inband=True)
BatonRouge = net.addSwitch( 's13', cls=OVSKernelSwitch, inband=True)
Albuquerque = net.addSwitch( 's14', cls=OVSKernelSwitch, inband=True)
LosAngeles = net.addSwitch( 's15', cls=OVSKernelSwitch, inband=True)
Atlanta = net.addSwitch( 's16', cls=OVSKernelSwitch, inband=True)
Memphis = net.addSwitch( 's17', cls=OVSKernelSwitch, inband=True)
Jacksonville = net.addSwitch( 's18', cls=OVSKernelSwitch, inband=True)
Miami = net.addSwitch( 's19', cls=OVSKernelSwitch, inband=True)
KansasCityMO = net.addSwitch( 's20', cls=OVSKernelSwitch, inband=True)
Missoula = net.addSwitch( 's21', cls=OVSKernelSwitch, inband=True)
Philadelphia = net.addSwitch( 's22', cls=OVSKernelSwitch, inband=True)
Tucson = net.addSwitch( 's23', cls=OVSKernelSwitch, inband=True)
Buffalo = net.addSwitch( 's24', cls=OVSKernelSwitch, inband=True)
Houston = net.addSwitch( 's25', cls=OVSKernelSwitch, inband=True)
Boston = net.addSwitch( 's26', cls=OVSKernelSwitch, inband=True)
Minneapolis = net.addSwitch( 's27', cls=OVSKernelSwitch, inband=True)
NewYork = net.addSwitch( 's28', cls=OVSKernelSwitch, inband=True)
SaltLakeCity = net.addSwitch( 's29', cls=OVSKernelSwitch, inband=True)
Cleveland = net.addSwitch( 's30', cls=OVSKernelSwitch, inband=True)
JacksonMS = net.addSwitch( 's31', cls=OVSKernelSwitch, inband=True)
Portland = net.addSwitch( 's32', cls=OVSKernelSwitch, inband=True)
Seattle = net.addSwitch( 's33', cls=OVSKernelSwitch, inband=True)
AshburnVA = net.addSwitch( 's34', cls=OVSKernelSwitch, inband=True)

# ... and now hosts
info ( '*** Adding hosts\n' )
SunnyvaleCAHost = net.addHost( 'h1', cls=Host, ip='10.0.0.201',
    ↪ defaultRoute=None)
NashvilleHost = net.addHost( 'h2', cls=Host, ip='10.0.0.202', defaultRoute
    ↪ =None)
RaleighNCHost = net.addHost( 'h3', cls=Host, ip='10.0.0.203', defaultRoute
    ↪ =None)
ChicagoHost = net.addHost( 'h4', cls=Host, ip='10.0.0.204', defaultRoute=
    ↪ None)
```

```
ElPasoTXHost = net.addHost( 'h5', cls=Host, ip='10.0.0.205', defaultRoute=
    ↪ None)
DenverHost = net.addHost( 'h6', cls=Host, ip='10.0.0.206', defaultRoute=
    ↪ None)
DallasHost = net.addHost( 'h7', cls=Host, ip='10.0.0.207', defaultRoute=
    ↪ None)
LouisvilleHost = net.addHost( 'h8', cls=Host, ip='10.0.0.208',
    ↪ defaultRoute=None)
VancouverHost = net.addHost( 'h9', cls=Host, ip='10.0.0.209', defaultRoute
    ↪ =None)
WashingtonDCHost = net.addHost( 'h10', cls=Host, ip='10.0.0.210',
    ↪ defaultRoute=None)
IndianapolisHost = net.addHost( 'h11', cls=Host, ip='10.0.0.211',
    ↪ defaultRoute=None)
PittsburghHost = net.addHost( 'h12', cls=Host, ip='10.0.0.212',
    ↪ defaultRoute=None)
BatonRougeHost = net.addHost( 'h13', cls=Host, ip='10.0.0.213',
    ↪ defaultRoute=None)
AlbuquerqueHost = net.addHost( 'h14', cls=Host, ip='10.0.0.214',
    ↪ defaultRoute=None)
LosAngelesHost = net.addHost( 'h15', cls=Host, ip='10.0.0.215',
    ↪ defaultRoute=None)
AtlantaHost = net.addHost( 'h16', cls=Host, ip='10.0.0.216', defaultRoute=
    ↪ None)
MemphisHost = net.addHost( 'h17', cls=Host, ip='10.0.0.217', defaultRoute=
    ↪ None)
JacksonvilleHost = net.addHost( 'h18', cls=Host, ip='10.0.0.218',
    ↪ defaultRoute=None)
MiamiHost = net.addHost( 'h19', cls=Host, ip='10.0.0.219', defaultRoute=
    ↪ None)
KansasCityMOHost = net.addHost( 'h20', cls=Host, ip='10.0.0.220',
    ↪ defaultRoute=None)
MissoulaHost = net.addHost( 'h21', cls=Host, ip='10.0.0.221', defaultRoute
    ↪ =None)
PhiladelphiaHost = net.addHost( 'h22', cls=Host, ip='10.0.0.222',
    ↪ defaultRoute=None)
TucsonHost = net.addHost( 'h23', cls=Host, ip='10.0.0.223', defaultRoute=
    ↪ None)
BuffaloHost = net.addHost( 'h24', cls=Host, ip='10.0.0.224', defaultRoute=
    ↪ None)
HoustonHost = net.addHost( 'h25', cls=Host, ip='10.0.0.225', defaultRoute=
    ↪ None)
```

```
BostonHost = net.addHost( 'h26', cls=Host, ip='10.0.0.226', defaultRoute=
    ↪ None)
MinneapolisHost = net.addHost( 'h27', cls=Host, ip='10.0.0.227',
    ↪ defaultRoute=None)
NewYorkHost = net.addHost( 'h28', cls=Host, ip='10.0.0.228', defaultRoute=
    ↪ None)
SaltLakeCityHost = net.addHost( 'h29', cls=Host, ip='10.0.0.229',
    ↪ defaultRoute=None)
ClevelandHost = net.addHost( 'h30', cls=Host, ip='10.0.0.230',
    ↪ defaultRoute=None)
JacksonMSHost = net.addHost( 'h31', cls=Host, ip='10.0.0.231',
    ↪ defaultRoute=None)
PortlandHost = net.addHost( 'h32', cls=Host, ip='10.0.0.232', defaultRoute
    ↪ =None)
SeattleHost = net.addHost( 'h33', cls=Host, ip='10.0.0.233', defaultRoute=
    ↪ None)
AshburnVAHost = net.addHost( 'h34', cls=Host, ip='10.0.0.234',
    ↪ defaultRoute=None)

info( '*** Creating link to controller\n' )
net.addLink('srv', 's20' )

# add edges between switch and corresponding host
info( '*** Creating links to hosts\n' )
net.addLink( SunnyvaleCA , SunnyvaleCAHost )
net.addLink( Nashville , NashvilleHost )
net.addLink( RaleighNC , RaleighNCHost )
net.addLink( Chicago , ChicagoHost )
net.addLink( ElPasoTX , ElPasoTXHost )
net.addLink( Denver , DenverHost )
net.addLink( Dallas , DallasHost )
net.addLink( Louisville , LouisvilleHost )
net.addLink( Vancouver , VancouverHost )
net.addLink( WashingtonDC , WashingtonDCHost )
net.addLink( Indianapolis , IndianapolisHost )
net.addLink( Pittsburgh , PittsburghHost )
net.addLink( BatonRouge , BatonRougeHost )
net.addLink( Albuquerque , AlbuquerqueHost )
net.addLink( LosAngeles , LosAngelesHost )
net.addLink( Atlanta , AtlantaHost )
net.addLink( Memphis , MemphisHost )
net.addLink( Jacksonville , JacksonvilleHost )
```

```
net.addLink( Miami , MiamiHost )
net.addLink( KansasCityMO , KansasCityMOHost )
net.addLink( Missoula , MissoulaHost )
net.addLink( Philadelphia , PhiladelphiaHost )
net.addLink( Tucson , TucsonHost )
net.addLink( Buffalo , BuffaloHost )
net.addLink( Houston , HoustonHost )
net.addLink( Boston , BostonHost )
net.addLink( Minneapolis , MinneapolisHost )
net.addLink( NewYork , NewYorkHost )
net.addLink( SaltLakeCity , SaltLakeCityHost )
net.addLink( Cleveland , ClevelandHost )
net.addLink( JacksonMS , JacksonMSHost )
net.addLink( Portland , PortlandHost )
net.addLink( Seattle , SeattleHost )
net.addLink( AshburnVA , AshburnVAHost )

# add edges between switches
info( '*** Creating links between switches\n' )
net.addLink( Vancouver , Seattle, bw=10, delay='0.981327242448ms')
net.addLink( Seattle , Portland, bw=4, delay='1.18964695145ms')
net.addLink( Seattle , SaltLakeCity, bw=2, delay='5.72307999339ms')
net.addLink( Seattle , Missoula, bw=2, delay='3.22205220818ms')
net.addLink( Portland , SunnyvaleCA, bw=2, delay='4.60768220921ms')
net.addLink( SunnyvaleCA , LosAngeles, bw=10, delay='2.5576487616ms')
net.addLink( SunnyvaleCA , SaltLakeCity, bw=10, delay='4.84338687672ms')
net.addLink( SaltLakeCity , Denver, bw=8, delay='3.0285160699ms')
net.addLink( Missoula , Minneapolis, bw=1, delay='8.19416113684ms')
net.addLink( LosAngeles , Tucson, bw=8, delay='3.59383432864ms')
net.addLink( Tucson , ElPasoTX, bw=10, delay='2.16385297881ms')
net.addLink( ElPasoTX , Albuquerque, bw=10, delay='1.8804042425ms')
net.addLink( Albuquerque , Denver, bw=10, delay='2.73372668506ms')
net.addLink( ElPasoTX , Houston, bw=2, delay='5.51266089799ms')
net.addLink( Denver , KansasCityMO, bw=2, delay='4.5554374722ms')
net.addLink( Houston , Dallas, bw=10, delay='1.83892529045ms')
net.addLink( Dallas , KansasCityMO, bw=8, delay='3.71415948227ms')
net.addLink( Houston , BatonRouge, bw=10, delay='2.08076995771ms')
net.addLink( BatonRouge , Jacksonville, bw=2, delay='4.64500830895ms')
net.addLink( Jacksonville , Miami, bw=10, delay='2.6951145462ms')
net.addLink( Jacksonville , Atlanta, bw=10, delay='2.33311908223ms')
net.addLink( Houston , JacksonMS, bw=10, delay='2.89268053888ms')
net.addLink( JacksonMS , Memphis, bw=10, delay='1.61221095621ms')
```

```
net.addLink( Memphis , Nashville, bw=10, delay='1.60789062383ms')
net.addLink( Nashville , Atlanta, bw=10, delay='1.75814505891ms')
net.addLink( Nashville , Louisville, bw=10, delay='1.26422085221ms')
net.addLink( Louisville , Indianapolis, bw=10, delay='0.870831823627ms')
net.addLink( Indianapolis , Chicago, bw=10, delay='1.35390966832ms')
net.addLink( Minneapolis , Chicago, bw=10, delay='2.89717670035ms')
net.addLink( KansasCityMO , Chicago, bw=10, delay='3.37360840214ms')
net.addLink( Chicago , Cleveland, bw=10, delay='2.51580035832ms')
net.addLink( Cleveland , Buffalo, bw=10, delay='1.41225696982ms')
net.addLink( Buffalo , Boston, bw=8, delay='3.26472097927ms')
net.addLink( Boston , NewYork, bw=10, delay='1.55559299529ms')
net.addLink( NewYork , Philadelphia, bw=10, delay='0.658169135175ms')
net.addLink( Philadelphia , WashingtonDC, bw=10, delay='1.01292714509ms')
net.addLink( WashingtonDC , AshburnVA, bw=10, delay='0.782530310362ms')
net.addLink( AshburnVA , Pittsburgh, bw=10, delay='0.775958619848ms')
net.addLink( Pittsburgh , Cleveland, bw=10, delay='0.940531004777ms')
net.addLink( WashingtonDC , RaleighNC, bw=10, delay='1.89777711892ms')
net.addLink( RaleighNC , Atlanta, bw=10, delay='2.90545728302ms')
net.addLink( SaltLakeCity , LosAngeles, bw=2, delay='4.74057138487ms')

