Universitat
de les Illes Balears

DOCTORAL THESIS
2022

FORMAL ALGEBRAIC MODELLING FOR FOG COMPUTING NETWORK ARCHITECTURE

Pedro Juan Roig Roig

Universitat
de les Illes Balears

DOCTORAL THESIS
2022

Doctoral Programme in Information and Communications Technology

FORMAL ALGEBRAIC MODELLING FOR FOG COMPUTING NETWORK ARCHITECTURE

Pedro Juan Roig Roig

Thesis Supervisor: Dr. Carlos Juiz García
Thesis Supervisor: Dr. Salvador Alcaraz Carrasco
Thesis Tutor: Dr. Carlos Juiz García

Doctor by the Universitat de les Illes Balears

Dedicated to my parents

Acknowledgments

I would like to thank my thesis supervisors Dr. Salvador Alcaraz, Dr. Katja Gilly and Dr. Carlos Juiz, as well as all those who supported me on my way to get here.

Thesis supervisor
Carlos Juiz and Salvador Alcaraz

Author
Pedro Juan Roig Roig

Abstract

Fog computing is basically an extension of cloud computing where the computing resources are located on the edge of the network, allowing for better performance regarding latency and bandwidth. Hence, data centres (DC) being used in fog computing may be far smaller and so may the number of hosts and switches in use. In this context, the present thesis dissertation undertakes the modelling of some DC designs for fog computing, setting the focus on just simple network topologies, even though more complex ones might achieve better performance. Such topologies may be modelled in different ways, exposing the minimal path, or equal-cost multiple paths, through which a live Virtual Machine (VM) migration of computing assets may take place from a source to a destination host. This way, a user moving throughout a fog domain will have their computing assets following it as close as possible with a minimal time interval, which is the key point in Internet of Things (IoT) moving environments. It is to be stressed that the models are going to be exposed by following an analytical approach, with the combination of different mathematical branches to get the models ready, such as geometry and topology to obtain the appropriate designs, arithmetic to forward the moving assets to the proper destination, logic to implement those actions in flow charts and pseudocode, or algebra to exhibit a formal description of the whole model. The main contribution in this thesis dissertation goes about obtaining models of optimal paths for VM migrations in DC topologies related to fog computing deployments by means of an abstract process algebra called Algebra of Communicating Processes (ACP), which has been used in order to formally specify and verify such models, as it allows to reason about process terms on an analytical basis, getting back to basics regarding the information technology field. In summary, regarding each topology, different models may be proposed, such as by means of flow charts, pseudocode and a formal algebraic model, followed by the presentation of a formal algebraic model for a whole fog/cloud system.

Directores de tesis
Carlos Juiz and Salvador Alcaraz

Autor
Pedro Juan Roig Roig

Resumen

Fog computing es básicamente una extensión del cloud computing, donde los recursos de computación se encuentran en el borde de la red, lo cual permite un mejor rendimiento en cuanto a latencia y ancho de banda. Por lo tanto, los centros de datos (DC) que se utilizan en fog computing deben ser mucho más pequeños, así como el número de hosts y switches instalados. En este contexto, esta disertación de tesis aborda el modelado de algunos diseños de DC para fog computing, poniendo el foco solo en topologías de red simples, aunque otras más complejas puedan lograr mejor rendimiento. Dichas topologías pueden modelarse de diferentes maneras, exponiendo la ruta mínima, o rutas mínimas de igual costo, a través de las cuales se pueden llevar a cabo las migraciones de recursos entre un host origen y otro destino. De esta manera, un usuario moviéndose por un dominio fog tendrá sus recursos de computación asociados siguiéndole lo más cerca posible en un intervalo de tiempo mínimo, siendo éste el punto clave en los entornos de movilidad IoT. Se debe destacar que los modelos se van a exponer siguiendo un enfoque analítico, mediante la combinación de diferentes ramas matemáticas para preparar los modelos de la manera más directa, como la geometría y la topología para obtener los diseños adecuados, la aritmética para llevar a cabo el movimiento de los recursos de computación al destino adecuado, la lógica para implementar esas acciones en diagramas de flujo y pseudocódigo, o el álgebra para proporcionar una descripción formal de todo el modelo. La principal contribución en esta disertación de tesis consiste en obtener modelos de rutas óptimas para migraciones de VM en topologías de DC relacionadas con despliegues fog computing por medio de un álgebra de procesos abstracta denominada ACP, con la cual se han especificado y verificado formalmente dichos modelos, ya que permite razonar sobre términos de procesos de forma analítica, volviendo a los orígenes en el campo de las tecnologías de la información. En resumen, con respecto a cada topología, se proponen diferentes modelos, tales como diagramas de flujo, pseudocódigo y un modelo algebraico formal, para posteriormente presentar un modelo algebraico formal para un sistema completo de fog/cloud.

Directors de tesi

Carlos Juiz and Salvador Alcaraz

Autor

Pedro Juan Roig Roig

Resum

Fog computing és bàsicament una extensió del cloud computing, on els recursos de computació es troben al límit de la xarxa, la qual cosa permet un millor rendiment pel que fa a latència i ample de banda. Per tant, els centres de dades (DC) que s'utilitzen en fog computing han de ser molt més petits, així com el nombre de hosts i switches instal·lats. En aquest context, aquesta dissertació de tesi aborda el modelatge d'alguns dissenys de DC per fog computing, posant el focus només en topologies de xarxa simples, encara que altres més complexes puguin aconseguir millor rendiment. Aquestes topologies poden modelar-se de diferents maneres, exposant la ruta mínima, o rutes mínimes del mateix cost, a través de les quals poden tenir lloc les migracions de recursos entre un host origen i un altre destí. Així, un usuari movent-se per un domini fog tindrà els seus recursos de computació associats seguir-lo el més a prop possible amb un interval de temps mínim, que és el punt clau en els entorns de mobilitat IoT. S'ha de destacar que els models van a ésser exposats seguint un enfocament analític, mitjançant la combinació de diferents branques matemàtiques per preparar els models, com la geometria i la topologia per obtenir els dissenys adequats, l'aritmètica per dur a terme el moviment dels recursos de computació a la destinació adequada, la lògica per implementar aquestes accions en diagrames de flux i pseudocodi, o l'àlgebra per a proporcionar una descripció formal de tot el model. La principal contribució en aquesta dissertació de tesi es basa en obtenir models de rutes òptimes per a migracions de VM en topologies de DC relacionades amb desplegaments fog computing mitjantçant un àlgebra de processos abstracta anomenada ACP, amb la qual s'han especificat i verificat formalment aquests models, ja que permet raonar sobre termes de processos de forma analítica, tornant als orígens en el camp de les tecnologies de la informació. En resum, pel que respecta a cada topologia, es proposen diferents models, como ara diagrames de flux, pseudocodi i un model algebraic formal, per a posteriorment presentar un model algebraic formal per a un sistema complet de fog/cloud.

Contents

Title Page i
Dedication iii
Acknowledgments v
Abstract vii
Resumen ix
Resum xi
List of Figures xix
List of Tables xxiii

1 Introduction **1**
1.1 Context 1
1.2 Objectives 6
1.3 Outline 7

2 Main pillars and related work **9**
2.1 Computing, cloud and fog 9
2.1.1 Cloud and Fog computing 12
2.1.2 VM migration techniques 19
2.2 Modelling with ACP 22
2.2.1 Using FDT modelling in cloud/fog environments 27
2.2.2 ACP modelling 29
2.3 Topologies for data centres 35
2.3.1 Network topologies 36
2.3.2 Redundant topologies proposed 39
2.4 Summary 40

3 Regular *N*-polytopes, trees, graphs and tori applied to data centres **43**
3.1 *N*-dimensional geometry 43
3.1.1 Platonic solids 45
3.1.2 Schläfli notation 47
3.1.3 Regular polygons 48
3.1.4 Regular polyhedra 50

3.1.5 Regular polychora 53
3.1.6 Regular polytera 55
3.1.7 Regular N-polytopes 57
3.1.7.1 N-simplex: k-facets 58
3.1.7.2 N-hypercube: k-facets 60
3.1.7.3 N-orthoplex: k-facets 61
3.2 Tree-like data structures 62
3.2.1 Abstract data types 63
3.2.2 Vocabulary about trees 63
3.2.3 Structural induction 65
3.2.4 Binary trees 65
3.2.5 M-ary trees 68
3.3 Graph-like data structures 70
3.3.1 Seven bridges of Königsberg 71
3.3.2 Fundamentals on graphs 72
3.3.3 Eulerian and Hamiltonian graphs 73
3.3.4 Geometrical graphs 74
3.3.5 De Bruijn graphs 75
3.4 Torus topologies 81
3.4.1 Properties of a torus 81
3.4.2 Toroidal network designs 84
3.4.3 De Bruijn torus 86
3.4.4 De Bruijn hypertorus 89
3.5 Summary 92

4 Common network topologies for Data Centres 97
4.1 Partial mesh: tree-like designs 98
4.1.1 Fat tree 99
4.1.2 Leaf and spine 101
4.2 Partial mesh: graph-like designs 103
4.2.1 N-hypercube 104
4.2.2 Folded N-hypercube 107
4.3 Comparing tree-like -vs- graph-like designs 112
4.3.1 Average distance in links among hosts 116
4.3.2 Average distance in links among end switches 117
4.3.3 Results and analysis 118
4.4 Other mesh and star topologies 120
4.4.1 Full mesh 120
4.4.2 Quasi full mesh 121
4.4.3 Star 122
4.5 Toroidal grid topologies 124
4.5.1 Redundant ring 125

4.5.2 Toroidal ring 125
4.6 Directed-graph topologies 126
4.6.1 De Bruijn graph 126
4.6.2 De Bruijn reverse graph 128
4.7 Undirected-graph topologies 129
4.7.1 Binary grid 130
4.7.2 Hamming graph 132
4.7.3 Cage graphs 135
4.7.3.1 Petersen graph 136
4.7.3.2 Heawood graph 137
4.8 Summary 137

5 Modelling network Data Centre topologies with ACP 139
5.1 Hosts, switches and ports 140
5.2 Modelling hosts 141
5.3 Fat tree 145
5.3.1 Edge switches 146
5.3.2 Aggregation switches 147
5.3.3 Core switches 150
5.3.4 Consolidating fat tree 151
5.4 Leaf and spine 158
5.4.1 Leaf switches 160
5.4.2 Spine switches 161
5.4.3 Consolidating leaf and spine 163
5.5 N-hypercube 165
5.6 Folded N-hypercube 173
5.7 Full mesh 179
5.8 Quasi full mesh 183
5.9 Redundant star 186
5.9.1 Spoke switches 187
5.9.2 Hub switches 188
5.9.3 Consolidating hub and spoke 189
5.10 Redundant ring 191
5.11 Toroidal ring 194
5.12 De Bruijn graph 197
5.13 De Bruijn reverse graph 203
5.14 Binary grid 206
5.15 Hamming graph 209
5.16 Petersen graph 213
5.17 Heawood graph 217
5.18 Summary 223

6 Generalization of Data Centre models with ACP 227
6.1 Topology framework for a generic fog model 227
6.2 Modelling VM actions within a fog environment with ACP 230
6.2.1 Action *cloud in* 231
6.2.2 Action *migrate in* 233
6.2.3 Action *init VM* 235
6.2.4 Action *cloud out* 236
6.2.5 Action *migration out* 238
6.2.6 Action *kill VM* 239
6.3 Verification of the models 240
6.4 Summary 242

7 Use case of Data Centre models 245
7.1 Sequential relay - schematic diagram 245
7.2 Topology framework for linear deployments 247
7.3 Scenario 1: one-way and no resource restrictions 250
7.4 Scenario 2: two-way and resource restrictions 255
7.5 Model verification 263
7.6 Summary 267

8 Conclusions and Future Work 269
8.1 Conclusions 269
8.2 Future work 275

9 Publications 279

10 Appendix 285
10.1 Algorithm for a host 285
10.2 Algorithm for an edge switch in fat tree 286
10.3 Algorithm for an aggregation switch in fat tree 287
10.4 Algorithm for a core switch in fat tree 287
10.5 Algorithm for a leaf switch in leaf and spine 288
10.6 Algorithm for a spine switch in leaf and spine 288
10.7 Algorithm for an N-hypercube switch 289
10.8 Algorithm for a folded N-hypercube switch 290
10.9 Algorithm for a full mesh switch $\{N$-simplex$\}$ 291
10.10Algorithm for a quasi full mesh switch $\{N$-orthoplex$\}$ 292
10.11Algorithm for a spoke switch in a redundant star 293
10.12Algorithm for a hub switch in a redundant star 293
10.13Algorithm for a redundant ring switch 294
10.14Algorithm for a toroidal ring switch 295
10.15Algorithm for a de Bruijn graph switch 296

10.16Algorithm for a de Bruijn reverse graph switch 297
10.17Algorithm for a binary grid switch 298
10.18Algorithm for a Hamming graph switch 299
10.19Algorithm for a Petersen graph switch 300
10.20Algorithm for a Heawood graph switch 301

Bibliography **307**

List of Figures

2.1 Cloud service models. 13
2.2 VM migration timeline. 21
2.3 Example of bisimilarity. 31
2.4 Most common network topologies. 37
2.5 Instances of topologies with a tree-like design and a graph-like design. 38

3.1 Platonic solids. 46
3.2 Convex regular polygons. 49
3.3 Star regular polygons. 50
3.4 Regular tessellation of the $1D$ Euclidean line. 50
3.5 Star regular polyhedra. 52
3.6 Regular tessellation of the $2D$ Euclidean plane. 53
3.7 Regular tessellation of the $3D$ Euclidean space. 56
3.8 N-simplex shapes for lower dimensions. 59
3.9 N-hypercube shapes for lower dimensions. 61
3.10 N-orthoplex shapes for lower dimensions. 62
3.11 Perfect binary tree with height 3. 68
3.12 Perfect ternary tree with height 3. 69
3.13 Perfect quaternary tree with height 3. 69
3.14 Topology of the 7 bridges of Königsberg, along with its graph. 71
3.15 Directed -vs- undirected graphs. 72
3.16 De Bruijn graph: Eulerian $B(2,4)$ and Hamiltonian $B(2,3)$. 78
3.17 Ring torus geometry. 82
3.18 Forming a torus from a square. 84
3.19 Toroidal structures: line (along $1D$) and ring (along $2D$). 85
3.20 De Bruijn binary square torus with a squared pattern: $(4,4;2,2)_2$ . . 87
3.21 De Bruijn torus $(4,4;2,2)_2$: squared patterns. 88
3.22 Various bidimensional toroidal arrays. 89
3.23 Three-dimensional toroidal array ($2 \times 2 \times 2$ nodes). 91
3.24 Three-dimensional toroidal array ($16 \times 4 \times 4$ nodes). 91

4.1 Fat tree architecture for $K = 4$ and oversubscription ratio 1:1. 101
4.2 Leaf and spine architecture with 8 leaves and 4 spines. 103
4.3 N-hypercube architectures for $N \in \{0 \cdots 4\}$. 105
4.4 Pascal's triangle for the first rows. 107
4.5 Converting a plain N-hypercube into a folded N-hypercube. 108
4.6 Folded N-hypercube architectures for $N \in \{2 \cdots 4\}$. 108
4.7 Converting a Pascal's triangle into a folded Pascal's triangle. 110
4.8 Folded Pascal's triangle for the first rows. 110
4.9 Full mesh topology. 121
4.10 Quasi full mesh topology. 122
4.11 Hub and Spoke topology. 123
4.12 Redundant Hub and Spoke with a direct communication link. 124
4.13 Hamiltonian paths for $B(2, 3)$. 126
4.14 Eulerian path for $B(2, 4)$. 127
4.15 Paths in and out of a node in a binary de Bruijn graph. 127
4.16 De Bruijn graph B(3,2). 128
4.17 De Bruijn reverse graph: Hamiltonian $\overline{B}(2, 3)$ and Eulerian $\overline{B}(2, 4)$. . 129
4.18 Binary grid for $n = 2$, $n = 3$, $n = 4$. 130
4.19 Hamming graph $H_2(n, n - 1)$ for $n = 3$ and $n = 4$. 133
4.20 Petersen graph. 136
4.21 Heawood graph. 137

5.1 Model for a generic switch i, with its connected hosts. 141
5.2 Model for a generic host h. 143
5.3 Flow chart for hosts, with a unique user. 144
5.4 Flow chart for hosts, with several users. 145
5.5 Model for switches and their ports in fat tree. 146
5.6 Fat tree device nomenclature for $K = 4$. 146
5.7 Flow chart for the behaviour of an edge switch (fat tree). 148
5.8 Flow chart for the behaviour of an aggregation switch (fat tree). . . . 149
5.9 Flow chart for the behaviour of a core switch (fat tree). 151
5.10 Model for switches and their ports in leaf and spine. 159
5.11 Leaf and spine device nomenclature. 160
5.12 Flow chart for the behaviour of an leaf switch (leaf and spine). 161
5.13 Flow chart for the behaviour of a spine switch (leaf and spine). 162
5.14 Model for switches and their ports in N-hypercube. 167
5.15 Node nomenclature using N binary values: 3 (left) and 4 (right). . . . 167
5.16 Node nomenclature translated into decimal values: 3 (left) and 4 (right).168
5.17 Model the interconnected switches i for 3-hypercube. 168
5.18 Flow chart for the behaviour of a given switch (N-hypercube). 169
5.19 Model for switches, along with its ports (folded N-hypercube). 175
5.20 Node nomenclature employing N binary values: 3 (left) and 4 (right). 175

5.21 Node nomenclature converted into decimal values: 3 (left) and 4 (right). 176
5.22 Flow chart for the behaviour of a given switch (folded N-hypercube). 177
5.23 Model for the uplink ports in a full mesh switch (N-simplex). 181
5.24 Model for the interconnected switches in a 3-simplex. 181
5.25 Flow chart for the behaviour of a given switch (N-simplex). 182
5.26 Model for the uplink ports in a quasi full mesh switch (N-orthoplex). 184
5.27 Model for the interconnected switches in a 3-orthoplex. 184
5.28 Flow chart for the behaviour of a given switch (N-orthoplex). 185
5.29 Model for switches, along with its ports (redundant hub and spoke). . 187
5.30 Nomenclature of switches in redundant hub and spoke. 188
5.31 Flow chart for the behaviour of a spoke (redundant hub and spoke). . 189
5.32 Flow chart for the behaviour of a hub (redundant hub and spoke). . . 190
5.33 Model for switches, along with its ports (redundant ring). 192
5.34 Node setup for a redundant ring topology. 192
5.35 Flow chart for the behaviour of a given switch (redundant ring). . . . 193
5.36 Node setup for a toroidal ring topology for $n = 4$. 196
5.37 Model for switches, along with its ports (toroidal ring). 196
5.38 Flow chart for the behaviour of a given switch (toroidal ring). 198
5.39 Model for a given node i in a de Bruijn graph, along with its ports. . 200
5.40 Flow chart for the behaviour of a switch (de Bruijn graph). 201
5.41 Flow chart for the behaviour of a switch (de Bruijn reverse graph). . 205
5.42 Model for a given node i in a Binary grid, along with its ports. 207
5.43 Flow chart for the behaviour of a given switch (binary grid). 208
5.44 Model for a node i in a Hamming graph $H_k(n, n-1)$ and its ports. . 211
5.45 Flow chart for the behaviour of a switch (Hamming graph $H_k(n, n-1)$). 212
5.46 Node distribution in a Petersen graph. 214
5.47 Model for a given node i in a Petersen graph, along with its ports. . . 216
5.48 Flow chart for the behaviour of a given switch (Petersen graph). . . . 218
5.49 Node distribution in a Heawood graph. 219
5.50 Model for a given node i in a Heawood graph, along with its ports. . 222
5.51 Flow chart for the behaviour of a given switch (Heawood graph). . . . 224

6.1 Generic topology scheme for an IoT/fog model. 228

7.1 Schematic diagram of a fat tree sequential wireless relays. 246
7.2 External behaviour of scenario 1 and scenario 2. 265

List of Tables

3.1 Platonic solids. . . . 46

3.2 Convex regular polyhedra in Schläfli notation. . . . 51

3.3 Star regular polyhedra in Schläfli notation. . . . 52

3.4 Star regular polyhedra and their main features. . . . 52

3.5 Convex regular polychora in Schläfli notation (I). . . . 54

3.6 Convex regular polychora in Schläfli notation (II). . . . 54

3.7 Star regular polychora in Schläfli notation. . . . 55

3.8 Convex regular polytera in Schläfli notation. . . . 56

3.9 Convex regular N-polytopes in Schläfli notation ($N \geq 5$). . . . 57

3.10 Meaning of k-facets. . . . 59

3.11 Amount of k-facets present in each N-simplex. . . . 60

3.12 Amount of k-facets present in each N-hypercube. . . . 61

3.13 Amount of k-facets present in each N-orthoplex. . . . 62

3.14 Number of nodes in a perfect binary tree of height h. . . . 68

3.15 Number of nodes in a perfect M-ary tree of height h. . . . 70

3.16 Wong algorithm to achieve de Bruijn sequence $B(2, 3)$. . . . 80

3.17 Wong algorithm to achieve de Bruijn sequence $B(2, 4)$. . . . 81

3.18 De Bruijn sequences expressed as de Bruijn tori with length up to 64. 90

3.19 An instance of a $16 \times 4 \times 4$ de Bruijn hypertorus. . . . 93

3.20 An instance of all unique 256 b-cubes in a $16 \times 4 \times 4$ de Bruijn hypertorus. 94

4.1 Fat tree most relevant values. . . . 101

4.2 Number of redundant paths in fat tree among hosts. . . . 102

4.3 Relevant number of items for the scenarios proposed (fat tree). . . . 113

4.4 Relevant number of items for the scenarios proposed (leaf and spine). 114

4.5 Relevant number of items for the scenarios proposed (N-hypercube). 114

4.6 Relevant number of items for the scenarios proposed (folded N-Hyp). 115

4.7 Average distance in links (λ) among hosts. . . . 118

4.8 Average distance in links (λ) among end switches. . . . 119

4.9 Known $(3, g)$-cages. . . . 136

4.10 Topologies proposed along with some relevant features. . . . 138

5.1 Relevant variables and operations to be applied in the models. 142
5.2 List of devices and ports between 1-hop away hosts in fat tree. 154
5.3 List of devices and ports between 2-hops away hosts in fat tree. . . . 154
5.4 List of devices and ports between 3-hops away hosts in fat tree. . . . 155
5.5 List of devices and ports between 1-hop away hosts in leaf and spine. 163
5.6 List of devices and ports between 2-hops away hosts in leaf and spine. 164
5.7 Searching mismatching bits by arithmetic and logical means. 169
5.8 List of devices and ports between 1-hop away hosts in N-hypercube. . 170
5.9 Permutations from source switch to destination switch in N-hypercube. 172
5.10 Searching the shortest path and its next hops by using of logical means. 176
5.11 List of devices and ports between 1-hop away hosts in folded N-hyp. . 179
5.12 Permutations from source switch to destination switch in folded N-hyp. 180
5.13 List of devices and ports between 1-hop away hosts in full mesh. . . . 181
5.14 List of devices and ports between 2-hops away hosts in full mesh. . . 183
5.15 List of devices and ports between 1-hop away hosts in quasi full mesh. 186
5.16 List of devices and ports between 2-hops away hosts in quasi full mesh. 186
5.17 List of devices and ports between 1-hop away hosts in R. hub & spoke. 189
5.18 List of devices and ports between 2-hops away hosts in R. hub & spoke. 190
5.19 Searching the hops from source to destination node (de Bruijn graph). 202
5.20 Searching the hops from source to destination node (de Bruijn rev.). . 206

6.1 Communication channels in the model. 230

List of Algorithms

1 Host() 285

2 Fat tree - edge() 286

3 Fat tree - aggregation() 287

4 Fat tree - core() 287

5 Leaf and spine - leaf() 288

6 Leaf and spine - spine() 288

7 N-hypercube() 289

8 Folded N-hypercube() 290

9 Full mesh() $\{N$-simplex$\}$ 291

10 Quasi full mesh() $\{N$-orthoplex$\}$ 292

11 Redundant hub and spoke - spoke() 293

12 Redundant hub and spoke - hub() 293

13 Redundant ring() 294

14 Toroidal ring() 295

15 De Bruijn graph() 296

16 De Bruijn reverse graph() 297

17 Binary grid() 298

18 Hamming graph() $\{H_k(n, n-1)\}$ 299

19 Petersen graph() 300

20 Heawood graph() 301

Chapter 1

Introduction

1.1 Context

Cloud computing paradigm has been consolidated in recent years, as traditional computing deployments are quite often being extended with the addition of extra assets, usually off-site, those being external resources basically regarding processing power and storage facilities, which are reached by service users through any sort of network connection.

Cloud scenarios provide many advantages compared to on-premise scenarios, such as ubiquitous computing, cost savings and on-demand resource flexibility. However, cloud's core nature may impose implicit disadvantages, such as not being appropriate in real-time scenarios, due to the latency involved in reaching the cloud and getting back, or otherwise, not being fit for its use in low-resource devices.

In order to cope with those limitations, fog computing paradigm represents an extension of the cloud by bringing the computing resources to the edge of the network. This way, latency and jitter rates get reduced and resource utilization gets decreased, whilst keeping the advantages of remote computation.

On the other hand, IoT (Internet of Things) devices are ordinary items equipped with computing and networking capabilities so as to get them connected in order to achieve a more efficient performance, thus making possible the enhanced deployment of smart cities and smart grids or the use of cyber physical systems in Industry 4.0.

Those devices usually have limited resources regarding processing power, memory allocation, storage capacity, bandwidth restrictions and low-power batteries.

