Facing the challenges of network services on the edge, such as routing considering the quality of service, is a crucial issue for the current research efforts in the area of the network. Multicast routing is an essential technique for delivering routing services with a high level of optimization from the perspective of operators and application providers when there are user groups. Furthermore, the routing that considers latency constraints has obstacles to merging different conditions in the solution, mainly when there is a fairness perspective to be accomplished in the users’ communication. This work aims to deal with this fairness requirement in multicasting, showing efficient solutions to increase the equilibrium in the routing by choosing better path options for fair group interaction.

Keywords: Distributed Interactive Application, Edge Computing, Software Defined Network, Fairness, Playability, Multicast Routing, Shortest Path, Delay and Delay-variation.
Universidade Federal da Bahia
Instituto de Computação

Programa de Pós-Graduação em Ciência da Computação

FAIRNESS-ORIENTED MULTICAST ROUTING FOR DISTRIBUTED INTERACTIVE APPLICATIONS WITH COMPUTING ON THE NETWORK EDGE

Ibirisol Fontes Ferreira

DISSERTAÇÃO DE MESTRADO

Salvador
27 de abril de 2023
FAIRNESS-ORIENTED MULTICAST ROUTING FOR DISTRIBUTED INTERACTIVE APPLICATIONS WITH COMPUTING ON THE NETWORK EDGE

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Gustavo Bittencourt Figueiredo

Salvador
27 de abril de 2023
103p.: il.

Orientador: Prof. Dr. Gustavo Bittencourt Figueiredo.

“Algoritmos de Roteamento Multicast Orientados a Justiça para Aplicações Distribuidas Interativas Usando Computação na Borda”

Ibirisol Fontes Ferreira

Dissertação apresentada ao Colegiado do Programa de Pós-Graduação em Ciência da Computação na Universidade Federal da Bahia, como requisito parcial para obtenção do Título de Mestre em Ciência da Computação.

Banca Examinadora

Prof. Dr. Gustavo Bittencourt Figueiredo (Orientador PGCOMP)

Prof. Dr. Tiago de Oliveira Januario (IC-UFBA)

Prof. Dr. Maycon Leone Maciel Peixoto (PGCOMP)

Prof. Dr. José Augusto Suruagy Monteiro (CIN-UFPE)
To my family, extended family, and ancestors.
ACKNOWLEDGEMENTS

I want to express my deep gratitude to all those who have supported me throughout this challenging journey. In particular, I would like to thank Professor Gustavo Bittencourt Figueiredo for his invaluable guidance during my Master’s degree. Additionally, I am immensely grateful to my committee members, Prof. José Augusto Suruagy Monteiro, Prof. Maycon Leone Maciel Peixoto, and Prof. Tiago de Oliveira Januario, for their valuable and insightful comments.

Furthermore, I am thankful for my colleagues and coordinators’ unwavering aid and incentive at the National Research and Education Network Point of Presence in Bahia (PoP-BA/RNP). I also sincerely thank the Research, Development & Innovation Division (DPDI) of the Brazilian National Research and Education Network (RNP) for their valuable assistance. Likewise, I also thank the Intelligent Optical Network Lab (ION Lab) colleagues for their camaraderie. Similarly, I openly thank the professors, staff, and community of the Federal University of Bahia for their contributions throughout my studies.

Likewise, I must express gratitude to the Free Software community, which has provided me with a long list of software instrumentally used in this work and for help in building and promoting a sharing perspective for our society.

In addition, I am grateful to my family, parents, sister, extended family, and friends for their unwavering support and assistance throughout my educational saga and until completing my research and Master’s thesis. I also extend my gratitude to everyone who, whether directly or indirectly, has helped me achieve this academic objective.

Lastly, I kindly thank my lifemate Lilian and my son Raion for their love, encouragement, and support during this long journey.
Enfrentar os desafios dos serviços de rede na borda, como o roteamento considerando a Qualidade de Serviço, é uma questão crucial para os atuais esforços de pesquisa na área de redes. O roteamento multicast é uma técnica essencial para entregar serviços de roteamento com alto nível de otimização sob a perspectiva das operadoras e provedores de aplicações quando há grupos de usuários. Além disso, o roteamento considerando restrições de latência tem obstáculos para mesclar diferentes condições na solução, principalmente quando há uma perspectiva de equidade a ser realizada na comunicação dos usuários. Este trabalho visa lidar com este requisito de justiça no multicasting, apresentando soluções eficientes para aumentar o equilíbrio no roteamento através da escolha de melhores opções de caminhos para interação justa do grupo.

Palavras-chave: Distributed Interactive Application, Edge Computing, Software Defined Network, Fairness, Playability, Multicast Routing, Shortest Path, Delay and Delay-variation.
Facing the challenges of network services on the edge, such as routing considering the quality of service, is a crucial issue for the current research efforts in the area of the network. Multicast routing is an essential technique for delivering routing services with a high level of optimization from the perspective of operators and application providers when there are user groups. Furthermore, the routing that considers latency constraints has obstacles to merging different conditions in the solution, mainly when there is a fairness perspective to be accomplished in the users’ communication. This work aims to deal with this fairness requirement in multicasting, showing efficient solutions to increase the equilibrium in the routing by choosing better path options for fair group interaction.

**Keywords:** Distributed Interactive Application, Edge Computing, Software Defined Network, Fairness, Playability, Multicast Routing, Shortest Path, Delay and Delay-variation.
CONTENTS

Chapter 1—Introduction ................................................. 1
  1.1 Work Context and Motivation ................................... 1
  1.2 Theoretical Foundations ......................................... 3
    1.2.1 Graph Theory ............................................. 3
    1.2.2 Computational Complexity Theory ......................... 5
    1.2.3 Statistics Fundamentals ................................ 7
    1.2.4 Quality of Service ...................................... 8
  1.3 Research Objectives ............................................. 9
  1.4 Thesis Contributions ........................................... 10
  1.5 Thesis Outline ................................................ 11

Chapter 2—Literature Review ........................................... 13
  2.1 Edge Computing ................................................ 13
  2.2 Distributed Interactive Application ......................... 15
  2.3 Playability and Fairness ...................................... 24
  2.4 Statistical Dispersion Computation ............................ 28

Chapter 3—Multicast Routing .......................................... 29
  3.1 Multicast Communication ...................................... 29
  3.2 Shortest Path Tree ............................................. 33
  3.3 K Shortest Path ............................................... 33
  3.4 Priority Queue ................................................. 36
  3.5 Performance Metrics .......................................... 38

Chapter 4—Multicasting with Delay and Delay Variation Constraints ........................................... 41
  4.1 Related Works .................................................. 41
  4.2 Network Model .................................................. 47
  4.3 Problem Definition ............................................ 48

Chapter 5—Fairness Routing ........................................... 49
  5.1 Fairness Measure ............................................... 49
  5.2 Operationalization of Statistical Dispersion .................. 50
  5.3 Fair Shortest Path Tree (FSPT) ................................ 51
  5.4 Narrow ........................................................ 56
Chapter 6—Experimental Evaluation

6.1 Benchmark Overview ........................................... 61
6.2 Simulation Results ............................................. 63
6.3 Emulation Results ............................................. 80

Chapter 7—Conclusion and Future Work

Bibliography
# LIST OF FIGURES

1.1 DIA scenario with programmable infrastructures and distributed MEC hosting ............................................. 2
2.1 Simple MEC representation .................................................. 14
2.2 DIA interaction process through the network ....................... 18
2.3 The interchange between DIA servers across the network .......... 20
3.1 Multicast trees kind comparison ........................................... 30
3.2 Loopless shortest paths $k = 3$ .......................................... 34
3.3 Loopy shortest paths $k = 3$ .............................................. 35
5.1 FSPT trees compared to precomputing methods ....................... 54
5.2 FSPT Relaxation Step by Step precomputed by EBSPT ............. 55
5.3 Narrow Step by Step to compute solution with $k = 2$ ............... 59
5.4 Narrow and Chains tree comparison for a generic topology ......... 60
6.1 Emulated Environment .......................................................... 62
6.2 Mean Link stress from $s$ to $R$ (%CH from SPT) ..................... 64
6.3 Mean Node stress from $s$ to $R$ (%CH from SPT) ..................... 65
6.4 Mean Cost from $s$ to $R$ (%CH from SPT) ............................. 66
6.5 Mean Average latency from $s$ to $R$ (%CH from SPT) ............... 67
6.6 Mean Inter-destination delay dispersion from $s$ to $R$ (%CH from SPT) ................................................................. 68
6.7 Mean Average latency from $s$ to $R$ (%CH from SPT) ............... 69
6.8 Mean Inter-destination delay dispersion from $s$ to $R$ (%CH from SPT) ................................................................. 70
6.9 Mean Average latency from $s$ to $R$ (%CH from SPT) ............... 71
6.10 Mean Inter-destination delay dispersion from $s$ to $R$ (%CH from SPT) ............................................................... 72
6.11 Mean Inter-destination delay dispersion from $s$ to $R$ (%CH from SPT) ............................................................... 73
6.12 Mean Execution time from $s$ to $R$ (%CH from SPT) ............... 74
6.13 Mean Inter-destination delay dispersion from $s$ to $R$ (%CH from SPT) with Narrow .............................................. 75
6.14 Mean Maximum inter-destination delay variation from $s$ to $R$ (%CH from SPT) with Narrow ........................................... 76
6.15 Mean Execution time from $s$ to $R$ (%CH from SPT) with Narrow .... 77
6.16 Mean Average latency from $s$ to $R$ (%CH from SPT) with Narrow .... 78
6.17 Mean Inter-destination delay dispersion from $s$ to $R$ (%CH from SPT) with Narrow .............................................. 79
6.18 Normalized mean maximum inter-destination response time ........ 80
6.19 Normalized mean inter-destination response time dispersion ........ 81
<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.20</td>
<td>Normalized mean of routing metrics (FDDSPT)</td>
<td>82</td>
</tr>
<tr>
<td>6.21</td>
<td>Normalized mean of routing metrics (FEBSPT)</td>
<td>82</td>
</tr>
<tr>
<td>6.22</td>
<td>Normalized mean of routing metrics (FSPT)</td>
<td>83</td>
</tr>
<tr>
<td>6.23</td>
<td>Normalized mean of routing metrics (Narrow)</td>
<td>83</td>
</tr>
<tr>
<td>6.24</td>
<td>Normalized mean <em>maximum inter-player scores</em></td>
<td>84</td>
</tr>
<tr>
<td>6.25</td>
<td>Normalized mean <em>inter-player scores dispersion</em></td>
<td>84</td>
</tr>
</tbody>
</table>
LIST OF TABLES

1.1 Set and Graph Theory Notations .............................................. 5
1.2 Frequently used asymptotic notations ...................................... 6
1.3 Most common asymptotic functions .......................................... 7
1.4 Network Routing Metrics Types .............................................. 9

2.1 Names and acronyms used for Distributed Interactive Applications .... 16
2.2 Network technologies related to the most common use cases for each interactive application definition ........................................ 18
2.2 Network technologies related to the most common use cases for each interactive application definition ........................................ 19
2.3 Most used quality and fairness indexes related to the network resources employed ......................................................... 27

3.1 k Shortest Path algorithms characteristics .................................. 36
3.2 Multicast Networks Performance Metrics ................................... 38
3.2 Multicast Networks Performance Metrics ................................... 39

4.1 Algorithms for DVBMT problem and its computational complexity ... 46
4.1 Algorithms for DVBMT problem and its computational complexity ... 47
LIST OF ALGORITHMS

1. Fair-Shortest-Path Tree Construction ............................................ 52
2. Narrow Algorithm for DVBMT problem ............................................. 58
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
<td>21</td>
</tr>
<tr>
<td>AR</td>
<td>Augmented reality</td>
<td>22</td>
</tr>
<tr>
<td>BAM</td>
<td>Branch-Aware Modification</td>
<td>31</td>
</tr>
<tr>
<td>CBT</td>
<td>Core-Based Trees</td>
<td>42</td>
</tr>
<tr>
<td>CDN</td>
<td>Content Delivery Networks</td>
<td>29</td>
</tr>
<tr>
<td>DBMS</td>
<td>Delay Bounded Multi-Source</td>
<td>45</td>
</tr>
<tr>
<td>DCSBLVT</td>
<td>Directional Core Selection Build Lower Variation Tree</td>
<td>44</td>
</tr>
<tr>
<td>DCS</td>
<td>Directional Core Selection</td>
<td>44</td>
</tr>
<tr>
<td>DDCMA</td>
<td>Delay and Delay Variation Constraint Multicast Algorithm</td>
<td>43</td>
</tr>
<tr>
<td>DDMC</td>
<td>Destination-Driven MultiCasting</td>
<td>31</td>
</tr>
<tr>
<td>DDOF</td>
<td>Delta Degrees of Freedom</td>
<td>28</td>
</tr>
<tr>
<td>DDSPT</td>
<td>Destination Driven Shortest Path Tree</td>
<td>31</td>
</tr>
<tr>
<td>DDVBM</td>
<td>Distributed Delay and Delay Variation Bounded Multicast</td>
<td>42</td>
</tr>
<tr>
<td>DDVCA</td>
<td>Delay and Delay Variation Constraint Algorithm</td>
<td>42</td>
</tr>
<tr>
<td>DFS</td>
<td>Depth-first search</td>
<td>45</td>
</tr>
<tr>
<td>DIA</td>
<td>Distributed Interactive Application</td>
<td>1</td>
</tr>
<tr>
<td>DMOS</td>
<td>Degradation Mean Opinion Score</td>
<td>27</td>
</tr>
<tr>
<td>DPDVBM</td>
<td>Dynamic Program for Delay Variation Bound</td>
<td>41</td>
</tr>
<tr>
<td>DVBMT</td>
<td>Delay and Delay Variation Bounded Multicast Tree</td>
<td>3</td>
</tr>
<tr>
<td>DVE</td>
<td>Distributed Virtual Environments</td>
<td>23</td>
</tr>
<tr>
<td>DVMA</td>
<td>Delay Variation multicasting Algorithm</td>
<td>41</td>
</tr>
<tr>
<td>EBSPT</td>
<td>Early Branching Shortest Path Tree</td>
<td>31</td>
</tr>
<tr>
<td>EDVBM</td>
<td>Economic Delay and Delay-Variation Bounded Multicast</td>
<td>43</td>
</tr>
<tr>
<td>FDDSP</td>
<td>Fair Destination Driven Shortest Path Tree</td>
<td>53</td>
</tr>
<tr>
<td>FEBSPT</td>
<td>Fair Early Branching Shortest Path Tree</td>
<td>53</td>
</tr>
<tr>
<td>FPS</td>
<td>First-Person Shooter</td>
<td>22</td>
</tr>
<tr>
<td>FSPT</td>
<td>Fair Shortest Path Tree</td>
<td>xiii</td>
</tr>
<tr>
<td>ICN</td>
<td>Information Centric Networking</td>
<td>22</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>ILP</td>
<td>Integer Linear Programming</td>
<td>32</td>
</tr>
<tr>
<td>IMA</td>
<td>Interval Multicast Algorithm</td>
<td>45</td>
</tr>
<tr>
<td>IMS</td>
<td>Interval Multicast Subgraph</td>
<td>45</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
<td>14</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
<td>32</td>
</tr>
<tr>
<td>KSP</td>
<td>K Shortest Path</td>
<td>29</td>
</tr>
<tr>
<td>LMC</td>
<td>Localized Multicast</td>
<td>30</td>
</tr>
<tr>
<td>MCDVBM</td>
<td>Multi-Core Delay Variation Bound Multicast Trees</td>
<td>45</td>
</tr>
<tr>
<td>MCMA</td>
<td>Multi-Constrained Multicast Algorithm</td>
<td>43</td>
</tr>
<tr>
<td>MEC</td>
<td>Mobile Edge Computing</td>
<td>1</td>
</tr>
<tr>
<td>MF-AM</td>
<td>Minimizing total Flow entries of All Multicast demands</td>
<td>32</td>
</tr>
<tr>
<td>MF-EM</td>
<td>Minimizing Flow entries of Each Multicast demand</td>
<td>32</td>
</tr>
<tr>
<td>MMO</td>
<td>Massively Multiplayer Online game</td>
<td>15</td>
</tr>
<tr>
<td>MOS</td>
<td>Mean Opinion Score</td>
<td>27</td>
</tr>
<tr>
<td>MST</td>
<td>Minimum Spanning Tree</td>
<td>30</td>
</tr>
<tr>
<td>MU-AM</td>
<td>Minimizing Unicast entries of All Multicast demands</td>
<td>32</td>
</tr>
<tr>
<td>MVNFP</td>
<td>Multicast-oriented Virtual Network Function Placement</td>
<td>32</td>
</tr>
<tr>
<td>nEDA</td>
<td>Novel Estimation of Distribution Algorithm</td>
<td>32</td>
</tr>
<tr>
<td>NFV</td>
<td>Network Functions Virtualization</td>
<td>1</td>
</tr>
<tr>
<td>NSF</td>
<td>National Science Foundation</td>
<td>32</td>
</tr>
<tr>
<td>NVE</td>
<td>Networked Virtual Environment</td>
<td>23</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
<td>17</td>
</tr>
<tr>
<td>PiMF</td>
<td>Plug-in Multicast Framework</td>
<td>32</td>
</tr>
<tr>
<td>QoE</td>
<td>Quality of Experience</td>
<td>2</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
<td>1</td>
</tr>
<tr>
<td>REA</td>
<td>Recursive Enumeration Algorithm</td>
<td>33</td>
</tr>
<tr>
<td>ROIA</td>
<td>Real-Time Online Interactive Applications</td>
<td>21</td>
</tr>
<tr>
<td>RPG</td>
<td>Role-Playing Game</td>
<td>26</td>
</tr>
<tr>
<td>SDN</td>
<td>Software-Defined Networking</td>
<td>1</td>
</tr>
<tr>
<td>SMT</td>
<td>Steiner Minimal Tree</td>
<td>30</td>
</tr>
<tr>
<td>SON</td>
<td>Service Overlay Network</td>
<td>44</td>
</tr>
<tr>
<td>SPDVR</td>
<td>Server Selection Problems with Delay Constraints and Delay Variation Reduction</td>
<td>43</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>SPF</td>
<td>Shortest Path First</td>
<td>30</td>
</tr>
<tr>
<td>SPT</td>
<td>Shortest Path Tree</td>
<td>29</td>
</tr>
<tr>
<td>ST</td>
<td>Steiner Tree</td>
<td>30</td>
</tr>
<tr>
<td>VMNDDVC</td>
<td>Virtual Multicast service-oriented Network subject to Delay and Delay Variation Constraints</td>
<td>xxiii</td>
</tr>
<tr>
<td>VMNDDVCM</td>
<td>Virtual Multicast service-oriented Network subject to Delay and Delay Variation Constraints (VMNDDVC) Mapping algorithm</td>
<td>43</td>
</tr>
<tr>
<td>VNFs</td>
<td>Virtual Network Functions</td>
<td>32</td>
</tr>
<tr>
<td>VR</td>
<td>Virtual Reality</td>
<td>22</td>
</tr>
</tbody>
</table>
This chapter provides a comprehensive overview of our work, giving it context, outlining the research objectives, and emphasizing its notable contribution.