# starting network
info( '*** Starting network\n')
net.build()
c0.start()

for n in range (1,35):
       net.get('s%d' %n).start([c0])

# set switches ip address
SunnyvaleCA.cmd('sudo ifconfig s1 10.0.0.101 up');
Nashville.cmd('sudo ifconfig s2 10.0.0.102 up');
RaleighNC.cmd('sudo ifconfig s3 10.0.0.103 up');
Chicago.cmd('sudo ifconfig s4 10.0.0.104 up');
ElPasoTX.cmd('sudo ifconfig s5 10.0.0.105 up');
Denver.cmd('sudo ifconfig s6 10.0.0.106 up');
Dallas.cmd('sudo ifconfig s7 10.0.0.107 up');
Louisville.cmd('sudo ifconfig s8 10.0.0.108 up');
Vancouver.cmd('sudo ifconfig s9 10.0.0.109 up');
WashingtonDC.cmd('sudo ifconfig s10 10.0.0.110 up');
Indianapolis.cmd('sudo ifconfig s11 10.0.0.111 up');
Pittsburgh.cmd('sudo ifconfig s12 10.0.0.112 up');
BatonRouge.cmd('sudo ifconfig s13 10.0.0.113 up');
```

```
Albuquerque.cmd('sudo␣ifconfig␣s14␣10.0.0.114␣up');
LosAngeles.cmd('sudo␣ifconfig␣s15␣10.0.0.115␣up');
Atlanta.cmd('sudo␣ifconfig␣s16␣10.0.0.116␣up');
Memphis.cmd('sudo␣ifconfig␣s17␣10.0.0.117␣up');
Jacksonville.cmd('sudo␣ifconfig␣s18␣10.0.0.118␣up');
Miami.cmd('sudo␣ifconfig␣s19␣10.0.0.119␣up');
KansasCityMO.cmd('sudo␣ifconfig␣s20␣10.0.0.120␣up');
Missoula.cmd('sudo␣ifconfig␣s21␣10.0.0.121␣up');
Philadelphia.cmd('sudo␣ifconfig␣s22␣10.0.0.122␣up');
Tucson.cmd('sudo␣ifconfig␣s23␣10.0.0.123␣up');
Buffalo.cmd('sudo␣ifconfig␣s24␣10.0.0.124␣up');
Houston.cmd('sudo␣ifconfig␣s25␣10.0.0.125␣up');
Boston.cmd('sudo␣ifconfig␣s26␣10.0.0.126␣up');
Minneapolis.cmd('sudo␣ifconfig␣s27␣10.0.0.127␣up');
NewYork.cmd('sudo␣ifconfig␣s28␣10.0.0.128␣up');
SaltLakeCity.cmd('sudo␣ifconfig␣s29␣10.0.0.129␣up');
Cleveland.cmd('sudo␣ifconfig␣s30␣10.0.0.130␣up');
JacksonMS.cmd('sudo␣ifconfig␣s31␣10.0.0.131␣up');
Portland.cmd('sudo␣ifconfig␣s32␣10.0.0.132␣up');
Seattle.cmd('sudo␣ifconfig␣s33␣10.0.0.133␣up');
AshburnVA.cmd('sudo␣ifconfig␣s34␣10.0.0.134␣up');

# set spanning tree
SunnyvaleCA.cmd('ovs-vsctl␣set␣bridge␣s1␣stp-enable=true')
Nashville.cmd('ovs-vsctl␣set␣bridge␣s2␣stp-enable=true')
RaleighNC.cmd('ovs-vsctl␣set␣bridge␣s3␣stp-enable=true')
Chicago.cmd('ovs-vsctl␣set␣bridge␣s4␣stp-enable=true')
ElPasoTX.cmd('ovs-vsctl␣set␣bridge␣s5␣stp-enable=true')
Denver.cmd('ovs-vsctl␣set␣bridge␣s6␣stp-enable=true')
Dallas.cmd('ovs-vsctl␣set␣bridge␣s7␣stp-enable=true')
Louisville.cmd('ovs-vsctl␣set␣bridge␣s8␣stp-enable=true')
Vancouver.cmd('ovs-vsctl␣set␣bridge␣s9␣stp-enable=true')
WashingtonDC.cmd('ovs-vsctl␣set␣bridge␣s10␣stp-enable=true')
Indianapolis.cmd('ovs-vsctl␣set␣bridge␣s11␣stp-enable=true')
Pittsburgh.cmd('ovs-vsctl␣set␣bridge␣s12␣stp-enable=true')
BatonRouge.cmd('ovs-vsctl␣set␣bridge␣s13␣stp-enable=true')
Albuquerque.cmd('ovs-vsctl␣set␣bridge␣s14␣stp-enable=true')
LosAngeles.cmd('ovs-vsctl␣set␣bridge␣s15␣stp-enable=true')
Atlanta.cmd('ovs-vsctl␣set␣bridge␣s16␣stp-enable=true')
Memphis.cmd('ovs-vsctl␣set␣bridge␣s17␣stp-enable=true')
Jacksonville.cmd('ovs-vsctl␣set␣bridge␣s18␣stp-enable=true')
Miami.cmd('ovs-vsctl␣set␣bridge␣s19␣stp-enable=true')
```

```
        KansasCityMO.cmd('ovs-vsctl␣set␣bridge␣s20␣stp-enable=true')
        Missoula.cmd('ovs-vsctl␣set␣bridge␣s21␣stp-enable=true')
        Philadelphia.cmd('ovs-vsctl␣set␣bridge␣s22␣stp-enable=true')
        Tucson.cmd('ovs-vsctl␣set␣bridge␣s23␣stp-enable=true')
        Buffalo.cmd('ovs-vsctl␣set␣bridge␣s24␣stp-enable=true')
        Houston.cmd('ovs-vsctl␣set␣bridge␣s25␣stp-enable=true')
        Boston.cmd('ovs-vsctl␣set␣bridge␣s26␣stp-enable=true')
        Minneapolis.cmd('ovs-vsctl␣set␣bridge␣s27␣stp-enable=true')
        NewYork.cmd('ovs-vsctl␣set␣bridge␣s28␣stp-enable=true')
        SaltLakeCity.cmd('ovs-vsctl␣set␣bridge␣s29␣stp-enable=true')
        Cleveland.cmd('ovs-vsctl␣set␣bridge␣s30␣stp-enable=true')
        JacksonMS.cmd('ovs-vsctl␣set␣bridge␣s31␣stp-enable=true')
        Portland.cmd('ovs-vsctl␣set␣bridge␣s32␣stp-enable=true')
        Seattle.cmd('ovs-vsctl␣set␣bridge␣s33␣stp-enable=true')
        AshburnVA.cmd('ovs-vsctl␣set␣bridge␣s34␣stp-enable=true')

        info( '***␣Running␣CLI\n' )
        CLI( net )

        info( '***␣Stopping␣network' )
        net.stop()