IoT devices make the perfect candidates for the fog computing paradigm, and that is why the combination of both is often referred to as IoT/fog environments. In this context, it is to be said that such environments are usually divided into three hierarchical layers, where in the lower one there are IoT devices like sensors, actuators, smartphones, tablets or cars, whereas in the middle one there are fog nodes, also known as hosts, with higher computing power than the former, thus dealing with most of the data generated by those, whilst in the upper one there are back-end cloud nodes so as to manage data requiring an extensive use of computing power or storage.

The most typical scenario is that of a group of IoT items, ranging from a few to a myriad, being all connected to some central servers forming the fog layer, according to a hub and spoke topology, where those hosts undertake most of the workload, although they may divert some of it to a cloud backup node.

Special attention has to be paid to those IoT devices in fog environments where they may be moving around the fog facilities, those being distributed along a limited geographic area. In such a case, it is important that the computing resources allocated to each user on the fog infrastructure, normally deployed as virtual instances in the form of a VM (Virtual Machine) or a virtual container, get as close as possible to the physical locations of their respective users, hence, latency and jitter get reduced, whereas bandwidth needs get minimized. It is to be noted that, in the context of this thesis, when these virtualized entities associated to users are mentioned, they will be generically referred to as VMs, although the approach using virtual containers would be very similar.

Additionally, when the IoT devices have mobility, the communication delay between a moving device and its associated computing assets may increase with the distance among them, thus getting to degrade the performance as such a distance grows. In this context, the concept of distance refers to number of links making the path between two given source and destination entities, which depends on the network topology of the DC (Data Centre) in use. Hence, in order to keep such a distance within a reasonable range at all times whilst allowing the device to keep moving, it may be necessary to also move its related assets in some way to get them close enough to each other.

VM migration is the mechanism for a VM to try and follow its associated user while it moves through the fog environment. This migration takes place between the source host where the VM is currently held to the destination host where the VM is ready to move so as to minimise the distance towards its related user.

The migration path will try to follow the shortest available path between source and destination hosts. There may well be redundant paths, and in such a case, load balancing policies may be applied. Anyway, those paths definitely depend on the topology of the fog environment, thus, they will be related to the interconnections put in place among switches.

There are many different topologies being used in fog computing to interconnect hosts, each of them with their own benefits and drawbacks. The ideal situation would be a full-mesh infrastructure, where all hosts would be just one hop apart from each other. However, as the number of hosts increases, this approach is not feasible, so partial-mesh solutions come into play.

Among partial-mesh solutions, a distinction between hierarchical and flat views may be noted, the former presenting more advantages as the number of hosts grow, making tree-like designs, whereas the latter allowing for good solutions for smaller numbers, presenting graph-like designs.

As per the tree-like category, *fat tree* and *leaf and spine* get the most referenced instances in the literature related to fog computing, whilst regarding the graph-like category, *plain N-hypercube* is a typical instance, along with a well-known extension called *folded N-hypercube.*

Furthermore, *hub and spoke* designs are widely considered in real deployments, where there is a central entity, called hub, which establishes a point-to-point connection to each of the other entities, named spokes. Therefore, all communications between any pair of spokes need to go through the hub.

There may be many variations in all those categories, some of them presenting advantages on the performance side, in spite of introducing some degree of complication on the design and on the packet forwarding side. Therefore, in order to achieve a good trade-off between both parameters, the most basic instances are still widely used in many fog deployments.

Regarding the IoT devices being found in a fog environment, it is to be remarked that those IoT items are usually ordinary devices equipped with limited resource capabilities, which bring some sensors or actuators attached, thus playing the part of end devices for fog communications. Furthermore, the fog infrastructure is made up of a series of hosts being located anywhere between the end devices and the cloud, whose role is to undertake most of the computation and storage necessities of the aforesaid IoT devices, using the cloud as a backup system.

Those hosts may be interconnected by a network of switches, which are linked together according to a certain topology in order to allow the VM migration of a given instance of VM from any source host to any destination host, where different topology deployments may be put in place in relation to the needs of each fog ecosystem.

Hence, each switch joins together a bunch of hosts, resulting in only two links between any pair of those, whereas all switches communicate to each other through a certain topology, which determines the number of links between any pair of switches, resulting in a variable number of links between hosts connected to different switches. In this context, some key performance indicators, such as latency, jitter and bandwidth, will depend on the precise number of such links between any couple of hosts.

Furthermore, it is to be considered that switching topologies offer scalability, resilience and load balancing among the diverse hosts within the fog domain, thus making possible that the aforementioned VM migrations between any given pair of hosts may be carried out by means of alternative paths, which may depend on the switching topology being chosen within the fog environment.

Additionally, it is to be noted that fog environments may not contain a great amount of hosts due to the limited geographic scope of fog domains. Therefore, the number of necessary switches to interconnect all those hosts will not be too high, hence, fog deployments may not need complicated topologies.

On the other hand, some complex topologies might obtain better performance indicators that the ones presented, even though such a complexity in those designs may make packet forwarding more difficult to be achieved. Thus, such designs may not be taken into account herein, so in order to keep things simple, just the aforesaid topologies are going to be considered herein.

With that in mind, all of the moving devices within a fog environment may need its associated computing resources to be the closest possible. This fact implies moving those computing resources among hosts, and if those are considered as VMs, then it may be said that live VM migrations may take place among hosts.

With respect to the minimal paths in each topology to achieve the fastest VM migration by means of taking the shortest path between a particular pair of hosts, it is to be said that it may be expressed by applying the appropriate mathematical expressions, which may result in the construction of different models according to diverse mathematical areas, such as arithmetic, logic or algebra.

In this sense, flow charts and pseudocode algorithms are presented for all topologies selected, although special attention needs to be paid to algebraic models, specifically designed by means of a process algebra called ACP (Algebra of Communicating Processes), as they may be further used in order to find out whether the algebraic models presented get verified, which basically means that the behaviour of the model proposed matches that of the real system, in a way that all possible inputs in both generate the desired outputs in both.

It is to be remarked that ACP does not take time into consideration, thus the measurement of performance may not based on time, as it is the usual case where performance is evaluated by means of timed process algebras, queueing networks or timed Petri nets. Hence, the focus on performance has been swapped from time to space, related to switching infrastructure and redundant paths, leading to distances, and this way, performance may be evaluated by means of timeless Petri nets or timeless process algebras, where ACP belongs to.

Regarding networking, bandwidth and latency are the key players when dealing with performance analysis. This is usually measured in time units, even though an alternative view may be to measure distance as a measure of time, as long as all links within the DC have the same bandwidth and the same features, whereas their link lengths are also the same. If that is the case, distance is directly proportional to time, such that if distance increases, so does time. In view of that, distance is going to be used herein to measure performance as the number of links between any given pair of devices because it is more intuitive and straightforward, where the shorter the better.

In this context, routing is crucial to enhance performance, as it minimizes the number of links to forward packets between a particular source and a particular destination within the DC. Other important parameters may also be cited, such as the protocol being used and the link utilization due to inefficiencies of implementation or environmental conditions, although only routing is going to be considered herein.

Therefore, the main working hypothesis of this thesis dissertation may be whether it is possible to model the VM migration path within a given DC topology being deployed in a fog environment by means of ACP. Furthermore, some other main working hypotheses may be proposed, such as the construction of a generic model for a fog ecosystem in ACP, where the DC topology is abstracted away, followed by the built-up of a particular instance of it, being focused on linear deployments, which is also designed with ACP. Additionally, some secondary working hypotheses may be brought forward, such as getting such models by means of flow charts and pseudocode algorithms, along with getting the list of devices and their appropriate ports for all redundant minimal paths (measured in number of links) for the models in ACP, as well as getting the appropriate verification for them.

This way, the target of this work may be seen mainly as threefold: first, to obtain formal algebraic models of VM migration for the aforementioned topologies, then, to achieve a formal algebraic model for any user moving around a generic fog environment, and from there on, to build up a specific scenario with a sequential layout, which may be adapted to bring fog services throughout a trajectory.

1.2 Objectives

Therefore, the goals in this thesis are:

- To review the state of the art regarding the main pillars of this research, such as fog computing environments, Formal Description Techniques (FDT) and network topologies for DC.

- To go over some mathematical objects to be used in the models proposed, such as regular N-polytopes, trees, graphs and tori.
- To revise the main characteristics of the DC topologies presented.
- To obtain different mathematical models for VM migration in fog ecosystems for such topologies, where the amount of nodes within a given topology depends on the number of hosts needed in the fog deployment.
- To establish a generic algebraic model for a fog computing domain.
- To propose an application of such a model for a sequential deployment.

This way, the main hypotheses proposed above are going to be developed in order to prove them right:

a) Is it possible to model the VM migration path within a given DC topology being deployed in a fog environment by means of ACP?

b) Is it possible to construct a generic model for a fog ecosystem in ACP, where the DC topology is abstracted away?

c) Is it possible to build up a particular instance of such a general model in ACP which is being focused on linear deployments?

Additionally, the secondary hypotheses cited above will also be developed on the way to prove the main one quoted in the first place.

1.3 Outline

The development of this thesis involves three parts as follows:

- **Part one: background.** This includes a brief overview on the main pillars sustaining this thesis. In particular, chapter 1 introduces some of the main features of fog computing, whilst chapter 2 discusses the role of live VM migration in fog domains, modelling with ACP and DC topologies. Furthermore, chapter 3 goes through some mathematical foundations to be applied on the network topology models proposed later on, whereas chapter 4 reviews some of the most common DC topologies for fog deployments.

- **Part two: contribution.** It is focused on the new proposals given in this thesis. In this sense, chapter 5 exhibits different mathematical modellings for each of the aforementioned topologies, including algebraic representations to be used in the verification of such designs, whilst chapter 6 presents an algebraic model of a generic fog environment, abstracting away from the topology being used, and chapter 7 carries out a real-life application of such a model for sequential deployments, which may be implemented in railways, highways or pipelines.

- **Part three: conclusions.** That contains the final references associated with this thesis. Basically, chapter 8 draws the conclusions reached out of this thesis dissertation, as long as the future work proposals. Afterwards, chapter 9 shows the contributions achieved with this thesis, resulting in a collection of research papers published in international conferences and journals related to the foundational pillars cited above. Then, there is an appendix showing the algorithms proposed for each topology studied above. Next, there is a section explaining the meaning of the acronyms being used. After that, the last section displays the bibliography being used when building up this thesis.

Chapter 2

Main pillars and related work

This work is about getting a formal algebraic modelling of a fog computing architecture, which is based on some different pillars to be presented herein.

2.1 Computing, cloud and fog

First of all, it is to be noted that the world of computing undertakes significant technological developments on a regular basis, thus enhancing the way they operate, as well as gaining compactness and efficiency, whilst reducing cost and power consumption [143]. Therefore, each major improvement induced a new change of paradigm when it comes to how computers work, hence, that fact brought up a new generation of computers.

Initially, it is widely accepted that the first electronic digital computers are considered to be part of the first generation of computers appeared in the 1940s in the context of World War II and the corresponding postwar, which were based on *vacuum tubes* to process information and *binary code* was used to interact with hardware.

Later on, a new generation emerged in the late 1950s with the creation of the first Operating System (OS), along with the use of *transistors* for information processing, which led to the appearance of faster and more reliable computers. Besides, the use of symbolic languages like *assembly* prevailed instead of binary code, and even the first high-level programming languages were launched, such as *Cobol* and *Fortran*.

Afterwards, in the middle 1960s, another generation kicked off with the use of integrated circuits, also known as *microchips*, along with the appearance of the first instances of OSs being multitask, multiuser, multiprocessor or real-time, as well as the first computer networks, which required manual intervention to accomplish file sharing. It is to be noted that *Unix*, which is the basis of most current OSs, was designed at that time.

Later on, in the early 1970s, a new generation arose due to the invention of *microprocessors*, those being able to integrate much more circuits into a single chip, which led to the development of both microcomputers and supercomputers, where the former popularized the use of personal computers, whilst the latter were devoted to problems needing intensive calculations. Furthermore, it is to be said that core developments in nowadays computing systems were made in this period, such as the creation of *C* programming language and the Transmission Control Protocol / Internet Protocol (TCP/IP) practical model and Open Systems Interconnection (OSI) reference model for network transmissions.

Eventually, the current generation is supposed to have been started in the middle 1980s, where diverse *artificial intelligent technologies* are being implemented in both hardware and software to try to improve different problem-solving skills, with the target of ever achieving more complex and accurate solutions. It is to be said that the vast majority of current OSs, programming languages, services, applications, network solutions, security designs and software developments are included therein.

Focusing on the way computers work, it is to be noted that all previous stages did it in a centralized way, meaning that each individual computer ran its own tasks. This fact is also known as *monolithic computing*, regardless whether they only support a single user at a time or multiple users through time sharing techniques.

However, in the last stage, the continuous rising in the power of microprocessors and the speed of Local Area Network (LAN) made possible the *distributed computing*, where hardware or software components may be located in different computers being interconnected and communication among them is undertaken by means of message passing techniques. This way, distributed systems permit to cope with ever higher load requirements and performances, thus outperforming centralized systems [54].

Some of the main features of such systems are concurrency, as tasks are executed in a simultaneous manner as opposed to a sequential one, the need of a common time for coordination and synchronization purposes, the failure tolerance in computers related to either its hardware, software or network components, the variability of the system structure, which may change during the execution time, and even the incomplete view of the whole system from each individual computer.

Different architectures are allowed in distributed systems, where maybe the most typical one is the client-server, also known as master-slave, where a client requests a resource or service being located on the server, which in turn provides it back to the client through a response. Some well known protocols of this architecture may be Hypertext Transfer Protocol (HTTP) protocol for web browsing, Simple Mail Transfer Protocol (SMTP) and Post Office Protocol revision 3 (POP3) protocols for email exchanging or File Transfer Protocol (FTP) protocol for file transferring.

Furthermore, peer-to-peer is also a typical architecture, where both entities establishing a communication channel may act as either a client or a server, depending on which is the requester and which is the provider of a specific resource. Some well-known examples of this architecture may be file sharing, Voice over IP (VoIP) services, videoconferencing applications or virtual currencies like bitcoin.

Additionally, distributed systems may be found in most types of wired and wireless telecommunication networks, as well as a wide range of network applications, real time process control for industrial automation and flight control. Furthermore, parallel computing may also be cited, where a task is divided into similar subprocesses to be run simultaneously, whose results are combined upon completion.

In that sense, multicore and multiprocessor keep the processing within the same computer, where the former implies a processor having more than one independent processing unit working at a time in a single Central Processing Unit (CPU) and the latter does more than one CPU working simultaneously.

On the contrary, cluster and grid share the processing among different computers, where the former implies that each node is homogeneous and performs the same task as a unique unit, whilst the latter states that all nodes may be homogeneous or heterogeneous and performs the same of a different task.

2.1.1 Cloud and Fog computing

A special instance of parallel computation called *cloud computing* needs to be cited, also referred to as cloud [223]. It may be seen as on-demand computing resources, typically related to storage or computing power, being offered over a network connection. This way, many services might be available such as file saving and restoring, online application executing, remote data processing, repository managing or collaborative environment working, all being delivered on an ad hoc basis.

Clouds may be classified into three main categories, such as public cloud, private cloud and hybrid cloud. As per the first one, it is dedicated to anyone and it is delivered through the Internet, which may or may not require an access fee, then, regarding the second one, it is restricted only to a company, being offered with different levels of access to employees, providers and clients, and finally, with respect to the third one, it combines both clouds in order for the companies to keep with both the availability requirements and the data protection legislation.

It may be said that the number of *as-a-service* types offered are growing by the day, even though the most well-known models of cloud are the following:

- Infrastructure as a Service (IaaS): it permits to rent tailor-made virtualized hardware, by choosing the number of processors, Random Access Memory (RAM) memory and storage space, as well as the OSs.

- Platform as a Service (PaaS): it permits to rent development platforms, by choosing software development tools and data base managers, leaving out hardware and software resources.

- Software as a Service (SaaS): it permits to rent a specific enterprise software and some determined space for the data associated to it, permitting the access through any device, regardless its type.

- IT as a Service (ITaaS): it permits to rent virtualized IT services according to the needs and business model of each organization, chosen on demand.

Besides, those different types of cloud services models may involve diverse conceptual blocks within a computing server [27], as shown in Figure 2.1.

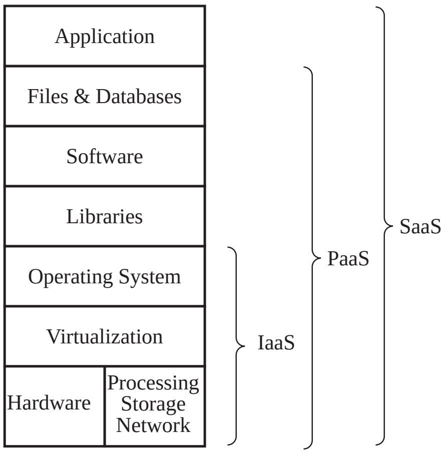


Figure 2.1: Cloud service models.

Therefore, it may be said that cloud computing offers a new model which may be customized for each kind of user and situation, as it allows ubiquity, different levels of virtualization and on-demand scalability. This way, the main benefits when adopting cloud services are cost reduction, including hardware and software acquisition and maintenance, enhanced security as cloud premises implement high-end hardening measures and reliability in the event of potential shortages due to backup policies.

According to the aforementioned characteristics, it may appear that cloud computing might be an interesting candidate for providing services to low-resource computing devices, as most of their tasks could be derived to the cloud. In this context, IoT may be described as the aggregation and interconnection of physical objects through the Internet by means of integrated sensors and specific software in order to forward data on to some servers or systems. In other words, IoT may be seen as the connection of regular objects to the Internet so as to analyze the data they provide [141].

It may be noted that basically any physical device, such as domestic, industrial or any other kind, might be connected to the Internet, hence leading to a hyperconnected scenario where the physical and the virtual worlds may cooperate, allowing for the deployment of digitalization solutions or cyber physical systems. This is possible thanks to some recent advances in diverse technologies, such as low-cost and low-power sensors, machine learning, data analytics, big data and artificial intelligence.

With respect to the architectures appropriate for IoT networks, it is to be considered that IoT devices have constraint features regarding processing, storage, memory, bandwidth and power consumption. Therefore, it may seem clear that the use of a central server in order to collect and redistribute all messages among IoT devices might be the ideal solution, as they may minimize the amount of resources needed.

In this context, the central server, also known as broker, may register all IoT devices, as well as it may receive all the messages sent from the source IoT devices, and in turn, it may forward all those message on to the destination IoT devices, according to some predefined criteria. Hence, it could be said that the broker might act as the hub in a star topology, whereas the IoT devices might act as the spokes.

Furthermore, the main approach followed by IoT communications is called Publisher/Subscriber (Pub/Sub), where a software or hardware agent in an IoT device called the publisher, which usually comes equipped with some kind of sensor to measure the environment, gets a measure from it and sends a message with a certain topic to the broker, which in turn, forwards it on to the agents in all other IoT devices being subscribed to that topic, called the subscribers, which usually come equipped with some sort of actuator to act on the environment upon the messages received.

Moreover, there are two ways to implement the message pattern of Pub/Sub. The first one is called message queue, where the broker creates a specific queue of messages of each subscriber, which will be displayed when it gets connected, as it happens in an online forum. Otherwise, the second one is called message service, where the broker forwards messages on as they get in, hence if a subscriber is not connected, it will not receive such a message, as it happens in an online chat. Anyway, the most popular Pub/Sub protocol is Message Queue Telemetry Transport (MQTT), which works according to the message service way, but having some message queue features.

Focusing on the features of IoT devices and how their communication protocols work, it may be clear that cloud computing might not be the optimal solution, as their resource constraints regarding bandwidth and computing power may not make them feasible due to excessive latency and jitter issues because of the overdue time elapsed when sending messages to the cloud and receiving them back according to IoT needs. Besides, the same reasoning applies to communications in real time.

However, those drawbacks associated to cloud environments might be overcome if the remote computing facilities may get closer to the end users, as the time elapsed may get considerably reduced. For that reason, an extension of cloud computing was designed where the computing facilities were located just at the edge of the network, which was called fog computing [31]. That way, each IoT device may get an associated computing resource located on a host within the fog facility, hence accounting for very short values of latency and jitter [197].

On the other hand, it is to be noted that many IoT devices may be moving around, which are known as moving IoT devices. In this sense, it might be necessary to migrate their computing resources located in a certain physical host within the fog DC to another physical host being closer to the current location of the IoT device [39]. If those remote computing resources are considered to be a VM for simplicity purposes, regardless the fact that those may be a VM, a docker container or another type, then that migration process may be called VM migration and takes place from a source host to a destination host, where the associated VM goes through one of the shortest internetworking paths available between both hosts.

In other words, users getting into a fog environment have some computing assets allocated [109], which are situated within any of the hosts being part of the fog architecture [153]. Those computing resources may usually be VMs, but may also have other forms [14], such as containers or dockers. Additionally, it is to be mentioned that the computer power of the asset can also be allocated in alternative fog environments, such as Arduinos, Raspberry Pis, smartphones, tablets or computing facilities in cars.

Nevertheless, within the context of this thesis, a VM will be referred to as any computing asset allocated within a host being part of a DC, and as such, those VMs might be migrated from one host to another in order to keep them as close as possible to their corresponding associated users when they move around [98].

Regarding fog computing, it may be considered as an extension of cloud computing which brings the computer assets closer to the edge of the network [126]. It is to be noted that cloud deployments have been around for a while and, at this stage, it may well be said that this technology is totally integrated into our ordinary life, as cloud deployments account for many types of remote computing systems and storage capacities being used by a great deal of users on a daily basis.

Likewise, it may be said that cloud computing has contributed to achieve ubiquity related to computing assets [32], in a sense that those do not need to be next to the end user any more, but they may be located anywhere, as long as there is a necessary connection to get there. Some examples of popular cloud solutions regarding computing services [55] are Amazon Web Services, Microsoft Azure or Google Cloud Platform, whereas some instances of popular cloud storage solutions [50] are iCloud, Dropbox or Box.

However, those advantages turn into disadvantages when dealing with some specific conditions, such as real time scenarios or resource-limited devices [236]. The former case involves a reduced response time [117], which may not be enough for devices sending their requests up to the cloud and waiting to get a response back out of it [217], depending on the distance and the available bandwidth [109].

The latter case deals with scenarios with devices having limited resources, such as computing capacity, RAM memory scarcity, storage shortage, bandwidth insuffiency or power undersupply [218]. The typical scenarios for this conditions may be IoT devices, although others may also be affected.

Therefore, cloud paradigm has been extended in order to cope with the aforesaid situations and has evolved to fog paradigm, also known as fog computing [53]. This means that computing resources are located at the the edge of the network, thus, being far closer to the end user [63]. Hence, fog environments permit remote computing in the aforesaid scenarios, namely, real time implementations and IoT deployments, as it takes far less latency, jitter and bandwidth to reach the remote computing resources than it would if cloud facilities are used [237].

It is also to be considered that fog deployments are composed of a bunch of servers, those being interconnected by some switches, which are organized according to a certain topology within the fog domain [11]. Those servers are usually called hosts, as they basically *host* the remote computing resources assigned to the users within the fog environment, where those computing assets are generically called VMs, even though other types may arise, as stated before [232]. Obviously, each host may keep several VMs, depending on their sizes.

Another point to be taken into account is that fog premises are usually connected to some cloud deployment [52], thus allowing it to be a backup system for the fog, hence permitting a VM to be diverted into the cloud in case there are no enough resources to allocate a certain VM in a host being part of the fog [28]. It may seem clear that the capacity of the resources within a cloud may well be far bigger than that of a fog, just because of its geographical scope.

Additionally, on the one hand, it is to be noted that the fog environment is usually tied to the Open Fog Reference Architecture [142] (adopted by the Institute of Electrical and Electronics Engineers (IEEE) as the official standard for fog computing), containing many applications, such as surveillance, autonomous vehicles or smart cities [3]. On the other hand, Multi-Access Edge Computing (MEC) is defined by the Industry Specification Group within the European Telecommunications Standards Institute (ETSI) [58], including many applications, such as caching scenarios, location services or extended reality scenarios, such as Virtual Reality (VR), Augmented Reality (AR) or Mixed Reality (MR) [239].

As a side note, it is also important to note that both MEC, which is also known as edge computing [125], and fog may both be seen as instances of distributed resources being located on the edge of the network [75] and being able to connect to external cloud nodes. However, the former puts the focus on the execution of computing processes close to end users, whilst the latter does it on the overall architecture [24].

Nonetheless, in both cases, communications between hosts and computing assets at the edge of the network do usually take place by means of mobile and wireless communications [8], which is called as Radio Access Network (RAN), whose performance may well be enhanced by the rise of the next generation networks, such as 5th Generation of Cellular Networks (5G) and the 6th Generation of Wireless Fidelity (WiFi-6) [146], and those being under development now, such as 6th Generation of Cellular Networks (6G) [235] and the 7th Generation of Wireless Fidelity (WiFi-7) [110]. However, when dealing with fog, communications with cloud facilities may also involve the use of wired communications, such as twisted pairs and mostly fiber optics.

Moreover, it may be considered that fog does usually imply a small DC, whist MEC might not be the case. Regarding the DCs, their size ought to be according to the amount of users and their workloads expected within the fog domain [229]. Therefore, the capacity of each host might not generally be as high as those implemented in cloud deployments, and likewise, the number of them could not need to be as big [79]. Hence, the topologies required for such DCs may well be quite simple in most deployments, thus allowing for easier designs, although in some cases, it might be a bit more complex, depending on expected amount of users and traffic [231].

Focusing on IoT deployments, they perfectly fit the features of fog computing [122], where communications take place between each of those devices and a central server called broker [221]. This entity may have enough resources to act as a hub for all communications with all the IoT devices [199], while maintaining a register of those being connected at any time [130].