1.1 WORK CONTEXT AND MOTIVATION

Recent advances in programmability and virtualization of network components enabled by the paradigms Software-Defined Networking (SDN) and Network Functions Virtualization (NFV), combined with new technologies such as Mobile Edge Computing (MEC) (5G PPP SN Working Group, 2017), allow the customization of the operator infrastructure. In particular, SDN allows network operators to define dynamic network policies to ensure acceptable Quality of Service (QoS) levels between application users and hosting MEC servers. Utilizing SDN and NFV technologies offer several benefits to Distributed Interactive Application (DIA) (networked applications with real-time user interactions as their main characteristic), especially in meeting their demands for low latency and high bandwidth availability (KAWABATA et al., 2017).

Application providers typically rent a MEC server and host their services to enhance QoS, reduce latency, increase bandwidth, and prevent network bottlenecks. The MEC management and orchestration plane selects the MEC server closest to the users to receive the application instance based on the availability of computing and network resources. A MEC server is typically a virtual representation of physical resources, such as CPU, memory, and storage, which are provided on a physical server and abstracted as virtual machines, containers, and serverless computing functions.

The advent of MEC potentialized DIA, since the former allowed the delivery of applications with increased scale and quality due to border resource availability (XU; ZHANG, 2020; LIAO et al., 2020; TSIPIS; OIKONOMOU, 2021; LIANG et al., 2021; LU et al., 2022). The DIA applications exchange data between users using one-to-many or many-to-many communication schemes, regardless of the application architecture. Its main characteristics are real-time, immersive, collaborative, and even competitive user interactions. Video conferences, telemedicine, simulation-based e-learning, serious games, and
distributed and massive online games are some of the DIA showcases widespread in people’s lives.

The described scenario is depicted in Figure 1.1. The provider assigns the client’s session requests to an instance of its DIA server deployed on a physical machine and available in the operator’s infrastructure. The network operator routes client traffic within a multicast tree to connect clients with the lowest latency and the highest bandwidth while using the least amount of resources. For that, the application provider requests a multicast session using the network control plane interface that connects it to the network operator’s infrastructure. Next, using the outputted multicast topology, the routing algorithm connects each DIA client to the corresponding DIA distributed server based on the latencies of the clients through the chosen network paths.

Although integrating these applications with MEC leads to improved Quality of Experience (QoE) by bringing the processing of user interactions closer to the users, it also poses several challenges that operators must address, including efficient communication and data synchronization between multiple servers, managing user connections with load balancing, network routing, and more (DELANEY; WARD; MCLOONE, 2006b; GLINKA et al., 2008).

As the number of users simultaneously interacting through the network increases and their connections’ latency, throughput, and jitter vary significantly, many of DIA’s requirements become even more critical (DELANEY; WARD; MCLOONE, 2006a). For instance, game applications usually require a latency ranging from 0.1 to 2.0 seconds between the clients and the game server (KASENIDES; PASPALLIS, 2019; MEILÄNDER et al., 2012). This limit is crucial to preserve the consistency and playability of the game state (BRUN; SAFAEI; BOUSTEAD, 2006a; DELANEY; WARD; MCLOONE, 2006a), including latency for fine synchronization between the servers. However, as analyzed by
1.2 THEORETICAL FOUNDATIONS

(BRUN; SAFAEI; BOUSTEAD, 2006c), the variation in the response time between the participants creates an inequity in the level of fairness of the game. To overcome this, the response time of target users is artificially increased to equalize latencies (ZANDER; LEEDER; ARMITAGE, 2005; LI et al., 2018).

As an alternative, rather than increasing response time within the components of DIA, custom routing techniques such as multicast are employed by network operators to minimize latency and reduce its variation among users. This strategy was formulated and labeled as the Delay and Delay Variation Bounded Multicast Tree (DVBMT) problem in (ROUSKAS; BALDINE, 1996). Addressing this problem simplifies the network-related tasks for the application providers as the network operators become responsible for that.

On the other hand, devising routing strategies that balance latency variation minimization with ensuring application fairness requires a formulation that guarantees QoS requirements while also considering metrics that measure the significant impact on application interactions. This task poses a considerable challenge from a networking perspective, given DIA’s unique workings and characteristics.

Therefore, this thesis proposes two algorithms to construct multicast routing options oriented to interactive applications’ latency requirements, which can be implemented on programmable networks. The algorithms improve the level of fairness among users. Additionally, the experiments showed that both algorithms enhance equalization between users’ inter-destination response times, displaying balanced results in other metrics and performing under the computational time complexity of the literature heuristics.

1.2 THEORETICAL FOUNDATIONS

Our work is grounded in a comprehensive theoretical framework from computer science, mathematics, statistics, and interdisciplinary domains. This foundation provides us with the necessary tools and knowledge to approach our challenge in the computer network field rigorously, accurately, and leading to suitable solutions. This section describes the main theoretical elements necessary for this thesis.

1.2.1 Graph Theory

Graph Theory supports many science fields in modeling and analyzing relations between physical or abstracted elements. Usually, a graph expresses relationships between different entities, such as computers, people, or concepts. When represented as a diagram in a plane or a three-dimensional space, circles or dots represent the components in a graph, and lines express their associations. The lines may designate various relationships, such as physical connections, communication links, or shared resources. Specifically for the computer network field, using a graph, we can make an abstract representation of a network, which is a set of vertices (points or nodes) and a set of edges (arcs or links), and using an algorithm with this modeling is possible to find relations or patterns which make a solution for a problem instance. This work follows most terms and definitions of Graph Theory combined with symbols and properties of Set Theory, based on the following consolidated textbooks (WALLIS, 2007; ERCIYES, 2018; SAOUB, 2021; WEST, 2000).
Formally, a graph $G$, denoted as $G = (V, E)$, is a discrete structure made up of a set of vertices $V_G$ and a set of edges $E_G$, or simply $V$ and $E$ respectively, with a relation that links each edge to two vertices in the set $V$ called endpoints.

When describing an element of $V$, the symbol $v_i$ is used, likewise for an edge from $E$, the specific representation $e_{i,j} = (v_i, v_j)$ comes, with has $v_i$ and $v_j$ named as endpoints of the edge, and both are simultaneously considered adjacent to each other. Whenever two vertices are adjacent, they are also called neighbors, and the set of all neighbors of a vertex $v_i$ is denoted neighbors($v_i$).

Occasionally, the subgraph concept comes together, allowing for handling only a specific portion of a graph. A subgraph $G'$ is a graph formed by a subset of vertices and edges from the original graph $G$. Sometimes, the subgraph can be expressed as a different letter from the original graph.

Specifically, some circumstances need a weight associated with graph edges or vertices. The weights generally consist of real numbers associated with either edges or vertices. A vertex-weighted graph has a function $f : V \mapsto \mathbb{R}$ mapping $V$ to a real number. Likewise, an edge-weighted graph has a function $f : E \mapsto \mathbb{R}$ mapping $E$ to a real number.

When crossing a graph, some additional definition comes in. A walk is a linking sequence of edges in a graph that connects two vertices. A trail is a type of walk where every edge is unique, while a path is a trail that passes through distinct vertices. Additionally, a closed walk is a sequence of edges that starts and ends at the same vertex. A circuit is a closed trail that starts and ends at the same vertex, and it does not repeat any edges but can repeat vertices. Even more restrictive, a cycle is a closed path with no edge and vertex twice, except for the starting and ending vertex.

Unlike Graph Theory textbooks, when dealing with graph modeling networks and their routing algorithms, the definition of path changes and allows that path to have repeatedly vertices (HOFFMAN; PAVLEY, 1959), which turns the terms path to the sequence of nodes and edges that must be followed successively to traverse from a starting node to an ending node, also known as a route (BRANDER; SINCLAIR; others, 1995). A simple or elementary path has no loops and is used to denote a regular path, as described before using the Graph Theory definition.

A tree $T$, repeatedly utilized in this work, is a special kind of graph, commonly used to describe an acyclic and connected subgraph for which there are no cycles or circuits. A connected graph is a graph in that for every pair of vertices is a path between them. Furthermore, a tree is called rooted tree when there is a specially specified vertex named root, which is referenced as the source from which the other vertex positions are defined in the tree. For example, $v_i$ is called an ancestor of $v_j$ if $v_i$ is on the unique path between the root and $v_j$, when $v_i$ is a parent of $v_j$ then it is an ancestor of $v_j$ and it is adjacent to $v_j$ in the single path between the root and $v_j$ in the tree. In contrast, $v_j$ is a child of $v_i$ if $v_i$ is on the tree path between the root and $v_j$ and it is also adjacent to $v_j$ in this path. If necessary to reference the parent of a vertex $v_i$, it is denoted as parent($v_i$).

Table 1.1 summarizes the mathematical notation from the set and graph theory used in this work to handle network modeling.
### 1.2 THEORETICAL FOUNDATIONS

#### 1.2.2 Computational Complexity Theory

When expressing some concepts related to Computational Complexity Theory, this work takes on some well-known textbooks such as Knuth (1997), Cormen et al. (2022), Garey & Johnson (1990).

Essentially, an *algorithm* provides a sequence of instructions or steps to solve a problem or complete a task. It involves following a process to achieve a specific goal in a

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meanings</th>
<th>Usage Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{N}$</td>
<td>The set of natural numbers</td>
<td>$\mathbb{N}_0$ (Naturals except zero)</td>
</tr>
<tr>
<td>$\mathbb{Z}$</td>
<td>The set of integers</td>
<td>$\mathbb{Z}^-$ (Negative integers)</td>
</tr>
<tr>
<td>$\mathbb{R}$</td>
<td>The set of real numbers</td>
<td>$\mathbb{R}^+$ (Positive real numbers)</td>
</tr>
<tr>
<td>$a,...,z$</td>
<td>A element collection name</td>
<td>$v_1, v_2, v_3, ..., v_n$</td>
</tr>
<tr>
<td>$A,...,Z$</td>
<td>A set name</td>
<td>$A$</td>
</tr>
<tr>
<td>$(a_1,a_2)$</td>
<td>A ordered pair (collection of two elements)</td>
<td>$(v_i,v_j)$</td>
</tr>
<tr>
<td>${}$</td>
<td>A set representations with elements listing</td>
<td>${e_1,e_2,e_3,...,e_n}$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>A empty set</td>
<td>${}=\emptyset$</td>
</tr>
<tr>
<td>$\in$</td>
<td>The element is a member of a set</td>
<td>$v_i \in V$</td>
</tr>
<tr>
<td>$\notin$</td>
<td>The element is not a member of a set</td>
<td>$v_i \notin V$</td>
</tr>
<tr>
<td>$\subset$</td>
<td>The first set is a proper subset of the second set</td>
<td>$R \subset V$</td>
</tr>
<tr>
<td>$\subseteq$</td>
<td>The first set is a subset of or equal to the second set</td>
<td>$V_T \subseteq V$</td>
</tr>
<tr>
<td>$\supset$</td>
<td>The set is a proper superset of a set</td>
<td>$V \supset V_T$</td>
</tr>
<tr>
<td>$\supseteq$</td>
<td>The set is a superset of or equal to a set</td>
<td>$V \subseteq R$</td>
</tr>
<tr>
<td>$\cap$</td>
<td>The <em>intersection</em> of sets elements</td>
<td>$R \cap {v_i}$</td>
</tr>
<tr>
<td>$\cup$</td>
<td>The <em>union</em> of sets elements</td>
<td>$R \cup {v_i}$</td>
</tr>
<tr>
<td>$A - B$</td>
<td>The <em>difference</em> between two sets</td>
<td>$V_T - R$</td>
</tr>
<tr>
<td>$</td>
<td>A</td>
<td>$</td>
</tr>
<tr>
<td>$A(...),...,Z(...)$</td>
<td>A function name and its possible arguments</td>
<td>$W(v_i,v_j)$</td>
</tr>
<tr>
<td>$P_G(v_i,v_j)$</td>
<td>A path from $v_i$ to $v_j$ in $G$</td>
<td>$P_T(v_i,v_j)$</td>
</tr>
<tr>
<td>$v_i \rightarrow ... \rightarrow v_j$</td>
<td>A path with edges direction and vertices listing from $v_i$ to $v_j$</td>
<td>$s \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_1$</td>
</tr>
</tbody>
</table>

Table 1.1: Set and Graph Theory Notations
finite time computing an output (or output set) based on a given input (or input set) through predefined steps which transform it into the other. Algorithms are applied in various domains, such as mathematics, computer science, engineering, economics, and the humanities.

Considering that a given algorithm is correct, it receives a valid input, finishes in finite time, and always returns an output. The algorithm solves a problem when it is correct. In addition to the algorithm’s correctness, its performance is measured by its efficiency in dealing with the input size or the amount of data ingested as input based on a metric. Usually, the evaluated metrics are the amount of memory an algorithm will consume during its running and the time consumed to execute its steps, which are called space complexity and time complexity, respectively, or generalized as computational complexity.

Generally exempted from any implementation and running environment detail, an algorithm’s efficiency analysis uses a function that maps the computational complexity based on the input magnitude expressed as a natural or real number. Moreover, this function may be bounded explicitly by using the asymptotic notation, which limits how the function behaves by a mathematical notation that receives a specific value or infinity as the argument. This notation is frequently utilized to estimate an algorithm’s complexity and determine its efficiency compared to other algorithms for solving the same problem. Table 1.2 shows the most common notations.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O$</td>
<td>Big-Oh notation defines the upper bound.</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>Big-Omega notation defines the lower bound.</td>
</tr>
<tr>
<td>$\Theta$</td>
<td>Big-Theta notation defines the tight bound (the upper and lower bound).</td>
</tr>
<tr>
<td>$o$</td>
<td>Little-Oh notation defines the upper bound that is not tight.</td>
</tr>
<tr>
<td>$\omega$</td>
<td>Little-Omega notation defines the lower bound that is not tight.</td>
</tr>
</tbody>
</table>

Table 1.2: Frequently used asymptotic notations

Mainly because analyzing the computational complexity is not practical if necessary to evaluate all possible instances of a problem, the asymptotic notations are made up of considering scenarios in which the algorithms can run better or worse based on the selected set of inputs in the analysis. Typically, those input sets describe the algorithm’s behavior in the worst, average, and best cases. Thus, considering the worst case, it is possible to draw a maximum limit to evaluate whenever the algorithms should consume the most computational resources, and it will be taken during the asymptotic analysis made in this work.

Lastly, it is crucial to note that the asymptotic function bound describes how a function behaves as it approaches a value while undergoing a sequence of elementary computational operations. As a result, certain algorithms exhibit a specific bound, which can be found in Table 1.3. Therefore, Big-Oh notation is used to describe the worst-case scenario of an algorithm, which is the maximum amount of time it will take to complete,
that for a polynomial algorithm is its polynomial order. For example, suppose that the algorithm performs elementary operations described by a function \( a \times n^2 + b \times n + c \). In that case, the highest order is 2, and the algorithm is classified as \( O(n^2) \), i.e., bounded by a quadratic asymptotic function.

<table>
<thead>
<tr>
<th>Asymptotic Function</th>
<th>Bound Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Constant function</td>
</tr>
<tr>
<td>( \log(n) )</td>
<td>Logarithmic</td>
</tr>
<tr>
<td>( n )</td>
<td>Linear</td>
</tr>
<tr>
<td>( n \times \log(n) )</td>
<td>Log-linear</td>
</tr>
<tr>
<td>( n^2 )</td>
<td>Quadratic</td>
</tr>
<tr>
<td>( n^3 )</td>
<td>Cubic</td>
</tr>
<tr>
<td>( n^k )</td>
<td>Polynomial (k is a non-negative integer)</td>
</tr>
<tr>
<td>( 2^n )</td>
<td>Exponential</td>
</tr>
<tr>
<td>( n! )</td>
<td>Factorial</td>
</tr>
</tbody>
</table>

Table 1.3: Most common asymptotic functions

1.2.3 Statistics Fundamentals

The statistical methods and terms used in this work are based on widely used textbooks, such as Trivedi (2016) and Stuart & Ross (2021).

In statistics, a population is a set of elements under investigation, and a slice (or subset) of elements from it is called a sample. In helping to estimate the population characteristics, statistical methods are necessary when conducting data set sampling or evaluating evidence produced on its analysis (TRIVEDI, 2016).

Furthermore, statistical methods are crucial when working with conducted experiments in which only a numerical portion of the complete data is considered Stuart & Ross (2021).

Both scenarios need central tendencies and dispersion quantities, such as mean, variance, and standard deviation, to summarize the sample characteristics and the shape of a probability distribution, which describes a random variable observed in an experiment. These are important measures to summarize as a numerical quantity determined by a data set from the analyzed random variable \( X_i = \{x_1, x_2, x_3, ...\} \).

The mean, in some contexts also called average, is a central measure used to find the arithmetic mean value of a set of numbers, that is, the total sum of the number divided by the set size. In particular, the mean can suffer when there are outliers or extreme values in the data set. The sample mean is calculated using the bellow equation and can also estimate the population mean when it is unbiased.

\[
\mu = \frac{1}{n} \sum_{i=1}^{n} x_i
\]  

(1.1)
Furthermore, measuring how spread a set of data is carries other essential measures. First, by using the variance, which calculates how a data set diverges from its mean value. The variance of the sample mean is calculated utilizing the equation 1.2 for a finite population or through its unbiased version shown in equation 1.3.

\[
\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n} \quad (1.2)
\]

\[
\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n - 1} \quad (1.3)
\]

Secondly, the standard deviation, which also measures how a data set is distributed far from its mean value. However, in opposition to variance, the standard deviations preserve a data unit relation with the set elements. It is calculated as the square root of the variance and is typically used to quantify the variation or dispersion in a data set. A higher standard deviation indicates that the values are more spread out of the mean. In contrast, a lower standard deviation implies that the values are more tightly clustered around the mean. It is calculated ahead by the equations 1.4 or 1.5, biased or unbiased, respectively.

\[
\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}} \quad (1.4)
\]

\[
\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n - 1}} \quad (1.5)
\]

Lastly, it is necessary to note that the difference in the denominator of biased and unbiased dispersion metrics comes from the concept of degrees of freedom referring to the number of values within a data set that can vary freely. It is essential in identifying the number of independent observations within a sample and is also used to compute the variance of a population. In other words, the degrees of freedom also allow us to use a sample statistic to estimate a population parameter.

All these concepts will be used in this work to measure the network metrics dispersion, mainly for the fairness concept on the application level, which will use the standard deviation to evaluate how fair the network service is at providing the same level of service for a group of DIA users.

1.2.4 Quality of Service

In a computer network, the QoS emerges as a crucial element to meet the service requirements, bringing techniques to manage network traffic and prioritize certain data types. It ensures that specific service requests receive the necessary resources to transmit application data efficiently and effectively across the network.

Usually, the allocation of resources for QoS is handled by the network’s control plane, which considers the available resources to satisfy the request. However, selecting a route to deploy the request may be limited by residual bandwidth, delay, jitter, and packet loss, known as routing constraints.
1.3 RESEARCH OBJECTIVES

Some different constraint types are classified based on their mathematical proprieties when evaluated and measured along a path in the network (MEDHI; RAMASAMY, 2018; WANG; HOU, 2000; WANG; LI; XU, 2007). First, assuming that $F(e_{i,j})$ defines the performance metric for link $e_{i,j} = (v_i, v_j)$ and a path is defined as $P_G(v_k, v_n) = (v_k, v_l, ..., v_m, v_n)$. The additive constraints are measured by the sum of the values of each link along the path following the propriety $F(v_k, v_n) = F(v_k, v_l) + ... + F(v_m, v_n)$. Second, the multiplicative constraints are computed by the product of each link metric with the propriety of $F(v_k, v_n) = F(v_k, v_l) \times ... \times F(v_m, v_n)$. Lastly, the concave constraints are calculated with the minimum value along the path following the propriety $F(v_k, v_n) = \min(F(v_k, v_l), ..., F(v_m, v_n))$.