if __name__ == '__main__':
    setLogLevel( 'info' )
    emptyNet()
```

II. TAXA DE TRANSMISSÃO ENTRE OS NÓS

NÓ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	Inf	2,72	2,70	2,72	2,74	8,63	2,69	2,69	1,67	2,68	2,69	2,72	2,72	8,62	2,70	2,74	2,71
2	2,72	Inf	10,70	10,70	2,70	2,75	8,63	10,60	1,68	10,60	10,50	10,60	8,71	2,68	2,79	10,70	10,60
3	2,70	10,70	Inf	10,60	2,70	2,70	8,68	10,70	1,67	10,60	10,70	10,60	8,72	2,75	2,60	10,60	10,70
4	2,72	10,70	10,60	Inf	2,70	2,73	8,64	10,60	1,68	10,60	10,60	10,60	8,63	2,70	2,74	10,70	10,60
5	2,74	2,70	2,70	2,70	Inf	2,71	2,70	2,70	1,67	2,70	2,70	2,85	2,72	2,72	2,72	2,70	2,70
6	8,63	2,75	2,70	2,73	2,71	Inf	2,73	2,68	1,67	2,68	2,69	2,68	2,71	10,70	2,70	2,74	2,69
7	2,69	8,63	8,68	8,64	2,70	2,73	Inf	8,64	1,67	8,62	8,67	8,63	10,60	2,68	2,69	8,64	8,63
8	2,69	10,60	10,70	10,60	2,70	2,68	8,64	Inf	1,68	10,70	10,70	10,70	8,63	2,73	2,73	10,70	10,60
9	1,67	1,68	1,67	1,68	1,67	1,67	1,67	1,68	Inf	1,68	1,68	1,68	1,67	1,67	1,67	1,68	1,68
10	2,68	10,60	10,60	10,60	2,70	2,68	8,62	10,70	1,68	Inf	10,70	10,60	8,73	2,70	2,70	10,60	10,70
11	2,69	10,50	10,70	10,60	2,70	2,69	8,67	10,70	1,68	10,70	Inf	10,50	8,63	2,79	2,74	10,70	10,70
12	2,72	10,60	10,60	10,60	2,85	2,68	8,63	10,70	1,68	10,60	10,50	Inf	8,73	2,68	2,70	10,70	10,60
13	2,72	8,71	8,72	8,63	2,72	2,71	10,60	8,63	1,67	8,73	8,63	8,73	Inf	2,65	2,72	8,62	8,73
14	8,62	2,68	2,75	2,70	2,72	10,70	2,68	2,73	1,67	2,70	2,79	2,68	2,65	Inf	2,70	2,66	2,69
15	2,70	2,79	2,60	2,74	2,72	2,70	2,69	2,73	1,67	2,70	2,74	2,70	2,72	2,70	Inf	2,73	2,70
16	2,74	10,70	10,60	10,70	2,70	2,74	8,64	10,70	1,68	10,60	10,70	10,70	8,62	2,66	2,73	Inf	10,50
17	2,71	10,60	10,70	10,60	2,70	2,69	8,63	10,60	1,68	10,70	10,70	10,60	8,73	2,69	2,70	10,50	Inf
18	2,74	2,72	2,70	2,70	2,72	2,71	2,70	2,70	1,67	2,70	2,70	2,70	2,70	2,70	2,68	2,70	2,70
19	2,78	2,70	2,70	2,85	2,70	2,70	2,85	2,71	1,67	2,70	2,85	2,70	2,72	2,71	2,76	2,78	2,72
20	2,69	10,70	10,70	10,70	2,70	2,70	8,57	10,60	1,68	10,70	10,60	10,60	10,50	2,69	2,70	10,70	10,70
21	1,67	1,68	1,68	1,68	1,67	1,68	1,68	1,68	2,69	1,68	1,68	1,68	1,67	1,67	1,67	1,67	1,68
22	2,73	10,50	10,60	10,60	2,70	2,58	8,71	10,70	1,67	10,60	10,60	10,50	8,61	2,59	2,69	10,70	10,60
23	2,72	2,70	2,70	2,70	10,60	2,72	2,70	2,70	1,67	2,70	2,70	2,70	2,70	2,71	2,74	2,78	2,70
24	2,68	10,60	10,60	10,60	2,70	2,71	8,63	10,70	1,68	10,60	10,70	10,60	8,74	2,70	2,70	10,70	10,70
25	2,69	8,63	8,73	8,64	2,72	2,73	10,60	8,63	1,67	8,74	8,63	8,66	10,60	2,71	2,71	8,72	8,73
26	2,69	8,63	8,64	8,71	2,71	2,71	8,71	8,64	1,67	8,64	8,64	8,65	8,72	2,71	2,71	8,63	8,63
27	2,71	10,60	10,70	10,50	2,85	2,75	8,66	10,60	1,68	10,70	10,70	10,60	8,71	2,71	2,71	10,70	10,70
28	2,73	8,63	8,63	8,64	2,71	2,71	8,73	8,62	1,67	8,64	8,64	8,64	8,71	2,70	2,77	8,74	8,73
29	10,70	2,74	2,66	2,74	2,72	8,62	2,68	2,75	1,67	2,68	2,75	2,68	2,75	8,62	2,70	2,67	2,68
30	2,69	10,70	10,70	10,60	2,70	2,69	8,65	10,70	1,68	10,70	10,70	10,60	8,65	2,68	2,71	10,70	10,60
31	2,70	8,70	8,72	8,63	2,70	2,70	10,60	8,73	1,67	8,73	8,74	8,73	10,70	2,69	2,32	8,70	8,72
32	1,67	1,68	1,67	1,68	1,67	1,67	1,67	1,68	4,79	1,68	1,68	1,67	1,67	1,67	1,67	1,67	1,67
33	1,65	1,66	1,61	1,63	1,67	1,66	1,60	1,68	10,60	1,67	1,68	1,68	1,67	1,67	1,67	1,67	1,67
34	2,55	10,60	10,50	10,60	2,70	2,74	8,52	10,60	1,68	10,40	10,40	10,60	8,54	2,65	2,58	10,50	10,50
Nó	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**
1	2,74	2,78	2,69	1,67	2,73	2,72	2,68	2,69	2,69	2,71	2,73	10,70	2,69	2,70	1,67	1,65	2,55
2	2,72	2,70	10,70	1,68	10,50	2,70	10,60	8,63	8,63	10,60	8,63	2,74	10,70	8,70	1,68	1,66	10,60
3	2,70	2,70	10,70	1,68	10,60	2,70	10,60	8,73	8,64	10,70	8,63	2,66	10,70	8,72	1,67	1,61	10,50
4	2,70	2,85	10,70	1,68	10,60	2,70	10,60	8,64	8,71	10,50	8,64	2,74	10,60	8,63	1,68	1,63	10,60
5	2,72	2,70	2,70	1,67	2,70	10,60	2,70	2,72	2,71	2,85	2,71	2,72	2,70	2,70	1,67	1,67	2,70
6	2,71	2,70	2,70	1,68	2,58	2,72	2,71	2,73	2,71	2,75	2,71	8,62	2,69	2,70	1,67	1,66	2,74
7	2,70	2,85	8,57	1,68	8,71	2,70	8,63	10,60	8,71	8,66	8,73	2,68	8,65	10,60	1,67	1,60	8,52
8	2,70	2,71	10,60	1,68	10,70	2,70	10,70	8,63	8,64	10,60	8,62	2,75	10,70	8,73	1,68	1,68	10,60
9	1,67	1,67	1,68	2,69	1,67	1,67	1,68	1,67	1,67	1,68	1,67	1,67	1,68	1,67	4,79	10,60	1,68
10	2,70	2,70	10,70	1,68	10,60	2,70	10,60	8,74	8,64	10,70	8,64	2,68	10,70	8,73	1,68	1,67	10,40
11	2,70	2,85	10,60	1,68	10,60	2,70	10,70	8,63	8,64	10,70	8,64	2,75	10,70	8,74	1,68	1,68	10,40
12	2,70	2,70	10,60	1,68	10,50	2,70	10,60	8,66	8,65	10,60	8,64	2,68	10,60	8,73	1,67	1,68	10,60
13	2,70	2,72	10,50	1,67	8,61	2,70	8,74	10,60	8,72	8,71	8,71	2,75	8,65	10,70	1,67	1,67	8,54
14	2,70	2,71	2,69	1,67	2,59	2,71	2,70	2,71	2,71	2,71	2,70	8,62	2,68	2,69	1,67	1,67	2,65
15	2,68	2,76	2,70	1,67	2,69	2,74	2,70	2,71	2,71	2,71	2,77	2,70	2,71	2,32	1,67	1,67	2,58
16	2,70	2,78	10,70	1,67	10,70	2,78	10,70	8,72	8,63	10,70	8,74	2,67	10,70	8,70	1,67	1,67	10,50
17	2,70	2,72	10,70	1,68	10,60	2,70	10,70	8,73	8,63	10,70	8,73	2,68	10,60	8,72	1,67	1,67	10,50
18	Inf	10,60	2,70	1,67	2,63	2,85	2,70	2,70	2,70	2,85	2,70	2,73	2,70	2,70	1,67	1,67	2,70
19	10,60	Inf	2,70	1,67	2,70	2,85	2,70	2,70	2,78	2,71	2,78	2,69	2,70	2,72	1,67	1,67	2,62
20	2,70	2,70	Inf	1,68	10,70	2,70	10,70	8,64	8,63	10,70	8,63	2,75	10,60	8,64	1,68	1,68	10,60
21	1,67	1,67	1,68	Inf	1,68	1,67	1,68	1,68	1,68	1,63	1,68	1,67	1,68	1,67	2,70	2,70	1,68
22	2,63	2,70	10,70	1,68	Inf	2,70	10,70	8,73	8,64	10,70	8,64	2,75	10,70	8,72	1,67	1,67	10,30
23	2,85	2,85	2,70	1,67	2,70	Inf	2,70	2,70	2,72	2,85	2,70	2,75	2,85	2,72	1,67	1,67	2,70
24	2,70	2,70	10,70	1,68	10,70	2,70	Inf	8,63	8,64	10,70	8,60	2,68	10,60	8,73	1,68	1,68	10,50
25	2,70	2,70	8,64	1,68	8,73	2,70	8,63	Inf	8,73	8,63	8,72	2,71	8,69	10,60	1,67	1,67	8,49
26	2,70	2,78	8,63	1,68	8,64	2,72	8,64	8,73	Inf	8,64	10,60	2,74	8,65	8,71	1,67	1,67	8,52
27	2,85	2,71	10,70	1,63	10,70	2,85	10,70	8,63	8,64	Inf	8,63	2,75	10,70	8,73	1,68	1,63	10,70
28	2,70	2,78	8,63	1,68	8,64	2,70	8,60	8,72	10,60	8,63	Inf	2,68	8,64	8,70	1,67	1,67	8,61
29	2,73	2,69	2,75	1,67	2,75	2,75	2,68	2,71	2,74	2,75	2,68	Inf	2,72	2,67	1,67	1,67	2,73
30	2,70	2,70	10,60	1,68	10,70	2,85	10,60	8,69	8,65	10,70	8,64	2,72	Inf	8,71	1,68	1,68	10,60
31	2,70	2,72	8,64	1,67	8,72	2,72	8,73	10,60	8,71	8,73	8,70	2,67	8,71	Inf	1,67	1,67	8,50
32	1,67	1,67	1,68	2,70	1,67	1,67	1,68	1,67	1,67	1,68	1,67	1,67	1,68	1,67	Inf	4,79	1,68
33	1,67	1,67	1,68	2,70	1,67	1,67	1,68	1,67	1,67	1,63	1,67	1,67	1,68	1,67	4,79	Inf	1,68
34	2,70	2,62	10,60	1,68	10,30	2,70	10,50	8,49	8,52	10,70	8,61	2,73	10,60	8,50	1,68	1,68	Inf

III. LATÊNCIA ENTRE OS NÓS

NÓ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0,000	0,019	0,023	0,016	0,008	0,008	0,016	0,018	0,007	0,021	0,017	0,019	0,016	0,010	0,003	0,021	0,018
2	0,019	0,000	0,005	0,003	0,012	0,011	0,008	0,001	0,019	0,007	0,002	0,007	0,008	0,013	0,017	0,002	0,002
3	0,023	0,005	0,000	0,007	0,016	0,015	0,013	0,006	0,022	0,002	0,007	0,003	0,010	0,018	0,022	0,003	0,006
4	0,016	0,003	0,007	0,000	0,013	0,008	0,007	0,002	0,015	0,005	0,001	0,003	0,011	0,011	0,016	0,005	0,005
5	0,008	0,012	0,016	0,013	0,000	0,005	0,007	0,013	0,014	0,018	0,014	0,016	0,008	0,002	0,006	0,013	0,010
6	0,008	0,011	0,015	0,008	0,005	0,000	0,008	0,010	0,010	0,013	0,009	0,011	0,012	0,003	0,008	0,013	0,013
7	0,016	0,008	0,013	0,007	0,007	0,008	0,000	0,009	0,018	0,012	0,008	0,011	0,004	0,009	0,013	0,010	0,006
8	0,018	0,001	0,006	0,002	0,013	0,010	0,009	0,000	0,017	0,007	0,001	0,006	0,009	0,013	0,018	0,003	0,003
9	0,007	0,019	0,022	0,015	0,014	0,010	0,018	0,017	0,000	0,020	0,017	0,019	0,022	0,012	0,009	0,021	0,020
10	0,021	0,007	0,002	0,005	0,018	0,013	0,012	0,007	0,020	0,000	0,006	0,002	0,012	0,016	0,021	0,005	0,008
11	0,017	0,002	0,007	0,001	0,014	0,009	0,008	0,001	0,017	0,006	0,000	0,005	0,010	0,012	0,017	0,004	0,004
12	0,019	0,007	0,003	0,003	0,016	0,011	0,011	0,006	0,019	0,002	0,005	0,000	0,013	0,014	0,019	0,006	0,009
13	0,016	0,008	0,010	0,011	0,008	0,012	0,004	0,009	0,022	0,012	0,010	0,013	0,000	0,009	0,013	0,007	0,007
14	0,010	0,013	0,018	0,011	0,002	0,003	0,009	0,013	0,012	0,016	0,012	0,014	0,009	0,000	0,008	0,015	0,012
15	0,003	0,017	0,022	0,016	0,006	0,008	0,013	0,018	0,009	0,021	0,017	0,019	0,013	0,008	0,000	0,019	0,016
16	0,021	0,002	0,003	0,005	0,013	0,013	0,010	0,003	0,021	0,005	0,004	0,006	0,007	0,015	0,019	0,000	0,003
17	0,018	0,002	0,006	0,005	0,010	0,013	0,006	0,003	0,020	0,008	0,004	0,009	0,007	0,012	0,016	0,003	0,000
18	0,021	0,004	0,005	0,008	0,012	0,015	0,009	0,005	0,023	0,007	0,006	0,009	0,005	0,014	0,018	0,002	0,006
19	0,023	0,007	0,008	0,010	0,015	0,018	0,011	0,008	0,026	0,010	0,009	0,011	0,007	0,017	0,021	0,005	0,008
20	0,012	0,007	0,010	0,003	0,009	0,005	0,004	0,006	0,014	0,008	0,005	0,007	0,008	0,007	0,012	0,009	0,008
21	0,009	0,015	0,018	0,011	0,017	0,012	0,018	0,013	0,004	0,016	0,012	0,015	0,022	0,015	0,012	0,016	0,016
22	0,022	0,008	0,003	0,006	0,019	0,014	0,013	0,008	0,021	0,001	0,007	0,003	0,013	0,017	0,022	0,006	0,009
23	0,006	0,014	0,018	0,015	0,002	0,007	0,010	0,015	0,013	0,020	0,016	0,018	0,010	0,004	0,004	0,016	0,012
24	0,020	0,007	0,006	0,004	0,016	0,012	0,011	0,006	0,019	0,004	0,005	0,002	0,015	0,015	0,020	0,009	0,009
25	0,014	0,006	0,011	0,009	0,006	0,010	0,002	0,007	0,020	0,013	0,008	0,012	0,002	0,007	0,011	0,008	0,004
26	0,023	0,010	0,005	0,007	0,020	0,015	0,014	0,009	0,022	0,003	0,009	0,005	0,015	0,018	0,023	0,008	0,011
27	0,017	0,006	0,010	0,003	0,015	0,011	0,010	0,005	0,012	0,008	0,004	0,006	0,014	0,014	0,019	0,008	0,008
28	0,022	0,008	0,004	0,007	0,019	0,015	0,014	0,009	0,022	0,002	0,008	0,003	0,013	0,017	0,022	0,006	0,010
29	0,005	0,014	0,018	0,011	0,008	0,003	0,011	0,013	0,007	0,016	0,012	0,014	0,015	0,006	0,005	0,016	0,016
30	0,018	0,006	0,004	0,003	0,015	0,010	0,010	0,005	0,018	0,002	0,004	0,001	0,014	0,013	0,018	0,007	0,008
31	0,017	0,003	0,008	0,007	0,008	0,013	0,005	0,004	0,022	0,010	0,005	0,010	0,005	0,010	0,014	0,005	0,002
32	0,005	0,019	0,022	0,015	0,013	0,010	0,018	0,018	0,002	0,020	0,017	0,019	0,020	0,013	0,007	0,021	0,021
33	0,006	0,018	0,021	0,014	0,013	0,009	0,017	0,017	0,001	0,019	0,016	0,018	0,021	0,011	0,008	0,020	0,019
34	0,021	0,007	0,002	0,005	0,017	0,013	0,012	0,007	0,020	0,000	0,006	0,001	0,012	0,015	0,020	0,005	0,008
Nó	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
1	0,021	0,023	0,012	0,009	0,022	0,006	0,020	0,014	0,023	0,017	0,022	0,005	0,018	0,017	0,005	0,006	0,021
2	0,004	0,007	0,007	0,015	0,008	0,014	0,007	0,006	0,010	0,006	0,008	0,014	0,006	0,003	0,019	0,018	0,007
3	0,005	0,008	0,010	0,018	0,003	0,018	0,006	0,011	0,005	0,010	0,004	0,018	0,004	0,008	0,022	0,021	0,002
4	0,008	0,010	0,003	0,011	0,006	0,015	0,004	0,009	0,007	0,003	0,007	0,011	0,003	0,007	0,015	0,014	0,005
5	0,012	0,015	0,009	0,017	0,019	0,002	0,016	0,006	0,020	0,015	0,019	0,008	0,015	0,008	0,013	0,013	0,017
6	0,015	0,018	0,005	0,012	0,014	0,007	0,012	0,010	0,015	0,011	0,015	0,003	0,010	0,013	0,010	0,009	0,013
7	0,009	0,011	0,004	0,018	0,013	0,010	0,011	0,002	0,014	0,010	0,014	0,011	0,010	0,005	0,018	0,017	0,012
8	0,005	0,008	0,006	0,013	0,008	0,015	0,006	0,007	0,009	0,005	0,009	0,013	0,005	0,004	0,018	0,017	0,007
9	0,023	0,026	0,014	0,004	0,021	0,013	0,019	0,020	0,022	0,012	0,022	0,007	0,018	0,022	0,002	0,001	0,020
10	0,007	0,010	0,008	0,016	0,001	0,020	0,004	0,013	0,003	0,008	0,002	0,016	0,002	0,010	0,020	0,019	0,000
11	0,006	0,009	0,005	0,012	0,007	0,016	0,005	0,008	0,009	0,004	0,008	0,012	0,004	0,005	0,017	0,016	0,006
12	0,009	0,011	0,007	0,015	0,003	0,018	0,002	0,012	0,005	0,006	0,003	0,014	0,001	0,010	0,019	0,018	0,001
13	0,005	0,007	0,008	0,022	0,013	0,010	0,015	0,002	0,015	0,014	0,013	0,015	0,014	0,005	0,020	0,021	0,012
14	0,014	0,017	0,007	0,015	0,017	0,004	0,015	0,007	0,018	0,014	0,017	0,006	0,013	0,010	0,013	0,011	0,015
15	0,018	0,021	0,012	0,012	0,022	0,004	0,020	0,011	0,023	0,019	0,022	0,005	0,018	0,014	0,007	0,008	0,020
16	0,002	0,005	0,009	0,016	0,006	0,016	0,009	0,008	0,008	0,008	0,006	0,016	0,007	0,005	0,021	0,020	0,005
17	0,006	0,008	0,008	0,016	0,009	0,012	0,009	0,004	0,011	0,008	0,010	0,016	0,008	0,002	0,021	0,019	0,008
18	0,000	0,003	0,011	0,019	0,008	0,014	0,011	0,007	0,010	0,010	0,009	0,019	0,010	0,007	0,023	0,022	0,007
19	0,003	0,000	0,014	0,021	0,011	0,017	0,014	0,009	0,013	0,013	0,011	0,021	0,012	0,010	0,026	0,025	0,010
20	0,011	0,014	0,000	0,014	0,009	0,011	0,007	0,006	0,011	0,006	0,010	0,008	0,006	0,008	0,014	0,013	0,008
21	0,019	0,021	0,014	0,000	0,017	0,015	0,015	0,020	0,018	0,008	0,018	0,009	0,014	0,018	0,004	0,003	0,016
22	0,008	0,011	0,009	0,017	0,000	0,021	0,005	0,014	0,002	0,009	0,001	0,017	0,004	0,011	0,022	0,020	0,001
23	0,014	0,017	0,011	0,015	0,021	0,000	0,019	0,008	0,022	0,018	0,021	0,008	0,017	0,011	0,011	0,012	0,019
24	0,011	0,014	0,007	0,015	0,005	0,019	0,000	0,013	0,003	0,007	0,005	0,015	0,001	0,011	0,019	0,018	0,004
25	0,007	0,009	0,006	0,020	0,014	0,008	0,013	0,000	0,016	0,012	0,014	0,013	0,011	0,003	0,018	0,019	0,013
26	0,010	0,013	0,011	0,018	0,002	0,022	0,003	0,016	0,000	0,010	0,002	0,018	0,005	0,013	0,023	0,021	0,003
27	0,010	0,013	0,006	0,008	0,009	0,018	0,007	0,012	0,010	0,000	0,010	0,014	0,005	0,010	0,013	0,011	0,008
28	0,009	0,011	0,010	0,018	0,001	0,021	0,005	0,014	0,002	0,010	0,000	0,018	0,004	0,011	0,022	0,021	0,002
29	0,019	0,021	0,008	0,009	0,017	0,008	0,015	0,013	0,018	0,014	0,018	0,000	0,013	0,016	0,007	0,006	0,016
30	0,010	0,012	0,006	0,014	0,004	0,017	0,001	0,011	0,005	0,005	0,004	0,013	0,000	0,009	0,018	0,017	0,002
31	0,007	0,010	0,008	0,018	0,011	0,011	0,011	0,003	0,013	0,010	0,011	0,016	0,009	0,000	0,021	0,021	0,010
32	0,023	0,026	0,014	0,004	0,022	0,011	0,019	0,018	0,023	0,013	0,022	0,007	0,018	0,021	0,000	0,001	0,020
33	0,022	0,025	0,013	0,003	0,020	0,012	0,018	0,019	0,021	0,011	0,021	0,006	0,017	0,021	0,001	0,000	0,019
34	0,007	0,010	0,008	0,016	0,001	0,019	0,004	0,013	0,003	0,008	0,002	0,016	0,002	0,010	0,020	0,019	0,000

IV. IMPLEMENTAÇÃO DO MÉTODO EXAUSTIVO

evaluateSingleInstance.m