Traditional networking protocols working with paradigms such as client-server or peer-to-peer do not match the aforesaid behaviour, thus, the central-server paradigm may be put in place [1]. There are different protocols working that way, most of them employing a Pub/Sub approach [193], whilst others use a Remote Procedure Call (RPC) methodology [64]. The main difference between them is that in the former, the central server plays a crucial part, as it manages all of the traffic between publishers and subscribers, whereas in the latter, its role is just to forward the flows of traffic among end users, not to manage them.

Pub/Sub outlook is far more extended regarding IoT communication protocols, where the most common ones are MQTT and Constrained Application Protocol (CoAP). In this context, the former may seem to be the most popular as it brings extra features [123], such as different levels of Quality of Service (QoS) [194], the option of allowing an Secure Sockets Layer (SSL)/Transport Layer Security (TLS) extra layer for encrypted communications [159] and the possibility of retaining certain messages, or even the whole thread, hence, changing the way it behaves by default [131]. As per the latter, its use is not so widespread, although it is more lightweight and efficient, its energy consumption is lower and its functionality is based on REST architecture web services (RESTful) [224].

Eventually, IoT moving devices require a special attention, as the movement of the devices throughout the fog area implies that the remote computing assets may end up being too far away from its associated IoT device [133]. Therefore, a VM migration mechanism may be put in place in order for the computing resources to try and follow as close as possible its moving user [119], thus keeping at a minimum the values for latency, jitter and bandwidth required.

2.1.2 VM migration techniques

VM migration may be seen as the procedure to move remote computing assets, which will be referred to a generic VM within this thesis, no matter which type it may be, from one host to another one belonging to the same switching infrastructure [107]. This way, in the context of this thesis, VM migration in a fog environment may be considered as the action of migrating a VM from a source to a destination host [62], being both interconnected by the topology belonging to a single fog domain [26].

Focusing on VM migration, three main approaches may be specified, depending on how a VM is treated during the migration process from source host to destination host [38]. The first option is *cold* migration, which is the case where a VM is shut down prior to moving it. The second option is *hot* migration, where its OS is suspended prior to moving it. The third option is called *live* migration, where services keep running seamlessly throughout the whole process.

It may seem obvious that the last one accounts for the best solution. Moreover, in summary, when dealing with live VM migration in DC premises [115], some advantages may arise, such as load balancing, hardware maintenance, power management and fault tolerance. However, it is not possible to have a VM always on service, as there must be some time range off to bring the core features of the VM from source to destination. This point leads to the establishment of some key parameters to obtain the performance of a live VM migration process [85].

Three parameters are usually taken into consideration [135]. First off, the *downtime*, which accounts for the amount of time that the VM is stopped during the whole migration. Then, the *total migration time*, which represents the elapsing time to complete the process. Finally, the amount of dirty pages migrated, which constitutes the data that changed during the migration due to the current use of the source VM, which has to be sent to the destination VM.

As the third parameter is dynamic, in a way that it may not be measured beforehand, as it depends on the workload being carried out by the source VM at migration time, the first and second parameters are the ones to be dealt with in order to achieve a tradeoff between them both [13]. In order to attain this target, some factors are involved, such as connections to local devices and network interfaces, although the key player is memory transfer, which is the one to be tackled [156].

Regarding memory transfer, it may be divided into three different stages, called *push* phase, *stop-and-copy* phase and *pull* phase, which take place throughout the whole live VM migration process [203]. The first one involves the beginning of such a process, where source VM keeps running. The second one is the most critical, where source VM gets halted, dirty pages are copied around, and afterwards, destination VM is booted up. The third one requires that the destination VM is running and source VM provides any requested new page not being copied yet.

In order to attain an efficient way to undertake the process of live VM migration, a common approach is to focus on just one or two of the aforesaid stages. This way, some popular implementations make use of pure demand-migration, pure stop-and-copy or post-copy live VM migration. Nonetheless, it may appear that the technique being the most efficient is iterative *pre-copy* live migration [74]. Basically, it combines a bounded interactive push step along with a very short stop-and-copy step, considering that several iterations may occur until all of the dirty pages are transferred [97].

Focusing on the iterative pre-copy live VM migration, it gets developed through 6 stages, where the whole VM transaction between any two given hosts takes place [144]. The timeline of such six steps summarized in Figure 2.2.

- Stage 1 is called *pre-migration*, and it involves the process to select a candidate to be the destination host which has enough resources.
- Stage 2 is named *reservation*, and it covers the process of allocating resources on the destination host prior to start the migration

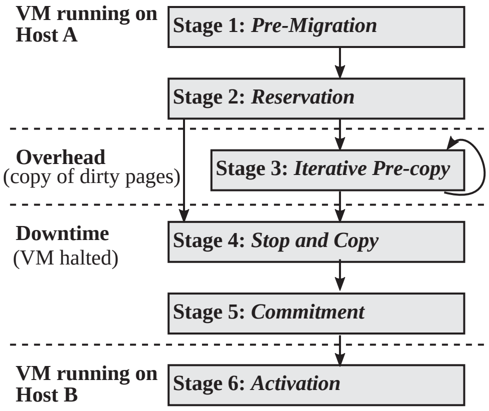


Figure 2.2: VM migration timeline.

- Stage 3 is called *iterative pre-copy*, and it includes two subphases. To start with, the complete RAM is sent over in the first interaction, and in turn, dirty pages get sent over in the next iterations.

- Stage 4 is named *stop-and-copy*, and it comprises the process where source VM gets halted to make possible the copy of the CPU state and the rest of inconsistent pages to the destination VM.

- Stage 5 is called *commitment*, and it contains the part where the destination host acknowledges that it has already received a consistent copy of the source VM from the source host, and this one acknowledges it back before discarding the source VM, which is the original VM.

- Stage 6 is named *activation*, and it embraces the point where the destination VM, which is the migrated VM, gets activated, and the proper device drivers get attached to the migrated VM.

As a side note, it may be said that *post-copy* live VM migration undertakes a reversal approach from the pre-copy technique [89]. The key point is that, in this case, source VM is halted right at the beginning and a copy of the CPU state is immediately transferred to destination, where the VM is then resumed, and in turn, dirty pages are sent across. This may account for a shorter downtime as the CPU state is sent earlier, but performance may well be degraded for some time until all dirty pages have been transferred [42].

A way to overcome this issue may be to propose a *hybrid pre-copy and post-copy* technique [48], where a *pre-copy* phase takes place at the beginning, where the VM working set, also called hot pages, are migrated from source to destination, then, the CPU state gets transferred and the execution swaps from host to destination, and in turn, the rest of pages, also called cold pages, are migrated across.

It is to be noted that the use of the post-copy fashion implies that not all dirty pages may be in the destination VM when it boots up, bringing the possibility of having degradation issues if some dirty pages are demanded and they have not already been transferred. However, this issue does not apply when using the iterative pre-copy technique described above [25].

Some other complex techniques have been proposed, such as performing the live VM migration with a machine learning approach [103], using prediction algorithms so as to forecast the most likely migrations [128], employing algorithms to assure energy-efficiency and quality-aware consolidation [121], or the utilization of adaptive algorithms to attain faster migrations [124].

Those methods might achieve better performance than the iterative pre-copy technique, although they all introduce a certain degree of complexity when dealing with the different algorithms proposed to accelerate the live VM migration [192], which may make them harder to follow because of the interaction of such algorithms [70]. Therefore, this point stands for the selection of the iterative *pre-copy* live VM migration technique to carry out the formal models to be proposed in this thesis.

2.2 Modelling with ACP

To start with, a model may be defined as an abstraction of reality according to a certain conceptualization [84]. This way, a particular set of concepts may need to be employed in order to obtain an abstract representation of some aspects related to certain entities within a given domain, although selecting different concepts might lead to attain diverse abstract representations of the same reality, hence assuming that a model is an instance of the conceptualization chosen. Likewise, the use of different modelling languages might lead to obtain diverse model specifications.

In this context, it may be noted that models are always approximations of reality [132], thus they should represent the most important features of the expected behaviour of reality in a predictable way, even though it may not hold for other features not being under consideration.

Focusing on systems, it is to be said that a system model may represent the main features of a given system and its environment, where different purposes might be pursued, such as understanding its behaviour or its structure [111]. Besides, a key point to be focused on when dealing with modelling is abstraction, which implies choosing some essential characteristics, while ignoring others not so relevant, thus allowing the design of a variety of models according to the set of parameters required and the level of detail expected in each particular instance.

As per the *language* to be used in the modelling, the first point to be discussed is related to informal and formal. The former is related to natural languages, such as English or Spanish, where each of them have their own syntax and semantics. Hence, it may well occur that some mismatches might be encountered among different descriptions related to the same set of actions [113].

Therefore, in order to avoid so, the latter comes into play, being usually referred to as FDT. Those are a group of heterogeneous tools whose target is to achieve common descriptions for the same actions, overcoming the aforesaid issue found in natural languages [158]. Basically, the target of those tools is to model, analyze and control aimed at non-deterministic discrete event systems [77].

One of the most well-known modelling formalisms are *Petri nets*, which were first developed by C.A. Petri back in the 1960s to describe distributed systems by means of places and transitions, in the form of circles and bars, where arcs play the part of joining them together, either pointing from places to transitions, or the other way around [150]. Besides, places may contain a certain number of tokens, usually shown as black circles, which permit the activation of a transition if all its incoming places contain at least a token in it. Putting toghether, Petri nets are aimed at describing concurrency and synchronization in distributed systems.

Apart from its ordinary behaviour, it allows for several variations, such as coloured, probabilistic or timed, which may be applied to a wide range of areas. In this sense, timeless Petri nets are useful when studying qualitative properties in concurrent asynchronous systems, although the concept of time needs to be included when assessing quantitative properties in concurrent synchronous systems, leading to the distinction between Deterministic Timed Petri nets and Stochastic Timed Petri nets [228].

Furthermore, all sorts of Petri nets may be broadly classified by the content of their places, either representing boolean values, such as those being marked by at most one unstructured token, or representing integer values, such as those being marked by more than one unstructured token, or even representing high-level values, such as those being marked by a set of structured tokens [23].

A different common tool is Specification and Description Language (SDL), which is an object-oriented language defined by International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) as recommendation Z.100, whose development kicked off back in the 1970s. It contains a set of extended Finite State Machines (FSM) running in parallel, consisting of different components, which are represented as a diagram [154]. The main ones are the structure, which comprises hierarchical levels such as system, blocks, processes and procedures, the communication, which includes asynchronous and synchronous signals along with channels, the behaviour being described as processes with their corresponding FSM, and eventually, different types of data.

Another widely used tool is Protocol/Process Meta Language (Promela), which is a specification language similar to *C* and developed in the 1980s. It emphasizes process synchronization and coordination by using asynchronous processes representing entities specifying behaviour, whereas synchronous and asynchronous message channels and shared variables define the environment where processes are running [61]. Furthermore, a program in Promela may be analyzed by means of a model checker called Simple Promela Interpreter (Spin), aimed at verifying its correctness and linear temporal logic properties [92].

An additional popular tool is Unified Modeling Language (UML), which was first developed back in the 1990s and it is supported by the Object Management Group. It is a graphical language consisting of different hierarchical diagrams, where the most broad categories are structural and behavioural diagrams, which are broken down into a range of subcategories [44]. It is based on object-oriented programming, as class diagrams contain classes, where some relationships are established among them, such as association and inheritance, thus conditioning their attributes and methods. Besides, classes are considered as templates to create objects, whose behaviour do not only depend on its inputs, but also on its preceding states, where all those possible states may be included into state machine diagrams.

Moreover, queueing networks is also a very popular formalism when dealing with stochastic models [12]. It got impulsed at the beginning of the 20th century by the studies of Erlang related to queueing theory. Basically, it takes into account the waiting time in order to get served, which is related to the arrival rate, and the processing time, which is related to the service rate. Both rates are usually stochastic, which means that the time being spent into the system regarding both tasks, also known as sojourn time, is non-deterministic.

In order to properly describe a queueing model, some factors need to be quoted with respect to the queueing system, such as the arrival rate, the queueing discipline, the buffer size or the number of queues, whereas different factors need to be cited with regards to the processing system, such as the number of servers, the capacities of each server or the service discipline [104].

Regarding the notation of queueing networks, the one proposed by Kendall [108] is commonly used in the literature. It has the form A/B/m/K/M, where A establishes the arrival process, B denotes the service process, m applies for the number of servers, K determines the capacity of the queue, and M does it for the customer population. In this context, the first two variables account for the type of probability distribution, which may result in M for Markovian (exponential), G for general or D for deterministic, whereas the rest of variables account for natural numbers or infinite. Not all variables are used at all times, but only the minimal amount of them so as to clearly describe the queueing network, such as M/M/1, M/M/∞, M/M/2/2 or D/G/2/4.

In summary, each of those may have different characteristics, making them more suitable for some given conditions, even though they may not be the fittest in different ones [195]. In other words, there is not an optimal FDT suitable for all scenarios, but each FDT may be more interesting in some circumstances, whereas that might not be the case in other ones [202].

Additionally, an algebraic approach may also be included into FDT, specifically regarding process algebras, those being a framework with diverse approaches to model concurrent systems in a formal way [17]. There are many of them around, some being implemented in software tools in order to automate strings of actions [82], whereas others being thought to be used in a manual approach [100], in the same way as traditional algebras. Also, they are usually timeless, unless a time extension is added.

Regarding these process algebras to be dealt with an analytical approach, it is to be said that those were the pioneers. Specifically, in the early 1980s, there were two main approaches differing in their semantics [149]. On the one hand, Communicating Sequential Processes (CSP), launched by Hoare [90], presented denotational semantics, generally as traces. On the other hand, Calculus of Communicating Systems (CCS), promoted by Milner [129], proposed operational semantics, generally as labelled transition systems.

Apart from their different syntax and semantics, it is to be said that both CSP and CCS share common grounds because they both build up mathematical models of processes, being considered as agents, which may act and interact with other agents and their common environment [139]. Moreover, both try to find out the behaviour of the model, even though the former does use the notion of failure equivalence if, and only if, two trees have identical failure sets, whilst the latter does employ the notion of observation equivalence, being defined as the smallest relations satisfying a given set of axioms, expressed as a synchronization tree [33].

However, neither of both approaches seemed to fully permit the establishment of an abstract and generalised system of axioms to deal with processes, and that is why Bergstra and Klop [21] took both as a starting point and developed a brand new approach called ACP, whose syntax is genuine, although inspired on them, and whose semantics are axiomatic and algebraic.

It is to be noted that ACP is aimed at capturing the behaviour of concurrent systems, such as network protocols being used in communication systems, or any other kinds of distributed systems [106]. ACP defines a set of axioms in order to perform algebraic derivations on process terms, in a similar manner as other abstract algebras do, such as group theory, ring theory or field theory [148].

Therefore, ACP may permit to specify and verify concurrent communication protocols by means of a set of rules defining their behaviours [16]. It is to be reminded that ACP abstracts away from the real nature of processes, just focusing on behavioural equivalences, being referred to as rooted branching bisimilarity, thus making possible to reason about relationships within distributed systems [81].

On the other hand, ACP may carry some disadvantages, mostly due to its abstract approach, which may appear to be too generic in some cases, or otherwise, it does not contain time variables, although some extensions does take time into account. Nonetheless, those features enforce ACP as a convenient tool to undertake the models proposed, as it allows to achieve simpler analytical models.

2.2.1 Using FDT modelling in cloud/fog environments

There is not a great deal of information in the literature regarding the formal modelling of fog computing domains. However, the following papers may well be cited as they do it by making use of the most important FDT:

Regarding *Petri nets*, [196] presents a model of integrated systems composed by cloud, fog and IoT all together. Furthermore, a stochastic Petri net is proposed in [208] to model a hybrid cloud and a fog scenario, where variables such as latency, workload or computational capacity are used to establish under which circumstances it is more interesting to use each scenario.

Moreover, [237] puts forward a model with further security settings for IoT devices using high level Petri nets. Likewise, [161] shows a model with coloured Petri nets. Besides, a dynamic resource allocation strategy for fog environments is presented based on priced timed Petri nets in [137], as well as in [138], aiming to achieve a higher efficiency than different static resource allocation strategies.

Focusing on SDL, [207] presents the challenges that IoT systems pose to SDL modelling, such as signal delays, interactions with external systems and the signal handling to multiple recipients. On the other hand, [225] makes use of SDL to simulate Wireless Sensor Networks (WSN) and [76] does it to simulate ubiquitous systems.

With respect to Spin/Promela, [47] depicts a model for the physical and Media Access Control (MAC) layers according to the Zigbee standard in order to analyze energy consumption of the radio interface. Besides, [41] proposes a model checking utility so as to target IoT behaviour, whereas [238] presents a process mining approach to IoT environments.

Additionally, [43] presents a formal approach using Promela and a system verification with Spin for MQTT in the context of communicating vehicles. This method is also employed by [226] focusing on CoAP, whereas [10] brings forward a formal approach to QoS in MQTT. It is to be reminded that both protocols, namely CoAP and MQTT, were specially designed for IoT scenarios with a scarcity of resources, as exposed in [91].

Talking about UML, it is to be said that [134] exhibits a model by means of an extension to iFogSim aiming at simulating scenarios searching for the optimization of data placement in fog and IoT contexts. Also, [9] exhibits a model and its verification for MQTT, whereas [73] proposes a model for predicting the power consumption in IoT devices.

With regards to other FDT, [147] introduces a LNT model so as to verify a failure management protocol for stateful IoT applications, specifically created for highly distributed scenarios. Likewise, [127] presents a BRS based approach to support interaction among IoT devices, whilst [145] presents a model verification of IoT usual behaviour through PRISM.

Moreover, [51] shows models and verifications for MQTT and CoAP by means of Event-B, whereas [2] presents a statistical model checking for MQTT, whilst [95] proposes a specification and verification of MQTT by using UPPAL. Also, [216] makes a review on different formal approaches for service composition, whereas [215] does it for IoT applications.

Eventually, sticking to process algebras, it may appear that those are quite adequate to model fog environments due to its abstraction and notation features. In this sense, there are some contributions, such as [118], where a model for a Smart

Emergency Evacuation System (SEES) is proposed using dT-Calculus, or [15], where a model for MQTT is presented in TPi.

Furthermore, [213] depicts a model for a generic IoT system employing dTP-Calculus, whereas [72] shows a similar model by means of π-Calculus, whilst [114] does it by using CaIT, and [29] makes use of IoT-LYSA.

In conclusion, there are many process algebras around, each one with its own characteristics. However, it may be taken into consideration that the point in this thesis is to use an analytical approach, thus discarding automatic tools, in a way that applying this perspective to process algebras permits to enhance the focus on the details occurring at each step within a procedure being modelled, with different entities taking part in an asynchronous manner. This analytical step-by-step approach may help better understand the basics and the dynamics that are happening along the way compared to simulation-based approaches, where some details may go unnoticed.

The procedure selected herein is the live VM migration occurring among the hosts being part of a fog deployment. In that sense, the more abstract a process algebra is, the better it may describe the behaviour of the system without having to deal with the real entities involved, along with their own particular features, and that is why ACP is chosen to undertake the models within this thesis.

2.2.2 ACP modelling

Focusing on ACP, it is to be remarked that there are some atomic actions, which may not be divided into smaller ones, such as send and read [18]. The former may be considered as writing a message into a communication channel, whereas the latter may be seen as receiving a message from a communication channel.

Specifically, sending a message d through channel i may be defined as $s_i(d)$, whereas reading a message d through channel i may be stated as $r_i(d)$.

Then, some operators are defined in order to deal with those atomic actions [67]. There are some of them, although the most important ones are the following:

- *Sequential* operator, denoted by the $\cdot$ sign, meaning that the first process is executed, and in turn, the second process will. It allows more than two arguments.

- *Alternate* operator, denoted by the + sign, meaning that either the first or the second processes are executed. It allows more than two arguments.
- *Merge* operator, denoted by the $||$ sign, meaning that the first and the second processes are executed concurrently. It allows more than two arguments.
- *Conditional* operator, denoted by $\mathit{True} \lhd \mathit{condition} \rhd \mathit{False}$, meaning that if the condition is met, then the left argument is executed, otherwise, the right argument will. It allows nesting other conditional operators.
- *Encapsulation* operator, denoted by ∂_H, meaning that only internal communications are allowed, whilst its related atomic actions go to deadlock, thus being irrelevant. It is to be said that set H contains all internal atomic actions, which may all be converted into deadlock after applying this operator.
- *Abstraction* operator, denoted by τ_I, meaning that only external communications are allowed, whereas internal communications go to silent step, thus being irrelevant. It is to be noted that set I contains all internal communications, which may all be turned into silent step after the application of this operator.
- *Guarded linear recursion*, meaning that a process may be described in a recursive manner, representing its repetition in a cyclic way.

The proper way to use ACP to specify and verify concurrent systems is achieved in three steps. First of all, a model for each relevant entity within the system is to be made up individually [71], by means of describing what each entity does with the help of a string of atomic actions along with its respective operators.

After that, a specification is to be built up, by connecting all those entities in a concurrent manner [112]. To do so, the first step is to run the merge operator, followed by executing the encapsulation operator, in order to find out which channels result in communications and which others are irrelevant.

Finally, a verification is to be performed [68]. The right way to proceed is to first obtain the external behaviour of the model by applying the abstraction operator on the specification previously achieved, and in turn, the external behaviour of the real system by means of ACP expressions. Then, both external behaviours may be compared, this is, that of the model and that of the real system, which may be both manipulated by using equational logic so as to try to equate them if at all possible, which may imply that their process graphs are both behaviourally equivalent.

After that, two conditions have to be checked, such as both external behaviours share the same string of actions and the same branching structure, and if both points are met, then it may be said that they are both *rooted branching bisimilar* [66], which is a sufficient condition to get the model verified [65], meaning that the model presents the desired external behaviour, where each input produces the expected output.

In this context, Figure 2.3 exhibits a simple example of a bisimulation relation between two process terms. By comparing the labelled transition systems referred to both expressions, it may seem clear that the *same string of actions* are executed by both, as at first, they run either action x or action y, and in turn, they run action z, which leads to a successful termination. Additionally, it may be obvious that the *same branching structure* is found in both, as the layout of the actions being run match each other. Therefore, there is a bisimulation equivalence between both actions, which implies that a behavioural equivalence may be established between both process terms, as the equational logic of ACP is sound and complete.

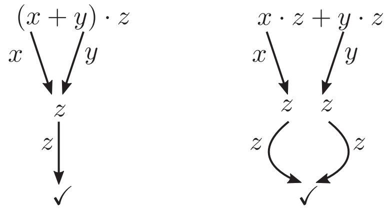


Figure 2.3: Example of bisimilarity.

On the other hand, when working with distributed systems, non-determinism may well be expected, which might involve ever growing terms as the number of concurrent entities gets higher, thus making harder to deal with them. However, the appropriate terms related to a given number of concurrent entities may be easily exposed by means of a particular expression, which permits to identify all available relationships among those terms.

It is to be reminded that when dealing with the concurrent operator, also known as merge operator and denoted by the sign $||$, it may be developed by means of two equivalent operations [22]. The first one is called left merge, given by the sign $\llfloor$, stating that the initial transition of the left entity is run in the first place, and then, the behaviour of the rest is that of a merge. The second one is named communication merge, given by the sign $|$, stating that a communication between the initial transitions of the entities involved are run together in the first place, and then, the behaviour of the rest is that of a merge.

The relationship among those three operators is given by the *Expansion Theorem*, by Bergstra and Klop [99], where the term X^i indicates all entities involved except the i-th one, this is, $X^i = \{X_1 \mid\mid \cdots \mid\mid X_n\} - \{X_i\}$, and the term $X^{i,j}$ denotes all entities involved except the i-th and the j-th ones, this is, $X^{i,j} = \{X_1 \mid\mid \cdots \mid\mid X_n\} - \{X_i, X_j\}$. This all is exhibited in (2.1).

$$\big(X_1 \mid\mid \cdots \mid\mid X_n\big) = \sum X_i \llfloor X^i + \sum \big(X_i \mid X_j\big) \llfloor X^{i,j} \tag{2.1}$$

This expression may be tailored for any number of entities being executed in a concurrent manner in order to obtain the behaviour of a concurrent group of objects, taking into account that the number of terms obtained for n entities is given by the n-th triangular number, which may be obtained by the binomial coefficient $\binom{n+1}{2}$. Hence, the overall number of terms attained grows according to an arithmetic progression of the second order for n concurrent items.

The following examples show the expressions to be applied for a small number of concurrent items, as seen in (2.2), (2.3),(2.4) and (2.5).