<table>
<thead>
<tr>
<th>Path Measure</th>
<th>Description</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay</td>
<td>The time it takes for a packet to travel from source to destination.</td>
<td>Additive</td>
</tr>
<tr>
<td>Jitter</td>
<td>The variation in delay between packets.</td>
<td>Additive</td>
</tr>
<tr>
<td>Loss Rate</td>
<td>The number of packets lost during the transmission window.</td>
<td>Multiplicative</td>
</tr>
<tr>
<td>Hop</td>
<td>The forwarding of the packet from one link to another.</td>
<td>Additive</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>The available capacity to transmit data.</td>
<td>Concave</td>
</tr>
</tbody>
</table>

Table 1.4: Network Routing Metrics Types

Linked to QoS emerges the QoE in the network field, which measures a user’s overall satisfaction when using a network service. It is typically measured by assessing the user’s perceived performance, reliability, and satisfaction with the service. QoE emerges as a vital aspect of network operation, as it helps identify improvement areas, optimizes the services for a better user experience, and, when necessary, drives QoS refinement. The discussion over user perception guides this work, and even though our experiments do not use proper user feedback in evaluation, they emanate from user-perspective network performance and use-impression metrics.

1.3 RESEARCH OBJECTIVES

The study was based on recent publications on multicast routing and fairness for distributed interactive applications, focusing on how they meet the quality of service and handle the user experience. Additionally, this work lists the strategies adopted to mitigate the negative consequences of different network conditions on group interactions through a DIA.
Furthermore, considering the interactive scenario between clients and servers of a DIA in the routing plan of a communication network as shown in Figure 1.1, the following question is posed as a research problem.

Is there a routing option for a multicast network that improves the communication fairness between a source node and the destination nodes of an interaction application group?

The main research objective is to investigate, propose, and deploy routing mechanisms that ensure better fairness when constructing multicast trees, facilitating fairer communication between users and interactive application servers.

The specific objectives are as follows:

- To analyze and implement algorithms to construct multicast trees.
- To propose, design, and develop fairness-oriented heuristics to build multicast trees based on the source.
- To employ and test the proposed multicast routing algorithms and compares them to the literature on a programmable network by simulating DIA’s communication.
- To examine the fairer multicast routing influences on DIA users’ performance on DIA session results.

### 1.4 THESIS CONTRIBUTIONS

This work has made several significant contributions, which are highlighted below:

- Analyzes the literature related to multicast routing for interactive applications.
- Presents a performance metric to measure the quality of multicast routing algorithms.
- Introduces a new evaluation criterion for DVBMT solutions.
- Offers an algorithm to construct multicast routing trees that meet the latency requirements of interactive applications while improving fairness among users in the same routing tree.
- Proposes an algorithm to solve the DVBMT problem applied for overlay networks that meet latency requirements of the interactive application with a near-optimal fairness level.
- Shows the practicality of these algorithms in fitting interactive real-time application requirements.
- Compares the disadvantages and advantages of algorithms in the literature with proposed solutions.
• Demonstrates the feasibility of implementing the proposed algorithm on programmable networks.

• Identifies the benefits of delay and delay-variation multicast algorithms in a practical DIA environment.

In addition to the contributions that came from the primary objectives, this work also pursues other lateral-aimed contributions, which are not mandatory for program completion and which are presented below:

• Provides a framework for multicast network simulation.

• Supplies an emulator for DIA behavior analysis due to different network conditions.

• Makes the generated data available in a portable way.

1.5 THESIS OUTLINE

This work is segmented as follows. First, Chapter 2 provides a detailed overview of the background topics. Further ahead, Chapter 3 delves into the literature review of multicast routing. Moving forward, Chapter 4 analyzes related work on delay and delay-variation, including the problem formulation. Then, Chapter 5 presents the proposed solutions along with some necessary concepts. In Chapter 6, all experiments and their respective results are discussed and shown. Finally, Chapter 7 offers concluding statements.
LITERATURE REVIEW

This chapter introduces the concepts and literature related to our proposed scenario, starting with edge computing, then moving on to the interactive applications, followed by playability and fairness, and lastly with the methods used for statistical dispersion computing as is necessary for fairness calculation.

2.1 EDGE COMPUTING

Many edge computing architectures (MANSOURI; BABAR, 2021) exist. However, MEC is standardized and has a higher maturity level than several other architectural proposals. Furthermore, MEC has several advantages for its deployment and operation (KEKKI et al., 2018). Also known as Multi-access Edge Computing, its core motivation was to fit computational resources into the network's edge.

Figure 2.1 exemplifies a MEC deployment on a mobile operator network. There are switches to commute the traffic that constitutes the Data Plane, servers to host and compute both network services and user-level applications above the abstraction provided by MEC platform components in each MEC host (REZNIK et al., 2017), physical sites to host the cyberinfrastructure components, and all necessary facilities such as racks, power supply and cooling. It is important to emphasize that MEC is also concerned with the dimensioning to achieve the operation and the business cost.

MEC has an orchestration and management plane based on the availability of computational and network resources, which selects a MEC server for receiving the demand. It generally does so from a third-party provider holding the user application. Often, this placement also considers the best choice to improve the performance of the application. The MEC server is typically a virtual abstraction of physical resources, such as CPU, memory, and storage, provided by virtual machines, containers, and functions deployment.

Furthermore, technologies for dynamic network control and management, as well as SDN, align with the MEC architecture. In particular, SDN allows the network operator to use the control plane to define network services policies to guarantee acceptable levels of QoS between the application users and the hosting MEC server. Besides, it brings the
possibility of handling the application traffic appropriately. Generally, it is carried out by a traffic steering routine that defines the alternative paths that application traffic can take when it crosses the network.

Nevertheless, MEC has several challenges regarding the placement of servers, services and applications in the available computing nodes, mainly when its placement is to host a real-time application sensitive to latency issues. Usually, all possible resource constraints are considered for the placement of MEC nodes.

The authors of the study (LO; NIANG, 2021) list the main benefits of having the computational resource available on the network edge when considering an Internet of Things (IoT) scenario. Specifically, they contrasted the importance of latency handling with edge computing for real-time interaction. They also highlighted the importance of centralized management techniques for supporting QoE.

In (SHAH; GREGORY; LI, 2021), the researchers investigated the edge network and its integration using cloud-native technologies with MEC architecture and controlling the network service. The new challenges and issues related to the chaining of services and network functions were also listed, mainly considering the computation chain, control and orchestration, and resource allocation to MEC compatible infrastructures. They also emphasized the importance of MEC in providing customized services in 5G and even 6G mobile edge networks.

Mansouri & Babar (2021) and Ranaweera, Jurcut & Liyanage (2021) evaluated the edge’s network resource and computational virtualization. The authors highlighted the difficulty in selecting and assigning the node to host some services, as well as the barriers
2.2 DISTRIBUTED INTERACTIVE APPLICATION

for real-time applications sensitive to latency and jitter.

Zhu & Huang (2017) proposed an algorithm to solve virtual machine placement in a MEC infrastructure. Although they had made analytical modeling of the problem, they used an approximation strategy with stochastic programming to solve it. In general, their work demonstrates that availability can be better achieved when there is much diversity in the use of the edge server and the minimization of resource usage. Additionally, in their extension of work (ZHOU; HUANG, 2018) to deal with function chaining, they show that the service order in the placement can be serialized to distribute compute load or even to improve the overall performance metrics on the servers.

Alabbasi, Wang & Cavdar (2018) proposed a mobile access network with a hybrid architecture with two levels of data hosting, specifically the central and edge cloud. They also assigned a NFV chaining between the clouds to balance the load, reduce the bandwidth usage, and decrease the energy consumption. A model and its implementation were shown by employing constraint programming to place the network function processing in the hosting spots. After some computational steps targeting those metrics minimization, their model reaches the equilibrium threshold between the energy and the bandwidth consumption. However, their work did not evaluate the dynamic environment when users joined and exited the network or alter the demands. Furthermore, they neither considered the service level characteristics in their analysis.

Tsipis & Oikonomou (2021) explored the MEC to provision a DIA server on the edge to optimize and improve the interactions of the user when they were in the same region. The server placement problem was studied, and a heuristic solution was proposed to balance two antagonistic metrics, the provider charge and user interaction quality, minimizing one and maximizing the other.

Meanwhile, Kumar et al. (2022) proposed a protocol to select the trade-off between the usage cost and the QoS, balancing the load demand on edge servers. Specifically, they formulated the application user allocation problem and proposed a solution that allocates the distributed edge servers. Although the problem is NP-Complete, they showed an efficient algorithm for the scenario described.

Parastar, Pakdaman & Hashemi (2020), Tsipis & Oikonomou (2021), Lo & Niang (2021), Ruan & Xie (2021), Liang et al. (2021), Kumar et al. (2022) all point to a growing literature attention to the edge infrastructures, mainly to address challenges and overcome its issues to deliver services with both high-level of quality and working guarantees to interactive applications.

2.2 DISTRIBUTED INTERACTIVE APPLICATION

The DIA is a networked application category with real-time user interactions as its main characteristic. That needs guarantees of lower response time and consistency for its virtual environment state shared by users. Those guarantees are critical to maintaining the user’s QoE effectively. Some recurrent DIA cases are video conferences, simulation-based e-learning, serious games, distributed, Massively Multiplayer Online game (MMO), and emerging networked interactive multimedia applications with immersive media technology. All those applications have the common condition of real-time interactions between
users inside a shared virtual environment.

DIA is defined to be:

“a networked software system that seeks to maintain global consistency when responding to multiple simultaneous nondeterministic inputs.” (DELANEY; WARD; MCLOONE, 2006a, p. 221).

DIAs are broadly widespread and studied in many computational science fields. DIAs also have numerous names, and acronyms (DELANEY; WARD; MCLOONE, 2006a) as well as listed on Table 2.1. Although those terms characterize these applications within the literature, this study will use DIA.

<table>
<thead>
<tr>
<th>Term</th>
<th>Acronyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Interactive Application</td>
<td>DIA</td>
</tr>
<tr>
<td>Distributed Virtual Environments</td>
<td>DVE</td>
</tr>
<tr>
<td>Real-Time Online Interactive Applications</td>
<td>ROIA</td>
</tr>
<tr>
<td>Distributed Interactive Simulation</td>
<td>DIS</td>
</tr>
<tr>
<td>Networked Virtual Environment</td>
<td>NVE</td>
</tr>
</tbody>
</table>

Table 2.1: Names and acronyms used for Distributed Interactive Applications

A DIA is a complex system with a considerable amount of elements. Generally, the components listed below are considered fundamental for their structure (DELANEY; WARD; MCLOONE, 2006a; GLINKA et al., 2008).

**Environment:** All necessary data to show and render the application state at a specific moment, that also contains the definitions and rules of how it must work. It is the most important as either a virtual universe or a world.

**Entity:** A virtual environment element that can interact passively or actively. The first can be represented by either stationary objects or elements with predefined and predictable moves, which the application can define. The second is generally controlled by the application’s users, who have a non-deterministic interaction pattern.

**State:** Every data related to an application virtual environment entity at a specific time. It must be updated and shared with all other distributed components of the applications, including the users’ elements.

**Event:** An occurrence that causes change in the DIA state. It can come from deterministic events, which may be foreseen, or non-deterministic, which have improbable predictability and come from real-time user interactions.

**Node:** Any computational device that processes or computes some DIA data involved in the application providing flow.
User: Character that interacts with DIA using a specific interface for each DIA application, for instance, a game joystick. This interface inputs user commands as a DIA event to be processed by DIA engine.

There are essential DIAs characteristics that are fundamental to be considered with regard to its architecture and serving. Among them, it is imperative to detail the following ones.

**Logic:** Entity, events, environment, all functional application descriptions and rules, and the powers that govern the relationship between entities and the DIA ecosystem.

**Engine:** A real-time loop that process all DIA logic to compute the new application state. When a user interacts continuously in the loop, it is called human-in-the-loop or also man-in-the-loop.

**Distribution:** Also named partitioning, it is the logical division of the DIA components among processing nodes, including the manager of the division and communication between the nodes. That allows both the interaction interface distribution and the division of state processing. The distribution can use an architecture with multiple primary nodes and other secondaries, without a primary node using a Peer-to-Peer (P2P) scheme between servers, or even without servers nodes when only clients coordinate the state computation and exchange.

Figure 2.2 depicted a DIA users’ interactions through the communicational network and its server node computing those interactions received. All interactions sent from clients to the DIA server are processed after they are received, and when the new state is generated, the server returns the new DIA state to all clients that render it to users. Next, following the human-in-the-loop aspects, the users can interact and generate new events to be processed by the server node again. This exchange process is why experienced interaction quality depends on the consistency of the received data for the DIA users actions.

Furthermore, many DIAs examples are perceived in literature as use cases, some of which are focused in studies reviewed in this work, shown in Table 2.2.

It should be noticed that several DIAs emerge continuously with new interaction technologies, and some of their operating requirements challenge the handling of users’ QoE.
Table 2.2: Network technologies related to the most common use cases for each interactive application definition

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Network Technologies</th>
<th>Use cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIA</td>
<td>SDN and MEC</td>
<td>Distributed virtual environment, distributed interactive simulations, electronic games, massively multiplayer online games, entertainment, collaboration systems, telerobotics, virtual workspace, shared interactive board, telehealth, distributed simulations, video conferencing, teleconference, virtual and augmented reality, and serious games.</td>
</tr>
<tr>
<td>DVE</td>
<td>Cloud, BUS and Spatial Publish Subscribe</td>
<td>Massively multiplayer online games, entertainment, collaboration systems, telerobotics, training, engineering test and analysis.</td>
</tr>
<tr>
<td>NVE</td>
<td>QoS and Cloud</td>
<td>Electronic games, multiplayer online games, remote virtual meetings, training, teaching, collaborative working environment and social interaction environment.</td>
</tr>
</tbody>
</table>
2.2 DISTRIBUTED INTERACTIVE APPLICATION

Table 2.2: Network technologies related to the most common use cases for each interactive application definition

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Network Technologies</th>
<th>Use cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>SDN, QoS, OpenFlow, MEC, NFV, Cloud, AR/VR, ICN, and 5G</td>
<td>Video streaming, virtual and augmented reality, multiple sensorial media, online games, video conferencing, VoIP, virtual navigation, facial recognition, and telepresence.</td>
</tr>
</tbody>
</table>

Generally, a DIA with higher number of users, and built for large-scale interactions, adopts techniques for logical partitioning of the environment (GLINKA et al., 2008), mainly to achieve process and synchronize every generated event in all broad interactions and share it with an elevated level of consistency.

The three main techniques used for the virtual environment distribution are **zoning**, **instancing**, **replication** (GLINKA et al., 2008). Figure 2.3 shows a scenario with multiple server nodes. All servers with labels from 1 to \( n \) share the responsibility to reckon the virtual environment. The initial division is into two areas (or regions) and uses **zoning**, which splits the interaction responsibility areas between servers 1 and \( n - 1 \). These servers receive each client’s connection based on where the user-controlled entity currently is and interacts. They synchronize the processed events with other servers through updated messages to keep all crucial servers with the same virtual world knowledge. Additionally, some scenarios must have the virtual environment shared in independent copies that do not need critical global synchronization between the main servers. This is when the technique **instancing** can mainly be used (BRUN; SAFAEI; BOUSTEAD, 2006c) mainly to dive into some non-critical areas that do not have interactions that cause state incoherence. For example, MMO uses **instancing** to split dungeon areas where users can go on missions and not conflict with other groups in the same environment. Figure 2.3 illustrates an area under the trusting of server 1, despite sharing a specific sub-region with server 2 which maintains this **instancing** isolated. In addition, Figure 2.3 shows the approach to share the whole server \( n - 1 \) responsibility area with the server \( n \) through **replication**. This allows the processing load and user connectivity division between them. Besides, there must have a synchronization of this environment continuously shared.

All sharing techniques need a coherent state synchronization between the server nodes, as the scenario with only one server also does. Mainly because the users’ communication is already a critical synchronization-dependent component of the DIA. Moreover, those synchronizations can drastically affect the same-region interaction group if consistency is not granted in their intersection server nodes. Despite the overall time to process the
received users’ interactions, the users in fragile connection conditions, the latency above all, can also affect the whole DIA response time. Those users have the critical response time to DIA (BRUN; SAFAEI; BOUSTEAD, 2006c), and usually, when they are assigned to faster servers, it benefits an entire group of users.

Figure 2.3: The interchange between DIA servers across the network.

The DIA consistency and responsiveness play a crucial role in ensuring the users QoE. Especially for consistency, here are some aspects DIA architects ensure and pay attention to:

**Synchronisation:** Conserves the events and entity information at the same state across all DIA distributed components. For example, it guarantees that all remote DIA attendants at a game have the same perception of the match status during its execution.

**Causality:** Also called ordering or eventuality, maintains the events yield ordering whatever is made artificially or naturally, even when it was received in different times. For instance, when the participants of a DIA session have diverse interactions, it is processed in the same order they generated it.

**Concurrency:** Ensures conflict resolution, although users dispute after interacting with the same entities. When two DIA users are interacting in a game match, for exam-
Concerning another aspect, the responsiveness is evaluated mainly related to DIA’s response time, whereas the events are triggered by the users. Therefore, responsiveness and consistency have an inverse proportionality aspect called the consistency-responsiveness trade-off (DELANEY; WARD; MCLOONE, 2006a). In other words, an increased level of responsiveness can be reached locally unless the latency affects DIA global state.

In opposition to that, many problems make the DIA operation difficult, although the latency is vital. That means either networking or computation time aspects interfere drastically with DIA latency level. The second can be composed by processing events, environment, images, sounds, and other necessary media to immerse the user in local or remote manipulation. Providing the analysis of DIA networking behavior, remote interactions could be assured, and for that, the following latencies sources are considered within the literature.

**Packet processing delay:** Often required for sending, forwarding, and receiving, including queueing and buffering in each network node.

**Propagation and transmission delay:** Time duration related to the propagation of the data across the physical medium.

**Packet propagation delay:** Time needed to deliver a packet, also known as end-to-end delay or path latency, directly impacted by available bandwidth across networking from the source to the destination.

The DIA has many studies on its performance and integration with computational and networking infrastructure. Gorlatch, Humernbrum & Glinka (2014), Humernbrum, Glinka & Gorlatch (2014b), and Humernbrum, Glinka & Gorlatch (2014a) showed a framework that allows the interchange of services information between the interactive application and the SDN control plane, and they introduced a higher-level Application Programming Interface (API) to specify the network services requirements for Real-Time Online Interactive Applications (ROIA). The studies formalized the API requirements to understand network performance metrics and apply them correctly in the SDN control plane. Furthermore, the network programmability and its efficiency to deploy QoS were examined using a networked game scenario with up to 400 users. It was identified that the quality of metrics degradation negatively impacted the final application response time. Although they had investigated the programmability effects on network services delivery for ROIA in practical circumstances, the framework reaction to network changes was not assessed. In addition, there were limitations in the communication improvement after the directional QoS applied, mainly because they had used non-interactive synthetic traffic to consider the application perception of the network condition. Lastly, the placement of ROIA servers was neither considered.

(GORLATCH; HUMERNBRUM, 2015) brought up previous studies that show the connection between the higher-level network metrics and the network programmability
via API. Furthermore, they demonstrated the procedure of translation between these two specific levels of QoS, high and low, from the application to the network control point of view. The author analyzed the ROIA response time facing different high-level metrics on a practical evaluation with a First-Person Shooter (FPS) genre game.