```
% EVALUATESINGLEINSTANCE Evaluates different controller placements for a
% single instance of latency and bandwidth matrices.
%
% X = EVALUATESINGLEINSTANCE(A,B,C) evaluates different placements of C
% controllers for the single instance of distance matrices A and B.
%
% X = EVALUATESINGLEINSTANCE(A,B,C,D) evaluates different placements of C
% controllers for the single instance of distance matrices A and B with node
% weights D.
%
% X = EVALUATESINGLEINSTANCE(A,B,C,D,E) evaluates different placements of C
% controllers for the single instance of distance matrices A and B with node
% weights D and a subset of working controllers E.
%
% X = EVALUATESINGLEINSTANCE(A,B,[C],D,E,F) evaluates the single controller
% placement F for the single instance of distance matrices A and B with node
% weights D and a subset of working controllers E. In this case, the
% content of variable C is ignored.
%
% For example use cases, see also PLOTEXAMPLE.

% Copyright 2012–2013 David Hock, Stefan Geiler, Fabian Helmschrott,
% Steffen Gebert
% Chair of Communication Networks, Uni Wrzburg

% Updated in 2018 by Ana Carolina de Oliveira Christofaro
% Master degree of Electrical Engineering, University of Brasilia

function [evaluationResult]=evaluateSingleInstance(latencyMatrix,bandwidthMatrix,
    ↪ numberOfControllers,nodeWeights,workingsubset,singlePlacementInstance)

evaluationResult.numberOfControllers=numberOfControllers;
evaluationResult.failureType='SingleInstance';

latencydisttemp=latencyMatrix;
```

```
latencydisttemp(latencydisttemp==Inf)=nan;
latencydiameter=nanmax(nanmax(latencydisttemp));

bandwidthdisttemp=bandwidthMatrix;
bandwidthdisttemp(bandwidthdisttemp==Inf)=nan;
bandwidthdiameter=nanmax(nanmax(bandwidthdisttemp));

bandwidthMatrix(bandwidthMatrix==Inf)=nan;
bandwidthMatrix = - bandwidthMatrix;

n = size(latencyMatrix, 1);

if ~exist('nodeWeights','var')
   nodeWeights=ones(1,size(latencyMatrix,1));
end

if ~exist('workingsubset','var')
   if ~exist('singlePlacementInstance','var')
      workingsubset=1:numberOfControllers;
   else
      workingsubset=1:size(singlePlacementInstance,2);
   end
end

mylimit=2e7/4;
if ~exist('singlePlacementInstance','var')

   % Create help arrays for avoiding the repeated computation of combnk
   % (combnk creates all possible combinations of numberOfControllers elements out
       ↪ of n)
   if ~exist('variables','dir')
      mkdir('variables');
   end

   for i=1:numberOfControllers
      if nchoosek(size(latencyMatrix,1),i)*i<mylimit
         if ~exist(['variables/nk' num2str(size(latencyMatrix,1)) '_' num2str(i) '
             ↪ .mat'],'file') && i*nchoosek(size(latencyMatrix,1),i)<mylimit
            nk=combnk(1:size(latencyMatrix,1),i);
            save(['variables/nk' num2str(size(latencyMatrix,1)) '_' num2str(i) '.
                ↪ mat'],'nk');
         end
```

```
    else
        break;
    end
end

nksize=nchoosek(size(latencyMatrix,1),numberOfControllers);

if numberOfControllers*nksize<mylimit
    load(['variables/nk' num2str(size(latencyMatrix,1)) '_' num2str(
        ↪ numberOfControllers) '.mat']);

    [avgLatencyN2C,maxLatencyN2C,controllerlessNodes,controllerImbalance,
        ↪ avgLatencyC2C,maxLatencyC2C,avgBandwidthN2C,minBandwidthN2C,
        ↪ avgBandwidthC2C,minBandwidthC2C]=calculateMetrics(latencyMatrix,
        ↪ bandwidthMatrix,nk(:,workingsubset),nodeWeights);

else
    % find largest computable numberOfControllers (fitting in RAM)
        for i=numberOfControllers:-1:1
            if i*nchoosek(size(latencyMatrix,1),i)<mylimit
                rightNumberOfControllers=i;
                break;
            end
        end
    leftNumberOfControllers=numberOfControllers-rightNumberOfControllers;
        if leftNumberOfControllers>rightNumberOfControllers
            disp(' --> One side numberOfControllers to high, not solvable that
                ↪ way!');
            return;
        end
    lstruct=load(['variables/nk' num2str(size(latencyMatrix,1)) '_' num2str(
        ↪ leftNumberOfControllers) '.mat']);
    left=lstruct.(char(fieldnames(lstruct)));
    lstruct=load(['variables/nk' num2str(size(latencyMatrix,1)) '_' num2str(
        ↪ rightNumberOfControllers) '.mat']);
    right=lstruct.(char(fieldnames(lstruct)));
    clear lstruct;

    avgLatencyN2C=nan(1,nksize);
    maxLatencyN2C=nan(1,nksize);
    controllerlessNodes=nan(1,nksize);
    controllerImbalance=nan(1,nksize);
```