$n = 2$:

$$\big(X_1 \mid\mid X_2\big) = X_1 \llfloor X_2 + X_2 \llfloor X_1 + \big(X_1 \mid X_2\big) \tag{2.2}$$

$n = 3$:

$$\begin{aligned}\big(X_1 \mid\mid X_2 \mid\mid X_3\big) =& X_1 \llfloor \big(X_2 \mid X_3\big) + X_2 \llfloor \big(X_1 \mid X_3\big) + X_3 \llfloor \big(X_1 \mid X_2\big) + \\ & \big(X_1 \mid X_2\big) \llfloor X_3 + \big(X_1 \mid X_3\big) \llfloor X_2 + \big(X_2 \mid X_3\big) \llfloor X_1\end{aligned} \tag{2.3}$$

$n = 4$:

$$\begin{aligned}(X_1 \mid\mid X_2 \mid\mid X_3 \mid\mid X_4) =& X_1 \mathbin{⌊\!⌊} (X_2 \mid X_3 \mid X_4) + X_2 \mathbin{⌊\!⌊} (X_1 \mid X_3 \mid X_4) + \\ & X_3 \mathbin{⌊\!⌊} (X_1 \mid X_2 \mid X_4) + X_4 \mathbin{⌊\!⌊} (X_1 \mid X_2 \mid X_3) + \\ & (X_1 \mid X_2) \mathbin{⌊\!⌊} (X_3 \mid X_4) + (X_1 \mid X_3) \mathbin{⌊\!⌊} (X_2 \mid X_4) + \\ & (X_1 \mid X_4) \mathbin{⌊\!⌊} (X_2 \mid X_3) + (X_2 \mid X_3) \mathbin{⌊\!⌊} (X_1 \mid X_4) + \\ & (X_2 \mid X_4) \mathbin{⌊\!⌊} (X_1 \mid X_3) + (X_3 \mid X_4) \mathbin{⌊\!⌊} (X_1 \mid X_2) \end{aligned} \tag{2.4}$$

$n = 5$:

$$\begin{aligned}(X_1 \mid\mid X_2 \mid\mid X_3 \mid\mid X_4 \mid\mid X_5) =& X_1 \mathbin{⌊\!⌊} (X_2 \mid X_3 \mid X_4 \mid X_5) + X_2 \mathbin{⌊\!⌊} (X_1 \mid X_3 \mid X_4 \mid X_5) + \\ & X_3 \mathbin{⌊\!⌊} (X_1 \mid X_2 \mid X_4 \mid X_5) + X_4 \mathbin{⌊\!⌊} (X_1 \mid X_2 \mid X_3 \mid X_5) + \\ & X_5 \mathbin{⌊\!⌊} (X_1 \mid X_2 \mid X_3 \mid X_4) + (X_1 \mid X_2) \mathbin{⌊\!⌊} (X_3 \mid X_4 \mid X_5) + \\ & (X_1 \mid X_3) \mathbin{⌊\!⌊} (X_2 \mid X_4 \mid X_5) + (X_1 \mid X_4) \mathbin{⌊\!⌊} (X_2 \mid X_3 \mid X_5) + \\ & (X_1 \mid X_5) \mathbin{⌊\!⌊} (X_2 \mid X_3 \mid X_4) + (X_2 \mid X_3) \mathbin{⌊\!⌊} (X_1 \mid X_4 \mid X_5) + \\ & (X_2 \mid X_4) \mathbin{⌊\!⌊} (X_1 \mid X_3 \mid X_5) + (X_2 \mid X_5) \mathbin{⌊\!⌊} (X_1 \mid X_3 \mid X_4) + \\ & (X_3 \mid X_4) \mathbin{⌊\!⌊} (X_1 \mid X_2 \mid X_5) + (X_3 \mid X_5) \mathbin{⌊\!⌊} (X_1 \mid X_2 \mid X_4) + \\ & (X_4 \mid X_5) \mathbin{⌊\!⌊} (X_1 \mid X_2 \mid X_3) \end{aligned} \tag{2.5}$$

It may seem clear that the more entities are interacting in a concurrent manner, the more process terms are generated, hence, the more complicated get the calculations. However, the application of the *encapsulation operator*, over a set H containing all internal atomic actions, cancels a great deal of such process terms, as it converts atomic actions into communications, bringing those atomic actions to deadlock, except for those atomic actions related to getting in and out of the system, meaning terms coming in or going out of the system being modelled. Therefore, results are much more manageable, even though there are many entities working concurrently, as only a limited number of channels will be taken into account, namely, those where communication happens, and thus, discarding the rest of them.

Furthermore, the application of the *abstraction operator*, over a set I containing all internal computations and internal communications, masks all internal actions which are irrelevant for the external behaviour of the model, thus showing such a external behaviour, and making possible to compare it with that of the real system. Therefore, external behaviour of many concurrent devices making up a model, or otherwise a real system, may well end up being portrayed by a handful of process terms, hence making easy to deal with such models.

Additionally, the verification of a model by means of ACP has been described above, where the focus is set on finding a rooted branching bisimulation equivalence (which is a congruence with respect to ACP), where *rooted* imposes that an initial transition may never be a silent step τ (meaning an internal action) and *branching* does it for silent steps being truly silent (being irrelevant). It is to be noted that a bisimulation equivalence is just a particular case where no silent step is involved [80].

The use of ACP to specify and verify models of network protocols or other distributed protocols related to computing issues is not very common, although some attempts have been made, mostly with automated tools based on ACP, such as with Alternating Bit Protocol (ABP) [82], Bounded Retransmission Protocol (BRP) [83] or FireWire (IEEE-1394) [204]. However, it is to be reminded that an analytical approach is being taken in this thesis, thus all specifications and verifications carried out herein will be done that way so as to better appreciate the each step taken.

Along the preparation of this work, some papers have been written in this field, modelling network protocols belonging to different layers within the OSI model. For example, related to *data link layer*, also known as layer 2, Spanning Tree Protocol (STP) has been modelled according to IEEE 802.1D standard [163]. The role of STP is key to avoid logical layer-2 loops, resulting in broadcast storms and MAC inconsistency events [152].

With regards to *network layer*, also known as layer 3, some protocols have been modelled. To start with, Open Shortest Path First (OSPF) version 2, which is the most popular routing protocol among enterprises, standarized by Internet Engineering Task Force (IETF) on some Request for Comments (RFC) and whose target is Internet Protocol version 4 (IPv4) addressing networks [177], [179]. Likewise, Enhanced Interior Gateway Routing Protocol (EIGRP), which used to be a proprietary routing protocol from Cisco Systems, although it was partially released in 2013 to other manufacturers [176]. On the other hand, Protocol Independent Multicast (PIM) and Internet Group Management Protocol (IGMP) protocols, which are necessary for basic configuration of multicast, standardized by IETF by means of some RFCs [165].

With respect of *application layer*, also known as layer 7, some protocols have been modelled as well. For example, FTP, which is a protocol to transfer files from a server to a client, standarized by IETF on some RFCs [164]. Moreover, HTTP, which is the key player protocol regarding Internet browsing and RESTful applications, standarized by IETF on some RFCs [180]. Furthermore, MQTT, which is a message service protocol based on pub-sub paradigm, standardized by Organization for the Advancement of Structured Information Standards (OASIS) and being the most popular at this point for IoT devices [188].

On the other hand, ABP protocol has been extended to include some timing constrictions, in an attempt to study how its behaviour may vary, according to its traditional way. Besides, faulty channels have been included in both directions in order to make it more realistic. Putting all together, manual algebraic derivations have been performed to give a better understanding [162].

In conclusion, ACP syntax and semantics are going to be be used in this work, along with its set of axioms, to undertake manual derivations in order to model distributed hosts being part of a fog computing environment with a given topology, specifically their behaviour regarding live VM migrations related to moving IoT devices.

2.3 Topologies for data centres

To begin with, it is to be said that the network topology is a key factor when designing a DC (Data Centre), as it represents how the different switches supporting a given topology are interconnected, which further influence how far the hosts are located to each other.

With respect to the topology to be employed, it is to be remarked that there are many available options, going from some quite simple to others not that much. Many features are to be taken into account, not only the performance achieved, but also the easiness to implement, to maintain, to forward traffic or to apply load balancing policies. Therefore, a tradeoff may be reached among all those features.

2.3.1 Network topologies

The target in this research work is to establish formal algebraic models for DC topologies to be applied to a fog computing environment, where the number of nodes may be variable, as hosts could be ranging from a few to a great deal, depending on the geographical area or the amount of traffic expected. However, it may well be much lower than that in a cloud computing environment [157], hence the number of hosts may usually be small to medium-sized. This fact establishes that complex designs for DC architecture should be discarded beforehand, as they might introduce more drawbacks than benefits, thus, more simple designs may prevail.

Therefore, to start with the interconnection topologies, the most typical ones employed in LAN connections ought to be considered [222]. The following bullet list shows the most popular ones, along with the main features of each type [69], where Figure 2.4 exhibits the graphical layouts for all those topologies.

- *Point to point*: just two nodes are connected to each other, so it does not apply for more than two.
- *Line*: a string of sequential point to point links, so if a link gets down, the rest of nodes further that link are unreachable.
- *Bus*: a central bus through which all traffic is sent, thus all nodes get that traffic, making the topology inefficient.
- *Ring*: a central ring through which all traffic is sent, thus each intermediate node gets that traffic, making the topology inefficient.
- *Hub and spoke*: a central node manages all traffic to the end nodes, making up for logical star topology. This fits the best so as to interconnect end devices with little computing resources.
- *Full mesh*: all nodes are directly connected to each other, so the distance between any two nodes is just one hop. This is the ideal situation regarding performance, but it is not scalable, because costs grow too much as the number of nodes rise.

- *Partial mesh*: it is like a full mesh, but not all links are available. Most of them might be seen as a *graph-like* topology, although in some specific cases, they might be viewed as a *tree-like* topology.
  - *Tree*: it is a hierarchical design, allowing for better organization, making for a clear implementation, easy maintenance and load balancing options. They may be seen as a *tree-like* topology, where the costs rise with the complexity of its design.
  - *Graph*: it is usually a flat design, with no hierarchy, allowing for a great degree of customization. They may be seen as a *graph-like* topology, where the costs grow depending on the number of links available in the design.

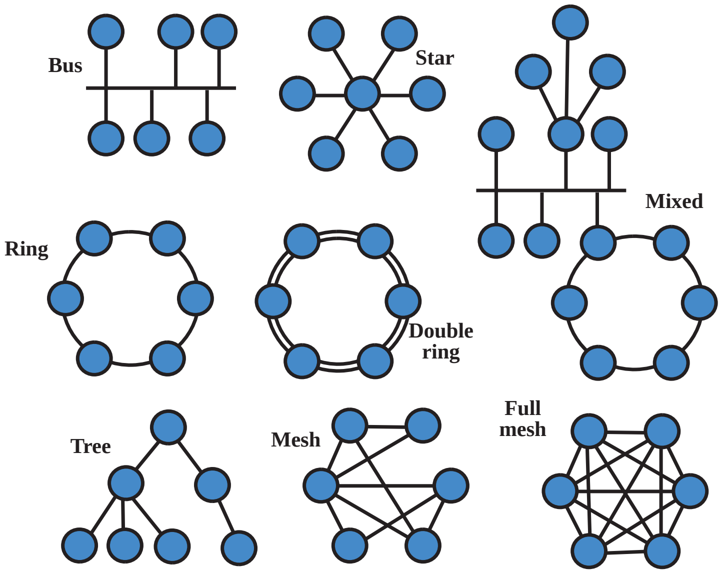


Figure 2.4: Most common network topologies.

Having a look at those topologies, it is to be noted a couple of details. To start with, the first five topologies do not possess redundancy, which may be a must when designing a DC, as single points of failure should be avoided at all costs. Hence, for those topologies to be adequate for DC designs may need to double each link between any two switches so as to provide more than one path for *interswitch* communications (where two hosts are not connected to the same switch), whereas a single link is provided between each host and its connected switch (thus *intraswitch* communications only have one possible path, although it might be possible to install two network interface cards to achieve redundant paths in this case).

Furthermore, it is to be remarked that full mesh topologies present the best performance, as it offers a direct link between any pair of switches, getting multiple redundant paths regardless the destination. However, the costs of acquiring, managing and maintaining such an infrastructure are usually too high, reserving it for just pretty small critical deployments. Hence, the topology being the most popular in real deployments is partial mesh, as implementation costs are lower at the expense of having a limited amount of redundant paths.

The options to be considered for partial mesh designs are usually divided into two different categories, such as *tree-like* and *graph-like*, as shown in Figure 2.5, taking into account that topologies should be easy to deal with. Therefore, some popular options in real DC deployments for tree-like architectures are fat tree and leaf and spine, whereas some well-known options for graph-like architectures are N-hypercube and folded N-hypercube. Hence, those designs are going to be more carefully revised at a later stage. As said before, there may be other topologies offering better performance, but designs are not as easy, and additionally, they are harder to manage, as the degree of difficulty in forwarding algorithms grows with the complexity of the topology.

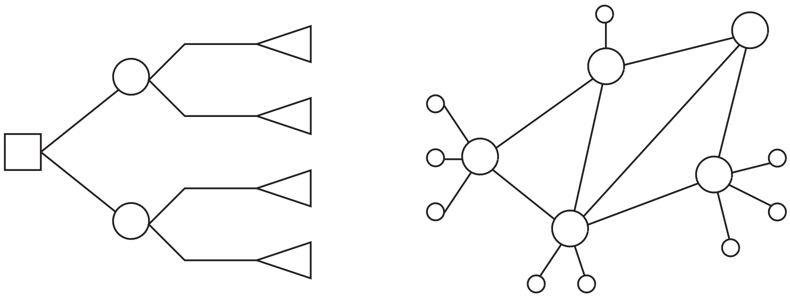


Figure 2.5: Instances of topologies with a tree-like design and a graph-like design.

2.3.2 Redundant topologies proposed

Taking into account the previous statements, the topologies to be studied herein are going to be presented in the following bullet list, comparing them to their LAN counterparts exposed above. It is to be reminded that they are all redundant topologies for switching interconnection in a DC aimed at fog computing, which implies a small to medium amount of hosts therein. Furthermore, simple designs are imposed in order to facilitate the modelling of message forwarding among VMs located in different hosts, as those designs may obtain a convenient tradeoff between the simplicity of forwarding algorithms and the performance achieved.

- *Partial mesh*: like a partial mesh LAN, which includes 4 different designs.
    - *Fat tree, leaf and spine, N-hypercube and folded N-hypercube*
- *Full mesh*: like a full mesh LAN.
- *Quasi full mesh*: similar to full line LAN, even though all opposite links are missing.
- *Redundant hub and spoke*: similar to a star LAN, adding up a second hub for redundancy.
- *Redundant ring*: like a ring LAN, adding up redundant links.
- *Toroidal ring*: similar to a grid LAN, adding up extra links to join outer nodes.

Apart from those, some 6 further graph-like designs are also proposed at a later stage because of their specific features when dealing with forwarding policies. Those may be grouped in pairs, as each couple is closely related. In this sense, de Bruijn graph and reverse de Bruijn graph are built up in an inverse manner, whereas binary grid and Hamming graph are constructed in an opposite fashion, whilst Petersen graph and Heawood graph share the same principles.

Focusing on *partial mesh* topologies, those are the ones being implemented the most in production environments, as they provide an interesting tradeoff between redundancy and complexity. There are many different design options available, although the following ones are going to be studied herein in a more detailed approach:

- tree-like designs
    - *fat tree*
    - *leaf and spine*
- graph-like designs
    - *plain N-hypercube*
    - *folded N-hypercube*

Therefore, the main characteristics of all those topologies may be exposed so as to be able to create mathematical models based on their physical behaviour at a later stage. However, some mathematical fundamentals need to be introduced first in order to better understand the models proposed.

2.4 Summary

In this chapter, the main pillars upon which this thesis dissertation has been built up are being presented. The first one is cloud and fog computing, and specifically, live VM migration among the hosts within a fog domain, where a brief story of computing has been cited from the initial computers back in the middle of the 20th century to the rise of cloud computing at the beginning of the 21st century and the arrival of fog computing nowadays, along with a literature review of those stages.

The second one is modelling, and specifically, doing it by means of a process algebra called ACP. In this sense, the most relevant FDT have been first described, following with the selection of process algebras for modelling communication systems, and choosing ACP as a convenient instance due to its abstract features. Furthermore, a review of literature regarding the application of FDT to the modelling of some aspects of IoT/fog environments have also been exhibited.

The third one is topologies, and specifically, those being applied to small to medium size DC, as fog environments are not dedicated to a big amount of users and traffic flows. The study on DC topologies is going to be completed in chapter 4 with a review on the most common of those in fog ecosystems. However, at an earlier stage, chapter 3 introduces some mathematical foundations related to geometry, trees, graphs and topology, which are needed in chapter 5 to construct the most appropriate models for each of the DC topologies described in chapter 4.

Therefore, those three pillars are going to act as the building blocks to develop this thesis dissertation.

Chapter 3

Regular N-polytopes, trees, graphs and tori applied to data centres

Fog deployments account for the implementation of basic DC designs, which might get modelled by means of certain geometrical shapes. Some of them are quite straightforward, such as the segment, the star or the torus. However, when dealing with any type of mesh topology, multidimensional geometry may come into play by using specific types of convex regular N-polytopes.

Therefore, some concepts about geometry are going to be exposed, not only about multidimensional geometry, but also regarding trees, graphs and tori shapes, in order to get the basic foundations for presenting the models at a later stage.

3.1 N-dimensional geometry

To start with, it is to be reminded that the concept of dimension of a mathematical object stands for the minimum number of coordinates being necessary so as to clearly specify any of its given points.

With that in mind, in our physical world, an isolated point may be seen as an object with no dimensions, also known as $0D$. Likewise, an isolated line may be stated as an object with only one dimension, also known as $1D$. Besides, an isolated plane may be associated as an object with just two dimensions, also known as $2D$.

Moreover, an isolated space may correspond to an object in three dimensions, also known as $3D$. Therefore, our physical world may be considered as three-dimensional, because any given point therein may be fully located by quoting the measures of its length, width and height [136].

The physical $3D$ space was already studied back in the ancient Greece, some 2.300 years ago, where Euclid of Alexandria wrote his book *The Elements*, which is considered the cornerstone of the so-called Euclidean geometry. Basically, Euclid was the first to assume a small system of axioms, just five of them, and from there on, he could deduce some other propositions and theorems, hence, they may fit into a deductive logic system, being the oldest deductive approach to Mathematics [20].

Regarding Euclid's axioms, all of them seem to be absolute truths, and so were any propositions and theorems being proved from them. Additionally, Euclid also proposed a small set of postulates, just five of them, which seemed to be intuitively obvious, where all of them were proved based on his axioms but the fifth one could not. Basically, the fifth postulate may be stated as if there is a line and an exterior point, there may be just one straight line passing through such a point being parallel to such a line. Likewise, this fifth postulate may be also exposed as the sum of all angles in any triangle is just 180°.

However, later on in the XIX century, some mathematicians proposed other geometries where that fifth postulate did not hold, such as Gauss or Lobachevsky, opening the door to the definition of different non-Euclidean geometries, as opposed to Euclidean geometry, that being the one presented by Euclid.

Focusing in $2D$ surfaces, there may be planes with a constant value of Gaussian curvature, or otherwise a non-constant value. If the former applies, the most well-known case where such a value is zero stands for an Euclidean plane, thus having no intrinsic curvature whatsoever, whereas the most typical case with a positive value does it for an elliptic plane, where the sum of all angles of a triangle accounts for more than 180°, and the most representative case with a negative value does it of a hyperbolic plane, where such a sum accounts for less than 180°. Otherwise, if the latter applies, Riemann manifolds apply, where all the former may also be included as a particular subset.

If a $3D$ space is considered as the intersection of two perpendicular planes, such planes may be the same type, such a pure Euclidean, elliptic or hyperbolic spaces, or otherwise, mixed spaces, where each of them may be of a different kind. Likewise, the same may happen with planes with non constant value of Gaussian curvature or anisotropic planes, where any combination is possible. Eventually, those spaces may be extended to any given number of dimensions, where those stands for a natural number, or even for infinite.

Focusing on *Euclidean spaces*, it may be said that multidimensional geometries are the Euclidean spaces where $N > 3$, which may be seen as an extension of our physical world [212]. Generically speaking, a space with any number of dimensions may be called Euclidean space if all its dimensions meet the axioms of the Euclidean geometry. Furthermore, every type of space may be considered to behave locally as an Euclidean space.

3.1.1 Platonic solids

There were great mathematicians dedicated to the study of geometry back in the ancient Greece apart from Euclid, such as Pythagoras of Samos, Thales of Miletus or Archimedes of Syracuse. However, Plato paid special attention to a bunch of three-dimensional shapes with some particular characteristics such as being formed with regular polygons, thus all angles are equal and so are all sides, being congruent, thus having the same size and shape, and with the same number of polygonal faces meeting in every node. Those special shapes were called *platonic solids* and accounted for convex regular polyhedra.

Specifically, he associated those shapes to the four classical elements in his dialogue *Timaeus*, such that fire was bound to the tetrahedron, Earth was attached to the cube, air was tied to the octahedron and water was linked to the icosahedron, whilst the dodecahedron was used to arrange the constellations in heaven. Later on, Aristotle considered that the fifth element was ether, postulating that heaven was made of it.

There are five convex regular polyhedra, composed by a single type of congruent convex regular polygons as their faces, which are listed with their main features in

Table 3.1 and are shown in Figure 3.1. Those polygons are either equilateral triangles, squares or regular pentagons.

Table 3.1: Platonic solids.

Polyhedron	**Polygonal faces**	**Faces**	**Edges**	**Nodes**
Tetrahedron	Equilateral Triangles	4	6	4
Cube	Squares	6	12	8
Octahedron	Equilateral Triangles	8	12	6
Dodecahedron	Regular Pentagons	12	30	20
Icosahedron	Equilateral Triangles	20	30	12

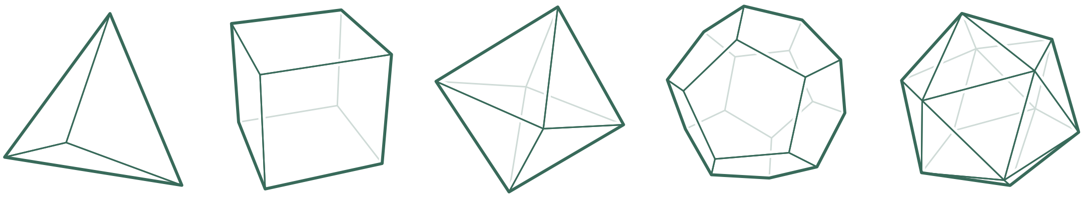


Figure 3.1: Platonic solids.

It is to be noted that there are two pairs of dual shapes, in a way that the number of vertices of a polyhedron may match the number of faces of its dual, whilst having the same number of edges. This fact may imply that the vertices of one may be right in the centre of the faces of the other, where any of both may the inscribed one, and the other, the circumscribed one. On the one hand, this closely related feature happens to the cube and the octahedron, whereas on the other hand, it also occurs to the dodecahedron and the icosahedron.

Additionally, there is also an autodual shape, such that its number of vertices and faces are equal. This feature implies that the vertices of it may be right in the centre of the faces of another polyhedron of the same type. This feature does only take place in the tetrahedron.

It is also to be pointed out that those convex regular polyhedra are all homeomorphic to a 2-sphere, meaning a sphere in $3D$. This may be proved by using *Euler*'s characteristic (χ), which works as a topological invariant for any type of geometrical shape, thus describing the topology of a given shape. Focusing on convex regular polyhedra, it says that the addition of the number of vertices and faces along with the subtraction of the number of edges must be equal to 2, such as in (3.1).

$$\chi = \sum_{i=0}^{N-1} N_i \xrightarrow{\text{N=3}} \sum_{i=0}^{2} N_i = \text{Vertices} - \text{Edges} + \text{Faces} = 2 \tag{3.1}$$

3.1.2 Schläfli notation

The study of multidimensional geometry is understood to have started its development in the middle of the XIX century thanks to different authors, such as Hamilton with the quaternions, or Graves and Cayley with the octonions. However, it may be considerd that its foundation was established by *Schläfli* in his masterpiece called *Theorie der vielfachen Kontinuität* [200].

Unluckily, he could not publish his work, whilst Riemann introduced the concept of N-dimensional manifolds [160] a couple of years later, thus it remains unclear whether both works related to higher dimensional geometry took place independently, even though both took different approaches. It is also worth to be mentioned that in the last decades of the XIX century, different mathematicians rediscovered Schläfli's results in an independent way, although it is considered that Coxeter made the best compilation of them in the middle of the XX century [46].

Regarding Schläfli's work in multidimensional geometry, he first studied polygons and polyhedra so as to define their extensions for higher dimensions, along with their corresponding properties and adapting Euler's formula to any dimension. By the way, he originally referred to those shapes as polyschemes, although one of his rediscoverers back in the XIX century, called Hoppe, first coined the term *polytope*, which is the one being used nowadays. This way, focusing on regular polygons and platonic solids, the generalised N-dimensional extension of them may be called *regular polytopes* or *regular N-polytopes.*

Additionally, Schläfli came up with a special nomenclature to classify and provide some key properties of any regular N-polytope, which is called either *Schläfli notation* or *Schläfli symbol.* This has the form $\{p, q, r, \cdots, y, z\}$, having as many variables as $N-1$, where N is the dimension of the shape [214]. It is to be said that this notation gives information about its types of regular *facets* and its kinds of regular *peak figures* (items incident on a facet), the former going from the $(N-1)$-dimensional one downwards and the latter does from the $0D$ figure upwards. Besides, all facets and peak figures are alike everywhere in a regular N-polytope, resulting in congruent regular facets and congruent regular peak figures for each type extracted from notation.

Therefore, if a regular polytope is defined by the Schläfli symbol $\{p, q, r, \cdots, x, y, z\}$, it means that it is made up by regular $\{p, q, \cdots, y\}$ as $(N-1)$-facets, regular $\{p, \cdots, x\}$ as $(N-2)$-facets, and so on up to $\{p\}$ as faces $(2D)$, whilst $\{\}$ stands for edges $(1D)$, whose type is obviously unique, whereas it shows $\{q, r, \cdots, y, z\}$ as its vertex figure, $\{r, \cdots, y, z\}$ as its edge figure, and so on up to $\{z\}$ as its $(N-3)$-facet figure, whilst $\{\}$ stands for its $(N-2)$-facet figure. Alternatively, it may be said that there are z $\{p, q, \cdots, y\}$ $(N-1)$-facets around each $(N-3)$-facet of a regular N-polytope.