Humernbrum, Ahlbrand & Gorlatch (2017) developed a simulator to mimic ROIA application behavior sourcing the parameter from an MMO. This strategy opposes the common journey described in the literature, assessing those applications that normally use traffic replay or previous data matches to evaluate a QoS policy. Subsequently, Humernbrum, Ahlbrand & Gorlatch (2018) compare, in practice, the simulator and MMO sessions.

Meiländer et al. (2012) and Meiländer, Lorke & Gorlatch (2015) introduced scalability models for a ROIA taking into consideration the computational and networking metrics to serve a high amount of users simultaneously. The models considered computing structures distributed where the application users could be assigned to maintain the session quality face up to the dynamic of new users arriving. Meiländer & Gorlatch (2018) merged those previous models and extended the benchmarking to a more realistic cloud environment, and they used a MMO as practically ROIA.

Scarlato & Perra (2017) reviewed the interactive application literature focused on the network aspect, and they showed the potential Information Centric Networking (ICN) use to distribute the interactive content, mainly when there is an immersive media application with Augmented reality (AR) and Virtual Reality (VR) technologies. Although they had argued about aspects related to transmission types and network metrics, their work fundamentally mapped architectural possibilities to allow the communication of that real-time and interactive application.

Then, Lo & Niang (2021) assessed the QoS schemes deployment for IoT. The Edge Computing emerges as an all-purpose utility for interactive applications to deal with latency, bandwidth, mobility, energy consumption, and other critical aspects to network efficiency and service quality.

Kasenides & Paspallis (2019) systematically reviewed works involved in developing solutions for MMO issues. They depict the trend and necessity to use centralized and distributed computational solutions, such as cloud and edge computing, to solve the common problem while delivering MMO applications. They also showed the research directions on how to use an environment with higher flexibility and scalability to address the application demands once there was user overflow. The networking challenges are also determined in this study, mainly when discussing the necessary QoS warrants for those applications to perform within the expected response time range.

Oliveira et al. (2021) reviewed the SDN architecture for QoS provisioning. In addition, works and marks related to QoS for interactive applications were evaluated by them. There is a QoS provisioning architecture for SDN networks. However, that was neither assessed with an interactive application, a non-static topology, nor a different number of clients.

Zhang & Tang (2011) analyzed the problem related to client assignment for continuous DIAs and proposed an algorithm to assign the users to distributed DIA servers. The problem had its NP-completeness demonstrated, and they presented the heuristic solution
to keep consistency and fairness. The latter was considered as the situation where the
casuality of the actions is preserved, and all users have the same participation probability
regardless of the quality of their network connections. Further, they concluded the user
lag in the modeling and assumed the shortest path in the routing.

In Brun, Safaei & Boustead (2006a) they discussed and analyzed Distributed Virtual
Environments (DVE) architectures targeted to MMO. They highlighted the importance
of topological and network metric aspects to user experience and fairness while using an
interactive application.

Afterwards, Behnke, Grecos & Luck (2014), Behnke et al. (2015), and Behnke et
al. (2016) showed architectural modeling to DVE synchronization aiming at ensuring
consistency between servers, and user grouping to maintain the quality of communication.
No supposed detailed network attributes or metrics could impact the user’s experience.

Ruan & Xie (2021) addressed intrinsic challenges to the new generation of interactive
applications based on AR and VR technologies, which need advanced network services to
perform in a distributed way. They also promoted MEC to fulfill the computational load
sharing and increased the QoE. However, even adopting MEC, there is no silver bullet.
That is because it still needs attention to accomplish interaction quality by looking at
user perception of the system.

Delaney, Ward & McLoone (2006a) and Delaney, Ward & McLoone (2006b) also stud-
ied the DIAs consistency and the QoE literature. They showed a diversity of important
and recurrent DIA characteristics. Besides, they listed the issues related to latency, jitter,
network bandwidth, and transmission reliability to maintain DIA functioning properly.

Skorin-Kapov et al. (2018) made an extrapolated review of the networked multimedia
applications and their incorporation of new technological trends and concepts applied
to QoE management. They listed the great potential to achieve better user experience
through the use of SDN, MEC, and NFV, which allow user-centered services with fine
control and keeping up with the demands.

Salinas et al. (2023) analyzed researching papers on the Networked Virtual Environ-
ment (NVE), focusing on similar works and the issues they addressed. They have listed
emerging challenges given by the new trends of high-connectivity applications. The con-
cern is that the QoE driven services is a developing topic in the study for NVE application.

In spite of the haptic interface, in order to deal with NVE, Win et al. (2021) evaluate
some components incorporated into the user interactions while solving a maze game. They
showed that when multiple users fronting network metrics change, the QoE decreases
dramatically.

Karakus & Durresi (2017) revisit SDN studies related to QoS to evaluate the most
attended topics and remaining challenges. They point the network programmability as
a way to provide services with a high level of QoS awareness regarding the application
necessities as long as the network resource allocation and routing paths applied are an-
alyzed with profound attention. Although the path selection based on the application
characteristics is still open for discussion, it is indeed relevant for interactive applications.

Despite the total budget to a DIA provider, Tsipis & Oikonomou (2021) faced the
DIA server users allocation problem taking into account the interaction between them.
Therefore, they used the heuristic technique to do so and evaluated it in a simulated
environment using a traffic dump. Their solution exposed good results compared to other research studies.

By addressing the latency and bandwidth issues in data center scenarios, (AMIRI et al., 2016) and (AMIRI et al., 2017) brought a routing optimization to ensure the interactive application components communicated adequately to the metric levels needed.

Xu & Zhang (2020) explored the SDN controller loading allocations and DIA flows routing to face fairness improvement. They assumed the controller’s position on the network and its effects on fairness. Nevertheless, they also acknowledged that bandwidth and latency are vital to fairly treating users’ interactions in the environment. They proposed two algorithms. The first one was to allocate an arriving request to the controller that centered the resource allocation on the SDN controller to process the requests. At the same time, they considered the communication destination to the DIA server. Then, they used deep learning to predict the group interaction pattern and, based on the controller resources available, allocate the best controller to process the demand of flow switches. The second algorithm was planned for proper routing to safeguard a balanced bandwidth allocation while selecting routing paths with acceptable latencies to operate the application. This algorithm selects paths based on available bandwidth, and it has an approximation parameter with deep learning to ponder how to penalize paths with divergent latencies. Besides, the residual bandwidth increases the overall throughput in compliance with the max-min rule for fairness. Lastly, they evaluated both algorithms that showed a good improvement related to other solutions. Nevertheless, there is no evaluation with different sizes and topological characteristics.

Orientating the application hosted in a MEC architecture, Parastar, Pakdaman & Hashemi (2020) also studied the resource allocation and routing problem. Simultaneously, they ran the heuristic technique for assignment and routing, considering the inter-player delay minimization as fairness constraints to assigning the best server that uses the shortest paths to select the best route to optimize users’ interactions.

2.3 PLAYABILITY AND FAIRNESS

Interactive applications have playability and fairness as crucial aspects to achieve satisfactory QoE levels and increase users experience regarding enjoyment, immersion, and other factors. Consequently, the network and application service architecture are essential when it comes to allowing a user-centered approach to boost application perceptions.

Nevertheless, there are different approaches to achieve fair scenarios (AMIRI et al., 2016). First, artificially fixing the user quality metrics differences is generally implemented in a DIA processing component. Second, networking and service tuning include, for example, selecting server nodes, provisioning resource options, and routing the clients and server traffic appropriately.

Ee & Bajcsy (2004) faced the bandwidth allocation balance and packet delivery in a Sensor Network with a fairness point of view. They evidenced problems related to many-to-one multi-hop routing for data processing communications between the sensors and the central node.

Baughman & Levine (2001) analyzed fairness as an anti-cheating guarantee, defined
as the improper use of the DIA states to achieve better race conditions, mainly by adding some delay or dropping messages to the update of the application state. They proposed an anti-cheating protocol to synchronize the DIA state, using MMO as a case study. The synchronization assumes a broad scenario for the environment state storing and processing, including centralized and distributed architectures. The authors also described an overload to execute the protocol, unless their asynchronous version was being used instead.

Zander, Leeder & Armitage (2005) focused on the fairness related to network metrics quality. They analyzed fairness as the similar condition for players in a match, to achieve the same conditions for the game. In other words, they described it as the race condition where the latencies between players and servers do not affect the player’s performance. There is a component ahead of the server to analyze the connection and metrics related to players; then, an artificial delay is applied to equalize all players’ connections down to a maximum delay for proper functioning.

Brun, Safaei & Boustead (2006a) described the problems associated with consistency, playability and fairness for centralized and distributed interactive application environments and their impacts on user experience. Alike the response time, playability was adopted as the guide for the players’ interaction quality, which is affected by the level of consistency reached in different topological arrangements and their corresponding latencies. Besides, the fairness was characterized by players with the same skills and with good scores. They used a simulated distributed game tool (BRUN; SAFAEI; BOUSTEAD, 2005), with avatars controlled by a specific application module with the same actions. This emulated environment avoids interactions among real users during the experiments. It also allows to measure the impact of the network metrics. They observed that latency and topological options significantly affect game fairness. The distributed scenario, with servers close to users and equidistant from other servers, has a higher level of fairness, playability, and better response time.

Brun, Safaei & Boustead (2006c), before their last work, defined a critical response time to evaluate the users in worse playability and fairness circumstances. This metric is predicted using different interactive application architectures, and it was used to optimize the server nodes positioning to achieve better latencies balance between players. When servers are centralized between two players with ampler critical response time in a user group, they are led into a fair playability scenario. The authors assumed that the shortest path was used to establish routes, then they used the path information to select better positioning. They focused on responsible servers attribution, although it needs to be adequated topologically.

Additionally, Brun, Safaei & Boustead (2006b) described some options to bypass playability and fairness problems for interactive application providers. However, their work evidenced only possibilities related to inconsistency mitigation regarding the application side and the server placement in the available infrastructure. Although they neglected the network-specific strategies on the network operator side, it has a great potential to be used, mainly because it is currently a trend in infrastructure management. Despite the dynamic difficulty change the application infrastructure, trading off inconsistencies and tuning the topology decision points showed that a distributed environment,
with proper parameter refinements, achieves better results of playability and fairness for the users (BRUN; SAFAEI; BOUSTEAD, 2006a) (BRUN; SAFAEI; BOUSTEAD, 2006b).

Zhang & Tang (2011) analyzed the DIA interaction time between the users that, if minimized, can conserve consistency and give fairness among DIA clients. They compared client assignments using the interaction time and the connection to the nearest server approaches, considering the server node resource, and then penalized users in better conditions to balance the match. They used the shortest path to build the routes between clients and the server node. In essence, they used fairness as the guarantee for operation execution orders issued by clients.

Regarding the impossibility of network changes to ensure fair match conditions to the MMO users, Hirota & Kuribayashi (2011), using the standard ITU-T Recommendation P.910 (1999), evaluated the user perception of fairness. They considered fairness a chance for equated competition even when there are heterogeneous network conditions. Furthermore, they identified that some inconsistency events are more significant to an MMO player believing they are in a less fair situation compared to other players, mainly when the events impact the player’s performance.

Amiri et al. (2016) optimized the game session assigned to the servers in a cloud environment, taking into consideration streamed games where the video and state processing are centralized in the server. That scenario is known by its need of increased computational and network bandwidth capacity. Furthermore, they evaluated the fairness considering the average score performance of the players (ZANDER; LEEDE; ARMITAGE, 2005), that the mean kill rate was for their FPS case study games. This index was used to evaluate the assignment and balance during attribution of players to the servers. Although the data center network infrastructure was assumed as controlled, they did not change the routing options.

Millar et al. (2017) noticed the fairness level measures within the response time and consistency aligning, mainly when it can impact the case application and its satisfactory playability. They emphasized that what is considered fairness can differ depending on the simulated DIA. In other words, an MMO Role-Playing Game (RPG), an MMO FPS, training and others have different levels of consistency and response time to be considered fair (BRUN; SAFAEI; BOUSTEAD, 2006a). Besides, they discussed some models to evaluate fairness. It was depicted that the most recurrent ones use single dimensional space to position the players’ witness to correlation, commonly called fairness space or playability space.
## 2.3 Playability and Fairness

Table 2.3: Most used quality and fairness indexes related to the network resources employed

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fairness Index (Jain’s Index)</strong></td>
<td>Initially proposed to measure the fairness between the bandwidth achieved by different traffic flows that share a communication channel, it is applied to several categories of network metrics.</td>
<td>(JAIN; CHIU; HAWE, 1984; XU; ZHANG, 2020)</td>
</tr>
<tr>
<td><strong>QoE Fairness</strong></td>
<td>Proposed as an improvement of the Fairness Index to measure fairness in the application of QoE, it is intended to capture the reception index between users, which will be fair when they receive the same QoE.</td>
<td>(GEORGOPOULOS et al., 2013; PETRANGELI et al., 2014; HOBFELD et al., 2017)</td>
</tr>
<tr>
<td><strong>Mean Opinion Score (MOS)</strong></td>
<td>A system quality mean score attribution based on users’ subjective ratings, mainly used for audio, video, and audiovisual quality evaluation. Users are assessed with a quality scale grade after experiencing the system services.</td>
<td>(ITU-T Recommendation P.800, 1996; WIN et al., 2021; MAZUR; RAK; NOWICKI, 2021)</td>
</tr>
<tr>
<td><strong>Degradation Mean Opinion Score (DMOS)</strong></td>
<td>A system quality mean score attribution based on users’ subjective ratings as MOS, but with users’ gauging based on comparison of a normal and a degraded sample of the system service.</td>
<td>(ITU-T Recommendation P.910, 1999; HIROTA; KURIBAYASHI, 2011)</td>
</tr>
<tr>
<td><strong>Mean Score Rate</strong></td>
<td>Simple measure to evaluate the divergence once players’ scores are compared. Nonetheless, it depends on the period the scores were analyzed.</td>
<td>(ZANDER; LEEDER; ARMITAGE, 2005; AMIRI et al., 2016)</td>
</tr>
<tr>
<td><strong>Normalized Standard Deviation</strong></td>
<td>A more general measure to evaluate dispersion, it is used to see the divergence between a set of users correlated to a metric value. Nonetheless, it depends on the analyzed population.</td>
<td>(BRUN; SAFAEI; BOUSTEAD, 2006a; CHOI, 2016)</td>
</tr>
</tbody>
</table>

There are numerous ways to measure fairness in a virtual environment, in quantitative as many as in qualitative ways (BASIL; MU; AL-SHERBAZ, 2019). Table 2.3 summarizes some of the most recurrent index options to gauge the perceived quality and fairness in
the evaluated studies.

2.4 STATISTICAL DISPERSION COMPUTATION

Many evaluations need to calculate the *sum of squared deviations* of the samples to measure a dispersion along their mean. Despite the diversity of algorithms to calculate these dispersion metrics, such as (LING, 1974; CHAN; GOLUB; LEVEQUE, 1983), and Welford’s method (WELFORD, 1962), the latter stands out in the literature mainly because it uses an online routine to compute the new mean, variance, and standard deviation values. **Welford’s method** approaches the samples incrementally and keeps the number of operations for each value small, using a simple algorithm to compute the running mean (2.1) and the sum of squared deviations (2.2) as an unweighted version presented in (KNUTH, 1981) (attributed to Welford).

\[
\mu_n = \mu_{n-1} + \frac{(x_n - \mu_{n-1})}{n} \tag{2.1}
\]

\[
S_n = S_{n-1} + ((x_n - \mu_{n-1}) \times (x_n - \mu_n)) \tag{2.2}
\]

To compute a new dispersion metric state, the next stage of the estimator needs to be derived. For this, it requires the nth sample \(x_n\), the previous mean \(\mu_{n-1}\), and last corrected *sum of squares* \(S_{n-1}\) (denoted as the estimator). With these new states for the mean (\(\mu_n\)) and the estimator (\(S_n\)), its variance (2.3) and standard deviation (2.4) can be calculated, both should consider degrees of freedom (represented by *ddof* as *Delta Degrees of Freedom* (DDOF)) for the sampling strategy.

\[
\sigma^2 = \frac{S_n}{(n - ddof)} \tag{2.3}
\]

\[
\sigma = \sqrt{\frac{S_n}{(n - ddof)}} \tag{2.4}
\]

**Welford’s method** can be used distributed and merged without loss of confidence to address the difficulty of dealing with large data sets to compute the statistical dispersion (SCHUBERT; GERTZ, 2018). Furthermore, the authors demonstrated the numerical stability of the calculation of statistical metrics using online algorithms and merge equations for the mean (2.5) and the sum of deviation products (2.6) of the data set partitions previously split \(A\) and \(B\), i.e. disjoint partitions of samples.

\[
\mu_{AB} = \mu_A + \frac{n_B}{n_{AB}} \times (\mu_B - \mu_A) \tag{2.5}
\]

\[
S_{AB} = S_A + S_B + \frac{n_A \times n_B}{n_{AB}} \times (\mu_B - \mu_A)^2 \tag{2.6}
\]

The equations shown here will be used in our solutions to perform some operations with the path weights when computing dispersion metrics for destination nodes in the multicast tree.
In this chapter, we examine each multicast topic associated with our contribution. First, we review multicast-related fundamental concepts. Then, we discuss Shortest Path Tree (SPT), one of the most widely used techniques to build a multicast tree, and K Shortest Path (KSP) algorithms, as they are used in our solution. Finally, we review the performance metrics for multicast networks.

3.1 MULTICAST COMMUNICATION

Unicast, broadcast, and multicast contrast among the various computer network communication schemes. The first associates the sender and destination in a one-to-one way. The second establishes one-to-all communication between a sender and all possible network destinations. Finally, the latter allows one-to-many communication, with the sender and destinations changing if necessary. In short, it is group communication. Although multicast was initially designed to optimize the number of transmissions, it is also applied to build other network group communication structures with different optimization requirements.

Predominantly, multicast is constructed by (i) the network forwarding and routing control or (ii) direct communication of the components of the application layer ignoring the network components, independently of the under-layer protocols (ALSAEED; AHMAD; HUSSAIN, 2018). The second option assumes that the network was built and cannot be modified to achieve the desired behavior, falling into the application of multicast planning for communication and data distribution. For example, Content Delivery Networks (CDN) uses this approach to synchronize or share its content between distributed servers, typically between core and edge servers. The P2P connections over the Internet are used to make a multicast in an overlay network. The first option, oppositely, with the recent paradigm change caused by programmability, in specific SDN, which allows the knowledge of network metrics to deploy and manage network devices such as switches and routers. It also allows deep packet flow processing (MERLING; LINDNER; MENTH, 2021), allowing different routing strategies to build multicast communication (ALSAEED; AHMAD; HUSSAIN, 2018).
Furthermore, a multicast group can be described as a destination group in a network that receives data through communication from the same origin, which can be represented as a tree where the sender \((\text{source})\) node is connected by only one path within the receiver nodes \((\text{destination})\). The origin, generally, is the root node or is connected to them. In opposition, destinations are intermediates, leaf nodes, or nodes connected to them.