```
avgLatencyC2C=nan(1,nksize);
maxLatencyC2C=nan(1,nksize);
avgBandwidthN2C=nan(1,nksize);
minBandwidthN2C=nan(1,nksize);
avgBandwidthC2C=nan(1,nksize);
minBandwidthC2C=nan(1,nksize);
arrayoffset=0;

   for j=1:size(left,1)
      if mod(j,ceil(size(left,1)/10))==0
         fprintf('.');
      end
      rightsubset=right(right(:,1)>left(j,end),:);
      nk=[repmat(left(j,:),size(rightsubset,1),1) rightsubset];
      % Returns a subset of all placements for numberOfControllers

      [avgLatencyN2Ctemp,maxLatencyN2Ctemp,controllerlessNodesTemp,
          ↪ controllerImbalanceTemp,avgLatencyC2Ctemp,maxLatencyC2Ctemp,
          ↪ avgBandwidthN2Ctemp,minBandwidthN2Ctemp,avgBandwidthC2Ctemp,
          ↪ minBandwidthC2Ctemp]=calculateMetrics(latencyMatrix,
          ↪ bandwidthMatrix,nk(:,workingsubset),nodeWeights);

      % Merges the subset to one total set
      avgLatencyN2C(arrayoffset+(1:length(avgLatencyN2Ctemp)))=
          ↪ avgLatencyN2Ctemp;
      maxLatencyN2C(arrayoffset+(1:length(avgLatencyN2Ctemp)))=
          ↪ maxLatencyN2Ctemp;
      controllerlessNodes(arrayoffset+(1:length(avgLatencyN2Ctemp)))=
          ↪ controllerlessNodesTemp;
      controllerImbalance(arrayoffset+(1:length(avgLatencyN2Ctemp)))=
          ↪ controllerImbalanceTemp;
      avgLatencyC2C(arrayoffset+(1:length(avgLatencyN2Ctemp)))=
          ↪ avgLatencyC2Ctemp;
      maxLatencyC2C(arrayoffset+(1:length(avgLatencyN2Ctemp)))=
          ↪ maxLatencyC2Ctemp;
      avgBandwidthN2C(arrayoffset+(1:length(avgLatencyN2Ctemp)))=
          ↪ avgBandwidthN2Ctemp;
      minBandwidthN2C(arrayoffset+(1:length(avgLatencyN2Ctemp)))=
          ↪ minBandwidthN2Ctemp;
      avgBandwidthC2C(arrayoffset+(1:length(avgLatencyN2Ctemp)))=
          ↪ avgBandwidthC2Ctemp;
```

```
            minBandwidthC2C(arrayoffset+(1:length(avgLatencyN2Ctemp)))=
                ↪ minBandwidthC2Ctemp;
            arrayoffset=arrayoffset+length(avgLatencyN2Ctemp);
        end
    end
else

    [avgLatencyN2C,maxLatencyN2C,controllerlessNodes,controllerImbalance,
        ↪ avgLatencyC2C,maxLatencyC2C,avgBandwidthN2C,minBandwidthN2C,
        ↪ avgBandwidthC2C,minBandwidthC2C]=calculateMetrics(latencyMatrix,
        ↪ bandwidthMatrix,singlePlacementInstance(:,workingsubset),nodeWeights);

end

Cost = [avgLatencyN2C' / latencydiameter, maxLatencyN2C' / latencydiameter,
    ↪ controllerlessNodes', controllerImbalance' / n, avgLatencyC2C' /
    ↪ latencydiameter, maxLatencyC2C' / latencydiameter, - avgBandwidthN2C' /
    ↪ bandwidthdiameter, - minBandwidthN2C' /bandwidthdiameter, - avgBandwidthC2C
    ↪ ' /bandwidthdiameter, - minBandwidthC2C' /bandwidthdiameter];

[paretoOptima,evaluationResult] = exhaustive(nk,Cost);

end
```

exhaustive.m

```
% EXAUSTIVE Evaluates different controller placements for a single instance
% of latency and bandwidth matrices, and returns the logical Pareto Front.
%
% X,Y = EXAUSTIVE(A,B) evaluates different placements defined in the set A
% with cost B.

% Copyright 2018 Ana Carolina de Oliveira Christofaro
% Master degree of Electrical Engineering, University of Brasilia

function [M,statsM]=exhaustive(nk, Cost)

    nPop=size(nk,1);
    kController=size(nk,2);
    nCost = size(Cost,2);
```

```
empty_individual.Position=[];
empty_individual.Cost=[];
empty_individual.DominationSet=[];
empty_individual.DominatedCount=[];

pop=repmat(empty_individual,nPop,1);

for i=1:nPop
   pop(i).DominatedCount=0;
   pop(i).Cost=Cost(i,:);
   pop(i).Position=nk(i,:);
end

F{1}=[];

for i=1:nPop

   p=pop(i);

   for j=i+1:nPop

      q=pop(j);

      if Dominates(p,q)
         q.DominatedCount=q.DominatedCount+1;
      end

      if Dominates(q,p)
         p.DominatedCount=p.DominatedCount+1;
      end

      pop(i)=p;
      pop(j)=q;

   end

   if pop(i).DominatedCount==0
      F{1}=[F{1} i];
   end

end
```

```
   F1=pop(F{1});

   M = vec2mat([F1.Position],kController);
   CostM = vec2mat([F1.Cost],nCost);

   statsM.nk = M;
   statsM.avgLatencyN2C=CostM(:,1)';
   statsM.maxLatencyN2C=CostM(:,2)';
   statsM.controllerlessNodes=CostM(:,3)';
   statsM.controllerImbalance=CostM(:,4)';
   statsM.avgLatencyC2C=CostM(:,5)';
   statsM.maxLatencyC2C=CostM(:,6)';
   statsM.avgBandwidthN2C=CostM(:,7)';
   statsM.minBandwidthN2C=CostM(:,8)';
   statsM.avgBandwidthC2C=CostM(:,9)';
   statsM.minBandwidthC2C=CostM(:,10)';

end

function b=Dominates(x,y)

   if isstruct(x)
      x=x.Cost;
      x(isnan(x)) = 0 ;
   end

   if isstruct(y)
      y=y.Cost;
      y(isnan(y)) = 0 ;
   end

   b=all(x<=y) && any(x<y);

end
```

calculateMetrics.m

```
% CALCULATEMETRICS Calculates all placement metrics for given distance
% matrices, placements, and node weights.
```

```
%
% [U,V,W,X,Y,Z,T,H,M,N] = CALCULATEMETRICS(A,B,C) calculates and returns
% placement metrics for given distance matrices A and B and placements C,
% where C is a matrix containing in each column one placement (consisting
% of the node IDs given in the rows of this column).
% U is a vector containing the average latency between the nodes and the
% controllers, V is a vector containing the maximum latency between the
% nodes and the controllers, W is a vector containing the maximum number
% of controller-less nodes, X is a vector containing the imbalance of the
% controllers, y is a vector containing the average latency between the
% controllers, Z is a vector containing the maximum latency between
% the controllers, T is a vector containing the average bandwidth between
% the nodes and the controllers, H is a vector containing the minimum
% bandwidth between the nodes and the controllers, M is a vector containing
% the average bandwidth between the controllers and N is a vector
% containing the minimum bandwidth between the controllers.
%
% [U,V,W,X,Y,Z,T,H,M,N] = CALCULATEMETRICS(A,B,C,D) calculates and returns
% placement metrics for given distance matrices A and B and placements C
% as well as node weights D, where C is a row vector containing for each
% node a node weight value.
%
% For example use cases, see also PLOTEXAMPLE.

% Copyright 2012-2013 David Hock, Stefan Geiler, Fabian Helmschrott,
% Steffen Gebert
% Chair of Communication Networks, Uni Wrzburg

% Updated in 2018 by Ana Carolina de Oliveira Christofaro
% Master degree of Electrical Engineering, University of Brasilia

function [avgLatencyN2C,maxLatencyN2C,controllerlessNodes,controllerImbalance,
    ↪ avgLatencyC2C,maxLatencyC2C,avgBandwidthN2C,minBandwidthN2C,avgBandwidthC2C,
    ↪ minBandwidthC2C]=calculateMetrics(latencyMatrix,bandwidthMatrix,
    ↪ placementArray,nodeWeights)

   global fitcount;

   % Check Matlab-version
   matlabVersion=str2num(regexprep(version('-release'),'[^0-9]',''));

   if matlabVersion < 2013
```

```
   %C = unique(A,'rows') treats each row of A as a single entity and
   %returns the unique rows of A in sorted order.
   [uniquePlacementArray,tildevar,backIdx]=unique(placementArray,'rows');
else
   [uniquePlacementArray,tildevar,backIdx]=unique(placementArray,'rows','legacy
      ↪ ');
end

clear tildevar;

%% Create temporary matrix T to latency
[temp_latencyMatrix,tempidx_latencyMatrix]=min(reshape(latencyMatrix(:,
    ↪ uniquePlacementArray),size(latencyMatrix,1),size(uniquePlacementArray,1),
    ↪ size(uniquePlacementArray,2)),[],3);

%% Create temporary matrix T to bandwidth
[temp_bandwidthMatrix,tempidx_bandwidthMatrix]=min(reshape(bandwidthMatrix(:,
    ↪ uniquePlacementArray),size(bandwidthMatrix,1),size(uniquePlacementArray
    ↪ ,1),size(uniquePlacementArray,2)),[],3);

%% Calculate load balancing

% Indices of all controllers with at least one node assigned
if matlabVersion < 2013
   usedidx=unique(tempidx_latencyMatrix);
else
   usedidx=unique(tempidx_latencyMatrix, 'legacy');
end

% Simple case — only one controller — imbalance is 0
if length(usedidx)<=1
      controllerImbalance=zeros(1,size(tempidx_latencyMatrix,2));
else

   if ~exist('nodeWeights','var')
      tempidx4_latencyMatrix=hist(tempidx_latencyMatrix,usedidx);
   else
      % If node weights are defined, they are included in calculation of
      % imbalance
      tempidx4_latencyMatrix=accumarray([reshape(repmat(1:size(
          ↪ tempidx_latencyMatrix,2),size(tempidx_latencyMatrix,1),1),numel(
          ↪ tempidx_latencyMatrix),1) reshape(tempidx_latencyMatrix,numel(
```

```
            ↪ tempidx_latencyMatrix),1)],repmat(nodeWeights,1,size(
            ↪ tempidx_latencyMatrix,2)),[], @sum)';
    end

    % Transpose necessary to repair one-column vector
    if size(tempidx4_latencyMatrix,1)==1 && size(tempidx4_latencyMatrix,2)>1
        tempidx4_latencyMatrix=tempidx4_latencyMatrix';
    end

    % Calculate imbalance as max minus min
        controllerImbalance=max(tempidx4_latencyMatrix(end-length(usedidx)+1:end
            ↪ ,:))-min(tempidx4_latencyMatrix(end-length(usedidx)+1:end,:));
end

%% Number of controller-less nodes is calculated with or without node weights
if exist('nodeWeights','var')
    controllerlessNodes=sum((temp_latencyMatrix==Inf).*repmat(nodeWeights',1,
        ↪ size(temp_latencyMatrix,2)));
else
    controllerlessNodes=sum(temp_latencyMatrix==Inf);
end

%% Maximum and average node-to-controller latency are calculated
temp_latencyMatrix(temp_latencyMatrix==Inf)=nan;
avgLatencyN2C=nanmean(temp_latencyMatrix);
maxLatencyN2C=nanmax(temp_latencyMatrix);

%% Maximum (negative) and average node-to-controller bandwidth are calculated
temp_bandwidthMatrix(temp_bandwidthMatrix==Inf)=nan;
avgBandwidthN2C=nanmean(temp_bandwidthMatrix);
minBandwidthN2C=nanmax(temp_bandwidthMatrix);