Furthermore, Schläfli symbol may be seen as a *recursive description*, as a regular N-polytope may be built up using regular k-facets, where $k = [0 \cdots N-1]$. This way, $k = 0$ stands for vertices, $k = 1$ does it for edges, $k = 2$ goes for faces, $k = 3$ means cells, and so on. For instance, $\{p\}$ defines a regular 2-polytope where obviously there are just one face $\{p\}$, $\{p, q\}$ defines a regular 3-polytope where the type of faces are $\{p\}$ and with a vertex figure $\{q\}$, $\{p, q, r\}$ defines a regular 4-polytope where cells are $\{p, q\}$, faces are $\{p\}$, edge figures are $\{r\}$ and vertex figures are $\{q, r\}$, and so on.

3.1.3 Regular polygons

To start with the $1D$ Euclidean space, the Schläfli symbol of any segment might be considered as an empty string enclosed into curly braces, such as $\{\}$, because that symbol may have $N-1$ variables, which happens to be 0 variables. On the other hand, it implies that there is only one possible type of shape in $1D$, which stands for the segment, regardless its length.

Moving on to the $2D$ Euclidean space, the Schläfli symbol of any regular polygon may be represented by just one variable. Regarding *convex regular polygons* of $\{p\}$ edges, the Schläfli symbol may be just $\{p\}$, that being a natural number going from 3 upwards so as to achieve a closed polygon. For instance, $\{3\}$ represents an equilateral triangle, {4} does it for a square, {5} for a regular pentagon, {6} for a regular hexagon, and so on. Basically, expression (3.2) exhibits this condition, whereas Figure 3.2 shows some of them. Obviously, all polygons have the same number of vertices and edges.

$$\forall p \in \mathbb{N} \ \ / \ \ p \geq 3 \tag{3.2}$$

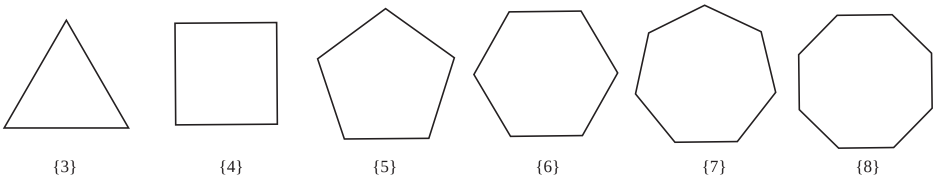


Figure 3.2: Convex regular polygons.

Otherwise, with respect to *star regular polygons*, they have p spokes whereas the number of times it winds around its center to built up the star is given by m, also known as the vertex interval step to get the star done. In such cases, the Schläfli symbol may be just $\{p/m\}$ where the fraction may result in a quotient strictly higher than two, and at the same time, it may be irreducible, meaning that both values may be coprime. For instance, {5/2} shows for a pentagonal star, or {7/2} and {7/3} makes it for two different sorts of heptagonal stars.

It is to be said that regular compound polygons appear when the coprime condition is not fulfilled, such as in the case of a hexagonal star {6/2} or one kind of octogonal star {8/2}, even though sometimes they are branded as 'improper' star regular polygons. Nonetheless, expression (3.3) exhibits the necessary conditions for achieving a proper star regular polygon, whilst Figure 3.3 shows some of them.

$$\begin{cases} \forall p \in \mathbb{N} \ \ / \ \ \dfrac{p}{m} > 2 \\ p, m \ \text{coprime} \end{cases} \tag{3.3}$$

Furthermore, if the aforesaid first condition is brought to the limit, in this case a line segment is attained, which might be seen as a degenerated star regular polygon. That segment line may be seen as a *regular linear tiling* of a $1D$ Euclidean space, as an infinite number of such any segment may completely tessellate any given line, which is

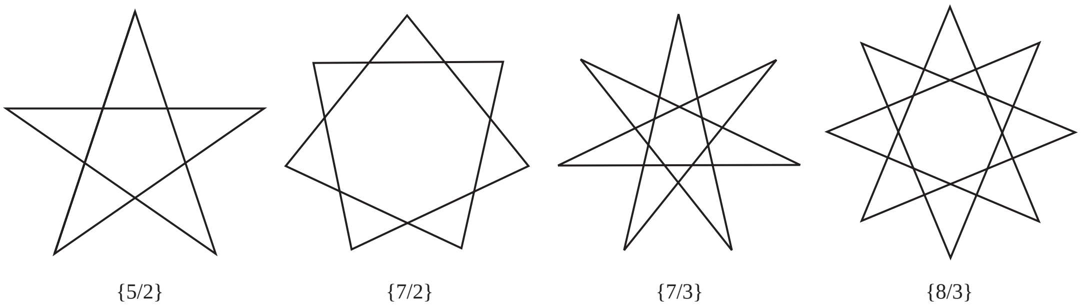


Figure 3.3: Star regular polygons.

homeomorphic to the Euclidean space for $N = 1$, also known as the real number line. Hence, expression (3.4) exhibits that condition, even though the coprime condition does not apply, where Figure 3.4 depicts the tiling. Some examples are $\{4/2\}$, $\{6/3\}$ or $\{8/4\}$, even though any fraction $\{p/m\}$ where p is twice m may be a solution.

$$\forall p \in \mathbb{N} \;\; / \;\; \frac{p}{m} = 2 \tag{3.4}$$

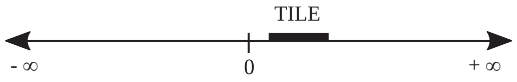


Figure 3.4: Regular tessellation of the $1D$ Euclidean line.

3.1.4 Regular polyhedra

Moving next to the $3D$ Euclidean space, the Schläfli symbol for any regular polyhedron may be represented by two variables. Concerning *convex regular polyhedra*, which happens to be the *platonic solids*, variable $\{p\}$ indicates the type of convex regular polygons making for their faces, also known as 2-facets, whereas variable $\{q\}$ does for the amount of faces meeting at each vertex, or alternatively, their vertex figure. Thus, the Schläfli symbol may be $\{p, q\}$, where both values need to be natural numbers. However, not all combinations are allowed, as both conditions stated in (3.5) must be met. Furthermore, Table 3.2 presents the Schläfli notation for each of the five platonic solids.

$$\begin{cases} \dfrac{1}{p} + \dfrac{1}{q} > \dfrac{1}{2} \quad , \quad \forall p, q \in \mathbb{N} \\ \{p\}, \{q\} \subset \text{regular polygons} \end{cases} \tag{3.5}$$

Table 3.2: Convex regular polyhedra in Schläfli notation.

Polyhedron	**Polygonal faces**	$\{\mathbf{p}\}$	$\{\mathbf{q}\}$	$\{\mathbf{p},\mathbf{q}\}$
Tetrahedron	Equilateral Triangles	$\{3\}$	$\{3\}$	$\{3,3\}$
Cube	Squares	$\{4\}$	$\{3\}$	$\{4,3\}$
Octahedron	Equilateral Triangles	$\{3\}$	$\{4\}$	$\{3,4\}$
Dodecahedron	Regular Pentagons	$\{5\}$	$\{3\}$	$\{5,3\}$
Icosahedron	Equilateral Triangles	$\{3\}$	$\{5\}$	$\{3,5\}$

Otherwise, with regards to *star regular polyhedra*, also known as *Kepler-Poinsot polyhedra*, are obtained if either $\{p\}$ or $\{q\}$ are a pentagonal star, whose Schläfli notation happens to be $\{5/2\}$, and the other value is either $\{3\}$ or $\{5\}$, so as to deal with a dodecahedron or an icosahedron. The first condition proposed for their convex counterpart may apply, which gets restricted to expression (3.6), thus limiting the possible combinations to four shapes, which are presented in Table 3.3 and are depicted in Figure 3.5.

Any other star regular polygon would not meet the conditions established above, meaning that a closed shape is not obtained by using any other values. On the other hand, regarding the four available star regular polyhedra, $\{p\}$ may represent the type of faces, whilst $\{q\}$ may show the vertex figure.

$$\frac{1}{p} + \frac{1}{q} > \frac{1}{2} \quad , \quad \begin{cases} p = 3, 5; q = 5/2 \\ \text{OR} \\ p = 5/2; q = 3, 5 \end{cases} \tag{3.6}$$

Furthermore, those four star regular polyhedra are dual in pairs, as it may be seen in Table 3.4, although it also may be seen in the previous table by realizing that Schläfli symbols are the other way around when it comes to dual polyhedra.

Moreover, taking the condition for regular polyhedra to the limit, it may be possible to find the regular polygons permitting a *regular plane tiling* of the whole $2D$

Table 3.3: Star regular polyhedra in Schläfli notation.

Polyhedron	**Polygonal faces**	**{p}**	**{q}**	**{p, q}**
Great dodecahedron	Regular Pentagons	$\{5\}$	$\{5/2\}$	$\{5, 5/2\}$
Small stellated dodecahedron	Pentagonal stars	$\{5/2\}$	$\{5\}$	$\{5/2, 5\}$
Great icosahedron	Equilateral Triangles	$\{3\}$	$\{5/2\}$	$\{3, 5/2\}$
Great stellated dodecahedron	Pentagonal stars	$\{5/2\}$	$\{3\}$	$\{5/2, 3\}$

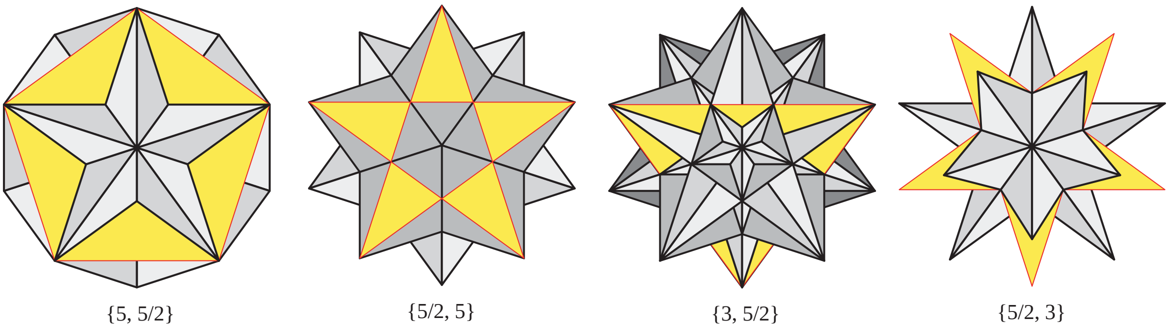


Figure 3.5: Star regular polyhedra.

Table 3.4: Star regular polyhedra and their main features.

Polyhedron	**Euler Characteristic**	**Faces**	**Edges**	**Nodes**
Great dodecahedron	$\chi = -6$	12	30	12
Small stellated dodecahedron	$\chi = -6$	12	30	12
Great icosahedron	$\chi = 2$	20	30	12
Great stellated dodecahedron	$\chi = 2$	12	30	20

Euclidean plane. The condition is exhibited in (3.7), and there are only three possible solutions, in which $\{p\}$ stands for the type of face and $\{q\}$ does it for the number of such shapes surrounding each of the vertices, which may also be referred to as the vertex figure.

The first one is $\{6, 3\}$, also known as hexagonal tiling, meaning that three regular hexagons may meet at each vertex. The second one is $\{4, 4\}$, also known as square tiling, where four squares meet at each vertex. And the third one is $\{3, 6\}$, also known as triangular tiling, where six equilateral triangles meet at each vertex.

Those regular tessellations offer both translational and rotational symmetry with a unique regular polygon for a $2D$ Euclidean space. It is to be mentioned that Penrose proposed an aperiodic tessellation mainly based on regular pentagons, offering rotational symmetry but not translational one, like the structure of a quasicrystal [78]. However, other types of shapes had to be included in order to get the tessellation of the whole Euclidean plane, hence, it may not be considered as a regular tiling. Thus, Figure 3.6 shows the only regular tessellations available for the $2D$ Euclidean space.

$$\begin{cases} \frac{1}{p} + \frac{1}{q} = \frac{1}{2} \quad , \quad \forall p, q \in \mathbb{N} \\ \{p\}, \{q\} \subset \text{regular polygons} \end{cases} \tag{3.7}$$

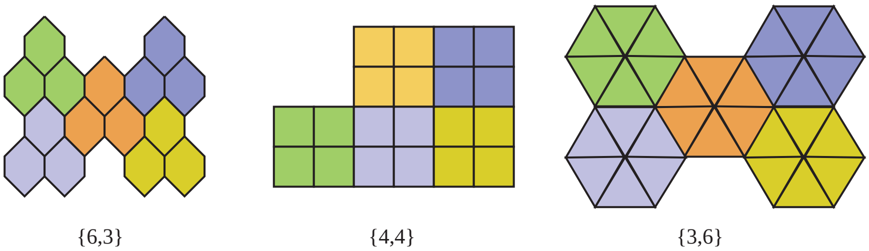


Figure 3.6: Regular tessellation of the $2D$ Euclidean plane.

3.1.5 Regular polychora

Moving ahead to the $4D$ Euclidean space, the Schläfli notation for any regular polychoron may need to be represented by three variables. Looking at *convex regular polychora*, variables $\{p, q\}$ determine the kind of polyhedral cells it possess, variable $\{p\}$ indicated the sort of polygonal faces each cell is made of, variable $\{r\}$ designates the edge figure and variables $\{q, r\}$ states for the vertex figure. Hence, the Schläfli symbol may be $\{p, q, r\}$, where all values must be natural numbers, and besides, the conditions stated in (3.8) has to be met. Additionally, Tables 3.5 and 3.6 present the Schläfli notation for all six possible combinations.

$$\begin{cases} \sin\frac{\pi}{p} + \sin\frac{\pi}{r} - \cos\frac{\pi}{q} > 0 \quad , \quad \forall p, q, r \in \mathbb{N} \\ \{p, q\}, \{q, r\} \subset \text{regular polyhedra} \end{cases} \tag{3.8}$$

Table 3.5: Convex regular polychora in Schläfli notation (I).

Name	**Alternative name**	$\{\mathbf{p},\mathbf{q},\mathbf{r}\}$
Pentachoron	4-simplex	$\{3,3,3\}$
Tesseract	4-hypercube	$\{4,3,3\}$
Hexadecachoron	4-orthoplex	$\{3,3,4\}$
Icositetrachoron	24-cell	$\{3,4,3\}$
Hecatonicosachoron	120-cell	$\{5,3,3\}$
Hexacosichoron	600-cell	$\{3,3,5\}$

Table 3.6: Convex regular polychora in Schläfli notation (II).

Polychoron	$\{\mathbf{p},\mathbf{q}\}$	$\{\mathbf{p}\}$	$\{\mathbf{r}\}$	$\{\mathbf{q},\mathbf{r}\}$	$\{\mathbf{p},\mathbf{q},\mathbf{r}\}$
4-simplex	$\{3,3\}$	$\{3\}$	$\{3\}$	$\{3,3\}$	$\{3,3,3\}$
4-hypercube	$\{4,3\}$	$\{4\}$	$\{3\}$	$\{3,3\}$	$\{4,3,3\}$
4-orthoplex	$\{3,3\}$	$\{3\}$	$\{4\}$	$\{3,4\}$	$\{3,3,4\}$
24-cell	$\{3,4\}$	$\{3\}$	$\{3\}$	$\{4,3\}$	$\{3,4,3\}$
120-cell	$\{5,3\}$	$\{5\}$	$\{3\}$	$\{3,3\}$	$\{5,3,3\}$
600-cell	$\{3,3\}$	$\{3\}$	$\{5\}$	$\{3,5\}$	$\{3,3,5\}$

It is to be observed that a 4-simplex (also known as 5-cell) is bounded of 5 tetrahedral cells ($\{p,q\}$), with 3 of them joining together around each edge ($\{r\}$). With respect to the 4-hypercube (also known as 8-cell), it is limited by 8 cubic cells, with 3 of them located around each edge. With regards to the 4-orthoplex (also known as 16-cell), it is enclosed by 16 tetrahedral cells, with 4 of them surrounding each edge. Furthermore, a 24-cell is bounded by 24 octahedral cells, with 3 of them limiting each of the edges, a 120-cell is limited by as many dodecahedral cells, with 3 of them meeting at each of the edges, and a 600-cell is surrounded by as many tetrahedral cells, with 5 of them meeting at every edge.

Otherwise, regarding *star regular polychora*, also known as *Schläfli-Hess polychora*, they are possible only with star regular polyhedra involved, thus, the value of $5/2$ may be involved in some way, along with the values of 3 and 5, apart from the fulfillment of the requirements given in (3.9), taken from their convex regular counterparts. There are just 10 combinations available, even though other solutions are available, because some of those solutions do not form a finite figure, so they have to be rejected.

Therefore, the resulting star regular polychora are presented in Table 3.7, where the different Schläfli symbols for each figure may be appreciated.

$$\begin{cases} \sin\dfrac{\pi}{p} + \sin\dfrac{\pi}{r} - \cos\dfrac{\pi}{q} > 0 \quad , \quad \forall p,q,r \in 3,5,\dfrac{5}{2} \\ \{p,q\},\{q,r\} \subset \text{regular polyhedra} \end{cases} \tag{3.9}$$

Table 3.7: Star regular polychora in Schläfli notation.

Polychoron	$\mathbf{\{p,q\}}$	$\mathbf{\{p\}}$	$\mathbf{\{r\}}$	$\mathbf{\{q,r\}}$	$\mathbf{\{p,q,r\}}$
Icosahedral 120-cell	$\{3,5\}$	$\{3\}$	$\{5/2\}$	$\{5,5/2\}$	$\{3,5,5/2\}$
Small stellated 120-cell	$\{5/2,5\}$	$\{5/2\}$	$\{3\}$	$\{5,3\}$	$\{5/2,5,3\}$
Great 120-cell	$\{5,5/2\}$	$\{5\}$	$\{5\}$	$\{5/2,5\}$	$\{5,5/2,5\}$
Grand 120-cell	$\{5,3\}$	$\{5\}$	$\{5/2\}$	$\{3,5/2\}$	$\{5,3,5/2\}$
Great stellated 120-cell	$\{5/2,3\}$	$\{5/2\}$	$\{5\}$	$\{3,5\}$	$\{5/2,3,5\}$
Grand stellated 120-cell	$\{5/2,5\}$	$\{5/2\}$	$\{5/2\}$	$\{5,5/2\}$	$\{5/2,5,5/2\}$
Great grand 120-cell	$\{5,5/2\}$	$\{5\}$	$\{3\}$	$\{5/2,3\}$	$\{5,5/2,3\}$
Great icosahedral 120-cell	$\{3,5/2\}$	$\{3\}$	$\{5\}$	$\{5/2,5\}$	$\{3,5/2,5\}$
Grand 600-cell	$\{3,3\}$	$\{3\}$	$\{5/2\}$	$\{3,5/2\}$	$\{3,3,5/2\}$
Great grand stellated 120-cell	$\{5/2,3\}$	$\{5/2\}$	$\{3\}$	$\{3,3\}$	$\{5/2,3,3\}$

Besides, bringing the condition for regular polychora to the limit allows to find out the regular polyhedra necessary to achieve a *regular cubic tiling* of the whole $3D$ Euclidean space. That condition is given in (3.10), and there is only one possible solution, that being $\{4,3,4\}$. In that case, $\{p,q\}$ represents the type of cell, which happens to be a cube, and $\{r\}$ does it for the number of those being around each edge, as well as the edge figure. Moreover, Figure 3.7 shows such a cubic tessellation for the $3D$ Euclidean space.

$$\begin{cases} \sin\dfrac{\pi}{p} + \sin\dfrac{\pi}{r} - \cos\dfrac{\pi}{q} = 0 \quad , \quad \forall p,q,r \in \mathbb{N} \\ \{p,q\},\{q,r\} \subset \text{regular polyhedra} \end{cases} \tag{3.10}$$

3.1.6 Regular polytera

Moving on to the $5D$ Euclidean space, the Schläfli notation for any regular polyteron may take four variables. Putting the focus on *convex regular polytera*, variables

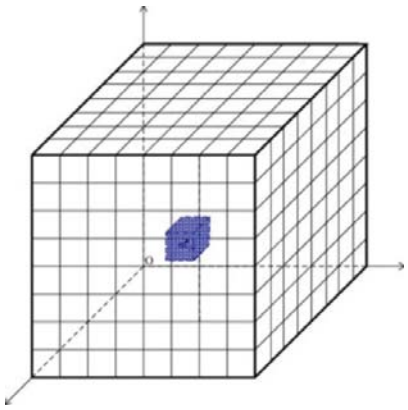


Figure 3.7: Regular tessellation of the $3D$ Euclidean space.

$\{p, q, r\}$ show the type of convex regular polychoron forming the shape, variables $\{p, q\}$ give for the sort of convex regular polyhedral cells making for each polychoron, variable $\{p\}$ indicates the kind of convex regular polygonal faces building up each polyhedra, whereas variable $\{s\}$ shows the face figure, variables $\{r, s\}$ give the edge figure and variables $\{q, r, s\}$ indicates the vertex figure.

Therefore, the Schläfli symbol may be $\{p, q, r, s\}$, where all values must be natural numbers, and moreover, the conditions stated in (3.11) has to be fulfilled. Additionally, Table 3.8 present the Schläfli notation for the only three possible combinations, making up for the hexateron, also known as 5-simplex, the penteract, also known as 5-hypercube, and the pentacross, also known as 5-orthoplex.

$$\begin{cases} \dfrac{\cos^2 \frac{\pi}{q}}{\sin^2 \frac{\pi}{p}} + \dfrac{\cos^2 \frac{\pi}{r}}{\sin^2 \frac{\pi}{s}} < 1 \quad , \quad \forall p, q, r, s \in \mathbb{N} \\ \{p, q, r\}, \{q, r, s\} \subset \text{regular polychora} \end{cases} \tag{3.11}$$

Table 3.8: Convex regular polytera in Schläfli notation.

Polyteron	$\{\mathbf{p}, \mathbf{q}, \mathbf{r}\}$	$\{\mathbf{p}, \mathbf{q}\}$	$\{\mathbf{p}\}$	$\{\mathbf{s}\}$	$\{\mathbf{r}, \mathbf{s}\}$	$\{\mathbf{q}, \mathbf{r}, \mathbf{s}\}$	$\{\mathbf{p}, \mathbf{q}, \mathbf{r}, \mathbf{s}\}$
5-simplex	$\{3, 3, 3\}$	$\{3, 3\}$	$\{3\}$	$\{3\}$	$\{3, 3\}$	$\{3, 3, 3\}$	$\{3, 3, 3, 3\}$
5-hypercube	$\{4, 3, 3\}$	$\{4, 3\}$	$\{4\}$	$\{3\}$	$\{3, 3\}$	$\{3, 3, 3\}$	$\{4, 3, 3, 3\}$
5-orthoplex	$\{3, 3, 3\}$	$\{3, 3\}$	$\{3\}$	$\{4\}$	$\{3, 4\}$	$\{3, 3, 4\}$	$\{3, 3, 3, 4\}$

Otherwise, respecting *star regular polytera*, there are not any instance available, in a way that no type of star regular polychora may be the foundation to build up any such a star regular polytera. In other words, it is not possible to meet both the conditions exposed in (3.12).

$$\begin{cases} \dfrac{\cos^2 \frac{\pi}{q}}{\sin^2 \frac{\pi}{p}} + \dfrac{\cos^2 \frac{\pi}{r}}{\sin^2 \frac{\pi}{s}} < 1 \quad , \quad \forall p, q, r, s \in 3, 5, \dfrac{5}{2} \\ \{p, q, r\}, \{q, r, s\} \subset \text{regular polychora} \end{cases} \tag{3.12}$$

Additionally, moving the condition for regular polytera to the limit may permit to find out the regular polychora to attain a *regular four-dimensional tiling* of the whole $4D$ Euclidean space. Such a condition is given in (3.13), and there are three solutions available, such that $\{4, 3, 3, 4\}$, $\{3, 3, 4, 3\}$ and $\{3, 4, 3, 3\}$, where each of them is valid to obtain a complete regular four-dimensional tessellation of the $4D$ Euclidean space.

$$\begin{cases} \dfrac{\cos^2 \frac{\pi}{q}}{\sin^2 \frac{\pi}{p}} + \dfrac{\cos^2 \frac{\pi}{r}}{\sin^2 \frac{\pi}{s}} = 1 \quad , \quad \forall p, q, r, s \in \mathbb{N} \\ \{p, q, r\}, \{q, r, s\} \subset \text{regular polychora} \end{cases} \tag{3.13}$$

3.1.7 Regular N-polytopes

Generalizing a higher-dimensional Euclidean space where N>5, it is clear that only 3 types of *convex regular polytopes* are possible, as no other type of shape is allowed in the $5D$ Euclidean space. Therefore, the Schläfli notation for any given convex regular polytope available for $N \geq 5$ is listed in Table 3.9.

Table 3.9: Convex regular N-polytopes in Schläfli notation ($N \geq 5$).

Regular N-polytope	**Schläfli notation**
N-simplex	$\{3, 3, \cdots, 3, 3\}$
N-hypercube	$\{4, 3, \cdots, 3, 3\}$
N-orthoplex	$\{3, 3, \cdots, 3, 4\}$

It is to be said that N-simplices may also be called regular simplices, whereas N-hypercubes are also known as measure polytopes, whilst N-orthoplexes are also named as cross polytopes. On the other hand, Schläfli symbols may give some information about the duality of regular polytopes, in a way that two shapes are dual if their Schläfli notations are just flipped around. Besides, it is to be reminded that the dual of the dual is the original one.

This is the case of the measure polytope and the cross polytope of the same dimension, which means that one of them may has its i-dimensional facets in the same place where its dual one has its $(N-i)$-dimensional facets. Otherwise, the regular simplices are autodual or self-dual, as they are the dual of themselves, which also may show in the Schläfli symbol being read the same way either forwards of backwards.