There are some possibilities to build a tree for multicast communication using (a) source-based trees and (b) shared trees. The (a) construction process uses the shortest paths from the source to the destination nodes, normally classified as SPT, employs shortest path algorithms for examples using Shortest Path First (SPF) designed by Dijkstra (DIJKSTRA, 1959). For (b), any group node can participate in the construction as a reference, not only the source node, mainly because communication needs to be efficient in taking any node as the sender. A widely used strategy to build it is to use Steiner Minimal Tree (SMT), mainly when the union of sender and destinations is the totality of network nodes or the modeling graph. An alternative is to use Minimum Spanning Tree (MST), which also uses a group-driven solution and can be built with the Prim spanning tree algorithm (PRIM, 1957). Both types (a) and (b) can be addressed as the solution found for the Steiner Tree (ST) problem (ALSAEED; AHMAD; HUSSAIN, 2018), which is well known by its computational complexity (KARP, 1972), and for most formulations options is \(NP\)-complete. Unlike SPT and SMT, both have polynomial solutions (KARP, 1972), and a variety of heuristics are applied to find a solution for a wide variety of different topological arrangements of a graph. Figure 3.1 shows the comparison between multicast trees built with methods (a) and (b). Both have distinct values related to tree size, which is expressed as the cost for connecting the group \((C_T)\), while it has equal individual path sizes mean between sender and receiver nodes \((\mu_T)\).

Based on greedy algorithms Dijkstra and Prim to build a tree Shaikh, Lu & Shin (1995) that brings the Localized Multicast (LMC) to compute a minimal cost tree for a multicast group. As a benefit, their algorithm uses local information from adjacent

![Multicast Trees Comparison](image-url)
3.1 MULTICAST COMMUNICATION

nodes to evaluate the current node in the building solution, which allows a distributed approach to calculate the communication tree. In addition, LMC is directed to destination nodes to select the graph edge to be included in the solution. Preferably, it chooses the paths that cross destination nodes instead of nondestination nodes. In Shaikh & Shin (1997), they extended the previous work and renamed the algorithm *Destination-Driven MultiCasting* (DDMC) to show its characteristics. They examined DDMC in a simulated environment using many topologies and problem instances, seeing that DDMC delivered trees with a lower cost than SPT and performed better than many other heuristics to build SMT.

Zhang & Mouftah (2002) utilized the same approach as Shaikh, Lu & Shin (1995) to set up the minimal cost tree for a multicast group. They called the solution *Destination Driven Shortest Path Tree* (DDSPT), which works given preference to paths with destination nodes when there are multiple shortest paths for the currently evaluated multicast group node. Furthermore, DDSPT delivers trees at less cost than SPT using the *Dijkstra* shortest path algorithm.

Further ahead, Zhang & Mouftah (2006) faced the source-based tree construction in a two-step method. They use DDSPT to precompute a tree and then call a custom procedure to select an alternative parent chain for each multicast group node. They relaxed the parent choosing routine to select parents with its distance to the source larger than the shortest path, increasing it in maximum $H$ extra hops. They call it DDSP-H, where $H$ is the maximum number of extra hops a member can connect to the source to decrease its distance from other destination nodes. Although the generated tree has paths larger than SPT, it brings trees with lower total cost.

In contrast, *Early Branching Shortest Path Tree* (EBSPT) (Humernbrum; Hagedorn; Gorlatch, 2016) based on DDSPT uses the inverse logic to evict paths with a destination node on the way. This strategy creates multicast trees with forks close to the root. Furthermore, Humernbrum, Hagedorn & Gorlatch (2016) achieved a multicast tree with fewer rules on its deployment on the SDN enable devices and called it *Branch-Aware Modification* (BAM). Thus, they optimized the amount of computational resources used in the *OpenFlow* enabled switches, which is an important research topic (Nallusamy; Saravanen; Krishnan, 2021).

Centered in SDN paradigm, Biswas & Wu (2021a) explored the programmability of the network in a data center to mitigate link congestion, dealing with regular and anomalous congestion, and used *Dijkstra* SPF to redirect flows to an uncongested path. In addition, they formulated a multiple-flow redirection problem to consider one congested link, which is *NP-hard*, and proved a grouping-based heuristic that was compared to EBSPT (Humernbrum; Hagedorn; Gorlatch, 2016) and DDSPT (Zhang; Mouftah, 2002). In Biswas & Wu (2021b), they utilized programmability to group flows destined for the same place and use one forwarding rule by grouping, which decreased the number of forwarding rules in programmable network switches. However, they had not focused on the amount of packet reduction. Instead, they reduced the computational resource used in the network nodes.

Kotachi et al. (2019) considered many multicast routing groups and sessions on a SDN,
then proposed to reuse the rules for the same destinations even with different group sessions intended to minimize the use of the flow table. Approaching the proposed model as a Integer Linear Programming (ILP) problem, they compared it with other multicast tree construction solutions EBSPT and DDSPT with late application of the algorithm BAM, including two benchmark models called Minimizing Flow entries of Each Multicast demand (MF-EM) and Minimizing Unicast entries of All Multicast demands (MU-AM). Although their comparison does not consider dynamic scenarios or different networks, mainly because they focused on the National Science Foundation (NSF) reference topology in the simulation, the proposed model Minimizing total Flow entries of All Multicast demands (MF-AM) reduced total flow entries much more than other solutions.

Sun et al. (2021) applied the multicast scheme to build a communication tree with the chaining of services at computational nodes with the support of NFV, promoting the relief of demand for edge services. They merged MEC and SDN in the case scenario to allow custom configuration and deployment of the distribution tree. Although they used multicast group communication to formulate the model, their final communication was intended to be unicast. Their results indicated that the solution has a lower overhead for control messages and network signaling.

Facing Multicast-oriented Virtual Network Function Placement (MVNFP), known as NP-hard, on a SDN infrastructure with support of NFV and MEC, Wang et al. (2021) routed traffic and allocated network functions at nodes in a two-mode approach. Initially, they grouped the users and built the routing tree using SPF. Lastly, to place Virtual Network Functions (VNFs), they used the previously calculated tree to find the nodes with available resources and compliance with requirements iteratively, resulting in Novel Estimation of Distribution Algorithm (nEDA). Although nEDA has considerable time complexity, its two-phase heuristic performs better than many other algorithms in terms of average cost and mean execution time. However, they do not evaluate it in dynamic conditions and user experience metrics.

Despite the overlap of protocol layers in multicast communication for overlay networks, Eager, Escolar & Calero (2020) debated its efficiency and pointed to the necessity of specific protocols. In addition, they evaluated the overlapping ineffectiveness as unnecessary extras data to allow communication, which generally has data duplicity in some layers. They analyzed many multicast protocols in a Multi Tenant architecture. They found that ineffectiveness can reach 176%, even when using GRE, a well-known market solution that produces 72% overloading.

Zhang, Humernbrum & Gorlatch (2020) proposed a plugin named Plug-in Multicast Framework (PiMF) to manage multicast sessions on a SDN infrastructure. Their extension improved the dynamic usage of the multicast service and maintained compatibility with other communication frameworks. They showed the possibility of simultaneously operating different multicast services to meet QoS requirements.

In Diab, Lee & Hefeeda (2020) is shown Oktopus an algorithm to deal with the multicast service chaining problem. Their solution calculates multicast distribution graphs that minimize the routing cost per multicast session in a two-phase approach. Furthermore, they evaluated Oktopus using Internet Service Provider (ISP) topologies and compared it to the tightest algorithm in the literature. Although Oktopus has a routing cost op-
timality gap in all evaluated topologies, it is faster than the optimal algorithm to bring the solution and increase the number of multicast sessions served.

To minimize latency in the synchronization of video content between data centers of various providers, Liu, Niu & Li (2016) introduced programmability to inter-domain networks, mainly to build communication paths for different application requirements. Their routing proposal distinguishes between delay-sensitive sessions and deals that give specific routes. In addition, they implemented the solution at the application layer to overcome any challenge related to the change in the ISP network infrastructure. Finally, even though they had not compared it with similar work, it was benchmarked with the shortest path routing.

3.2 SHORTEST PATH TREE

There are different approaches for computing SPT. These approaches differ in the way paths with identical costs are selected. The DDSPT (ZHANG; MOUFTAH, 2002) uses paths with the lowest distance from a destination node, and in opposition, the EBSPT (HUMERNBRUM; HAGEDORN; GORLATCH, 2016) selects paths with the longest distance from a destination node. Although the final tree produced by each approach has the same cost, metrics such as link stress, node stress, and total tree cost can be affected differently.

In this sense, if used as a building block for our proposed fairness relaxation to improve the tree level of compliance with delay-variation, each method would produce trees with different characteristics. Therefore, in Chapter 5, we assess the impact of these approaches when used in the first round of the multicast tree computation.

3.3 K SHORTEST PATH

The KSP algorithms are used recurrently to build alternative tree versions connecting destination nodes with different path characteristics. These algorithms solve the KSP problem, a generalization of the shortest path problem. Specifically, the algorithms must output the number of paths requested (represented by letter \( k \)) for each destination in increasing order of distance from a source if it exists.

Many of the KSP algorithms are typically characterized in two families, delimited according to the probable occurrence or not of a loop in the path calculation, also known as k-shortest-path with or without any cycle restriction.

Figure 3.2 shows a graph and its simple paths alternatives calculated (loopless or also loop free) from source node \( s \) to destination \( r_1 \) using the Yen’s algorithm (YEN, 1971). Alternatively, Figure 3.3 depicts the graph and its alternative computed paths with cyles (loopy) using Recursive Enumeration Algorithm (REA) (JIMÉNEZ; MARZAL, 1999). Both figures have the \( k \) set to 3 during the algorithms running, the path computed by Yen’s algorithm and REA are identical for \( k \) equal to 1 following the order \( s \rightarrow v_1 \rightarrow v_2 \rightarrow v_8 \rightarrow r_1 \) (Figures 3.2a and 3.3a). However, when \( k \) is equal to 2 the extra path provides by Yen’s algorithm is \( s \rightarrow v_1 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_7 \rightarrow v_8 \rightarrow r_1 \) (Figure 3.2b) and REA output the cycled path \( s \rightarrow v_1 \rightarrow v_2 \rightarrow v_8 \rightarrow v_7 \rightarrow v_8 \rightarrow r_1 \) (Figure 3.3b), with
(a) First path from $s$ to $r_1$
length = 4

(b) Second path from $s$ to $r_1$
length = 7

(c) Third path from $s$ to $r_1$
length = 7

Figure 3.2: Loopless shortest paths $k = 3$
Figure 3.3: Loopy shortest paths $k = 3$
length 7 and 6, respectively. Then, when \( k \) receives the value of 3 the *Yen’s algorithm* additionally computed the path \( s \rightarrow v_1 \rightarrow v_3 \rightarrow v_4 \rightarrow v_6 \rightarrow v_7 \rightarrow v_8 \rightarrow r_1 \) and REA the path \( s \rightarrow v_1 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_7 \rightarrow v_8 \rightarrow r_1 \) both with length 7 (Figures 3.2c and 3.3c).

Table 3.1 summarizes the well-known algorithms in the literature to compute the \( k \) alternatives path for each destination on a graph.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Computational Complexity</th>
<th>Characteristic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dreyfus’s algorithm</td>
<td>( O(k \times</td>
<td>V</td>
<td>^2) )</td>
</tr>
<tr>
<td>Yen’s algorithm</td>
<td>( O(k \times</td>
<td>V</td>
<td>\times (</td>
</tr>
<tr>
<td>Hershberger et al. algorithm</td>
<td>( O(k \times</td>
<td>V</td>
<td>\times (</td>
</tr>
<tr>
<td>Eppstein’s algorithm</td>
<td>( O(</td>
<td>E</td>
<td>+</td>
</tr>
<tr>
<td>Martins and Santos’ algorithm</td>
<td>( O(</td>
<td>E</td>
<td>+ k \times</td>
</tr>
<tr>
<td>Recursive Enumeration Algorithm</td>
<td>( O(</td>
<td>E</td>
<td>+ k \times</td>
</tr>
</tbody>
</table>

Table 3.1: \( k \) Shortest Path algorithms characteristics

### 3.4 PRIORITY QUEUE

Typically, numerous routing algorithms employ a specialized data abstraction called priority queue to control graph node manipulation and visit order during the path search. For example, the classical implementation of *Dijkstra*’s SPF algorithm uses an array to handle the ordering for operation tracking needed to represent a priority queue. This can be achieved by maintaining a sorted list or keeping the list of elements in any order. It still selected the appropriate element each time it was needed (KNUTH, 1998). However, the element selection policy returns the largest or the smallest item in the queue, which can also be characterized as largest-in-first-out or smallest-in-first-out. Depending on one of these two element recovery policies, it is specially named as the maximum or minimum priority queue, respectively. Usually, the largest and smallest decisions obey a test criterion of some element value, such as distance, cost, latency, etc.

A priority queue is formally defined as:

“an abstract data structure consisting of a set of items, each with a real-valued key” (LEEUWEN, 1991, p. 326).
In the context of routing algorithms to enable it to operate using a queue $Q$, the implementation of the priority queue should support the following operations at least (LEEUVEN, 1991; BRODAI, 1996; CORMEN et al., 2022):

**Initiation:** This function creates and returns a priority queue data structure $Q$ that is initially empty.

**Insertion:** Add an element in $Q$ while ensuring its position constrained by a key value.

**Extraction:** This operation finds and returns the element in $Q$ with either the highest or lowest key value. Sometimes, this operation is broken into two procedures, find-max (or find-min) and delete-max (or delete-min).

**Update:** To decrease or increase the priority of an element in the priority queue. It is essential to know the element’s location in $Q$ before performing this operation.

Generally, priority queues are implemented with arrays, linked lists, heap data structures, or binary search trees. However, each type of data structure has its characteristics when performing different operations to maintain the priority order of the elements during its manipulation. Although the space complexity required for the priority queue is linear for most implementations, it is slightly different for the time complexity, which can change from $O(1)$ as the minimum for some operations to at least $\Omega(n)$ for each deletion call during an extraction (BRODAI, 1996; CORMEN et al., 2022). Particularly for find-min, delete-min, insertion, and update operations, their implementation specificities and influences on performance are carefully analyzed in the literature. Furthermore, to provide consistent performance even in the worst-case scenario, the amortized analysis technique is used, ensuring an average performance of operations (TARJAN, 1985).

Currently, the lowest computational complexity to operate a priority queue is given by a particular type of heap data structure called *Fibonacci Heaps* (FREDMAN; TARJAN, 1987), which is $\Theta(1)$ to and has an amortized time of $O(\lg(n))$ for queue element deletion. However, the *Fibonacci Heaps* is known for its challenging practical implementation, although there is also a heap implementation that provides the time bounds of *Fibonacci Heaps* in the worst case for deletion of $O(\lg(n))$ called *Strict Fibonacci Heaps* (BRODAL; LAGOGIANNIS; TARJAN, 2012). Besides, *Fibonacci Heaps* delivers the best theoretical bound for priority queue operations and offers a threshold for overall queue improvement. For this reason, it was used to analyze the theoretical computational complexity of the algorithms shown in this work. Moreover, in practice, the most commonly used implementation of priority queues uses a regular heap with a binary tree. Consequently, to implement the proposed algorithms for the practical analysis, the module `heapq` (O’CONNOR, 2023) available in the *Python* language standard library was utilized to instantiate the priority queues. Using `heapq`, with minor adjustments, it was possible to employ a maximum priority queue for FSPT and a minimum priority queue for Narrow, which will be discussed in Chapter 5.
3.5 PERFORMANCE METRICS

In the literature, different and additional performance metrics have been used to assess the quality level of multicast networks (CHU et al., 2001; CHU et al., 2002; EL-SAYED; ROCA; MATHY, 2003; FAHMY; KWON, 2007; LIN, 2014).

The authors of (CHU et al., 2001) explored how the requirements of the Internet application can influence the design of multicast systems, especially in the context of audio and video conferencing applications. They extensively evaluated overlay network schemes considering performance metrics such as bandwidth, latency, resource usage, and normalized resource usage.

In (CHU et al., 2002), the author studied overlay networks and how to adapt them to meet application-level performance requirements. To evaluate the performance of the proposed protocol, they used application-level and network metrics such as latency, bandwidth, stress (link stress), resource usage, and protocol overhead.

In (EL-SAYED; ROCA; MATHY, 2003), the researchers discuss several proposals for building a group communication service. First, they classified several proposals from the literature into some categories based on the characteristics of the solutions. Then, they listed the performance metrics used, such as stress, resource usage, stretch, losses after failures, time to first package, and control overhead. Eventually, they concluded that there is no unique performance metric to classify some solutions as the best.

In (FAHMY; KWON, 2007), the authors evaluate several protocols for application-level overlay networks, including their performance penalty over router-level solutions. They provide some evaluation through their experimental data and simulations, quantifying and describing the performance penalty for multicast overlay trees using different performance metrics, such as overlay cost, link stress, and resource usage.

The researchers studied overlay multicast routing algorithms in (LIN, 2014), comparing the performance of some algorithms through a comparative simulation experiment that evaluates the metrics of latency, average latency, stress (link stress), resource usage, and overlay cost.

Some of these metrics are summarized in Table 3.2 mainly due to their excellent indication for multicast networks, even if the solution has a more specific objective, as used in the DVBMT problem.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Summary</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>The sum of total weights for each edge of the multicast tree.</td>
<td>Well-known</td>
</tr>
<tr>
<td>Latency</td>
<td>The end-to-end delay from the source to the receiving host, including all queueing times.</td>
<td>(CHU et al., 2001; CHU et al., 2002; LIN, 2014)</td>
</tr>
</tbody>
</table>
### Table 3.2: Multicast Networks Performance Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Summary</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average latency</td>
<td>The average delay from the source to the hosts in the communication group (represented by $\mu_T$).</td>
<td>(FAHMY; KWON, 2007; LIN, 2014)</td>
</tr>
<tr>
<td>Link stress</td>
<td>The number of times the same packet is transmitted over a physical link $\text{stress}(e)$.</td>
<td>(CHU et al., 2002; EL-SAYED; ROCA; MATHY, 2003; FAHMY; KWON, 2007; LIN, 2014)</td>
</tr>
<tr>
<td>Resource usage</td>
<td>Summarizes the cost and stress of each edge used to build the multicast network. It is defined by the formula $\sum_{e \in E_T} D(e) \times \text{stress}(e)$.</td>
<td>(CHU et al., 2001; CHU et al., 2002; EL-SAYED; ROCA; MATHY, 2003; FAHMY; KWON, 2007; LIN, 2014)</td>
</tr>
<tr>
<td>Overlay cost</td>
<td>The summation of the downward intermediates connecting devices $\text{hops}(p)$ for each overlay multicast path $p$ given by $\sum_{r \in R} \text{hops}(p_r)$.</td>
<td>(FAHMY; KWON, 2007; LIN, 2014)</td>
</tr>
<tr>
<td>Node stress</td>
<td>The number of times the same packet is transmitted over a commuting node $\text{stress}(v)$ (where $v$ represents the node in the path).</td>
<td>This work</td>
</tr>
</tbody>
</table>

In this work, the metric node stress was introduced to consider stressed routers and switches in the physical infrastructure. This gave us an essential estimate of how a node is used repetitively and its possibility of being overloaded, including the stress due to the implementation of the communication, such as the flow rules in SDN networks.
Chapter 4

MULTICASTING WITH DELAY AND DELAY VARIATION CONSTRAINTS

In this chapter, we discuss related works. First, we examine the literature on the problem DVBMT. Furthermore, we introduce network modeling. Then, we show the DVBMT problem definition and its parameters for assessing solution quality.