%% Maximum and average controller-to-controller latency are calculated -
    ↪ matrix needs to be of square size
% Simple case - only one controller
if length(usedidx)<=1

    maxLatencyC2C=zeros(1,size(tempidx_latencyMatrix,2));
    avgLatencyC2C=zeros(1,size(tempidx_latencyMatrix,2));

elseif diff(size(latencyMatrix))==0
```

```
    temp_latencyMatrix=reshape(latencyMatrix(:,uniquePlacementArray),size(
        ↪ uniquePlacementArray,1)*size(latencyMatrix,1),size(
        ↪ uniquePlacementArray,2));

    %Maximum controller-to-controller latency for each position combination:
    maxtemp_latencyMatrix=reshape(nanmax(temp_latencyMatrix,[],2),size(
        ↪ latencyMatrix,1),size(uniquePlacementArray,1));
    maxtemp_latencyMatrix(maxtemp_latencyMatrix==Inf)=nan;
    maxLatencyC2C=nanmax(maxtemp_latencyMatrix(sub2ind(size(
        ↪ maxtemp_latencyMatrix),uniquePlacementArray',repmat(1:size(
        ↪ uniquePlacementArray,1),size(uniquePlacementArray,2),1))),[],1);

    %Average controller-to-controller latency for each position combination:
    meantemp_latencyMatrix=reshape(nanmean(temp_latencyMatrix,2),size(
        ↪ latencyMatrix,1),size(uniquePlacementArray,1));
    meantemp_latencyMatrix(meantemp_latencyMatrix==Inf)=nan;
    avgLatencyC2C=nanmean(meantemp_latencyMatrix(sub2ind(size(
        ↪ meantemp_latencyMatrix),uniquePlacementArray',repmat(1:size(
        ↪ uniquePlacementArray,1),size(uniquePlacementArray,2),1))),1);

else
    %%Maximum controller-to-controller latency for each position combination:
    maxLatencyC2C=nan(size(maxLatencyN2C));

    %Average controller-to-controller latency for each position combination:
    avgLatencyC2C=nan(size(maxLatencyN2C));
end

%% Maximum (negative) and average inter-controller bandwidth are calculated -
    ↪ matrix needs to be of square size
% Simple case - only one controller
if length(usedidx)<=1

    minBandwidthC2C=zeros(1,size(tempidx_bandwidthMatrix,2));
    avgBandwidthC2C=zeros(1,size(tempidx_bandwidthMatrix,2));

elseif diff(size(bandwidthMatrix))==0

    temp_bandwidthMatrix=reshape(bandwidthMatrix(:,uniquePlacementArray),size(
        ↪ uniquePlacementArray,1)*size(bandwidthMatrix,1),size(
        ↪ uniquePlacementArray,2));
```

```
        %Maximum (negative) controller-to-controller bandwidth for each position
            ↪ combination:
        mintemp_bandwidthMatrix=reshape(nanmin(temp_bandwidthMatrix,[],2),size(
            ↪ bandwidthMatrix,1),size(uniquePlacementArray,1));
        mintemp_bandwidthMatrix(mintemp_bandwidthMatrix==Inf)=nan;
        minBandwidthC2C=nanmax(mintemp_bandwidthMatrix(sub2ind(size(
            ↪ mintemp_bandwidthMatrix),uniquePlacementArray',repmat(1:size(
            ↪ uniquePlacementArray,1),size(uniquePlacementArray,2),1))),[],1);

        %Average controller-to-controller bandwidth for each position combination:
        meantemp_bandwidthMatrix=reshape(nanmean(temp_bandwidthMatrix,2),size(
            ↪ bandwidthMatrix,1),size(uniquePlacementArray,1));
        meantemp_bandwidthMatrix(meantemp_bandwidthMatrix==Inf)=nan;
        avgBandwidthC2C=nanmean(meantemp_bandwidthMatrix(sub2ind(size(
            ↪ meantemp_bandwidthMatrix),uniquePlacementArray',repmat(1:size(
            ↪ uniquePlacementArray,1),size(uniquePlacementArray,2),1))),1);

    else
        %Maximum (negative) controller-to-controller bandwidth for each position
            ↪ combination:
        minBandwidthC2C=nan(size(minBandwidthN2C));

        %Average controller-to-controller bandwidth for each position combination:
        avgBandwidthC2C=nan(size(minBandwidthN2C));
    end

    avgLatencyN2C=avgLatencyN2C(backIdx);
    maxLatencyN2C=maxLatencyN2C(backIdx);
    controllerlessNodes=controllerlessNodes(backIdx);
    controllerImbalance=controllerImbalance(backIdx);
    avgLatencyC2C=avgLatencyC2C(backIdx);
    maxLatencyC2C=maxLatencyC2C(backIdx);
    avgBandwidthN2C=avgBandwidthN2C(backIdx);
    minBandwidthN2C=minBandwidthN2C(backIdx);
    avgBandwidthC2C=avgBandwidthC2C(backIdx);
    minBandwidthC2C=minBandwidthC2C(backIdx);

return
```

V. IMPLEMENTAÇÃO DO PSA

psa.m

```
% Copyright 2012  2013 David Hock, Stefan Geiler, Fabian Helmschrott,
% Steffen Gebert
% Chair of Communication Networks, Uni Wrzburg
%
% Updated in 2018 by Ana Carolina de Oliveira Christofaro
% Master degree of Electrical Engineering, University of Brasilia

function [M, statsM]=psa(T0, rho, m, s, distanceMatrix, bandwidthMatrix, k, seed)

   %PSA Pareto Simulated Annealing
   % T0: starting temperature (~50)
   % rho: cooling rate (~.9)
   % m: number of iterations per temperature level
   % s: number of elements in the set of generating solutions S
   % k: number of controllers per placement
   % seed: seed for the RNG

   randn('state', seed);
   nStats = 10;
   alpha = 1.05;
   n = size(distanceMatrix, 1); %number of nodes
   T = T0;

   latencydisttemp=distanceMatrix;
   latencydisttemp(latencydisttemp==Inf)=nan;
   latencydiameter=nanmax(nanmax(latencydisttemp));

   bandwidthdisttemp=bandwidthMatrix;
   bandwidthdisttemp(bandwidthdisttemp==Inf)=nan;
   bandwidthdiameter=nanmax(nanmax(bandwidthdisttemp));

   bandwidthMatrix(bandwidthMatrix==Inf)=nan;
   bandwidthMatrix = - bandwidthMatrix;

   % Random placements
```

```
S = generateS(n, s, k);
D = zeros(size(S, 1));

[avgLatencyN2C, maxLatencyN2C, controllerlessNodes, controllerImbalance,
    ↪ avgLatencyC2C, maxLatencyC2C, avgBandwidthN2C,minBandwidthN2C,
    ↪ avgBandwidthC2C,minBandwidthC2C] = calculateMetrics(distanceMatrix,
    ↪ bandwidthMatrix, S);

statsS = [avgLatencyN2C' / latencydiameter, maxLatencyN2C' / latencydiameter,
    ↪ controllerlessNodes', controllerImbalance' / n, avgLatencyC2C' /
    ↪ latencydiameter, maxLatencyC2C' / latencydiameter, avgBandwidthN2C' /
    ↪ bandwidthdiameter, minBandwidthN2C' /bandwidthdiameter, avgBandwidthC2C'
    ↪ /bandwidthdiameter, minBandwidthC2C' /bandwidthdiameter];

M = S(paretoGroup(statsS), :); % M is the set of Pareto optima from S

Lambda = zeros(s, nStats);

itercount = 0;

while T > 1

   % Permute rowwise (allows more efficient implementation of the next step).
   S = shuffleRows(S);

   % For each placement x in S, draw a random neighbor y by replacing the
   % location of one controller location with a random location.
   % draw up to k/2 new neighbors, this increases diversity while in higher
       ↪ temperature regions
   Y = drawNeighbors5(S, n, ceil(T*k / (2* T0)));

   % Calculate the stats for the new placements.
   [avgLatencyN2C, maxLatencyN2C, controllerlessNodes, controllerImbalance,
       ↪ avgLatencyC2C, maxLatencyC2C, avgBandwidthN2C, minBandwidthN2C,
       ↪ avgBandwidthC2C, minBandwidthC2C] = calculateMetrics(distanceMatrix,
       ↪ bandwidthMatrix, [Y; M]);

   statsYM = [avgLatencyN2C' / latencydiameter, maxLatencyN2C' /
       ↪ latencydiameter, controllerlessNodes',controllerImbalance' / n,
       ↪ avgLatencyC2C' / latencydiameter, maxLatencyC2C' / latencydiameter,
       ↪ avgBandwidthN2C' / bandwidthdiameter, minBandwidthN2C' /
       ↪ bandwidthdiameter,avgBandwidthC2C' /bandwidthdiameter, minBandwidthC2C
```

```
    ↪ ' /bandwidthdiameter];

statsY = statsYM(1:size(Y, 1), :);

SYM = [S; Y; M]; % Update M.
optima = paretoGroup([statsS; statsYM]);

M = SYM(optima, :);

if (itercount == 0)

   % (Initially random) objective weights for each solution (normalized
       ↪ rowwise).
   Lambda = rand(s, nStats);
   Lambda = bsxfun(@times, Lambda, 1./(sum(Lambda, 2)));

else

   % For each solution s in S, find a solution closest to it.

   % D contains the Lambda weighted, source dependent distance between
       ↪ elements in S.
   [p,q] = meshgrid(1:size(statsS,1), 1:size(statsS,1));
   pairs = [p(:) q(:)];

   dists = abs(statsS(pairs(:,1),:)-statsS(pairs(:,2),:));
   D(sub2ind(size(D), pairs(:, 1), pairs(:, 2))) = sum(Lambda(pairs(:, 1),
       ↪ :) .* dists((pairs(:, 1) - 1) * size(S, 1) + pairs(:, 2), :), 2);

   D = D + diag(Inf(size(diag(D))));

   % sidx == indices of x' in S where (x' are the elements in S closest to x
       ↪ )
   [~,sidx] = min(D,[],2);

   % combinations of x and x' as indices of S
   combs = [(1:size(S,1))', sidx];

   % update Lambda
   % \lambda^{x_i}_j * \alpha, if (f_j(x_i) - f_j(x'_i)) \leq 0
   % \lambda^{x_i}_j / \alpha, else
   A = ones(size(Lambda)) * alpha;
```

```
        A(logical(dists((combs(:, 1) - 1) * size(S, 1) + combs(:, 2), :) > 0)) =
            ↪ 1 / alpha;
        Lambda = A .* Lambda;

    end

    % update S
    % if x dominates y, reject y (keep x) (I) FIXME actually, there's still a
        ↪ probability to accept y
    % if y dominates x, accept y (replace x with y) (II)
    % else, accept y with probability P (exponentiated weighted sum,
    % temperature dependent) (III)
    YdomX = find(sum(statsY <= statsS) == nStats);
    nondom = setdiff(1:s, [YdomX]);
    % (II)
    S(YdomX, :) = Y(YdomX, :);

    % (III)
    P = min(1, exp(sum(Lambda(nondom, :) .* (statsS(nondom, :) - statsY(nondom,
        ↪ :)), 2)));

    acc = logical(rand(length(nondom), 1) < P);
    S(acc, :) = Y(acc, :);

    [avgLatencyN2C, maxLatencyN2C, controllerlessNodes, controllerImbalance,
        ↪ avgLatencyC2C, maxLatencyC2C,avgBandwidthN2C, minBandwidthN2C,
        ↪ avgBandwidthC2C, minBandwidthC2C] = calculateMetrics(distanceMatrix,
        ↪ bandwidthMatrix, S);

    statsS = [avgLatencyN2C' / latencydiameter, maxLatencyN2C' / latencydiameter
        ↪ , controllerlessNodes',controllerImbalance' / n, avgLatencyC2C' /
        ↪ latencydiameter, maxLatencyC2C' / latencydiameter,avgBandwidthN2C' /
        ↪ bandwidthdiameter, minBandwidthN2C' /bandwidthdiameter,avgBandwidthC2C
        ↪ ' /bandwidthdiameter, minBandwidthC2C' /bandwidthdiameter];

    itercount = itercount + 1;