Additionally, regarding the Euler's characteristic previously exposed for $3D$ spaces, Schläfli got it generalized for any convex regular polytope of dimension N, which is shown in (3.14).

$$\chi = \sum_{i=0}^{N-1} N_i = N_0 - N_1 + N_2 - N_3 \cdots = 1 - (-1)^N \tag{3.14}$$

This expressions happens to give $\chi = 2$ if N is odd, whereas $\chi = 0$ if N is even, even though it just happens with any shape being homeomorphic to a topological $(N-1)$-sphere.

Eventually, the three different kinds of mesh topologies (namely full mesh, quasi full mesh and some types of partial mesh) are going to be related to the three convex regular N-polytopes, showing the curious relationships among them regarding the number of nodes and the number of links.

3.1.7.1 N-simplex: k-facets

With respect to the N-simplex, it may be said that each node has a connection to all of the rest, regardless the dimension, as it may be shown in Figure 3.8.

On the other hand, it is to be reminded that in set theory, the set including all faces of a given polytope also contains the polytope itself, as well as the empty set, which is assigned a dimension of -1, just for consistency reasons. Hence, for any N-dimensional polytope, the scope of dimensions may include all dimensions ranging

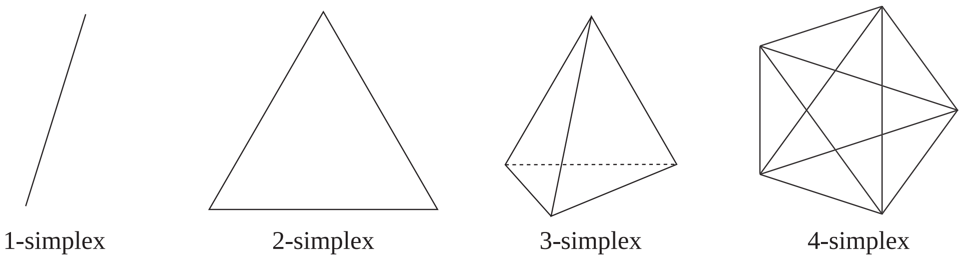


Figure 3.8: N-simplex shapes for lower dimensions.

from $-1 \leq k \leq N$. Thus, the meaning of the k-facets corresponding to the lower values of k are listed in Table 3.10.

Additionally, it is to be noted that the values of the k-facets for $k = -1$ and $k = N$ are always 1. As per the former, the boundary of a point, which is $0D$, is the empty set, which is considered as $-1D$, because all N-polytopes have $(N-1)$-dimensional boundaries, whereas the latter accounts for the N-polytope itself.

Table 3.10: Meaning of k-facets.

Value of k	**Geometrical item**	**Dimensions of that item**
-1	$\emptyset$ (empty set)	-1
0	nodes	0
1	edges	1
2	faces	2
3	cells	3
4	polychora	4
5	polytera	5

Therefore, the values for the different k-facets of an N-simplex for a given dimension N are obtained by expression (3.15), through the use of combinatory calculations.

$$\binom{N+1}{k+1} = \frac{(N+1)!}{(k+1)! \times \big((N+1)-(k+1)\big)!} \tag{3.15}$$

It is to be noticed that the same result may be obtained either through the aforesaid expression, or otherwise, by spotting the value located in the $(N+1)$-th row and the $(k+1)$-th column of the Pascal's Triangle [116]. Anyway, the values obtained for the k-facets related to some N-simplex shapes are presented in Table 3.11.

Table 3.11: Amount of k-facets present in each N-simplex.

Dim.	**Name**	$\mathbf{k = -1}$	$\mathbf{k = 0}$	$\mathbf{k = 1}$	$\mathbf{k = 2}$	$\mathbf{k = 3}$	$\mathbf{k = 4}$	$\mathbf{k = 5}$
N	-	$\binom{N+1}{0}$	$\binom{N+1}{1}$	$\binom{N+1}{2}$	$\binom{N+1}{3}$	$\binom{N+1}{4}$	$\binom{N+1}{5}$	$\binom{N+1}{6}$
-1	empty set ($\emptyset$)	1	N/A	N/A	N/A	N/A	N/A	N/A
0	point	1	1	N/A	N/A	N/A	N/A	N/A
1	segment	1	2	1	N/A	N/A	N/A	N/A
2	triangle	1	3	3	1	N/A	N/A	N/A
3	tetrahedron	1	4	6	4	1	N/A	N/A
4	4-simplex	1	5	10	10	5	1	N/A
5	5-simplex	1	6	15	20	15	6	1

By looking at the results obtained, it is easy to spot that the values corresponding to each row are just the sequence of values in the whole $(N+1)$-th row of the Pascal's Triangle, and that is because the expressions to find out both match. Otherwise, it is easy to see that the values in a single row are palindromic, as the same sequence is attained from left to right than from right to left, thus proving that the N-simplex is autodual, as previously stated.

3.1.7.2 N-hypercube: k-facets

With regards to the N-hypercube, it may be said that each node has a connection to N neighbours, each one in a different dimension, as it may be shown in Figure 3.9.

Hence, the amounts of the diverse k-facets within an N-hypercube for a particular dimension N are given by expression (3.16) by undertaking combinatory calculations.

$$2^{N-k} \times \binom{N}{k} = 2^{N-k} \times \frac{N!}{k! \times (N-k)!} \tag{3.16}$$

As in the previous case, the combinatory calculations may be made either by using the expression $C(N,k) = {}^{N!}/_{k! \cdot (N-k)!}$, or otherwise, by looking at the proper cell in the Pascal's Triangle to obtain the combinatory expression. Such values obtained for the k-facets related to some N-hypercube shapes are exhibited in Table 3.12.

By taking a look at the results, and knowing that the N-hypercube is dual with the N-orthoplex, it may be easy to imagine the values to be obtained for it.

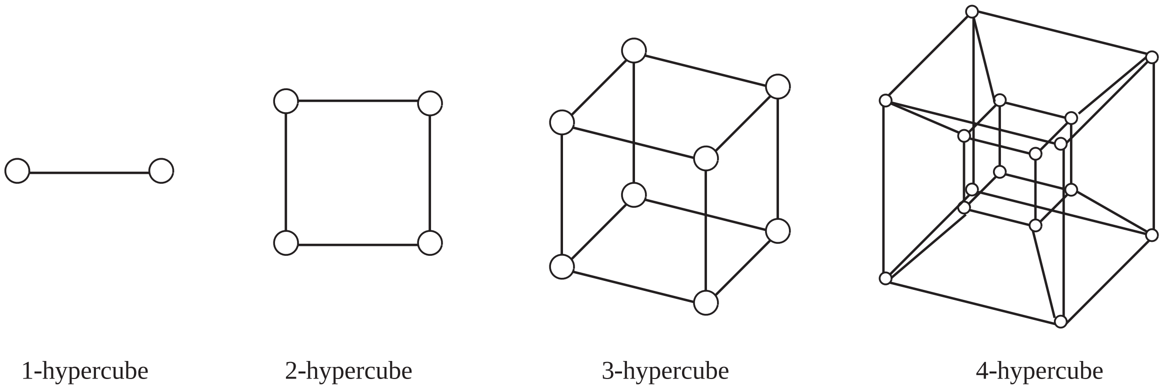


Figure 3.9: N-hypercube shapes for lower dimensions.

Table 3.12: Amount of k-facets present in each N-hypercube.

Dim.	**Name**	$\mathbf{k=-1}$	$\mathbf{k=0}$	$\mathbf{k=1}$	$\mathbf{k=2}$	$\mathbf{k=3}$	$\mathbf{k=4}$	$\mathbf{k=5}$
N	-	$2^{N+1}\binom{N}{-1}$	$2^{N}\binom{N}{0}$	$2^{N-1}\binom{N}{1}$	$2^{N-2}\binom{N}{2}$	$2^{N-3}\binom{N}{3}$	$2^{N-4}\binom{N}{4}$	$2^{N-5}\binom{N}{5}$
-1	empty set ($\emptyset$)	1	N/A	N/A	N/A	N/A	N/A	N/A
0	point	1	1	N/A	N/A	N/A	N/A	N/A
1	segment	1	2	1	N/A	N/A	N/A	N/A
2	square	1	4	4	1	N/A	N/A	N/A
3	cube	1	8	12	6	1	N/A	N/A
4	4-hypercube	1	16	32	24	8	1	N/A
5	5-hypercube	1	32	80	80	40	10	1

3.1.7.3 N-orthoplex: k-facets

Regarding the N-orthoplex, it may be said that each node has a connection to all its neighbours, except for its opposite one, no matter the dimension, as it may be depicted in Figure 3.10.

Thus, the values corresponding to the k-facets within an N-orthoplex for a given dimension N are attained by expression (3.17) thanks to combinatory calculations.

$$2^{k+1} \times \binom{N}{k+1} = 2^{k+1} \times \frac{N!}{(k+1)! \times \big(N-(k+1)\big)!} \tag{3.17}$$

As stated before, the combinatory calculations may be undertaken by either of both methods exposed in the N-hypercube. Anyway, those values attained for the k-facets related to some N-orthoplex shapes are depicted in Table 3.13.

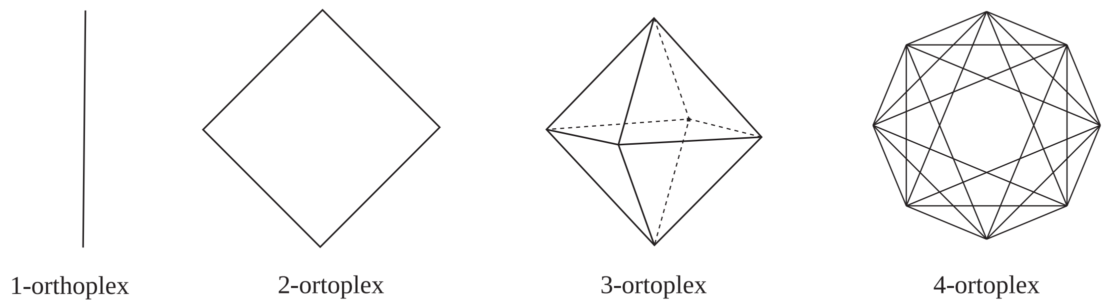


Figure 3.10: N-orthoplex shapes for lower dimensions.

Table 3.13: Amount of k-facets present in each N-orthoplex.

Dim.	**Name**	$\mathbf{k = -1}$	$\mathbf{k = 0}$	$\mathbf{k = 1}$	$\mathbf{k = 2}$	$\mathbf{k = 3}$	$\mathbf{k = 4}$	$\mathbf{k = 5}$
N	-	$2^0\binom{N}{0}$	$2^1\binom{N}{1}$	$2^2\binom{N}{2}$	$2^3\binom{N}{3}$	$2^4\binom{N}{4}$	$2^5\binom{N}{5}$	$2^6\binom{N}{6}$
-1	$\emptyset$	1	N/A	N/A	N/A	N/A	N/A	N/A
0	point	1	1	N/A	N/A	N/A	N/A	N/A
1	segment	1	2	1	N/A	N/A	N/A	N/A
2	square	1	4	4	1	N/A	N/A	N/A
3	octahedron	1	6	12	8	1	N/A	N/A
4	4-orthoplex	1	8	24	32	16	1	N/A
5	5-orthoplex	1	10	40	80	80	32	1

By comparing the results obtained for the N-orthoplex with those achieved for the N-hypercube, it is clear that both shapes are the dual of each other, as the values attained for a certain dimension in one shape are the same in the other one, but ordered the other way around.

3.2 Tree-like data structures

Tree-like structures have many utilities in mathematics and computer science, such as being the foundation of different applications, going from building up decision trees in statistics and probability to designing DC architectures. Therefore, it may be useful to review some of their main features [211].

3.2.1 Abstract data types

Abstract Data Types (ADTs) are logical descriptions on how data are viewed, putting the focus on what those data are representing and their sets of behaviours, no matter how they are implemented. ADTs provide a level of abstraction around the actual data, creating a sort of encapsulation around them, thus acting as an interface between the user and the real data.

Furthermore, the term *abstract* relates to hiding low-level details by using high-level ideas, thus providing only the essentials. To do so, an ADT defines the behaviour of an object or entity by a set of values (attributes) and a set of operations (methods), leaving aside the details of its physical implementation. Additionally, ADTs furnish modularity, thus allowing the replacement of any part without affecting the rest.

Some ADTs provide linear relationships, such as stacks, queues, lists or sets, but oftentimes more complex relationships are necessary, such as hierarchical structures in trees or flat designs in graphs, in order to define a wide range of structures as different as file directories, decision processes or taxonomies.

3.2.2 Vocabulary about trees

It may be necessary to define some key words regarding the use of trees:

- *Node:* any entity being part of the tree (also known as vertex)
- *Edge:* any connection between two nodes (also known as arcs or links)
- *Root:* the single node being the highest in the tree
- *Child:* a node directly connected to another one, getting away from the root
- *Parent:* a node directly connected to another one, getting closer to the root
- *Leaf:* a node without any child (also known as external node)
- *Internal node:* a node with children
- *Sibling:* a collection of nodes having the same parent

- *Common Ancestor:* a common node for two other given nodes

- *Path:* the string of nodes to go from one initial node to one final node

- *Depth:* the path length from the root down to a given node

- *Height:* the path length from a given node up to the root

- *Layer:* the set of nodes being at the same depth

Regarding trees as hierarchical ADTs with no loops, the *arity* may be considered as the maximum number of children allowed per node. Putting aside the 1-ary tree, which may be seen as a degenerated tree, behaving just like a list or a daisy chain, the simplest tree may be considered the 2-ary tree, also known as a *binary tree.* It is to be said that this is the kind of tree being studied the most for its simplicity, as trees with greater arity may be considered just as extensions of it.

There are different designs when it comes to distributing the nodes in a tree, although the most typical one is trying to fill it up evenly throughout all layers. However, that depends on the number of nodes available, thus, the following cases may be observed:

- *full binary trees*, which are defined as the ones having just two children for every single node, except for the leaves, which obviously have none

- *complete binary trees*, whose difference with the former is that all leaves hanging out of a single node are at the same level

- *full and complete binary trees*, also known as *perfect binary tree*, whose difference with the above is that all leaves are at the lowest depth, this is, at height zero

Therefore, full and complete binary trees are going to be the preferred choice when deploying a tree on a resource optimization basis, meaning the allocation of the greatest amount of nodes within the minimum amount of layers.

3.2.3 Structural induction

Whilst mathematical induction is applied on natural numbers, it may be generalized to recursively defined structures by means of structural induction [34]. Their principles are similar, as both are based on two fundamental facts, such as an Inductive Basis and an Inductive Step. However, mathematical induction is built around the existence of an initial number 0 and a unique successor $n + 1$ for every single number n, whereas structural induction is founded on a minimal element being the initial one and a well-ordered partial order relation.

Structural induction is used to prove properties about a set of objects defined in a recursive manner, such as a set T of trees. For that matter, the two steps cited above apply, such as the following:

- First, show that the claim holds for the Basis Step of the definition of T
- Then, for the recursive cases of T's, namely Recursive Step, show that if the claim holds for all objects being part of a new one, then it also holds for it

Therefore, a proof by structural induction requires that every member of any recursively defined set has some particular property, which may be proved by first verifying it for the simplest elements, and in turn, for each way of building up complex elements out of simpler ones.

3.2.4 Binary trees

A set B of Binary trees may be defined in a recursive way as follows [37]:

- A tree with a single node r is inside B
- If r is a node and T_1 and T_2 are disjoint binary trees, this is, $T_1 \in B$ and $T_2 \in B$, then, the tree $T = (r, T_1, T_2)$ is also a binary tree, this is, T is inside B. T may be seen as a tree with root r, where r has the tree T_1 as left child and the tree T_2 as right child

After having defined binary trees, properties on such trees may be proofed by using structural induction, such as following the aforementioned definition of a tree.

- *Proposition P:* The number of nodes $|V|$ of a non-empty binary tree T is the number of its edges $|E|$ plus one, such that $|V| = |E| + 1$
- *Proof:*

  1. *Basis Step:* If T consists of a single root node r, then, by definition, $|V| = 1$ and $|E| = 0$, so $P(r)$ holds
  2. *Recursive Step:* If the vertex and edge sets of binary subtrees T_1 and T_2 are denoted by V_1, E_1, V_2, E_2, and assuming that both are already defined, then $P(T_1)$ and $P(T_2)$ hold.
     Therefore: $|V| = |V_1| + |V_2| + 1 = (|E_1| + 1) + (|E_2| + 1) + 1 = (|E_1| + |E_2| + 2) + 1 = |E| + 1$
  3. So, $P(T)$ holds

The aforesaid proposition proves that a given binary tree keeps all its nodes interconnected, hence, there may always be an available path between one source node and another destination node, and as a consequence, any given pair of leaves in a binary tree will have at least one path between them.

Taking that proposition into consideration, some functions may be also defined in order to further reason about trees and its properties, such as $|\cdot| : B \to \mathbb{N}$, returning the *number of nodes* being part of a binary tree, thus building up a relationship between binary trees and natural numbers.

* Definition of $|\cdot| : B \to \mathbb{N}$:

  i) $|r| = 1$

  ii) $|(r, T_1, T_2)| = 1 + |T_1| + |T_2|$

Additionally, another function may be defined, such as $h : B \to \mathbb{N}$, giving back the *height* of that tree.

* Definition of $h : B \to \mathbb{N}$:

i) $h(r) = 0$

ii) $h(r, T_1, T_2) = 1 + \max(h(T_1), h(T_2))$

With all that in mind, another proposition may be proofed by structural induction relating both the number of nodes and the height of a tree. Specifically, it sets the *upper bound* for the number of nodes $|T|$ within a tree of height $h(T)$, which is the case of a full and complete binary tree.

- *Proposition P:* $\forall T \in B : |T| \leq 2^{h(T)+1} - 1$

- *Proof:*

1. *Basis Step:* If T consists of a single root node r, then, by definition, $|r| = 1$, thus $h(r) = 0$, hence it is clear that $|r| \leq 2^{h(r)+1} - 1$

2. *Recursive Step:* If $T = (r, T_1, T_2)$, then T_1 and T_2 are both part of T, thus it may be assumed that $|T_1| \leq 2^{h(T_1)+1} - 1$ and $|T_2| \leq 2^{h(T_2)+1} - 1$. Hence:

$$\begin{aligned} |T| &= |(r, T_1, T_2)| \\ &= 1 + |T_1| + |T_2| \\ &\leq 1 + 2^{h(T_1)+1} - 1 + 2^{h(T_2)+1} - 1 \\ &\leq 2 \times 2^{\max(h(T_1, h(T_2))+1} - 1 \\ &= 2 \times 2^{h(T)} - 1 = 2^{h(T)+1} - 1 \end{aligned}$$

3. So, $P(T)$ holds

The aforesaid proposition proves that the number of nodes of a perfect binary tree is just that amount, as it may be checked out in Figure 3.11. Otherwise, the leaf nodes account for two to the power of the height of the whole tree, whilst the non-leaf nodes does it for the sum of all the upper layers, as it is exhibited in Table 3.14. Additionally, it is to be noted that the number of nodes standing in a particular layer within the perfect binary tree may be obtained by calculating two to the power of its corresponding height. However, it is to be reminded that binary trees might not always be full and perfect, and in such cases, the values obtained might differ from the aforesaid mentioned.

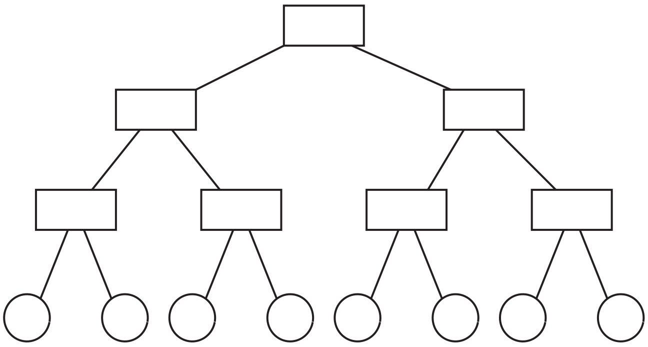


Figure 3.11: Perfect binary tree with height 3.

Table 3.14: Number of nodes in a perfect binary tree of height h.

Overall nodes	**Leaf nodes**	**Non-Leaf nodes**
$2^{h(T)+1} - 1$	$2^{h(T)}$	$2^{h(T)} - 1$

3.2.5 M-ary trees

Taking the definition of a binary tree as a base case scenario, it may well be extended to a set M of M-ary trees, which may also be defined recursively in the following way [120]:

1. A tree with a single node r is inside M
2. If r is a node and $\{T_1, T_2, \cdots, T_M\}$ are disjoint M-ary trees, which may be also expressed as $\{T_1, T_2, \cdots, T_M\} \in M$, then, the tree $T = (r, T_1, T_2, \cdots, T_M)$ is also an M-ary tree, this is, T is inside M. T may be seen as a tree with root r, where r have trees from T_1 to T_M as children.

M could be any natural number, such as 3-ary for trees with a maximum of three children, which are also known as ternary, as depicted in Figure 3.12, or 4-ary for trees being able to hold up to four children, which are also known as quaternary, as shown in Figure 3.13, or any other natural number.

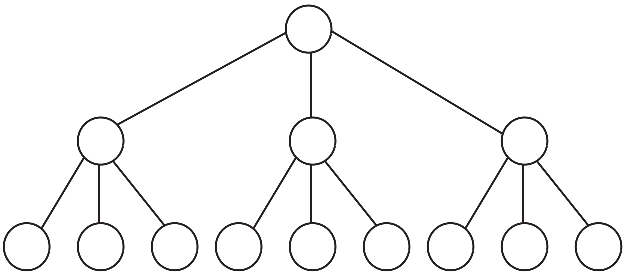


Figure 3.12: Perfect ternary tree with height 3.

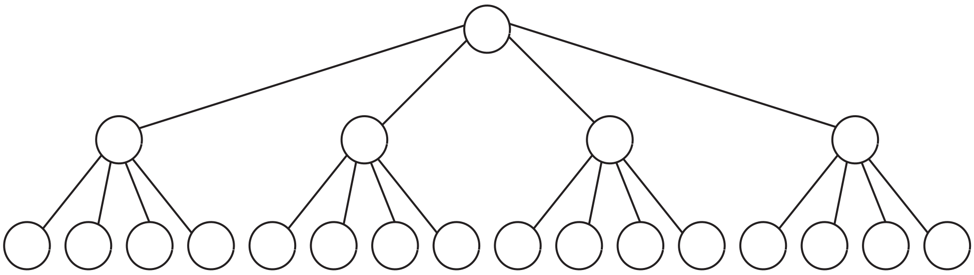


Figure 3.13: Perfect quaternary tree with height 3.

All properties previously defined for binary trees may be adapted for M-ary trees accordingly. Some of them remain unchanged, such as definitions of full and complete, which may be used both in binary and M-ary trees. Additionally, the functions defined above for getting the number of nodes or the height of a binary tree may easily be tailored to M-ary trees by just taking into account all subtrees involved.

However, the expression giving the upper bound for the number of nodes of an M-ary tree of height $h(T)$ must be adjusted, as shown in (3.18). First of all, the case where $M = 1$ must be distinguished from the rest of values of $M > 1 \,/\, M \in \mathbb{N}$ in order to avoid division by zero. It is to be reminded that such a tree may be called a degenerated tree, as it is effectively equivalent to a list or a daisy chain. With that in mind, the upper bound for the amount of nodes within a given M-ary tree of height $h(T)$ results as follows:

- If $M = 1 \quad \rightarrow \quad \forall T \in M : \quad |T| = h(T) + 1$

- If $M \geq 2 \quad \rightarrow \quad \forall T \in M :$

$$|T| \leq \frac{M^{h(T)+1} - 1}{M - 1} \tag{3.18}$$

Hence, if $M = 2$, the expression proved earlier for binary trees obviously apply, where $|T| \leq 2^{h(T)+1} - 1$.

Additionally, if $M = 3$, the expression obtained is $|T| \leq {}^{(3^{h(T)+1}-1)}/_{2}$, or if $M = 4$, the expression attained is $|T| \leq {}^{(4^{h(T)+1}-1)}/_{3}$, and so on.

Likewise, this expression may be tailored to calculate the leaf nodes and non-leaf nodes according in a perfect M-ary tree by extending those previously given for the binary tree, as seen in Table 3.15.

Table 3.15: Number of nodes in a perfect M-ary tree of height h.

Overall nodes	**Leaf nodes**	**Non-Leaf nodes**
$\frac{M^{h(T)+1}-1}{M-1}$	$M^{h(T)}$	$\frac{M^{h(T)}-1}{M-1}$

Hence, all conclusions obtained for binary trees regarding number of nodes and their interconnection may be extrapolated to M-ary trees.

Eventually, some topologies associated to partial mesh, along with the hub and spoke design, are going to be related in some way to the tree-like structures presented, showing their different behaviour according to the layer where each item is located.

3.3 Graph-like data structures

Graph theory have many applications in computer science, such as representing communication networks, computation flows, data structures or algorithms, as well as being a main topic within discrete mathematics. Hence, it may be interesting to review some of its key points.

3.3.1 Seven bridges of Königsberg

It is considered that graph theory started back in the 18th century, where Euler was thinking in a way to walk through the seven bridges of Königsberg, a city whose current name is Kaliningrad, crossing each one just once and coming back to the starting point. He came up with a solution, concluding that it was not possible to do so [6], but along the way, he set up the basics of topology and graph theory.

The former is related to the fact that he considered just the structural properties of the objects, and not their measures or their real shapes, thus keeping apart from geometry, whereas the latter refers to the fact that he abstracted away from the real features of the objects, just sticking to a set of *nodes* being joint together by a set of *edges* (both sets being obviously disjoint), where only the presence or absence of an edge between any pair of nodes is relevant.