4.1 RELATED WORKS

Rouskas & Baldine (1996) were the first to analyze and formulate the DVBMT problem. They have proven its NP-Completeness and proposed Delay Variation Multicasting Algorithm (DVMA) to solve the problem near optimality. However, the drawback of DVMA is its high computational complexity, which is $O(k \times l \times |R| \times |V|^4)$. The DVMA is suitable for scenarios with a multicast group of little sizes, particularly when the nodal degree is considered medium. The DVMA’s parameters $k$ ($k$-shortest paths to the analyzed destination) and $l$ ($l$-shortest paths to each remaining destination to merge in the under-construction tree) are defined by the user to determine the maximum number of iterations to minimize the delay-variation.

The work in Kapoor & Raghavan (2000) extends the DVBMT problem by considering the multicast tree cost. The authors propose two dynamic programming algorithms to solve the DVBMT problem. The first optimizes the original problem Dynamic Program for Delay Variation Bound (DPDVB), and the second reduces the tree cost by relaxing its objective. Both have pseudo-polynomial time complexity and achieve faster execution times than DVMA. However, the solutions were evaluated in a static topology with predefined multicast groups. The analyzed topologies have a nodal degree of 3.5, but the DVMA parameters $k$ and $l$ were not limited in the comparison. Both algorithms are pseudo-polynomial with complexity bounded by $O(\Delta \times |E| \times |R| \times \delta)$. An increase in $\Delta$ can dramatically affect the execution time, mainly when the need to expand $\Delta$ comes with an increase in $k$ to find even more paths.

Wang & Hou (2000) categorized many studies related to multicast trees and the most different building requirements. In addition, they made many appointments about the computational complexity of solutions.
In Low & Lee (2000), the authors introduced a heuristic called Distributed Delay and Delay Variation Bounded Multicast (DDVBM) to solve the DVBMT problem and minimize the final tree cost. DDVBM is based on a distributed approach, and its computational complexity is $O(|V|^3)$. However, the authors did not look at the intermediate nodes for each path as a possible destination node, which impacts the route’s usage due to some forwarded messages coming across some destination nodes many times without delivery.

The authors of Sheu & Chen (2001) propose a different solution for the problem DVBMT that focuses on the concept Core-Based Trees (CBT). It produces trees with a central node (core node) as the multicast point of equilibrium or synchronization in each final tree. The solution named Delay and Delay Variation Constraint Algorithm (DDVCA) has a complexity equal to $O(|R|\times|V|^2)$. An intrinsic characteristic of solutions based on this approach is that, in some cases, it could have a superposition of paths reaching the central node. This causes an unexpected delay variation on a forwarded packet in the final multicast topology constructed. Moreover, it can affect the expected delay for some destination nodes (HARUTYUNYAN; TERZIAN, 2018c).

Shi & Turner (2002) focused on the DVBMT problem in the construction of multicast trees for overlay networks. In this problem variation, all nodes have connectivity to each destination node, which is challenging due to the search for the tree with the lowest cost and the used bandwidth. They proposed a heuristic to build a minimal spanning tree based on the Prim algorithm. They showed different variations of the heuristic with computational complexities of $O(|V|^3)$, $O(|V|^2 \times \log |V|)$, and $O(K^2 \times |V|^2)$, respectively. The last heuristic uses a parameter $k$ to add the remaining destinations, outside the initially computed tree, to the final tree. The solution is applied to minimize the bandwidth used at each node and, at the same time, minimize the tree cost. This design option allows the usage of the remaining bandwidth in the overlay nodes for other multicast instances.

In Banik, Radhakrishnan & Sekharan (2007), the authors proposed the algorithm Chains. To the best of our knowledge, Chains has the lowest computational complexity in the literature compared to near-optimal solutions such as DVMA, DPDVB, and DDVCA. The solution is applied to overlay networks in special, mainly because the final solution may generate a non-tree solution. The central idea of the algorithm is to find the best $k$ shortest paths for each destination node in the topology and to create a table with those paths ordered by distance. After that, the algorithm looks for the best path sequence with the lowest delay-variation between the destinations. Finally, Chains uses REA (JIMÉENEZ; MARZAL, 1999) to search for $k$ alternatives. Taken together, the complexity of Chains is $O(|E| + |V| \times k \times \log(\frac{|V|}{k}) + |V|^2 \times k)$, to dense-graphs the complexity is $O(|V|^2 \times k)$. An important aspect evaluated in (BANIK; RADHAKRISHNAN; SEKHARAN, 2007) is the change in the multicast group and the entrance and exit of a node. The authors showed that Chains does not have the same execution time of a new instance of the problem as many other solutions in the literature because the changes update the table from a specific and direct execution of REA and its merging operation based on delay-variation ($\delta$).

The study in Zhang et al. (2010) tackles the DVBMT problem in the resource al-
location scenario for virtual overlay networks. In addition to latency requisites, they consider the resource cost of each node in the solution. The proposed algorithm is called VMNDDVCM Mapping algorithm (VMNDDVCM). It uses $k$-paths to compute the input paths and a sliding window to search for the latency requirement. The algorithm also checks if the candidate $i$-th shortest path to a destination does not overcome the remaining resource. The total time complexity is $O(\delta \times |R|^2 \times \Delta)$. The authors benchmarked the algorithm against DPDVB because both minimize the multicast tree cost (the total underlying resources such as CPU and bandwidth) and delay constraints. The metrics evaluated were the execution time, blocking, and serving of new virtual network instances, and the number of overlay networks served. VMNDDVCM performed better than DPDVB in metrics.

The authors of Kabat, Patel & Tripathy (2010) proposed the Economic Delay and Delay-Variation Bounded Multicast (EDVBM) algorithm for the overlay networks. The algorithm reduces the computational complexity compared to their previous proposal (Zhang et al., 2010). The EDVBM is based on the distance vector algorithms to compute the best routes. It could be carried out in a distributed approach with $O(|V|^3)$, or in a centralized way with $O(l_{max} \times |V| \times \lg(|V|))$ where $l_{max}$ is the maximum graph nodal degree.

The Delay and Delay Variation Constraint Multicast Algorithm (DDCMA) was developed by Aissa, Mnaouer & Belghith (2011) developed the DDCMA to solve the DVBMT problem in heterogeneous networks, including more latency restrictions and challenges correlated to the incorporated link structures. The DDCMA deals with the problem using a solution inspired by SPF and CBT. The time complexity is generally bounded by $O(|E| \times |V|)$, which is better than other solutions given the specific details of heterogeneous networks.

The study in Chen et al. (2011) deals with the problem DVBMT and the minimal assignment of servers. The servers are selected from a dedicated infrastructure and communicate in client-server or P2P models. The servers also host game sessions and are the contact point of their clients. The authors presented a new model and problem formulation and introduced an algorithm for each communication model. The Chains (Banik; Radhakrishnan; Sekharan, 2007) algorithm was used as a framework to select paths and paths chains to use after the selection of the server. The tree-building process called Server Selection Problems with Delay Constraints and Delay Variation Reduction (SPDVR) has a time complexity of $O(|R|^2 \times |V|^2)$ for the client-server version and $O(|R|^2 \times |V|^2)$ in the P2P version, which may still have the complexity affected by $\Delta$.

In Sekharan, Banik & Radhakrishnan (2011), the same authors of Chains (Banik; Radhakrishnan; Sekharan, 2007) brought the technique to deal with multicast networks when they have the Heterogeneous Postal Delivery model. That network has efficiency requirements for ordering messages delivered within the multicast network to meet latency requirements. At the same time, the order affects the final delay experienced by each destination. The algorithm shown has the complexity of $O(n^{5/2})$. However, the algorithm does not deal with delay-variation.

Belghith, Aissa & Mnaouer (2012) deepened their work on heterogeneous networks. However, the proposed algorithm Multi-Constrained Multicast Algorithm (MCMA) fo-
cused on parameterizing the delay-variation rather than minimizing it. They keep the core concepts based on shortest-path, SPF from Dijkstra and CBT. The MCMA maintains the overall time complexity of $O(|E| \times |V|)$. The comparison has shown better results than other compared algorithms from delay and delay-variation references with behavior close to the proposal from (ZHANG et al., 2010).

The authors in Lin (2014) classified the literature on multicast overlay networks. Categories were defined according to the model used in the routing structure. They compared the main algorithms in each category using a simulated environment. They looked at its behavior related to performance metrics, including latency, link stress (link level of usage by destinations), and overlay cost. The algorithms evaluated were OMNI (BANERJEE et al., 2006), Chains (BANIK; RADHAKRISHNAN; SEKHARAN, 2007), and Overcast (JANNOTTI et al., 2000). Although Chains performed slightly better for delay-variation metrics, it had the worst overlay cost and stress levels. Moreover, it performs unsatisfactorily concerning other delay aspects, such as mean and maximum latency.

The study in Chen et al. (2015) introduced a new ILP modeling aiming to place interactive application sessions on client and server components connected by a Service Overlay Network (SON). The modeling considered a scenario segmented by latency, such as the communications from the application server to SON root server, from SON root server to the SON contact server, and from SON closest contact server to its clients. The objective is to minimize the number of SON servers assigned to the overlay service, ensuring minimal delay and delay-variation. The authors introduced two algorithms to solve different problems in different scopes. The first, named SON-D, considers the maximum delay for each destination, and the proposed algorithm was named D-H. The second formulation, called SON-DDV, considers delay-variation, and its proposed solution was DDV-H. As a result, the computational complexity is $O(|V|^3 \times |R|)$ and $O(|V|^4 \times |R|^2)$, for algorithms D-H and DDV-H, respectively.

The studies Rhodes & Banik (2015), Rhodes & Banik (2016) benchmarked some algorithms for the DVBMT problem at an overlay multicast network prototype in the PlanetLab testbed. The starting point was the algorithm Chains, and they compared its performance with other algorithms in the literature. Chains showed a better performance ratio in delay and delay-variation metrics for members of the multicast group. On the other hand, the authors demonstrated that the delay metrics optimization in virtual environments does not guarantee a generally good performance, mainly because queueing delay in each path node has a lousy drawback. This influences the final latency perceived by the destination nodes. The authors suggested as the source of the problem how Chains deals with alternative paths. It prefers paths with lower latency, even if it carries some nodes repetitively in the procedure.

In Harutyunyan & Terzian (2016), the authors proposed a method based on a central node to build the multicast tree with all the constraints of the DVBMT problem. The approach’s most different point was the central node selection procedure, which tries to fix the difference between the computed and the perceived delay and delay-variation. They released the algorithms Directional Core Selection (DCS) and Directional Core Selection Build Lower Variation Tree (DCSBLVT), and the second has a routine derived
4.1 RELATED WORKS

from K Shortest Path from Yen. Those algorithms have the computational complexity of \(O(|V|^3 \times \lg(|V|))\) and \(O(|R| \times k \times |V| \times |E| \times \lg(|V|))\) in the order mentioned previously.

The researchers explored in Harutyunyan & Terzian (2018b) their previous work (HARUTYUNYAN; TERZIAN, 2016) utilizing a multi-core perspective to improve the options of paths to achieve the delay and delay-variation requirements. However, it builds a more resilient tree without unique node dependency. The algorithm labeled Multi-Core Delay Variation Bound Multicast Trees (MCDVBM) showed better results in terms of delay metrics compared to a mono-core approach. However, it performs worse in final tree cost and end-to-end latency, mainly because longer paths are selected. The total computational complexity of MCDVBM is \(O(|V|^3 \times \lg(|V|))\).

In Harutyunyan & Terzian (2018c), the authors extrapolate their prior work (HARUTYUNYAN; TERZIAN, 2016) to consider the dynamicity in joining and exiting of a multicast group. The approach improves computational performance, but in the worst case, it has the same computational complexity of DCS and DCSBLVT. The authors show that the worst case is a little portion of instances with the percentage of 3.4% and 2.8% to each algorithm variation. In the subsequent work in (HARUTYUNYAN; TERZIAN, 2018a), the authors brought the multicast group change analysis to their multi-core version MCDVBM (HARUTYUNYAN; TERZIAN, 2018b). The re-execution of the algorithm has the worst case only for 5.2% of the instances. However, generated trees suffer from re-execution for more than 50%. This behavior can drastically impact the algorithm’s performance when used in other topologies that the authors did not evaluate.

In Andrus et al. (2018), the authors reformulated the DVBMT problem. Instead of \(\Delta\) and \(\delta\), the authors used a latency interval to define the problem as Interval Multicast Subgraph (IMS). They argued that from an interactive application perspective, the path works properly with latency within an expected range. The algorithm called Interval Multicast Algorithm (IMA) has a fixed approach for paths, trying to change it until its latency reaches within the range. The algorithm computes a base tree using SPF and fixes some destination nodes’ paths using Depth-first search (DFS). The overall computational complexity is \(O(|R| \times \frac{|(|V|-1)|}{k})\).

The authors in (SEMONG et al., 2018) evaluate the multicast tree construction process in SDN to minimize latency and tree cost (amount of links used and its related weight). Furthermore, they present the scheme Delay Bounded Multi-Source (DBMS), which uses multiple sources to build the multicast tree, and their approach reduces the overload at the source node and the bandwidth usage. The overall computational complexity is \(O((|V| + 1)^4)\).

Bachmann, Bauer & Heseding (2019) dealt with latency constraints in the multicast tree building procedure using time synchronization for destination delivery in a SDN network, and they named it TopoSync. They showed an ILP to allocate the multicast session taking into account the bandwidth, latency, and path options, among other aspects. After all, they gauged the level of synchronization with the delay-variation achieved and expected when using SPF. Although the modeling shown had been effective for fixed topologies, they did not carry it out to scenarios with large multicast groups and alternative topologies. Bachmann et al. (2020) extended the previous work to complement it with programmable function placement and chaining infrastructure. Their work also
prototyped a SDN solution and compared it with SPF.

Ren et al. (2020) focused on chaining services in a programmable infrastructure using the technologies MEC, NFV, and SDN. Nevertheless, they serve it considering the expected delay of NFV and multicast sessions. They proposed the allocation of resources, disciplining the intermediate nodes, and maximizing the network *throughput*. They evaluated it in a *testbed*.

Table 4.1 recaps the algorithms, focusing on their computational time complexity and strategies to solve the DVBMT problem.

Table 4.1: Algorithms for DVBMT problem and its computational complexity

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Computational Complexity</th>
<th>Characteristic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVMA</td>
<td>$O(k \times l \times</td>
<td>R</td>
<td>\times</td>
</tr>
<tr>
<td>DPDVB</td>
<td>$O(\Delta \times</td>
<td>E</td>
<td>\times</td>
</tr>
<tr>
<td>DDVBM</td>
<td>$O(</td>
<td>V</td>
<td>^3)$</td>
</tr>
<tr>
<td>DDVCA</td>
<td>$O(</td>
<td>R</td>
<td>\times</td>
</tr>
<tr>
<td><em>Chains</em></td>
<td>$O(</td>
<td>E</td>
<td>+</td>
</tr>
<tr>
<td>VMNDDVCM</td>
<td>$O(\delta \times</td>
<td>R</td>
<td>^2 \times \Delta)$</td>
</tr>
<tr>
<td>EDVBM</td>
<td>$O(l_{max} \times</td>
<td>V</td>
<td>\times \log(</td>
</tr>
<tr>
<td>DDCMA</td>
<td>$O(</td>
<td>E</td>
<td>\times</td>
</tr>
<tr>
<td>MCMA</td>
<td>$O(</td>
<td>E</td>
<td>\times</td>
</tr>
<tr>
<td>DCSBLVT</td>
<td>$O(</td>
<td>R</td>
<td>\times k \times</td>
</tr>
</tbody>
</table>
4.2 NETWORK MODEL

Table 4.1: Algorithms for DVBMT problem and its computational complexity

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Computational Complexity</th>
<th>Characteristic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCDVBMT</td>
<td>$O(</td>
<td>V</td>
<td>^3 \times \lg(</td>
</tr>
<tr>
<td>IMA</td>
<td>$O(</td>
<td>R</td>
<td>\times (</td>
</tr>
<tr>
<td>DBMS</td>
<td>$O((</td>
<td>V</td>
<td>+1)^4)$</td>
</tr>
</tbody>
</table>

4.2 NETWORK MODEL

The multicast routing scheme for DIA applications must construct a distribution tree where all destination paths respect a given latency and latency-variation operational threshold. For this, the network is modeled as a weighted directed graph $G = (V, E)$, where $V$ represents the set of nodes, such as switches and routers, and $E$ designates the set of edges, which corresponds to the communication links that connect these nodes, such as wired or wireless links. There is a link delay function that maps each edge $e$ in $E$ to a non-negative weight $D : E \to \mathbb{R}^+$, and this value of $D(e)$ is a measure of the total delay that packets will have to cross this link. For multicast communication to occur, a sender node $s \in V$ and a group of destination nodes $R \subseteq (V - \{s\})$ are considered, all data traffic is transmitted through a multicast tree $T = (V_T, E_T)$ from the $s$ to $R$ (and vice versa), where $T$ spans the sender node $s$ and all destination nodes in $R$. The node $s$ is the root of $T$, and since $T$ is a subgraph of $G$, $V_T \subseteq V$ and $E_T \subseteq E$, it is possible that some nodes in $V_T$ will not be either $s$ or belong to $R$. In this case, this node has the forwarding function in $T$, and is $(V_T - (R \cup \{s\})) \subseteq V$. A path $P_T(s, v)$ in $T$ represents a unique path from $s$ to a node $v \in V_T$. The total delay experienced by a packet traversing a path is equal to $W(s, v) = \sum_{e \in P_T(s, v)} D(e)$. The following two crucial parameters are also introduced for the DVBMT problem. They are responsible for ensuring the service level needed by the multicast instance of the application related to the maximum delay and delay-variation among the paths selected for destinations.

**Source-destination delay ($\Delta$):** is a constraint that represents the upper bound for the end-to-end delay for which the path between the source and each destination does not overpass to keep the communication guarantees.

**Inter-destination delay variation ($\delta$):** is a constraint that defines the maximum difference between the end-to-end delay of any pair of paths connecting the source to a destination. It could be described as the level of synchronization between the receivers in the multicast group.
4.3 PROBLEM DEFINITION

The description of DVBMT below reproduces the problem as formulated by Rouskas and Baldine (ROUSKAS; BALDINE, 1996). There is some simplification for a better deep dive. The DVBMT problem is defined as follows:

**Definition 1 (DVBMT).** Let there be a graph $G = (V, E)$, a weight $D(e) \in \mathbb{R}^+$ for each $e \in E$, a subset $R \subseteq V$, and two positive integers bound $\delta$ and $\Delta$. Find a subtree $T$ of $G$ with a root $s | s \in V$ and that includes all vertices $R$, and such that for each $r \in R$ the total path weight to reach $r$ described as the path delay $W(s, r)$ is not greater than $\Delta$ and a delay-variation lower or equal to $\delta$.