    % update T
    if mod(itercount, m) == 0
         T = T * rho;
    end
end
```

```
   [avgLatencyN2C, maxLatencyN2C, controllerlessNodes, controllerImbalance,
       ↪ avgLatencyC2C, maxLatencyC2C,avgBandwidthN2C, minBandwidthN2C,
       ↪ avgBandwidthC2C, minBandwidthC2C] = calculateMetrics(distanceMatrix,
       ↪ bandwidthMatrix, M);

   statsM.maxNumberOfControllerlessNodes=controllerlessNodes;
   statsM.avgLatencyN2C=avgLatencyN2C./latencydiameter;
   statsM.maxLatencyN2C=maxLatencyN2C./latencydiameter;
   statsM.avgLatencyC2C=avgLatencyC2C./latencydiameter;
   statsM.maxLatencyC2C=maxLatencyC2C./latencydiameter;
   statsM.controllerImbalance=controllerImbalance./n;
   statsM.avgBandwidthN2C=-avgBandwidthN2C./bandwidthdiameter;
   statsM.minBandwidthN2C=-minBandwidthN2C./bandwidthdiameter;
   statsM.avgBandwidthC2C=-avgBandwidthC2C./bandwidthdiameter;
   statsM.minBandwidthC2C=-minBandwidthC2C./bandwidthdiameter;

end

function S = generateS(n, s, k)
   %GENERATES Generate the set of generating solutions S (i.e., controller
       ↪ placements).
   % n: network size
   % s: output size, |S|
   % k: number of controllers

   S = zeros(s + 1, k);
   ii = 1;
   nuniq = size(unique(S, 'rows'), 1);
   while nuniq ~= s + 1
      S(ii, :) = randsample(n, k);
      nuniq2 = size(unique(S, 'rows'), 1);
      if nuniq2 > nuniq
         ii = ii + 1;
         nuniq = nuniq + 1;
      end
   end

   S = S(1:s, :);

end
```

```
function B = shuffleRows(A)

   [nRows, nCols] = size(A);
   [vartilde, idx] = sort(rand(nRows, nCols), 2);
   % convert column indices into linear indices
   idx = (idx - 1) * nRows + ndgrid(1:nRows, 1:nCols);
   % rearrange
   B = A;
   B(:) = B(idx);

end

function X = drawNeighbors(S, n)

   X = S;
   k = size(S, 2);

   for i = 1:(size(S, 1))

      r = S(i, :);
      id = randi(k);
      sub = randi(n);
      while ismember(sub, r)
         sub = randi(n);
      end
      r(id) = sub;
      X(i, :) = r;

   end

end

function X = drawNeighbors5(S, n, nNeighbors)

   k = size(S, 2);
   X = S;

   for ii = 1:size(X,1)
      % draw k+1 candidates (this guarantees that at least one of them is new as
          ↪ randsample's default is w/o replacement)
      c = randsample(n, k + nNeighbors);
      % choose nNeighbors that are not part of the current placement
```

```
      % => the new placement has still k distinct controllers and does not equal
          ↪ the original one

      X(ii, 1:nNeighbors) = c(find(~ismember(c, X(ii, :)), nNeighbors, 'first'));
   end

end
```

paretoGroup.m

```
function front=paretoGroup(X)

% PARETOGROUP To get the Pareto Front from a given set of points.
% synopsis: front =paretoGroup (objectiveMatrix)
% where:
% objectiveMatrix: [number of points X number of objectives] array
% front: [number of points X 1] logical vector to indicate if ith
% point belongs to the Pareto Front (true) or not (false).
%
% by Yi Cao, Cranfield University, 31 June 2007
%
% Identify the Pareto Front from a set of points in objective space is the
% most important and also the most time-consuming task in multi-objective
% optimization. This code splits the given objective set into several
% smaller groups to be examined by the efficient paretofront algorithm.
% Then, the Pareto Fronts of each group are combined as one set to be
% checked by the paretofront algorithm to determine the overall Pareto
% Front. In this way, the overal computation time can be reduced about
% half.
%
% Example:
% X = rand(1000000,4);
% t0 = cputime;
% Y1=paretoGroup(X); %mex implementation without sorting.
% t1=cputime - t0;
% t0 = cputime;
% Y2=paretofront(X);
% t2=cputime - t0;
% isequal(Y1,Y2) %shoudl be 1
% disp([t1 t2])
```

```
% Computation time based on Intel(R) Core(TM)2 CPU T2500 @ 2.0GHz, 2.0 GB of RAM
% 0.6844 1.4404
%

[m,n]=size(X);
groupcut=floor(2^13/n);
gRoup=max(1,ceil(m/groupcut));
front=false(m,1);
for k=1:gRoup
   z0=(k-1)*groupcut;
   z=(z0+1):min(z0+groupcut,m);
   front(z)=paretofront(X(z,:));
end
if gRoup>1
   front(front)=paretofront(X(front,:));
end
```

paretofront.m

```
function [] = paretofront(varargin)
% PARETOFRONT returns the logical Pareto Front of a set of points.
%
% synopsis: front = paretofront(M)
%
%
% INPUT ARGUMENT
%
% - M n x m array, of which (i,j) element is the j-th objective
% value of the i-th point;
%
% OUTPUT ARGUMENT
%
% - front n x 1 logical vector to indicate if the corresponding
% points are belong to the front (true) or not (false).
%
% By Yi Cao at Cranfield University, 31 October 2007
%
% Example 1: Find the Pareto Front of a circumference
%
% alpha = [0:.1:2*pi]';
```

```
% x = cos(alpha);
% y = sin(alpha);
% front = paretofront([x y]);
%
% hold on;
% plot(x,y);
% plot(x(front), y(front) , 'r');
% hold off
% grid on
% xlabel('x');
% ylabel('y');
% title('Pareto Front of a circumference');
%
% Example 2: Find the Pareto Front of a set of 3D random points
% X = rand(100,3);
% front = paretofront(X);
%
% hold on;
% plot3(X(:,1),X(:,2),X(:,3),'.');
% plot3(X(front, 1) , X(front, 2) , X(front, 3) , 'r.');
% hold off
% grid on
% view(-37.5, 30)
% xlabel('X_1');
% ylabel('X_2');
% zlabel('X_3');
% title('Pareto Front of a set of random points in 3D');
%
%
% Example 3: Find the Pareto set of a set of 1000000 random points in 4D
% The machine performing the calculations was a
% Intel(R) Core(TM)2 CPU T2500 @ 2.0GHz, 2.0 GB of RAM
%
% X = rand(1000000,4);
% t = cputime;
% paretofront(X);
% cputime - t
%
% ans =
%
% 1.473529
```

```
error('mex file absent, type ''mex paretofront.c'' to compile');
```

paretofront.c

```
#include <math.h>
#include "mex.h"


/*
    paretomember returns the logical Pareto membership of a set of points.

    synopsis: front = paretofront(objMat)

    created by Yi Cao

    y.cao@cranfield.ac.uk

    for compiling type

    mex paretofront.c

*/


void paretofront(bool * front, double * M, unsigned int row, unsigned int col);

void mexFunction( int nlhs, mxArray *plhs[] , int nrhs, const mxArray *prhs[] )
{
        bool * front;
        double * M;
        unsigned int row, col;
        const int *dims;

        if(nrhs == 0 || nlhs > 1)
        {
            printf("\nsynopsis: front = paretofront(X)");
            plhs[0] = mxCreateDoubleMatrix(0 , 0 , mxREAL);
            return;
        }
```

```
        M = mxGetPr(prhs[0]);
        dims = mxGetDimensions(prhs[0]);
        row = dims[0];
        col = dims[1];



        /* ----- output ----- */

        plhs[0] = mxCreateLogicalMatrix (row , 1);
        front = (bool *) mxGetPr(plhs[0]);


        /* main call */
        paretofront(front, M, row, col);
}

void paretofront(bool * front, double * M, unsigned int row, unsigned int col)
{
    unsigned int t,s,i,j,j1,j2;
    bool *checklist, coldominatedflag;

    checklist = (bool *)mxMalloc(row*sizeof(bool));
    for(t = 0; t<row; t++) checklist[t] = true;
    for(s = 0; s<row; s++) {
        t=s;
        if (!checklist[t]) continue;
        checklist[t]=false;
        coldominatedflag=true;
        for(i=t+1;i<row;i++) {
            if (!checklist[i]) continue;
            checklist[i]=false;
            for (j=0,j1=i,j2=t;j<col;j++,j1+=row,j2+=row) {
                if (M[j1] < M[j2]) {
                    checklist[i]=true;
                    break;
                }
            }
            if (!checklist[i]) continue;
            coldominatedflag=false;
            for (j=0,j1=i,j2=t;j<col;j++,j1+=row,j2+=row) {
                if (M[j1] > M[j2]) {
```

```
                coldominatedflag=true;
                break;
            }
        }
        if (!coldominatedflag) { //swap active index continue checking
            front[t]=false;
            checklist[i]=false;
            coldominatedflag=true;
            t=i;
        }
    }
    front[t]=coldominatedflag;
    if (t>s) {
        for (i=s+1; i<t; i++) {
            if (!checklist[i]) continue;
            checklist[i]=false;
            for (j=0,j1=i,j2=t;j<col;j++,j1+=row,j2+=row) {
                if (M[j1] < M[j2]) {
                    checklist[i]=true;
                    break;
                }
            }
        }
    }
}
mxFree(checklist);
}
```

VI. IMPLEMENTAÇÃO DO NSGA-II

nsga2.m

```
% Copyright (c) 2015, Yarpiz (www.yarpiz.com)
% All rights reserved. Please read the "license.txt" for license terms.
%
% Project Code: YPEA120
% Project Title: Non-dominated Sorting Genetic Algorithm II (NSGA-II)
% Publisher: Yarpiz (www.yarpiz.com)
%
% Developer: S. Mostapha Kalami Heris (Member of Yarpiz Team)
%
% Contact Info: sm.kalami@gmail.com, info@yarpiz.com
%
% Adapted for the controller placements problem in SDN networks
% in 2018 by Ana Carolina de Oliveira Christofaro
% Master degree of Electrical Engineering, University of Brasilia

function [M, statsM] = nsga2(pCrossover,pMutation,kController,MaxIt,nPop,
    ↪ tournament,latencyMatrix,bandwidthMatrix)

%% Initialization
bandwidthMatrix(bandwidthMatrix==Inf)=nan;
bandwidthMatrix = - bandwidthMatrix;

empty_individual.Position=[];
empty_individual.Cost=[];
empty_individual.Rank=[];
empty_individual.DominationSet=[];
empty_individual.DominatedCount=[];
empty_individual.CrowdingDistance=[];

pop=repmat(empty_individual,nPop,1);

%% NSGA-II Parameters
randn('state', 42);
nCrossover=round(pCrossover*nPop); % Number of Parnets (Offsprings)
nMutation=round(pMutation*nPop); % Number of Mutants
```

```
n = size(latencyMatrix, 1);

randPop = GenRandPlacements (n,nPop,kController);
randPopCost = CostFunction(randPop,latencyMatrix,bandwidthMatrix);

for i=1:nPop
   pop(i).Position = randPop(i,:);
   pop(i).Cost = randPopCost(i,:);
end

% Non-Dominated Sorting
[pop, F]=NonDominatedSorting(pop);

% Calculate Crowding Distance
pop=CalcCrowdingDistance(pop,F);

% Sort Population
[pop, F]=SortPopulation(pop);

%% NSGA-II Main Loop

for it=1:MaxIt
   % Crossover
   popc=repmat(empty_individual,nCrossover,1);
   for k=1:2:nCrossover

      p1=TournamentSelection(pop,nPop,tournament);
      p2=TournamentSelection(pop,nPop,tournament);