Euler realized that the intermediate nodes along the path must have an even number of incident edges, being defined as *degree*, as it is always needed a way out of a node whenever you get in. Furthermore, he understood that the source and destination nodes might be allowed to have both an odd degree, but this condition do not meet the requirement of getting back to the starting point, concluding that such a case was not a solution for the problem proposed.

Hence, the only valid solution may be that where all nodes have an even degree. Unluckily, the scenario proposed showed 3 nodes with degree 3, and also 1 node with degree 5, as seen in Figure 3.14. Therefore, it is clear that the degree in all nodes is not even, as actually none of them are, thus there was no solution.

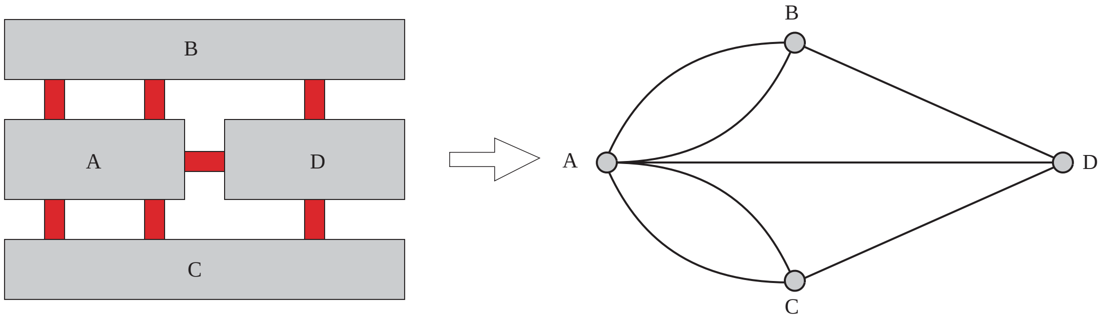


Figure 3.14: Topology of the 7 bridges of Königsberg, along with its graph.

3.3.2 Fundamentals on graphs

Basically, a graph may be seen as a diagram composed by some nodes, along with some edges joining certain pairs among such nodes. However, a mathematical abstraction may lead to the definition of a graph G as an ordered 3-tuple $(V(G), E(G), \psi_G)$, where the first term is a nonempty set of *nodes*, the second one is a set of *edges*, being disjoint from the former, and the third one is the *incidence function* that associates each of the edges with a pair of nodes, not necessarily being distinct [30].

It is to be mentioned that a graph is *connected* if there is a path between every pair of nodes, regardless how many intermediate nodes get involved in such a path. At this point, it is important to cite the difference between *directed* and *undirected* graphs, where the former states a direction in the edges, in a way that, given two ending points, paths in both ways might not be reciprocal, provoking the distinction between *indegree* of a node, namely the number of arriving edges, and *outdegree* of a node, namely the number of departing edges.

On the contrary, this does not apply in the latter, as the direction of the edges is irrelevant, hence, the *degree* of a node is the number of edges incident with it. In this context, it is to be noted that the number of nodes of odd degree happens to be even in every connected graph. Furthermore, it is worth saying that the sum of degrees of all nodes is always equal to twice the number of edges, applying to all graphs.

The difference between the aforesaid kind of graphs may be appreciated in Figure 3.15. On the right-hand side, a directed graph (V_1, E_1) is exhibited, where $V_1 = \{1, 2, 3\}$ and $E_1 = \{(1, 2), (1, 3), (2, 3), (3, 2)\}$, whilst on the left-hand side, an undirected graph (V_2, E_2) is exposed, where $V_2 = \{1, 2, 3\}$ and $E_2 = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$. Hence, both pictures clearly represent different graphs.

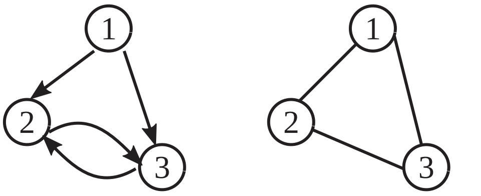


Figure 3.15: Directed -vs- undirected graphs.

On the other hand, many definitions in graph theory are related by graphical representations, such as two nodes being incident with a common edge are called *adjacent*, and so are two edges being incident with a common node, whilst an edge having the same nodes is called *loop*, or otherwise, it is called *link*, whereas a graph is *simple* if there are no loops and there are no more than one link joining together any given pair of nodes, or otherwise, it is known as *multigraph*. Additionally, regarding $2D$ graphs, it is to be pointed out that *flat* graphs are those where edges intersect only at nodes, thus meaning that all points may be embedded on the same plane.

Besides, graphs may be *weighted* if each edge bears a real number associated, called its weight. This is quite common in communication graphs, where weights account for a metric given to each edge, which leads to propose optimization problems in order to find a subgraph with maximum or minimum weight, such as determining the shortest path between any two given nodes. In this sense, Dijkstra algorithm does the job in an efficient way, by needing a polynomial time related to the number of nodes involved, as opposed to exponential [49].

Moreover, it is to be said that, while weighted graphs assign a given label to the edges according to any particular condition, it is also possible to associate some kind of *coloring* to the nodes, where the condition might be that adjacent nodes may need different colors. In this case, a graph is called M-colorable if M-colors may be used to paint all nodes in order not to get two adjacent nodes in the same color, where the minimum number needed is called chromatic number, whereas its counterpart for edges is named chromatic index. On a regular basis, it is considered that four colors are usually enough to cover up all nodes within a graph.

3.3.3 Eulerian and Hamiltonian graphs

Anyway, in homage to Euler, a nonempty connected *undirected* graph where there are no nodes with an odd degree is named *Eulerian cycle*, which permits to go across a path covering all edges just once and starting and finishing at the same node, whereas if there are just two nodes with an odd degree, it is called *Eulerian path*, which removes the constraint of returning to the start node. This brings about the definition of *Eulerian graphs* as those with a Eulerian cycle, along with *Semi-Eulerian graphs* as those with a Eulerian path.

Alternatively, regarding their *directed* counterparts, the conditions for a Eulerian graph are that it is balanced, meaning that the indegree and outdegree values for each node match, and additionally it is weakly connected, stating that the underlying undirected graph is indeed connected, which in other words means that there is a bidirectional path between any two given nodes. Likewise, the conditions for a Semi-Eulerian graph are the same as before, but accepting at most one node where its outdegree is a unit higher than its indegree, along with another one where its indegree is a unit higher than its outdegree.

On the other hand, focusing on nodes instead of edges, and regardless of whether it is directed or undirected, a nonempty connected graph where there is a path going through all nodes just once and starting and finishing at the same node is named *Hamiltonian cycle*, whilst if the restriction to return to the starting point is lifted, then it is called *Hamiltonian path.* This brings around the definition of *Hamiltonian graphs* as those with a Hamiltonian cycle, as well as *Semi-Hamiltonian graphs* as those with a Hamiltonian path. Furthermore, it is to be reminded that the task of testing a graph to verify whether it is Hamiltonian is an NP-Complete problem, as there is not a general expression to do it.

3.3.4 Geometrical graphs

When it comes to choosing a graph shape to implement a graph-like topology, many options are available, as graphs may have any type of distribution. Nevertheless, not all of them offer the same benefits in terms of performance or easiness of traffic forwarding, because the features of the shape selected may influence the outcome, hence choosing an appropriate one for a particular scenario may make a difference.

A typical option is to go for *regular geometrical shapes* when building up *undirected* graphs, as it may bring some types of symmetry, as well as some regular layout for the nodes and edges, including its lengths and angles. For instance, focusing on all of the regular polytopes presented above, they may do it for relatively easy implementations, which bring about *regular* graphs, as all nodes have the same degree, and they are also *cyclic* graphs, as all nodes in any given plane form a cycle. In this sense, the same geometrical pattern is repeated in all possible cycles, even though the value of the *girth* (which is defined as the shortest cycle contained inside a graph) depends on the number of sides of the regular polygon making up for their faces.

Furthermore, it is to be taken into account that all regular polytopes are considered to be *Hamiltonian* graphs, as they possess several Hamiltonian cycles for each node due to the diverse symmetries, which allows for different traffic forwarding strategies to be selected.

As a matter of fact, N-simplex shapes account for *complete* graphs, as every pair of nodes is joined by just one edge, thus all nodes are adjacents. With respect to N-orthoplex shapes, they are not complete, as each node is adjancent with the rest except for its opposite, as that link is missing, even though the opposite node may be reached directly through any of its adjacent nodes. Regarding N-hypercube shapes, each node only has as many edges as dimensions, which allows for each node to be adjacent to just other N vertices, although there is a number of redundant paths available to get to any other node. Therefore, these all are going to be considered as great options for graph designs as mesh topologies. Hence, some topologies associated to partial mesh may be referred to as graph-like structures, shaped as N-polytopes.

3.3.5 De Bruijn graphs

On the other hand, it is to be remarked the special properties of the *de Bruijn graphs*, which may be used when building up some grid network architectures. De Bruijn research was about finding the shortest circular superstring containing all available strings of length n within a given alphabet of length k, thus accounting for k^n overall combinations, where each of them had to be included just once [151].

The solution proposed for de Bruijn was a *directed* graph, which is often expressed as $B(k, n)$, where the B is a homage to de Bruijn. One of its main uses may be DNA shotgun sequencing, involving the reassembling of large strings, where the alphabet is composed by $k = 4$ nucleotides, known as $\{A,T,C,G\}$, although the binary alphabet

$k = 2$, being $\{0, 1\}$, may be the most commonly used in a variety of tasks, such as sequence alignment, rotating-drum angle encoding or even breaking key-lock systems.

In this context, the strings of length n may be also called *n-mers*, whereas any substring of length $n - 1$ may be also named $(n - 1)$*-mers*. Regarding the whole cyclic superstring, which brings embedded every single string happening just once, it is known as *de Bruijn sequence* and it may have a length k^n, thus being represented as a (k^n)*-mer*. Actually, there are a number of different de Bruijn sequences $B(k, n)$, depending on the values of k and n, according to the expression ${}^{(k!)^{k^{n-1}}}\!/_{k^n}$, where each one may be rotated leftwards or rightwards.

In order to build up a de Bruijn graph, the first thing to be done is to assign each one of the possible $(n - 1)$*-mers* to a different node. Therefore, a first $(n - 1)$*-mer* is linked to a second $(n - 1)$*-mer* so as to obtain an *n-mer*, where its $(n - 1)$-prefix is the former substring and its $(n - 1)$-suffix is the latter substring, hence the overlapping of two $n - 1$ substrings give out an n string, and the same procedure goes on to achieve each of the available strings.

This way, the edges of such a graph constitute all possible n strings, where every node within the directed graph may be balanced, having both an indegree and an outdegree as the size of the alphabet, and furthermore, the underlying undirected graph is obviously connected, thus proving that such a graph is Eulerian, meaning that all edges are crossed only once. In other words, a Eulerian cycle representing a shortest superstring containing every n string only once is attained.

Additionally, each de Bruijn graph with length n may be considered the *line digraph* of the length $n - 1$ by using an identical set of symbols, where the line digraph shows a dual representation of a direct graph, meaning that the edges in the original graph are substituted with nodes in the dual graph, and nodes in the original one are replaced by edges in the dual one, joining together the original adjacent edges.

This equivalence between nodes and edges through building up line digraphs provoke that Eulerian graphs in the original representations may be considered as Hamiltonian graphs in the dual representations, resulting that de Bruijn graphs are both Eulerian and Hamiltonian [40]. Therefore, it may be said that if a sequence $B(k, n)$ may be generated as a Hamiltonian cycle, then a sequence $B(k, n + 1)$ may be done as a Eulerian cycle, and this also applies the other way around, such that a sequence $B(k,n)$ created as Eulerian, induces a sequence $B(k,n-1)$ as Hamiltonian.

Due to its special features, the construction of *de Bruijn sequences* may be optimized by different techniques, even though some of them are only valid to binary alphabets and others may not be totally optimal. However, Wong proposed an efficient algorithm working for any type of alphabet [234], making possible to generate such sequences in $O(1)$-amortized time per symbol for any k-alphabet (bit for a binary alphabet), thus beating out the rest of generating methods described in the literature.

Focusing on *binary alphabets*, thus $k = \{0,1\}$, the method is quite straighforward, as it just involves checking whether a given string is a necklace, which may be defined as the lexicographically smallest string in an equivalence class of strings under rotation [198]. Hence, the expression to obtain the next string is (3.19).

$$f(b_1 b_2 \cdots b_n) = \begin{cases} b_2 b_3 \cdots b_n \overline{b_1} \text{ if } b_2 b_3 \cdots b_n 1 \text{ is a necklace;} \\ b_2 b_3 \cdots b_n b_1 \text{ otherwise.} \end{cases} \tag{3.19}$$

Likewise, generalizing for any *k-alphabet*, thus $k = \{0 \cdots k-1\}$, the aforesaid expression gets extended, where b is the highest value for $a_2 a_3 \cdots a_n b$ not being a necklace, or 0 otherwise. Hence, the expression to attain the next string is (3.20).

$$f_k(a_1 a_2 \cdots a_n) = \begin{cases} a_2 a_3 \cdots a_n b \text{ if } a_1 = k-1; \\ a_2 a_3 \cdots a_n (a_1+1) \text{ if } a_1 \neq k-1 \text{ and } a_2 a_3 ... a_n (a_1+1) \text{ is a necklace;} \\ a_2 a_3 \cdots a_n a_1 \text{ otherwise.} \end{cases} \tag{3.20}$$

As an example of generating de Bruijn graphs for binary alphabets, let us consider the construction of de Bruijn graphs for 3-length and 4-length words in both graphical and algorithmic ways. The graph for the former is built up in Figure 3.16, with $2^3 = 8$ nodes, where each one is assigned a 3-length word. This way, a Hamiltonian cycle may be achieved by picking up the value of each node, and coming back home.

If such a starting point is 000, then a de Bruijn sequence $B(2,3)$ may be obtained by taking the upper path when going rightwards and the middle path when coming back leftwards. Alternatively, a different sequence may be attained by taking the middle path when going rightwards and the lower path when coming back leftwards. Hence, the amount of different sequences is $2^{4-3} = 2$ possible distinct cycles, regardless the rotations performed in any of them.

Focusing on the first $B(2,3)$ sequence described above, it goes like this: 000, 001, 011, 111, 110, 101, 010, 100, 000. Leaving out the last string for being the initial one and taking the first bit of each string, it gives the 2^3-superstring 00011101 by taking the left-most bit of each string in a sequential way, which exhibits all possible combinations of the aforementioned strings only once by selecting each bit and the next ones in a circular way so as to form binary strings of length 3.

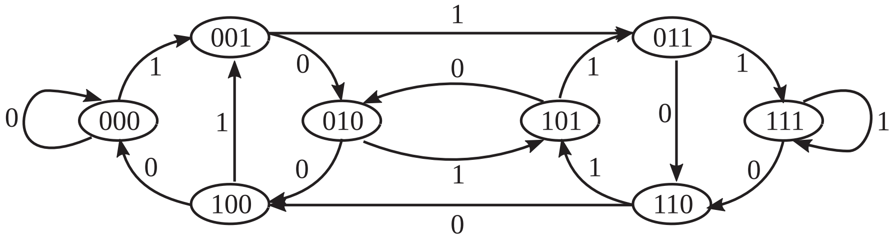


Figure 3.16: De Bruijn graph: Eulerian $B(2,4)$ and Hamiltonian $B(2,3)$.

Otherwise, a Eulerian cycle may be achieved by picking up the value of each node along with the weight of the next edge taken, and coming back to the starting point. If such a point is 000 and the edge taken is the loop with weight 0, making up 0000, then a de Bruijn sequence $B(2,4)$ may be attained by taking the upper path when going rightwards, then getting the whole right circumference, and in turn, making three sides of the rectangle, next visiting the middle nodes and finally coming back home to the initial point.

Overall, there are $2^{8-4} = 16$ possible distinct cycles, regardless the rotations. A particular $B(2,4)$ sequence taking 0000 as the initial point may go like this: 0000, 0001, 0011, 0111, 1111, 1110, 1101, 1011, 0110, 1100, 1001, 0010, 0101, 1010, 0100, 1000, 0000. Skipping out the last string for being the starting one and taking the first bit of each string, it is obtained the 2^4-superstring 0000111101100101, which exhibits the whole range of combinations of 4 binary strings by following the procedure described above for the $B(2,3)$ case.

Regarding the Wong algorithm, both sequences may be easily attained, considering that the set of necklaces for 3-length strings are $\{000, 001, 011, 111\}$, as all the other strings may be assimilated to any of those by applying rotation, whereas for 4-length words are $\{0000, 0001, 0011, 0111, 1111, 0101\}$. It is to be noted that a necklace may be seen as an equivalence class where bit rotation is the equivalence relation, considering the instance with lower numerical value as the equivalence class representative.

Necklaces are also known as a fixed necklaces, which may not be confused with free necklaces, also known as turnover necklaces or bracelets, where the strings are equivalent under both rotation and reflection. The calculation of how many different necklaces and bracelets are available for the strings of length n using a k-alphabet involves the Euler's totient function for a given integer n, expressed as $\varphi(n)$, which counts the positive integers lower than n being coprime with it.

Moreover, those concepts may also be applied to n circularly connected beads of certain colours, where k accounts for the colours available. It may be said that there are as many as $(n-1)!$ necklaces and $(n-1)!/2$ bracelets if all beads have a different colour, whereas those values get reduced as the range of colours at hand grows, which may be found out through the Pólya enumeration theorem.

Therefore, Table 3.16 shows the strings achieved by means of Wong algorithm, which are 000, 001, 011, 111, 110, 101, 010, 100, 000. Leaving out the final string achieved for being equal to the first one, and taking the first bit of each string in a sequential manner, the 2^3-superstring obtained is 00011101, which happens to be the same as that attained by following the Hamiltonian graph $B(2,3)$ presented above.

The order of the strings acquired are given by the output of the Wong algorithm for a binary alphabet ($k = 2$), starting with the all-zeros string of length n. Each given sequence, expressed by quoting its bits from left to right, such as $b_1 b_2 b_3$, is converted into a test sequence by adding up a bit 1 as a left bit shift, resulting in $b_2 b_3 1$. Afterwards, if the test sequence is a necklace, then the next sequence becomes $b_2 b_3 \overline{b_1}$, or else, the next sequence becomes $b_2 b_3 b_1$.

Likewise, table 3.17 shows the strings obtained by the Wong algorithm, which are 0000, 0001, 0011, 0111, 1111, 1110, 1101, 1011, 0110, 1100, 1001, 0010, 0101, 1010, 0100, 0000. Those strings are achieved following an analogous process to the

Table 3.16: Wong algorithm to achieve de Bruijn sequence $B(2,3)$.

Now: $\mathbf{b_1b_2b_3}$	**Test: $\mathbf{b_2b_3}$1**	**Necklace?**	**Next: YES: $\mathbf{b_2b_3\overline{b_1}}$ NO: $\mathbf{b_2b_3b_1}$**
000	001	Yes	001
001	011	Yes	011
011	111	Yes	111
111	111	Yes	110
110	101	No	101
101	011	Yes	010
010	101	No	100
100	001	Yes	000

previous case. Putting aside the final string for being the first one again, and taking the first bit of each string taken in an ordered manner, the 2^4-superstring attained is 0000111101100101, which is just the same one obtained by following the Eulerian graph $B(2,4)$ presented above.

Sticking to de Bruijn graphs for a generic k-alphabet, expressed as $B(k,n)$, it may be seen that each node i connects to other k nodes, which happen to be the ones inside the range going from $[k \times i] \ mod \ k^n$, all the way to $[k \times i + (k-1)] \ mod \ k^n$. Focusing on the binary alphabet ($k = 2$) along with an arbitrary length of strings (n), expressed as $B(2,n)$, each node i only gets connected to nodes $2i \ mod \ 2^n$ and $[2i+1] \ mod \ 2^n$, which effectively corresponds to apply a left bit shift to node i in binary, as the former injects a 0 at the least significant bit of i, whilst the latter does it with a 1.

When using binary alphabets, the forwarding path between any pair of nodes may be considered as left shifting one bit at a time in the source node, using as many bits as necessary to convert it into the destination node, which ensures a maximum path length of n hops. Therefore, de Bruijn graphs may be seen as a possible routing strategy for network communications, and in such a case, the key point may be to find out the largest substring of rightmost bits in the source node matching the substring of leftmost bits with the same length in the destination node, as only the remaining bits may have to be inserted in the source node in order to reach the destination node.

Table 3.17: Wong algorithm to achieve de Bruijn sequence $B(2, 4)$.

$\mathbf{b_1b_2b_3b_4}$	$\mathbf{b_2b_3b_4 1}$	**Necklace?**	**YES:** $\mathbf{b_2b_3b_4\overline{b_1}}$ **NO:** $\mathbf{b_2b_3b_4b_1}$
0000	0001	Yes	0001
0001	0011	Yes	0011
0011	0111	Yes	0111
0111	1111	Yes	1111
1111	1111	Yes	1110
1110	1101	No	1101
1101	1011	No	1011
1011	0111	Yes	0110
0110	1101	No	1100
1100	1001	No	1001
1001	0011	Yes	0010
0010	0101	Yes	0101
0101	1011	No	1010
1010	0101	Yes	0100
0100	1001	No	1000
1000	0001	Yes	0000

Additionally, this approach may be extended to any k-alphabet, by left shifting one symbol (going from 0 to $k-1$) at a time, thus reaching the destination node in at most n movements.

3.4 Torus topologies

Torus are shapes with particular mathematical characteristics from the point of view of geometry and topology, which may be applied to the design of multiprocessor interconnections and redundant designs in DCs [35].

3.4.1 Properties of a torus

Basically, a *torus* may be defined as a surface of revolution obtained when revolving a circle, also known as 1-sphere (S^1), around an axis, in a way that both geometrical figures are located in the same plane [230]. If such an axis is exterior to the circle, the resulting shape may be called ring torus, whereas if the axis is tangent to the circle, the resulting shape may be known as horn torus, or if the axis is interior to the circle, the resulting shape may be branded as spindle torus, and even if the axis passes through the central point of the circle, the resulting shape may be labelled as 2-sphere (S^2).

Among them all, the *ring torus* is the most usual one, that being the external shape of a doughnut, which is depicted in Figure 3.17, where it may be appreciated that r_1 indicates the section radius, also known as minor radius, and r_2 does it for the revolution radius, also known as major radius.

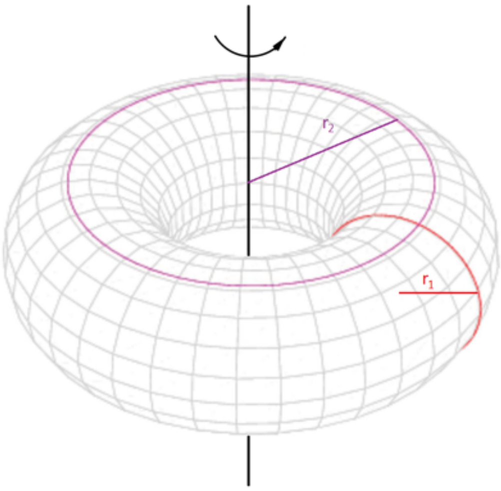


Figure 3.17: Ring torus geometry.

Topologically speaking, a ring torus may be defined as a closed surface, being the Cartesian product of two different circles, such as $S^1 \times S^1$, which may be seen as a 2-manifold with genus 1 (due to having a hole in its structure). Comparing that with a 2-sphere, both are compact and orientable, but the latter has genus 0. Hence, it may be said that a cup of coffee and a ring torus are both homeomorphic, as so are a convex regular polyhedra and a 2-sphere. However, a ring torus and the 2-sphere are not homeomorphic, for having different genus, which conveys a different Euler characteristic (χ), or neither are a cylinder and a Möbius strip, as only the former is orientable because the latter has a half-twist, or even neither are an $(N-1)$-sphere and an $\mathbb{R}^N$ space, as only the former is compact (provided that N is finite).

It is to be remarked the difference between a 1-sphere and a 2-disk (D^2), as the former is just the boundary of the latter, meaning that S^1 is a curve and D^2 is the surface comprising the circle and its interior space. Likewise, the same reasoning may apply to the difference betweeen a 2-sphere and a ball, where the former is a surface whilst the latter is a volume. And additionally, this also applies to a torus and a solid torus, that being built up by the expression ($S^1 \times D^2$), which comprises a torus and its interior volume. Therefore, the different pairs of items quoted are not homeomorphic, this is, they not share the same topological properties, as the envelope of a shape does not include the inner part of it.

The concept of torus may be extended to higher dimensions, considering that $T^N = S^1 \times S^1 \times \cdots \times S^1$. This means that the ring torus may be defined as the surface $T^2 = S^1 \times S^1$, whereas the lineal torus is $T^1 = S^1$, which is basically a circle, or the cubic torus is $T^3 = S^1 \times S^1 \times S^1$. This approach may be useful when dealing with torus-like designs, as those may be customizable by tailoring those compact N-dimensional manifolds.

It is to be pointed out that a ring torus is basically a $2D$ Euclidean surface, in spite of being deployed in a $3D$ Euclidean space. Additionally, the definition given for T^2 accounts for the Cartesian product of two 1-spheres, each of both having 2 dimensions, which accounts for an alternative representation as a surface in a $4D$ Euclidean space called the Clifford torus. However, the $3D$ representation may prevail, as it is far more intuitive.

Furthermore, Figure 3.18 reinforces the idea of a ring torus being a surface, as it shows how to convert a square whose opposite sides are glued together, also known as square torus or flat torus, into a ring torus in two steps. First of all, that square with identified sides gets wrapped in and a cylinder appears, getting the first circle, and afterwards, the cylinder is wrapped around and a ring torus shows up, getting the second circle. The inverse transformation may also be done by first unwrapping around the torus into a cylinder, and in turn, unwrapping it out into a square with identified sides. Hence, it is clear that a ring torus and a square torus are homeomorphic.