A feasible solution for this problem returns a tree that complies with both constraints (4.1) and (4.2). However, to evaluate whether a solution given by a tree $T$ reaches the QoS required by the application, the constraint $\delta_T$ is used as shown in (4.3). Usually, the amount $\delta_T$ is called maximum inter-destination delay variation.

$$W(s, r) \leq \Delta, \forall r \in R$$ (4.1)

$$\left|W(s, r_i) - W(s, r_j)\right| \leq \delta, \forall r_i, r_j \in R$$ (4.2)

$$\delta_T = \max_{r_i, r_j \in R} \left|W(s, r_i) - W(s, r_j)\right|$$ (4.3)

In this work, we utilize an additional criterion $\sigma_T$ (4.4) in conjunction with $\delta_T$. This measure is comparable to the one used by (LEE; YOUN, 2000; BELGHITH; AISSA; MNAOUIER, 2012), but it treats it as a group dispersion rather than a deviation from the average group delay. Thus, we refer to $\sigma_T$ as inter-destination delay dispersion.

$$\sigma_T = \sqrt{\frac{\sum_{r \in R}(W(s, r) - \bar{W})^2}{|R|}}$$ (4.4)

The parameter $\sigma_T$ estimates the adequacy of the multicast tree as a solution to the DVBMT problem since $\sigma_T$ shows a dispersion around the delay for all nodes in $R$, which translates an equalization sense between members. Although computing $\sigma_T$ involves some steps related to the size of $R$, it needs to compute this dispersion without an enormous number of operations and without penalizing the algorithm’s complexity. Here, we introduce an incremental approach with all complementary functions to calculate it in Section 5.2.
Our contributions to this work are divided into two main sections. First, in Section 5.3, we show the FSPT algorithm and its relaxation used in the multicast tree-building process to carry the equalization level to SPF. Our approach does not aim to be an exact solution but is relatively fast and straightforward as Dijkstra SPT algorithm. This characteristic brings this solution to any topology, number of nodes, and edges. Similarly to tree adequation from Zhang & Mouftah (2006), the algorithm precomputes the multicast tree using any SPF-based algorithm. After that, without affecting the computational complexity, we compute the standard deviation for all destination path weights used in the solution and try to find alternative paths to connect some destination or tree branches with a more close delay-variation for the remaining group members.

Second, in Section 5.4, we solve the DVBMT problem with a heuristic solution using our online path evaluation method to choose paths with a lower dispersion between multicast group members, even in comparison to Chains known as the tightest delay variation with the minimum execution time in the literature.

Before discussing the proposed algorithms, the following two sections show the requirements for computing and measuring multicast routing metrics through the runtime dispersion computation for the delay and delay-variation constraints.

5.1 FAIRNESS MEASURE

As discussed in Section 2.3, the authors in Brun, Safaei & Boustead (2006a) define the experimental measure of fairness as the standard deviation of the normalized players’ scores when it is closer to 0 being fairer.

This study uses standard deviation to assess the fairness level of performance metrics. The fairness criteria for DIA often include delay-originated network and systems metrics, like end-to-end latency and response time, as they directly impact players’ game match achievements. Therefore, as shown by Brun, Safaei & Boustead (2006a), Brun, Safaei & Boustead (2006c), response time and player scores follow a similar pattern, including when disturbed by any network inconsistency, and both can be used to estimate the users’
playability and score’ affecting. Therefore, to decrease the delay inequities in the network perspective among DIA users, a dynamic equalization using standard deviation was used to adjust the network routing based on the delay. The section hereafter introduces the technique’s requirements to work on an incremental and iterative approach, which squeezes the delay-variation analysis and enables the efficient computation of $\sigma_T$.

Furthermore, the following metrics were employed in this work to measure the impact on users’ perspectives when evaluating the routing changes’ effect on user performance and playability. However, they are currently utilized only for evaluation, as depicted in the analyses in Chapter 6.

**Maximum inter-destination response time**: The highest response time experienced among DIA users.

**Inter-destination response time dispersion**: The dispersion measure using the standard deviation of the response time experienced by all DIA users.

**Maximum inter-player scores**: The highest score achieved among DIA users.

**Inter-player scores dispersion**: The dispersion measure using the standard deviation of the scores achieved by all DIA users.

### 5.2 OPERATIONALIZATION OF STATISTICAL DISPERSION

To enable the custom relaxation after changing any adjacency in the tree, all used equations from the literature shown in Section 2.4 had been rearranged in a backward version $(n-1)$ as exhibited in (5.1) and (5.2), where the metrics without the umpteenth group element can be retrieved and then replaced with a different sample. Finally, an inversion was made to sample group disaggregation (5.3) and (5.4).

\[
\mu_{n-1} = \frac{n \times \mu_n - x_n}{n - 1} \quad (5.1)
\]

\[
S_{n-1} = S_n - ((x_n - \mu_n) \times (x_n - \mu_{n-1})) \quad (5.2)
\]

\[
\mu_A = \frac{\mu_{AB} \times n_{AB} - n_B \times \mu_B}{n_{AB} - n_A} \quad (5.3)
\]

\[
S_A = S_{AB} - S_B - \frac{n_A \times n_B}{n_{AB}} \times (\mu_B - \mu_A)^2 \quad (5.4)
\]

When a better path is identified, we use this strategy to perform the following operations with the weights of the destination path measures.

**Append**: Add a new path weight sample to the dispersion measure.

**Remove**: Delete a given path weight sample from the dispersion measure.

**Addition**: Sums a scalar value to the mean.
Subtraction: Deducts a scalar value from the mean.

Merge: Allows merging two partitions of a random variable. For example, it can merge two partitions of the path distance samples from the nodes in \( R \).

Split: Enables partitioning of random variables. For example, it can divide a partition of path distance samples from nodes in \( R \) into two smaller partitions.

It is essential to highlight that the algebraic operations above with a supplied constant were demonstrated and used recurrently (HOEHN; NIVEN, 1985; ACZÉL; PÁLES, 1988; PÁLES, 1991). Therefore, it will be used here to swap a parent edge weight in a partition of destination paths. This allows us to select the best combinations of routes for each group member’s communication that achieve a lower scattering in latency between them.

5.3 Fair Shortest Path Tree (FSPT)

In this section, we introduce our proposed approach called FSPT to solve the problem DVBMT. Its main contributions are the runtime dispersion measure for delay and delay-variation constraints and the relaxation in the multicast tree-building process, which brings the equalization to SPF. Although our approach may not provide an exact solution, it is designed to be fast and straightforward, much like the Dijkstra SPT algorithm. This makes it applicable to a wide range of topologies, as well as varying numbers of nodes and edges.

In summary, given a precomputed multicast tree, the FSPT idea is to calculate the standard deviation for all destination path weights used in the solution and then swap parents along the path from the farthest receivers until they reach the root node whenever it minimizes the distance dispersion between the group members since that does not violate any constraint.

Similarly to the tree adequation of (ZHANG; MOUFTAH, 2006), the algorithm precomputes the multicast tree using any algorithm based on SPF. After that, without affecting the computational complexity, we compute the standard deviation for all destination path weights used in the solution and try to find alternative paths to connect some destination or tree branches with a more close delay variation for the remaining group members.

The algorithm presented here includes a custom relaxation step that selects the optimal route to minimize the distance dispersion between the destination nodes. This step is performed at each visited node and is based predominantly on the Dijkstra SPT algorithm. Referred to as the “fair” relaxation, this method aims to reduce the differences in delay between receivers in a delay-sensitive multicast session, even if this increases the latency from some nodes to the source node. Importantly, this approach does not violate any latency limit \( \Delta \) for any destination node.

The FSPT algorithm is formally presented in Algorithm 1. The lines 1–3 initialize necessary variables and structures. Specifically, the variable \( T \) is initialized with a tree returned from variations of the SPT algorithm, as discussed in Section 3.2, while the dictionary \( U \) is initialized to control the nodes passing through the queue \( Q \).
Algorithm 1: Fair-Shortest-Path Tree Construction

Data: G, s, R, \( \Delta \), \( \delta \)

Result: \( T \)

1. \( T \leftarrow \text{treePreCalcAlg}(G, s, R) \);
2. \( st \leftarrow \text{runningStatistics}() \);
3. \( H, Q, U \leftarrow \emptyset \);
4. for \( r \in R \) do
   5. \( st_r \leftarrow \text{runningStatistics}() \);
   6. \( Q.push(W(s, r), r) \);
   7. \( st.append(W(s, r)) \);
   8. \( st_r.append(W(s, r)) \);
   9. \( h_r \leftarrow W(s, r) \);
10. end
11. while \( Q \neq \emptyset \) do
   12. \( d, i \leftarrow Q.pop() \);
   13. \( x_i \leftarrow W(s, i) \);
   14. \( \text{temp}_{st_i} \leftarrow st - st_i \);
   15. \( \sigma_T \leftarrow \text{st.std}() \);
   16. if \( \sigma_T \leq \delta \lor h_i > \Delta \) then \{ go to 31; \}
   17. for \( j \in G.neighbors(i) \) do
   18. \( x_j \leftarrow W(s, j) + D(e_{j,i}) \);
   19. \( h_j \leftarrow (h_i - x_i) + x_j \);
   20. \( \text{temp}_{st_j} \leftarrow (st_i - x_i) + x_j \);
   21. \( \text{temp}_{st} \leftarrow \text{temp}_{st_i} + \text{temp}_{st_j} \);
   22. \( \sigma_j \leftarrow \text{temp}_{st}.std() \);
   23. if \( j \notin U \land x_i \leq x_j \land h_j \leq \Delta \land \sigma_j < \sigma_T \) then
   24. \( W(s, i) \leftarrow x_j \);
   25. \( h_i \leftarrow h_j \);
   26. \( st_i \leftarrow \text{temp}_{st_j} \);
   27. \( st \leftarrow \text{temp}_{st} \);
   28. \( T.parent(i) \leftarrow j \);
29. end
30. end
31. \( U.push(i) \);
32. \( \text{parent} \leftarrow T.parent(i) \);
33. if \( st_{parent} = \emptyset \) then \{ \( st_{parent} \leftarrow \text{runningStatistics}() \); \}
34. \( st_{parent} \leftarrow st_{parent} + st_i \);
35. if \( h_{parent} = \emptyset \lor h_{parent} < h_i \) then \{ \( h_{parent} \leftarrow h_i \); \}
36. if \( \text{parent} \neq s \land \text{parent} \notin Q \) then \{ \( Q.push(W(s, \text{parent}), \text{parent}) \); \}
37. end
5.3 Fair Shortest Path Tree (FSPT)

Furthermore, from lines 4–10, the precomputed destination distances are used to initialize the maximum priority queue $Q$ (as discussed in Section 3.4), the global and local dispersion measures $st$ to compute the $\sigma$, and the destination path latency control structure $H$ containing a element variable $h_i$ for each member of the destination group.

Additionally, the outer loop between lines 11–37 selects a node $i$ with the highest delay (or distance) in the tree already calculated. Furthermore, within it, an inner loop between lines 17–30 defines the rule for selecting the parent node $j$ of the node $i$. The parent is changed whenever necessary, and all global metrics are updated (lines 23–29).

It is important to emphasize that the visit order of the node is made using the maximum priority queue $Q$, mainly because it allows recovery of the inserted node constrained by its distance from the sender. Each recovery operation ($pop$ on line 12) returns the node with the largest distance to $s$ and its delay. After evaluating possible new parents, it is placed in the queue with operation $push$ (line 36). However, before it, if the selected parent is not previously placed in the queue, its metrics are initialized and updated on lines 32–36.

The computational complexity of FSPT follows SPT. The complexity analysis is based on employing Fibonacci Heaps (FREDMAN; TARJAN, 1987) to operationalize the priority queue. The overall complexity of FSPT is $O(|E| + |V| \times \lg(|V|))$. In detail, the line 1 takes $O(|E| + |V| \times \lg(|V|))$, remains in the initiation steps until line 3 lasts $O(1)$. The lines 4–10 take $O(|R|)$ time. Inside the main loop, lines 17–30 take $O(|E|)$ time. Next, lines 31–36 take $O(1)$ except by line 36 which takes $O(\lg(|V|))$. Finally, the lines 11–37 take $O(|E| + |V| \times \lg(|V|))$ time duration, which implies that FSPT is $O(|E| + |V| \times \lg(|V|) + |R| + |E| + |V| \times \lg(|V|) + |V| \times \lg(|V|))$ which is $O(|E| + |V| \times \lg(|V|))$ due to $|R| \leq (|V| - 1)$.

Figure 5.1 exemplifies the trees calculated using different algorithms SPF and after trying to equalize the distances between the receivers using FSPT (all edges have weight 1, and the built tree is highlighted in yellow).

In particular, Figures 5.1a and 5.1b show the difference between the default SPT and FSPT, including lowering the tree metrics $\delta_T$ and $\sigma_T$. Next, in Figures 5.1c and 5.1d, there are no distinctions in the tree after the computation of FSPT (version tagged as Fair Destination Driven Shortest Path Tree (FDDSPT)) for this instance, precisely because DDSPT had chosen some paths that, in the event of a parent swap, their paths latencies will be affected in a chain reaction. Lastly, Figures 5.1e and 5.1f display differences using the EBSPT algorithm to compute the tree and its impact later with the routines FSPT, which was labeled Fair Early Branching Shortest Path Tree (FEBSPT).

Figure 5.2 shows off the FSPT steps from the base tree (Figure 5.1e) to the fairer relaxed tree (Figure 5.1f).

First, all dispersion metrics are computed for the source tree, and the queue is filled with all nodes in $R$, as shown in Figure 5.2a. Next, the node with the longest distance is visited (Figure 5.2b). Then, the edges evaluated can be swapped if there is any that makes the path’s distance dispersion lower. Otherwise, it follows only by inserting the selected parent node in the queue.

This visit by longest distance with parent queueing occurs node by node in the multicast tree without any modification, as shown in Figures 5.2c and 5.2d, until it reaches
Figure 5.1: FSPT trees compared to precomputing methods
Figure 5.2: FSPT Relaxation Step by Step precomputed by EBSPT
node $r_1$ in Figure 5.2e. Because node $r_1$ has a neighbor with a distance lower than or equal to its edge, which was not queued, the extra weight changing the parent to $v_3$ can be absorbed in the path without creating a cycle. The visit continues through the remaining nodes (Figures 5.2f and 5.2g) until it reaches the sender node $s$, as shown in Figure 5.2h. Finally, there are no more nodes in the tree to visit, and FSPT returns the relaxed tree achieved (as in Figure 5.1f).

### 5.4 NARROW

The algorithm shown in this section, named Narrow, had a different approach from the previous one. It aimed to provide a narrower solution than other algorithms for the DVBMT problem discussed in Chapter 4.

Briefly, the Narrow strategy consists in using a KSP algorithm to iteratively obtain the $k$th potential route from the root to the current closest destination. Next, it evaluates whether the dispersion metrics are minimized with path swapping. Then, whenever the dispersion decreases, it changes the path of the current analyzed destination in the under-construction solution.

Narrow uses REA, a KSP algorithm that calculates $k$ possible routes from a source to destinations dynamically and iteratively, first fetching the feasible paths with the shortest distances (see Section 3.3). Some paths selected for different destinations can overlap for these characteristics, generating a topology as a solution rather than a tree. Consequently, as Chains, Narrow is more suitable for overlay and programmable networks using data path virtualization, which virtually supports isolating each destination path. Also, Narrow employs the Welford’s algorithm to compute dispersion metrics online when evaluating the best path to merge in the final tree for a destination (see Chapter 5). Unlike Chains, which selects the paths with the shortest distances to each node in $R$ even when there is a possible optimization for delay-variation, Narrow uses the metric $\sigma_T$, which is more rigorous in dispersion estimation than $\delta_T$ used by Chains.

Narrow’s steps are shown in Algorithm 2. The lines 1–3 initialize some necessary variables and structures. Specifically, an iterative instance of REA is instantiated on line 1, followed by the creation of the dispersion metrics structure and the establishment of a priority queue. Subsequently, all necessary variables for each destination node $r$ in $R$ are initiated from 4 to 8, including the calculation of the first path $k = 1$ and its inclusion in the priority queue.

Next, the lines 9–11 set the window variables, which are copies of the previous variables, since it starts with the first values from the shortest path. Further down, on lines 12–39, the steps to verify the solution, obtain paths, and adjust the tree are executed incrementally. All steps consider window structures, replacing the current smallest distance destination path data with its next smallest path data.

In particular, the lines 13–15 evaluate the new calculated solution whenever it is below the specified threshold $\delta$. If so, the algorithm finishes the execution and returns the solution. Otherwise, there are two possibilities for proceeding.

First, from line 17 to 26, when there is no candidate path for a destination, the steps are followed to obtain a new path, getting the next path from REA for the current $r$. 
Next, it checks if the path found is below the constraints $\Delta$ and has no cycle (we design Narrow to discard paths with cycles to mimic Chains when dealing with node loops). In that case, it is turned into a candidate and finally pushed into the queue for evaluation in one of the next iterations.

Second, in the line interval 27–38, when there is a candidate path to incorporate, the algorithm computes the dispersion metrics for a window assessed and checks if it obtains a better solution in terms of $\delta$ (line 32). After that, Narrow updates the solution for the subsequent evaluation of the candidate tree. Finally, if there is no candidate path for the current destination evaluated and the algorithm sets the current distance and the path to None in the queue (line 37). After all, Narrow repeat the described loop routines.

The computational complexity of Narrow is $O(|E| + k \times |V| \times \log(|V|))$, assuming Fibonacci Heaps (FREDMAN; TARJAN, 1987) to implement the heap structures. In depth, line 1 takes $O((|E| + |V| \times \log(|V|) + |V|))$ to compute the shortest path and initialize the internal REA required variables. The remaining steps until reaching line 3 take $O(1)$. The lines 4–8 are bound by $O(|R| \times \log(|R|))$ to initialize Narrow destination-dependent basic structures. Next, lines 9–11 initiate the structures to analyze the path metrics before merging them to the solution. Inside the main loop, lines 13–15 are $O(1)$ to check the compliance of the tree. Lines 17–26 take $O((|E| + k \times |V| \times \log(|E|/|V|)) + (|E|) + (2 \times C))$ to obtain the $k$ paths to all destinations on the graph (the incorporated complexity analysis for REA is based on a general heap structure (CORMEN et al., 2022) as the authors had examined), to discard paths with node loops, and push them into the queue. Otherwise, lines 27–38 take $O(1)$ to manage stats and merge solutions. Finally, the main loop lines 12–39 is $O(|E| + k \times |V| \times \log(|E|/|V|))$ time complexity.

Figure 5.3 shows the narrow steps from the base solution (Figure 5.3a) within the shortest path to the final topology.

First, all dispersion metrics are calculated for the base solution, and the destination nodes are queued as depicted in Figure 5.3b. Second, the node with the smallest distance and within a new valid path is evaluated. The path swapping occurs if the dispersion can be reduced, as illustrated in Figure 5.3c. Next, the new topology is computed with each path considered independently, as shown in Figure 5.3d. This process is repeated for each destination node, ordered by its distance (in ascending order) from the source node. If the new path evaluation reduces the dispersion, then the path swapping occurs again, as depicted in Figure 5.3e. If no new valid path is evaluated or the limit of $k$ alternative paths is reached, the best solution is returned, as shown in Figure 5.3f.