      [popc(k).Position, popc(k+1).Position]=Crossover(p1,p2,kController);

      popc(k).Cost=CostFunction(popc(k).Position,latencyMatrix,bandwidthMatrix);
      popc(k+1).Cost=CostFunction(popc(k+1).Position,latencyMatrix,bandwidthMatrix
         ↪ );
   end

   % Mutation
   popm=repmat(empty_individual,nMutation,1);
   for k=1:nMutation

      p=TournamentSelection(pop,nPop,tournament);
```

```
        popm(k).Position=Mutate(p,n,kController);

        popm(k).Cost=CostFunction(popm(k).Position,latencyMatrix,bandwidthMatrix);
    end

    % Merge
     pop=[pop
        popc
        popm]; %#ok

    % Non-Dominated Sorting
    [pop, F]=NonDominatedSorting(pop);

    % Calculate Crowding Distance
    pop=CalcCrowdingDistance(pop,F);

    % Sort Population
    pop=SortPopulation(pop);

    % Truncate
    pop=pop(1:nPop);

    % Non-Dominated Sorting
    [pop, F]=NonDominatedSorting(pop);

    % Calculate Crowding Distance
    pop=CalcCrowdingDistance(pop,F);

    % Sort Population
    [pop, F]=SortPopulation(pop);

    % Store F1
    F1=pop(F{1});

end

%% Results

M = vec2mat([F1(:).Position],kController);
M = unique(M, 'rows');
CostM = CostFunction(M,latencyMatrix,bandwidthMatrix)';
```

```
statsM.nk=M;
statsM.controllerlessNodes=CostM(3,:);
statsM.avgLatencyN2C=CostM(1,:);
statsM.maxLatencyN2C=CostM(2,:);
statsM.avgLatencyC2C=CostM(5,:);
statsM.maxLatencyC2C=CostM(6,:);
statsM.controllerImbalance=CostM(4,:);
statsM.avgBandwidthN2C=-CostM(7,:);
statsM.minBandwidthN2C=-CostM(8,:);
statsM.avgBandwidthC2C=-CostM(9,:);
statsM.minBandwidthC2C=-CostM(10,:);

end
```

CalcCrowdingDistance.m

```
% Copyright (c) 2015, Yarpiz (www.yarpiz.com)
% All rights reserved. Please read the "license.txt" for license terms.
%
% Project Code: YPEA120
% Project Title: Non-dominated Sorting Genetic Algorithm II (NSGA-II)
% Publisher: Yarpiz (www.yarpiz.com)
%
% Developer: S. Mostapha Kalami Heris (Member of Yarpiz Team)
%
% Contact Info: sm.kalami@gmail.com, info@yarpiz.com

function pop=CalcCrowdingDistance(pop,F)

   nF=numel(F);
   nObj=size(pop(1).Cost,2);

   for k=1:nF

      Costs=vec2mat([pop(F{k}).Cost],nObj);
      Costs=Costs';
      n=numel(F{k});
      d=zeros(n,nObj);

      for j=1:nObj
```

```
            [cj, so]=sort(Costs(j,:));
            d(so(1),j)=inf;
            for i=2:n-1
               d(so(i),j)=abs(cj(i+1)-cj(i-1))/abs(cj(1)-cj(end));
            end
            d(so(end),j)=inf;
            d(isnan(d)) = 0 ;
        end
        for i=1:n
            pop(F{k}(i)).CrowdingDistance=sum(d(i,:));
        end
    end
end
```

CostFunction.m

```
% COSTFUNCION Calculates all placement metrics for given distance
% matrices and placements.

% Copyright 2018 Ana Carolina de Oliveira Christofaro
% Master degree of Electrical Engineering, University of Brasilia

function statsPop=CostFunction(pop,latencyMatrix,bandwidthMatrix)

    latencydisttemp=latencyMatrix;
    latencydisttemp(latencydisttemp==Inf)=nan;
    latencydiameter=nanmax(nanmax(latencydisttemp));

    bandwidthdisttemp=-bandwidthMatrix;
    bandwidthdisttemp(bandwidthdisttemp==Inf)=nan;
    bandwidthdiameter=nanmax(nanmax(bandwidthdisttemp));

    n = size(latencyMatrix, 1);

    [avgLatencyN2C, maxLatencyN2C, controllerlessNodes, controllerImbalance,
        ↪ avgLatencyC2C, maxLatencyC2C,avgBandwidthN2C,minBandwidthN2C,
        ↪ avgBandwidthC2C,minBandwidthC2C] = calculateMetrics(latencyMatrix,
        ↪ bandwidthMatrix, pop);
```

```
   statsPop = [avgLatencyN2C'./latencydiameter, maxLatencyN2C'./latencydiameter,
       ↪ controllerlessNodes', controllerImbalance'./n, avgLatencyC2C'./
       ↪ latencydiameter, maxLatencyC2C'./latencydiameter,(avgBandwidthN2C'./
       ↪ bandwidthdiameter),(minBandwidthN2C'./bandwidthdiameter),(avgBandwidthC2C
       ↪ './bandwidthdiameter),(minBandwidthC2C'./bandwidthdiameter)];

end
```

Crossover.m

```
% CROSSOVER Evaluates the crossover operator between two controller
% placements.
%
% [X,Y] = CROSSOVER[A,B,C] evaluates the crossover operator between the
% controller placements A and B with C controllers.

% Copyright 2018 Ana Carolina de Oliveira Christofaro
% Master degree of Electrical Engineering, University of Brasilia

function [x1, x2] = Crossover(y1,y2,k)

   C = intersect(y1,y2);
   D = setxor(y1,y2);

   x1 = C;
   t1 = size(x1, 2);
   p1 = D;

   x2 = C;
   t2 = size(x2, 2);
   p2 = D;

   while t1 < k
      t1 = t1 + 1;
      idx = randi(size(p1, 2));
      x1(t1) = p1(idx);
      p1(idx)=[];
   end

   while t2 < k
```

```
      t2 = t2 + 1;
      idx = randi(size(p2, 2));
      x2(t2) = p2(idx);
      p2(idx)=[];
   end

   x1 = sort(x1);
   x2 = sort(x2);

end
```

Dominates.m

```
% Copyright (c) 2015, Yarpiz (www.yarpiz.com)
% All rights reserved. Please read the "license.txt" for license terms.
%
% Project Code: YPEA120
% Project Title: Non-dominated Sorting Genetic Algorithm II (NSGA-II)
% Publisher: Yarpiz (www.yarpiz.com)
%
% Developer: S. Mostapha Kalami Heris (Member of Yarpiz Team)
%
% Contact Info: sm.kalami@gmail.com, info@yarpiz.com

% Adapted for the controller placements problem in SDN networks
% in 2018 by Ana Carolina de Oliveira Christofaro
% Master degree of Electrical Engineering, University of Brasilia

function b=Dominates(x,y)

   if isstruct(x)
      x=x.Cost;
      x(isnan(x)) = 0 ;
   end
   if isstruct(y)
      y=y.Cost;
      y(isnan(y)) = 0 ;
   end
   b=all(x<=y) && any(x<y);
```

```
end
```

GenRandPlacements.m

```
%GENRANDPLACEMENTS Generate the set of generating solutions S (i.e., controller
    ↪ placements).
% n: network size
% s: output size, |S|
% k: number of controllers
%
% Copyright 2012−2013 David Hock, Stefan Geiler, Fabian Helmschrott,
% Steffen Gebert
% Chair of Communication Networks, Uni Wrzburg

function S = GenRandPlacements (n, s, k)

   randPlacements = zeros(s + 1, k);
   ii = 1;
   nuniq = size(unique(randPlacements, 'rows'), 1);
   while nuniq ~= s + 1
      randPlacements(ii, :) = sort(randsample(n, k));
      nuniq2 = size(unique(randPlacements, 'rows'), 1);
      if nuniq2 > nuniq
         ii = ii + 1;
         nuniq = nuniq + 1;
      end
   end

   S = randPlacements(1:s, :);

end
```

Mutate.m

```
% MUTATE Evaluates the crossover operator between two controller
% placements.
%
```

```
% X = MUTATE[A,B,C] evaluates the mutation operator with the controller
% placements A for B placements and C controllers.

% Copyright 2018 Ana Carolina de Oliveira Christofaro
% Master degree of Electrical Engineering, University of Brasilia

function p = Mutate(p,n,k)

idx = randi(k);

   sub = randi(n);

   if ~ismember(sub, p)
      p(idx) = sub;
   end

   p = sort(p);

end
```

NonDominatedSorting.m

```
% Copyright (c) 2015, Yarpiz (www.yarpiz.com)
% All rights reserved. Please read the "license.txt" for license terms.
%
% Project Code: YPEA120
% Project Title: Non-dominated Sorting Genetic Algorithm II (NSGA-II)
% Publisher: Yarpiz (www.yarpiz.com)
%
% Developer: S. Mostapha Kalami Heris (Member of Yarpiz Team)
%
% Contact Info: sm.kalami@gmail.com, info@yarpiz.com

function [pop, F]=NonDominatedSorting(pop)

   nPop=numel(pop);

   for i=1:nPop
      pop(i).DominationSet=[];
      pop(i).DominatedCount=0;
```

```
end

F{1}=[];

for i=1:nPop
   p=pop(i);
   for j=i+1:nPop
      q=pop(j);

      if Dominates(p,q)
         p.DominationSet=[p.DominationSet j];
         q.DominatedCount=q.DominatedCount+1;
      end

      if Dominates(q,p)
         q.DominationSet=[q.DominationSet i];
         p.DominatedCount=p.DominatedCount+1;
      end
      pop(i)=p;
      pop(j)=q;
   end

   if pop(i).DominatedCount==0
      F{1}=[F{1} i];
      pop(i).Rank=1;
   end
end

k=1;

while true
   Q=[];
   for i=F{k}
      p=pop(i);

      for j=p.DominationSet
         q=pop(j);
         q.DominatedCount=q.DominatedCount-1;

         if q.DominatedCount==0
            Q=[Q j]; %#ok
            q.Rank=k+1;
```

```
            end

            pop(j)=q;
         end
      end

      if isempty(Q)
         break;
      end

      F{k+1}=Q; %#ok

      k=k+1;

   end

end
```

SortPopulation.m

```
% Copyright (c) 2015, Yarpiz (www.yarpiz.com)
% All rights reserved. Please read the "license.txt" for license terms.
%
% Project Code: YPEA120
% Project Title: Non-dominated Sorting Genetic Algorithm II (NSGA-II)
% Publisher: Yarpiz (www.yarpiz.com)
%
% Developer: S. Mostapha Kalami Heris (Member of Yarpiz Team)
%
% Contact Info: sm.kalami@gmail.com, info@yarpiz.com

function [pop, F]=SortPopulation(pop)

   % Sort Based on Crowding Distance
   [~, CDSO]=sort([pop.CrowdingDistance],'descend');
   pop=pop(CDSO);

   % Sort Based on Rank
   [~, RSO]=sort([pop.Rank]);
   pop=pop(RSO);
```

```
   % Update Fronts
   Ranks=[pop.Rank];
   MaxRank=max(Ranks);
   F=cell(MaxRank,1);
   for r=1:MaxRank
      F{r}=find(Ranks==r);
   end

end
```

TournamentSelection.m

```
% TOURNAMENTSELECTION Evaluates the tournament selection.

% Copyright 2018 Ana Carolina de Oliveira Christofaro
% Master degree of Electrical Engineering, University of Brasilia

function y = TournamentSelection(pop,nPop,type)

   id = [];
   id = randperm(nPop,type);

   for i = 1:type
      p(i) = pop(id(i));
   end

   val1=min([p(:).Rank]);
   val2=max([p([p(:).Rank]==val1).CrowdingDistance]);
   idx=find([p(:).Rank]==val1 & [p(:).CrowdingDistance]==val2,1,'first');

   y = p(idx).Position;

end
```