To illustrate the Narrow approach, Figures 5.4a and 5.4b represent the differences using the Narrow algorithm and Chains. There is a contrast in the path used for $r_2$, mainly because Chains uses a sequence of shortest paths for each destination, called a chain. It does not look at the intermediate path in Chains. In opposition, Narrow chooses this middle path to improve the tree fairness. For instance, the Chains’s path length to $r_2$ is equal to 3 with path hop sequence following $s \rightarrow v_1 \rightarrow v_3 \rightarrow r_2$, but Narrow took the sequence $s \rightarrow v_1 \rightarrow v_3 \rightarrow v_4 \rightarrow r_2$ with length of 4.
Algorithm 2: Narrow Algorithm for DVBMT problem

Data: $G, s, R, \Delta, \delta$

Result: $T$

1. $rea \leftarrow REA(G, s, R, K)$;
2. $st \leftarrow runningStatistics();$
3. $Q \leftarrow \emptyset;$
4. for $r \in R$ do
5. \hspace{1em} $W_0(s, r), P_0(s, r) \leftarrow rea.getPaths(r);$\hfill (1.0)
6. \hspace{1em} $st.append(W(s, r));$
7. \hspace{1em} $Q.push(W(s, r), r, None);$\hfill (1.0)
8. end
9. $window_st \leftarrow st;$
10. $window_dists \leftarrow W_0;$
11. $window_paths \leftarrow P_0;$
12. while $Q \neq \emptyset$ do
13. \hspace{2em} if $st.std() \leq \delta$ then
14. \hspace{3em} break;
15. end
16. \hspace{2em} $d, r, p \leftarrow Q.pop();$
17. \hspace{2em} if $p == None$ then
18. \hspace{3em} $dist, path \leftarrow rea.getPaths(r);$\hfill (1.0)
19. \hspace{3em} if $p \wedge dist \leq \Delta$ then
20. \hspace{4em} if $dist > d \wedge isLoopFree(path)$ then
21. \hspace{5em} $Q.push(dist, r, path);$\hfill (1.0)
22. \hspace{5em} continue;
23. \hspace{4em} end
24. \hspace{3em} $Q.push(d, r, None);$\hfill (1.0)
25. \hspace{2em} end
26. \hspace{2em} else
27. \hspace{3em} $window_st.remove(W_i(s, r));$
28. \hspace{3em} $window_st.append(W(s, r));$
29. \hspace{3em} $window_paths(r) \leftarrow path; $
30. \hspace{3em} $window_dists(r) \leftarrow dist; $
31. \hspace{3em} if $window_st.std() < st.std()$ then
32. \hspace{4em} $st \leftarrow window_st;$
33. \hspace{4em} $W_i(s, r) \leftarrow window_dists(r);$
34. \hspace{4em} $P_i(s, r) \leftarrow window_paths(r);$\hfill (1.0)
35. \hspace{4em} end
36. \hspace{3em} end
37. \hspace{2em} end
38. end
5.4 NARROW

Figure 5.3: Narrow Step by Step to compute solution with $k = 2$
Figure 5.4: Narrow and Chains tree comparison for a generic topology

(a) Narrow

$\mu_T = 3.8$, $\sigma_T = 0.4$, $\delta_T = 1$

(b) Chains

$\mu_T = 3.6$, $\sigma_T = 0.49$, $\delta_T = 1$
This chapter will delve into the comprehensive analysis of numerous experiments conducted. We will provide a detailed account of each experiment and their corresponding outcomes. Firstly, in Section 6.1, describing the environment and its specification used for benchmarking, followed by the simulations outcomes and its parameters led to measure the behavior of the algorithms in Section 6.2, and lastly in Section 6.3 showing the driven emulations and its results with practical DIA scenario.

6.1 BENCHMARK OVERVIEW

Two experiments were conducted to evaluate the efficiency of the routing algorithm in relation to multicast routing metrics and its impact on DIA sessions. The first experiment involved benchmarking the routing algorithms in a simulated environment. All algorithms were implemented using the Python programming language (Python Software Foundation, 2001), and the network was modeled using the NetworkX\(^1\) graph library (HAGBERG; SCHULT; SWART, 2008). Once the network topology and problem instance (sender, receivers, and DVBMT parameters) were inputted, the simulation computed the metrics associated with the resulting multicast routing topology and exported them.

In the second experiment, a virtual infrastructure was created using programmable switches to model network nodes. The clients and servers of the DIA were connected to the sender and receiver switches, creating a site abstraction. This allowed DIA sessions to run on the network using a multicast topology that a routing algorithm had computed. The emulator architecture is depicted in Figure 6.1, which includes an emulated link with latency characteristics of the paths. Each DIA component triggers its FPS gaming component (client or server) during execution, and all perform the number of game matches stipulated.

This strategy of using an emulated environment to measure the impact of network conditions in the DIA session is close to that used by (BREDEL; FIDLER, 2010), which

\(^1\)\(<http://networkx.github.io/>\)
goes in the opposite direction of synthetic traffic, which is used recurrently in the literature (LANG; BRANCH; ARMITAGE, 2004). The approach involves using artificial players instead of real players to evaluate the game matches and the players’ performance in some unexpected network circumstances. As an application DIA, the FPS game *Quake III* was used, and as an artificial player, *Slugbot* (CREW, 2004), a famous client-side bot that chose its actions to maintain good scores in the game, was executed to operate each player on the *Quake III* client. Employing artificial players to evaluate QoE is helpful because it allows a more controlled environment to measure network conditions’ impact on the gaming session. Artificial players, such as Slugbot, are programmed to make decisions that maintain good scores in the game and can be used to simulate real players in unexpected network circumstances. This approach is more reliable than synthetic traffic, commonly used in the literature, as it provides a more accurate representation of the gaming experience and players’ achievements than only the traffic behavior along the matches.

The overlay representation of the network had been prototyped by using libraries from the open source network emulator Mininet\(^2\) (LANTZ; HELLER; MCKEOWN, 2010) that allows the creation of virtual networks with customizable topologies, link characteristics, switches, routers, and host configurations. The prototype has a custom routine configuring the network’s abstraction with the latency attributes returned by the routing algorithm, which is triggered differently in each run using any of the implementations selected in this study. The Mininet also helped to create personalized host classes and service configurations to run the game’s server and clients.

It should be reported that both experimental scenarios previously shown use random

\[^2\]<http://mininet.org/>
topologies generated by the Waxman graph modeling method (WAXMAN, 1988). All experiments were conducted on a server with an Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz, 192 GiB of DIMM DDR4, and CentOS GNU/Linux 7.

6.2 SIMULATION RESULTS

The weight of all edges, used as delay, had been set to 1 in all topologies executed, obtaining a behavior equivalent to the hop count. The first simulation setup has a mean network degree of 4. Its number of nodes is incremented by 10 in each generated topology, started from 20 to 300. The portion of random receivers had been specified as a percentage of the number of network vertices, and the values used were 10%, 15%, 20%, 25%, 30%, 40%, and 50%. Each setup combination had 1000 instances and was replicated 5 times to compute execution time-dependent metrics. For example, for the topology with 10 nodes and mean network degree of 4, the load of 10% was evaluated by testing 1000 different permutations of sender and destinations. This given input was used repeatedly for 5 times to determine the performance of each algorithm.

Starting with pre-computing method analysis, Figure 6.2 shows how the algorithms in the SPT family behave for the link stress metric. For the node stress metric, Figure 6.3 shows the difference between these algorithms, and Figure 6.4 shows the main difference in the summation of all edge costs in the tree produced. In all figures, the algorithms keep the percentage differences (also known as percentage change - %CH) stable when the number of general nodes and the destinations increases.

Taking into account the differences in the metrics performed by each precomputing method, the FSPT evaluation follows considering the restrictions $\Delta$ and $\delta$, which were assigned as $\infty$ ms and 0. Then, a random sender $s$ was also defined to be imputed in the algorithm call.

Figure 6.5 reveals the differences between FSPT and SPT. There is an increase in delay means in all versions of FSPT, but the variation with EBSPT, the behaviors as an upper bound, indicates that the delay-variation was more attacked as shown in Figure 6.6. In both metrics, FSPT loses power with network growth and receiver loading.

The second simulation scenario fixed the size of 150 nodes. Its degree is incremented by 1 in each generated topology, started from 4 to 12. The receivers had been designed as in the previous setup. Figure 6.7 depicts a concave behavior, with EBSPT exhibiting the most impacted mean. In contrast, Figure 6.8 shows a convex shape with EBSPT revealing the best improvement in delay-variation. Both figures show maximum and minimum when the degrees are 8 and 9.

Therefore, the network degrees 8 had been considered to validate the network size and the impact of receiver loading, and in Figures 6.9 and 6.10 the superiority of FSPT with EBSPT pre-computing the tree with lower $\sigma_T$. However, there is an impact on average latency.

With this evaluation, the FSPT variation using EBSPT to precompute the trees showed the best results in building a multicast with delay and delay-variation constraints. The intuition of this behavior is that trees with more spreading branches had disjoint paths (as exemplified in Figure 5.1e), thus it does not use the same parents, which does
not generate blocking in the parent iteration to change its uplink.

Figures 6.11 and 6.12 show the percentage change comparison between FSPT and Chains to maximize the variability of the delay and the execution time, respectively. However, with low and medium loads, FSPT shows a good improvement compared to SPT with a small increase in computational time. In opposition, Chains increases the gain close to optimal, but with a huge drawback in execution time. It should be noted that the execution time of Chains can worsen, mainly because the $k$ was limited to $|R|$ in this experiment.

Using the same topologies with degree 4, Figures 6.13, 6.14, and 6.15 show the comparison of the percentage change between Narrow and Chains to SPT in terms of maximum inter-destination delay variation $\delta_T$, However, the inter-destination response time dispersion, even as $\sigma_T$, shows the inter-destination delay dispersion $\sigma_T$, and execution time, respectively. Although Narrow shows the same optimization level for $\delta_T$, it improved the standard deviation between the delay variation of the paths and outperformed Chains in
6.2 SIMULATION RESULTS

Then, with the exact degree change as in the previous scenario with a fixed size of 150 nodes, Figure 6.16 shows the curve change behavior that affected the mean path size. In addition, Figure 6.17 demonstrates the deviation of the pattern, revealing the best improvement in delay-variation of Narrow compared to Chains. This gain shows an evident global minimum when the network degrees are equal to or below the degree 8.

Figure 6.3: Mean Node stress from s to R (%CH from SPT)
Figure 6.4: Mean Cost from s to R (%CH from SPT)
Figure 6.5: Mean Average latency from $s$ to $R$ (%CH from SPT)
Figure 6.6: Mean *Inter-destination delay dispersion* from $s$ to $R$ (%CH from SPT)
Figure 6.7: Mean Average latency from $s$ to $R$ (%CH from SPT)
Figure 6.8: Mean *Inter-destination delay dispersion* from $s$ to $R$ (%CH from SPT)
6.2 SIMULATION RESULTS

Figure 6.9: Mean *average latency* from *s* to *R* (%CH from SPT)
Figure 6.10: Mean *Inter-destination delay dispersion from s to R* (%CH from SPT)
Figure 6.11: Mean *Inter-destination delay dispersion* from $s$ to $R$ (%CH from SPT)
Figure 6.12: Mean Execution time from $s$ to $R$ (%CH from SPT)
Figure 6.13: Mean *Inter-destination delay dispersion* from $s$ to $R$ (%CH from SPT) with Narrow
Figure 6.14: Mean Maximum inter-destination delay variation from $s$ to $R$ (%CH from SPT) with Narrow
Figure 6.15: Mean Execution time from $s$ to $R$ (%CH from SPT) with Narrow
Figure 6.16: Mean Average latency from $s$ to $R$ (%CH from SPT) with Narrow
Figure 6.17: Mean *Inter-destination delay dispersion* from s to R (%CH from SPT) with *Narrow*. 

Mean Network Degree vs. Receivers (%): CHAINS, FEBSPT, NARROW
6.3 EMULATION RESULTS

This experiment investigates the fairness level using the normalized standard deviations of the final path response time found for all destination nodes. The theoretical evaluation used a simulated environment (as in the previous subsection), and the practical evaluation utilized the emulated environment. There was a selected topology with 150 nodes, degree 4, and a load of 10% receivers. All edges had been configured with a weight of 10 ms, a random sender was selected, and 500 random instances of destination nodes, each with $\delta_T$ of 50 ms. For practical setup, 5 matches were executed, each lasting 5 minutes. The $\Delta$ and $\delta$ were bound as 100 ms and 0.

Figures 6.18 and 6.19 compare the algorithms for practical execution with metrics measure between clients and server communication. It is possible to visualize that the maximum response time measure and $\delta_T$ optimization are lower with FSPT enforcement. However, the inter-destination response time dispersion, even as $\sigma_T$, shows the variation in the response time is squeezed after FSPT is applied in the tree. The measured response time dispersion is practically the same between FSPT and Chains.

Figure 6.18: Normalized mean *maximum inter-destination response time*

Figures 6.20, 6.21, and 6.22 show the balance in performance metrics with hypothetical execution receiving the same inputs for each pre-computing method. Although there are drawbacks and non-optimality of FSPT, it can give good results. It is possible to establish that FSPT could be selected in some scenarios, such as networks with restrictions on network stress level, tree cost, and routing algorithm execution time. At the same time, it has good levels of fairness. That is, it increases the equalization of users’ perceived
response time.

Figure 6.23 also displays the balance in performance metrics with hypothetical execution. Despite the Narrow increase in many performance metrics compared to others, they minimize the fairness levels, i.e., it increases the equalization of the perceived response time of users with the lowest value of $\sigma_T$. At the same time, they have an execution time of the same or lesser magnitude of the algorithm Chains.

It is important to note that when comparing FSPT family and Narrow with Chains, we considered all performance metrics using a virtual path (similar to a unicast connection in an overlay network) from $s$ to nodes in $R$ since Chains is used in overlay networks and does not deliver a tree in physical infrastructure. Therefore, some metrics, such as link stress, node stress, and resource usage, can be further optimized if FSPT is implemented without this virtual abstraction for the paths.

In addition, we investigated the level of fairness using the normalized standard deviations of the players’ scores after the matches, using only the emulated environment. Figures 6.24 and 6.25 show the comparison between the algorithms for practical execution. Visualize that maximum inter-player scores (same as $\delta_T$) are lower with FSPT and Chains, but Narrow is the lowest. However, the inter-player scores dispersion (like as $\sigma_T$) shows the variation in the response time is squeezed after applying FSPT to the tree. The measured inter-player scores dispersion is practically the same between FSPT and Chains, but in contrast, Narrow achieves the lowest value.

Thus, the tests revealed a significant difference between Narrow and other algorithms in the delivery of the multicast routing option driven by fairness, additionally Narrow
delivery lower delay and delay-variation tree. FSPT also performed well on the fairness measures and improved compliance with delay and delay-variation. Additionally, FSPT demonstrated a positive correlation between the optimization of fairness, delay, and delay-variation against the time complexity, achieving a good tradeoff against other algorithms.
6.3 EMULATION RESULTS

Table 6.22: Normalized mean of routing metrics (FSPT)

<table>
<thead>
<tr>
<th>Metric</th>
<th>CHAINS</th>
<th>FSPT</th>
<th>SPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum inter-destination delay variation</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>Link stress</td>
<td>0.8</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Node stress</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overlay cost</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource usage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average latency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inter-destination delay dispersion</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Figure 6.22: Normalized mean of routing metrics (FSPT)

Figure 6.23: Normalized mean of routing metrics (Narrow)
Figure 6.24: Normalized mean *maximum inter-player scores*

Figure 6.25: Normalized mean *inter-player scores dispersion*
CONCLUSION AND FUTURE WORK

This thesis covered various topics, starting with the context and theoretical background. Then, delving into issues related to the defined scenario, including the importance of edge computing to reduce latency and improve the performance of DIA, and exploring the network design needs to ensure playability and fairness in the DIA sessions. For instance, to ensure that a game is playable and that all players have an equal chance of succeeding. We have intensely explored the topic of multicasting, focusing on a deep survey in the literature of the delay and delay-variation and correlating it with our proposal to achieve fairness in routing. Finally, we are introducing our contribution to achieving fairer routing oriented to the application experience and analyzing its results in multiple network performance metrics.

This work suggested a new performance metric to measure the quality of multicast routing algorithms and introduced a new evaluation criterion for DVBMT solutions. It offered an algorithm to construct multicast routing trees that meet the latency requirements of interactive applications while improving fairness among users in the same communication group. We also proposed an algorithm to solve the DVBMT problem applied for overlay networks that meet latency requirements of the interactive application with a near-optimal fairness level. Both proposals, FSPT and Narrow algorithms, have the same principle of improving fairness by using a dynamic adjustment in the routing paths that equalizes the players’ connection characteristics. They also showed practicality in meeting interactive real-time application requirements and, in the study case, delivered matches with more equilibrated players’ results. Furthermore, comparing their drawbacks and advantages with the literature algorithm allowed us to identify scenarios and conditions to utilize FSPT and Narrow. There were also demonstrated the feasibility of implementing the proposed algorithm on programmable networks and identified the benefits of delay and delay-variation multicast algorithms in a realistic DIA environment.

In addition, it is also important to emphasize that a framework was designed and developed to simulate multicast networks. It allowed us to employ virtual network topologies, validate FSPT and Narrow, and compare them to any other algorithm implemented in the literature. We also looked at the impacts of custom multicast routing when influenced by different network conditions using an emulated environment with DIA. The
routing option was computed by the algorithms implemented in the framework. Then, we tested the application behavior and analyzed users’ viewpoint consequences of this choice.

Future research should consider the potential effects of DIA sessions dynamism on the routing algorithm designer. For example, when a player joins an online game, they are added to the game’s server and have provisioning of the network resources before they begin playing. When players exit the game, they are removed from the server and can no longer participate, but they also need the network resources to be released. The DIA infrastructure must handle this dynamicity for hundreds or even thousands of users whenever they are joining and exiting to ensure that the game runs smoothly and that all players have an enjoyable experience. Furthermore, these extreme events cannot impact the other users, and fairness routing needs to be reliable in terms of stability.

Apart from a planned modularizing of the emulation environment to be incorporated in other initiatives, extending it to collect more gaming metrics and support different game topologies with multi-server synchronizing is necessary, for example, by incorporating statistics of users’ discarded states sent from client components. This improvement may allow a better understanding of the game’s performance drawbacks when exposed to distinct network conditions and make more informed decisions about improving the routing for better QoS provisioning. Additionally, it will allow for the examination of the potential direct effects of these circumstances on user performance during matches and in achieving the game goals.

Lastly, when dealing with dispersion statistics metrics, one of the main problems is the precision of the results due to floating-point arithmetic typically used in its implementation. Mainly because representing and operating numbers using floating points in a computer system can lead to inaccuracies due to the limited precision of the numbers, even when the number of significant digits is considerable, which can lead to errors in the results of dispersion statistics metrics, like standard deviation or variance, sometimes leading to incorrect conclusions from the data. Therefore, possible new research outcomes can be carried out by extensively analyzing dispersion metrics computation when operated by different algorithms and how it behaves not only with dynamic attachment in the data set (SCHUBERT; GERTZ, 2018) but also with dropping, as when adding and removing samples, summing and subtracting scalar values, and merging and splitting data partitions.

In summary, this thesis contributes to the field of computer networks, thoroughly exploring various studies related to multicast routing and performance metrics. Additionally, it examines QoS for interactive applications, including fairness evaluation. The literature review also covers the DVBMT problem. Furthermore, this work faced network-edge challenges handling a trendy and pervasive operator environment, converging technological paradigms SDN, NFV, and MEC. This ecosystem brings advances in programmability, virtualization of network components, and availability of computational resources, respectively, and has allowed us to deliver custom routing solutions through the use of FSPT and Narrow algorithms to deal with DIA latency and fairness issues that affect several users in many widely day-to-day applications utilized. Ultimately, we note that the following questions emerge from this thesis, which can be worth investigating
in future research. How can floating-point arithmetic impact the accuracy of dispersion statistics metrics for aggregation and distribution data set handling, and how can they be addressed? How would the proposed algorithm handle dynamic changes in the number of users in a game or network? Is there a better routing strategy to manage users’ dynamism and maintain an adequate fairness level?


89


BELGHITH, A.; AISSA, M.; MNAOUER, A. B. A novel multicast algorithm for collaborative multimedia applications in heterogeneous networks. Procedia Computer Science,


MAZUR, I.; RAK, J.; NOWICKI, K. Ensuring the qoe-related fairness to reduce the user abandonment ratio. *Sensors*, v. 21, n. 21, 2021. ISSN 14248220.