As a side note, squares with oriented edges are often used in topology to describe the features of surfaces, such as a torus, a Klein bottle, a sphere or a projective plane.

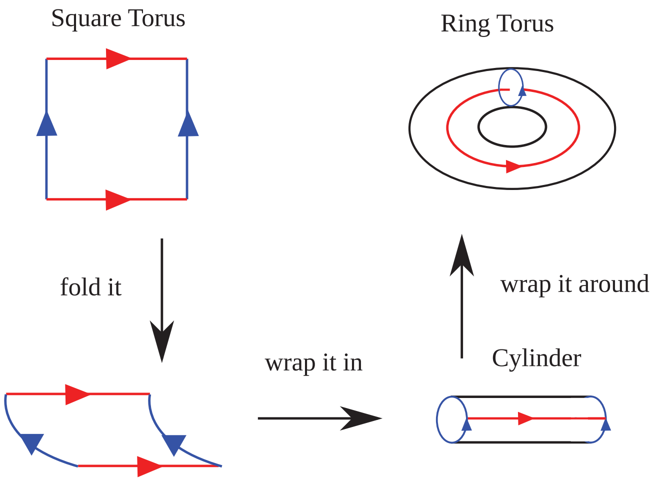


Figure 3.18: Forming a torus from a square.

3.4.2 Toroidal network designs

Focusing on topological network designs, it may be considered that a *linear torus* is equivalent to a ring network, because of its shape of a circle, getting all devices interconnected in a circular way. It is to be reminded that there is only one dimension available, thus the physical layout may be along a line, even though the logical layout may look like a circle, as the last item may have a wraparound link to the first one. Regarding its implementation, this design usually supports only a small amount of devices, hence, it may be reserved for simple deployments.

Moving on to a *ring torus*, it may seem equivalent to a squared mesh [206], or a grid, adding up further connections at the outer devices, known as wraparound paths, linking both external devices located in the same line, thus meeting the definition of a ring torus. Specifically, M should be a natural number being a perfect square, so that $\sqrt{M}$ items may be located per each side. All nodes get connected to their horizontal and vertical neighbours in a rectilinear fashion, and additionally, there are $2\sqrt{M}$ wraparound connections in the outward directions of each item located on the border, hence, all nodes have two horizontal-wise and two vertical-wise neighbours. With respect to its implementation, this design may support around 1000 devices, so it is recommended to put in place a higher dimension deployment for more than that.

Moving ahead to a *cubic torus*, it may look like a cubic mesh [205], and attaching extra wraparound links in order to interconnect devices being in the external positions of any of the three dimensional lines required. Specifically, M should be a natural number being a perfect cube, thus it is recommended to get aligned $\sqrt[3]{M}$ items in each dimension. On the other hand, this definition may be extended to any higher dimension, even though it might only be necessary in very large deployments.

It may seem clear that the higher the dimensions used, the benefits are better performances in terms of higher speed and lower latency, even though the drawbacks are the rise of costs and the complexity of interconnections and routing algorithms [19]. Therefore, a tradeoff must be always achieved, although the key point is to stick to the lower possible dimension. As a summary, the most basic types of toroidal designs are exhibited in Figure 3.19.

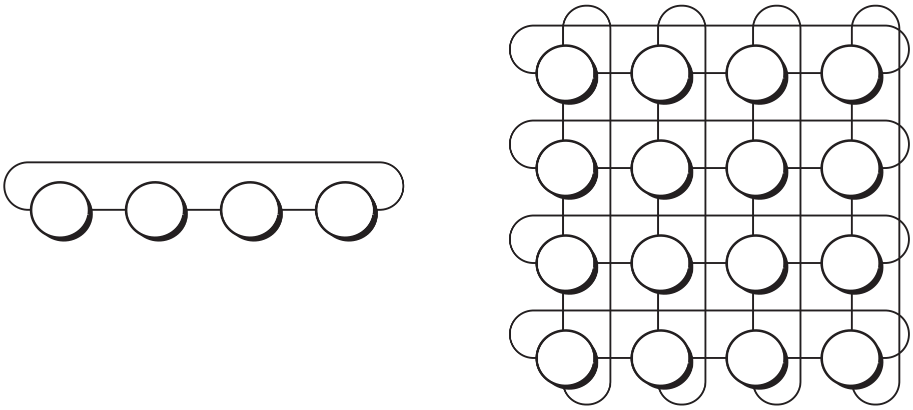


Figure 3.19: Toroidal structures: line (along $1D$) and ring (along $2D$).

Eventually, toroidal grid networks may be defined as grid networks of dimension N, with the particularity of having wraparound links for all nodes located on any edge. In this context, some topologies associated to toroidal grid designs are going to be proposed, such as toroidal ring and redundant ring, which are related to the torus structures as proposed for ring torus and linear torus with double links, respectively. With regards to cubic torus designs, those are not going to be considered herein as the number of hosts need not be that high in the proposed DCs.

3.4.3 De Bruijn torus

Regarding de Bruijn sequences, they may be considered as unidimensional, because they are represented just by strings. However, they may be extended to two dimensions, thus creating *de Bruijn tori* [93], which are composed by a toroidal array $r \times r'$ with wraparound edges, having the particularity that all $m \times m'$ toroidal subarrays available, formed with a particular k-alphabet, appear just once contiguously in it, thus the nomenclature goes like this: $(r, r'; m, m')_k$, where the most popular condition to achieve it is given in (3.21).

$$r \times r' = k^{m \times m'} \tag{3.21}$$

It is to be remarked that they have interesting applications in positional encoding and pattern location in two dimensions, thanks to the periodic perfect mapping of a whole surface without any repetition, which allows to associate a given area with its correspondent toroidal subarray [220].

It is to be mentioned that it may not always be possible to construct such a shape, even though in case of a squared pattern, thus $m = m'$, and m being even, a *square torus* always exist, such as $r = r'$, regardless the alphabet being employed [96], hence becoming $(r, r; m, m)_k$, where the aforesaid condition results in $r = r' = k^{m^2/2}$, therefore $(k^{m^2/2}, k^{m^2/2}; m, m)_k$. In this sense, $(4, 4; 2, 2)_2$ is the smallest binary square de Bruijn torus, meaning that a 4×4 toroidal array contains all $k^{m \times m} = 2^{2 \times 2} = 2^4 = 16$ binary 2×2 toroidal arrays just once, which are obtained by combining the 4 available positions in each one with the 2 binary values, as shown in Figure 3.20.

It is to be noted that the layout of 1's within that figure reminds of the shape of a Brigid's cross [201]. Furthermore, this picture may be called clockwise array, and along with its transposition, which may be obtained by matrix reflection over any of both its axis, also called counterclockwise array, they are both the only available representations of the smallest binary square torus, regardless of their row or column rotations, which may act as members of the same equivalence class under rotation.

The next binary square torus may be $(256, 256; 4, 4)_2$, which accounts for $2^{4 \times 4} = 2^{16}$ combinations of 4×4 toroidal arrays, whereas the following one may be $(2^{18}, 2^{18}; 6, 6)_2$, accounting for $2^{6 \times 6} = 2^{36}$ combinations of 6×6 toroidal arrays.

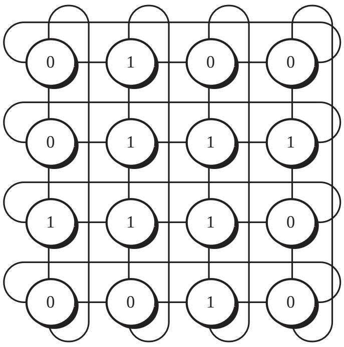


Figure 3.20: De Bruijn binary square torus with a squared pattern: $(4,4;2,2)_2$

On the other hand, the smallest ternary square torus may be $(9,9;2,2)_3$, accounting for $3^{2\times 2} = 3^4 = 81$ combinations of 2×2 toroidal arrays, whereas the smallest quaternary square torus may be $(16,16;2,2)_4$, having $4^{2\times 2} = 4^4 = 256$ combinations of 2×2 toroidal arrays.

Apart from the squared de Bruijn tori with squared patterns, the aforesaid expression permits some variations, such as designing a rectangular de Bruijn torus with a squared pattern, as $(16,32;3,3)_2$, or otherwise, attaining a squared de Bruijn torus with a rectangular pattern, as $(8,8;3,2)_2$. Likewise, a rectangular de Bruijn torus with a rectangular pattern is also possible, as $(8,32;2,4)_2$, or even different patterns such as L-shaped [155].

Focusing on the smallest binary square torus, thus $(4,4;2,2)_2$, and particularly in its clockwise version, all 2×2 toroidal arrays may be represented as in Figure 3.21, which exhibits a decomposition showing all 16 available combinations precisely once. It is to be noticed that the elements located in the last row and those in the last column form its matrices with the corresponding cells of the first row and the first column, respectively, due to the toroidal feature, which also supports another matrix formed by the elements located in the four corners. Besides, the 16 items in the 4×4 representation might be spotted in the top left corner of each of the 2×2 matrices, which are completed with the neighbors on the right, bottom and bottom right.

It may be appreciated that each 2×2 toroidal array has just 4 neighbours (top, bottom, left, right), where a shift is produced to the opposite way. This is, the top row

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \underset{\text{map}}{\longrightarrow} \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{bmatrix}$$

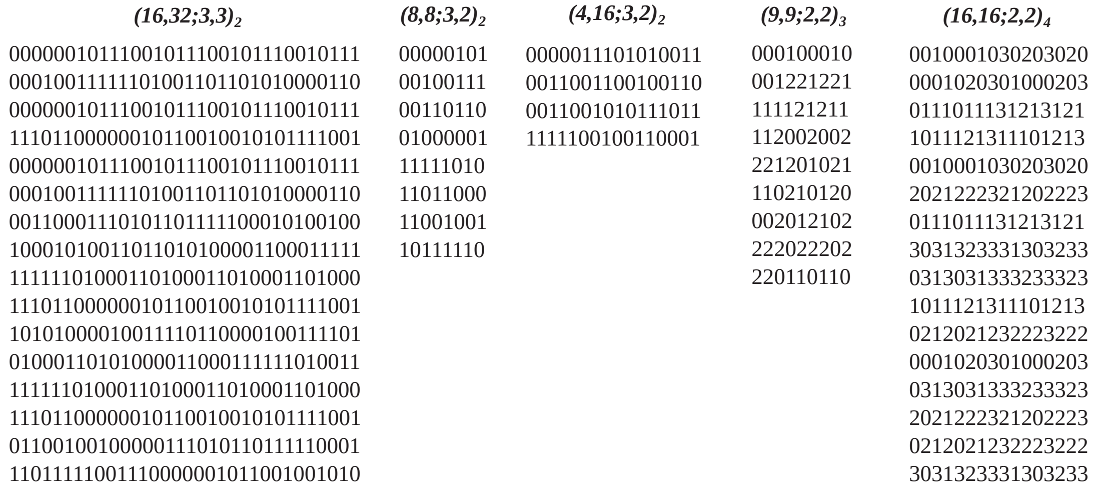
Figure 3.21: De Bruijn torus $(4,4;2,2)_2$: squared patterns.

is shifted to bottom row in the top neighbour, whereas the bottom row is shifted to the top row in the bottom neighbour. Likewise, the left column is shifted to the right column in the left neighbour, whilst the right column is shifted to the left column in the right neighbour, and that way, the toroidal shape remains.

Regarding the aforementioned mapping, it is also to be noted that the element located in the top left corner of a given pattern is identified in the same row and column within the de Bruijn torus as the squared pattern is located in the mapping. Therefore, the top left item within each pattern may be referred to as its *handle*, which is used to allocate such a pattern within the de Bruijn torus.

Furthermore, Figure 3.22 depicts instances of some other de Bruijn tori. Focusing on $(9,9;2,2)_3$, the de Bruijn torus is a 9×9 square and the pattern is a 2×2 square filled in with elements of a ternary alphabet, such as {0,1,2}, resulting in up to 81 unique patterns spotted with their corresponding handles, giving the distance from the top left item of each pattern to the top left item of the whole de Bruijn torus.

Additionally, de Bruijn sequences $B(k,n)$ may be considered as a $(k^n,1;n,1)_k$ de Bruijn torus, thus making for a linear torus with a linear pattern. As a side note, Table 3.18 exhibits some particular instances of the former whose length is up to 64, as well as its nomenclature as the latter.



$(16,32;3,3)_2$	$(8,8;3,2)_2$	$(4,16;3,2)_2$	$(9,9;2,2)_3$	$(16,16;2,2)_4$
00000010111001011100101110010111	00000101	0000011101010011	000100010	0010001030203020
00010011111101001101101010000110	00100111	0011001100100110	001221221	0001020301000203
00000010111001011100101110010111	00110110	0011001010111011	111121211	0111011131213121
11101100000010110010010101111001	01000001	1111100100110001	112002002	1011121311101213
00000010111001011100101110010111	11111010		221201021	0010001030203020
00010011111101001101101010000110	11011000		110210120	2021222321202223
00110001110101101111100010100100	11001001		002012102	0111011131213121
10001010011011010100001100011111	10111110		222022202	3031323331303233
11111101000110100011010001101000			220110110	0313031333233323
11101100000010110010010101111001				1011121311101213
10101000010011110110000100111101				0212021232223222
01000110101000011000111111010011				0001020301000203
11111101000110100011010001101000				0313031333233323
11101100000010110010010101111001				2021222321202223
01100100100000111010110111110001				0212021232223222
11011111001110000001011001001010				3031323331303233

Figure 3.22: Various bidimensional toroidal arrays.

3.4.4 De Bruijn hypertorus

The concept of de Bruijn torus may be easily extended to higher dimensions, hence achieving de Bruijn hypertorus. This way, such a shape of dimension N may be defined as $\overline{r} = (r_1 \cdots r_N)$, whereas a hypertoroidal pattern of such a dimension may be denoted by $\overline{m} = (m_1 \cdots m_N)$, with $r_i > m_i$ and $1 \leq i \leq N$, which may also be quoted as $(\overline{r}; \overline{m})_k^N$ [105].

Obviously, all combinations of such a hypertoroidal pattern using an alphabet k may be obtained exactly once within the de Bruijn hypertorus, thus attaining a kind of a periodic hypermapping. Likewise, the most popular condition cited for de Bruijn tori may also be extended to de Bruijn hypertori, resulting in expression (3.22), which applies to any dimension N, even up to infinite.

$$\prod_{i=1}^{N} r_i = k^{\prod_{i=1}^{N} m_i} \tag{3.22}$$

In this sense, it is possible to impose the condition that $r_1 = \cdots = r_N = r$, achieving k-ary N-cube de Bruijn hypertori, along with $m_1 \cdots m_N = m$, obtaining k-ary N-cube patterns. In such a case, it results in $r_1 = \cdots = r_N = k^{m^N/N}$, where each particular node has up to 2 neighbors in each dimension, thus its predecessor and successor modulo r, thus accounting for up to $2N$ neighbours overall.

Table 3.18: De Bruijn sequences expressed as de Bruijn tori with length up to 64.

items	$\mathbf{B(k,n)}$	$\mathbf{(k^n,1;n,1)_k}$	**An instance of that de Bruijn Sequence**
$\{0,1\}$	$B(2,2)$	$(4,1;2,1)_2$	0011
$\{0,1\}$	$B(2,3)$	$(8,1;2,1)_2$	00010111
$\{0,1\}$	$B(2,4)$	$(16,1;2,1)_2$	0000100110101111
$\{0,1\}$	$B(2,5)$	$(32,1;2,1)_2$	00001010111011000111110011010010
$\{0,1\}$	$B(2,6)$	$(64,1;2,1)_2$	0000001000011000101000111001001011001101001111010101110110111111
$\{0\cdots2\}$	$B(3,2)$	$(9,1;2,1)_3$	001122102
$\{0\cdots2\}$	$B(3,3)$	$(27,1;3,1)_3$	000111212220101200202102211
$\{0\cdots3\}$	$B(4,2)$	$(16,1;2,1)_4$	0011223302103132
$\{0\cdots3\}$	$B(4,3)$	$(64,1;3,1)_4$	00011101002022212112200303332322233131133013203210310231201230213
$\{0\cdots4\}$	$B(5,2)$	$(25,1;2,1)_5$	00102112041422430332313440
$\{0\cdots5\}$	$B(6,2)$	$(36,1;2,1)_6$	001020304051121314152232425334354455
$\{0\cdots6\}$	$B(7,2)$	$(49,1;2,1)_7$	00102112041306140315055162252353436442463326545660
$\{0\cdots7\}$	$B(8,2)$	$(64,1;2,1)_8$	0010203040506071121314151617223242526273343536374454647556576677

Sticking to the case where $N = 3$, the aforementioned expression becomes (3.23), regardless whether the de Bruijn 3D hypertorus or the 3D hypertoroidal pattern are cubic, where the size of all dimensions is the same, or otherwise, rectangular prism.

$$r_1 \times r_2 \times r_3 = k^{m_1 \times m_2 \times m_3} \tag{3.23}$$

In this sense, the smallest cubic de Bruijn hypertorus using a cubic pattern is $(256;2)_8$ [94], which may also be denoted by $(256,256,256;2,2,2)_8$, where the aforesaid condition may get converted into $(k^{m^3/3}, k^{m^3/3}, k^{m^3/3}; m,m,m)_k$.

Alternatively, the smallest rectangular prism de Bruijn hypertorus using a cubic pattern is $(16,4,4;2,2,2)_2$, where that pattern happens to be binary. In this context, a *b-cube* may be defined as a $2 \times 2 \times 2$ binary cubic hypertoroidal array, whose shape is shown in Figure 3.23, where the element at the front, top, left is its handle, which allows to allocate each pattern within the whole shape.

This way, a $16 \times 4 \times 4$ de Bruijn hypertoroidal array formed by such b-cubes contains each available b-cube just once, hence including only one instance of all $2^{2\times2\times2} = 256$ possible b-cubes. Therefore, $(16,4,4;2,2,2)_2$ makes for the smallest binary three-dimensional de Bruijn hypertorus, which is exhibited in Figure 3.24.

This picture may be seen as a sequence of 16 toroidal arrays with dimensions 4×4 being interconnected in a circular way, which may help visualize instances

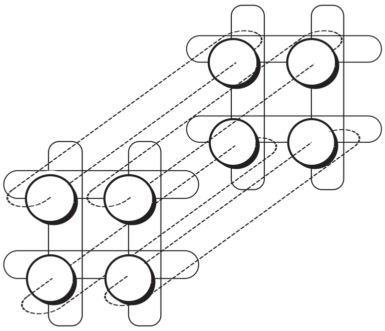


Figure 3.23: Three-dimensional toroidal array ($2 \times 2 \times 2$ nodes).

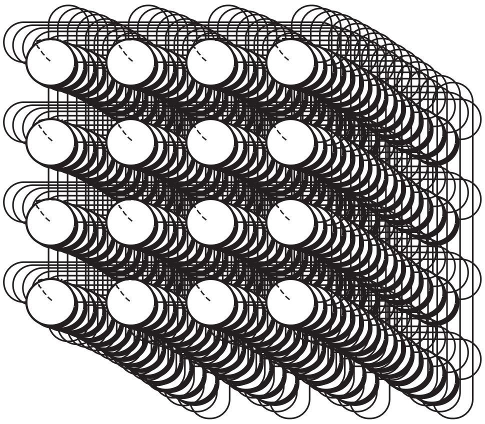


Figure 3.24: Three-dimensional toroidal array ($16 \times 4 \times 4$ nodes).

of $(16, 4, 4; 2, 2, 2)_2$. Hence, each of those 4×4 bidimensional arrays may be easily identified from 0 all the way to 15, as depicted in Table 3.19 showing a given instance.

Focusing on that particular instance, some interesting features may be spotted which might not necessarily be the case in all instances, such that it may be determined that each of the 16 toroidal arrays contains exactly the same number of zeros and ones, being 8 of each kind, or otherwise, every row and every column within any of those toroidal arrays contains one zero and three ones, or the other way around.

However, the main point of the instance proposed is the requirement to spot each available b-cube exactly once. In this context, it is to be said that each b-cube is being expressed as the product of its front 2×2 matrix by its back 2×2 matrix, thus the handle of a b-cube is the item being on the top left corner of the front matrix.

Taking just one of those 2×2 matrices, they are holding 4 nodes, each of them being occupied by a binary value, thus $\{0, 1\}$, resulting in a total amount of $4^2 = 16$ different combinations of such matrices. Furthermore, considering a b-cube as the product of two of those matrices (the front one and the back one), then there is an overall amount of $16^2 = 256$ different combinations of b-cubes.

In order to better organize all those combinations, 16 groups have been built up, each of them having one of the 16 combinations available as its front matrix. Furthermore, there are 16 elements within each group, thus representing each one of the combinations available for the back matrix.

In this sense, Table 3.20 exposes an instance of a $16 \times 4 \times 4$ de Bruijn hypertorus with its 256 b-cubes along with their handle, denoting the distance from that item to the front, top, left item of the rectangular prism de Bruijn hypertorus. Besides, all those b-cubes follow the same organization by groups and within each group [36].

Regarding the location of each b-cube, it may be said that the values of the front matrix are found in the 4×4 toroidal array denoted by the first value of the handle $(0 \cdots 15)$, where its row is given by the second one and its column is stated by the third one, both $(0 \cdots 3)$, whilst the rest of elements in the front matrix occupy the right, bottom and bottom right positions related to the handle, and the values of the back matrix are just behind those of the front matrix, in the successive 4×4 array.

Additionally, it is to be noted that the 16 items within each group have their handles in a different 4×4 array, meaning that the front top left element of each item within a group stands in a unique array. Also, if the handle of the first group is taken as a reference for each of the 16 arrays, it happens that each group has their handles in a different array, but in the same relative position out of the aforesaid reference.

3.5 Summary

In this chapter, some mathematical foundations have been presented to be applied on the topology models for DC, which will be studied in the following chapter. Regarding geometry, regular polytopes of dimension N have been inferred by starting with lower dimensions and adding up more dimensions until generic results for

Table 3.19: An instance of a 16 × 4 × 4 de Bruijn hypertorus.

Array ID	**Bidimensional Array**	**Array ID**	**Bidimensional Array**
0	$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$	1	$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$
2	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$	3	$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$
4	$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$	5	$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$
6	$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$	7	$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$
8	$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$	9	$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
10	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$	11	$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$
12	$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$	13	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$
14	$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$	15	$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$

Table 3.20: An instance of all unique 256 b-cubes in a 16 × 4 × 4 de Bruijn hypertorus.

b-cube	handle	b-cube	handle	b-cube	handle	b-cube	handle
$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(0,0,0)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(3,0,3)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(12,0,2)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(13,3,2)
$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(4,0,2)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(15,1,1)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(2,0,1)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(11,2,3)
$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(1,0,0)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(8,0,0)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(5,1,2)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(14,2,3)
$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(7,3,1)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(6,2,3)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(9,2,0)	$\begin{bmatrix}0&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(10,0,1)
$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(1,0,1)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(0,0,1)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(13,3,3)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(14,2,0)
$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(5,1,3)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(12,0,3)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(3,0,0)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(8,0,1)
$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(2,0,2)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(9,2,1)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(6,2,0)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(15,1,2)
$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(4,0,3)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(7,3,2)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(10,0,2)	$\begin{bmatrix}0&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(11,2,0)
$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(4,1,2)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(7,0,1)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(0,1,0)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(1,1,0)
$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(8,1,0)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(3,1,3)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(6,3,3)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(15,2,1)
$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(5,2,2)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(12,1,2)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(9,3,0)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(2,1,1)
$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(11,3,3)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(10,1,1)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(13,0,2)	$\begin{bmatrix}0&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(14,3,3)
$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(7,0,0)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(6,3,2)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(3,1,2)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(0,1,3)
$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(11,3,2)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(2,1,0)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(5,2,1)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(14,3,2)
$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(4,1,1)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(15,2,0)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(8,1,3)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(1,1,3)
$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(10,1,0)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(9,3,3)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(12,1,1)	$\begin{bmatrix}0&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(13,0,1)
$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(12,3,2)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(15,0,1)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(8,3,0)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(9,1,0)
$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(0,3,0)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(11,1,3)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(14,1,3)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(7,2,1)
$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(13,2,2)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(4,3,2)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(1,3,0)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(10,3,1)
$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(3,3,3)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(2,3,1)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(5,0,2)	$\begin{bmatrix}0&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(6,1,3)
$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(5,2,3)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(4,1,3)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(1,1,1)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(2,1,2)
$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(9,3,1)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(0,1,1)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(7,0,2)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(12,1,3)
$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(6,3,0)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(13,0,3)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(10,1,2)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(3,1,0)
$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(8,1,1)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(11,3,0)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(14,3,0)	$\begin{bmatrix}0&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(15,2,2)
$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(2,0,3)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(1,0,2)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(14,2,1)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(15,1,3)
$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(6,2,1)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(13,3,0)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(0,0,2)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(9,2,2)
$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(3,0,1)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(10,0,3)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(7,3,3)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(12,0,0)
$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(5,1,0)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(4,0,0)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(11,2,1)	$\begin{bmatrix}0&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(8,0,2)
$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(9,0,1)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(8,2,1)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(5,3,3)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(6,0,0)
$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(13,1,3)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(4,2,3)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(11,0,0)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(0,2,1)
$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(10,2,2)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(1,2,1)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(14,0,0)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(7,1,2)
$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(12,2,3)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(15,3,2)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(2,2,2)	$\begin{bmatrix}0&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(3,2,0)

b-cube	**handle**	**b-cube**	**handle**	**b-cube**	**handle**	**b-cube**	**handle**
$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(3,0,2)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(2,0,0)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(15,1,0)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(12,0,1)
$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(7,3,0)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(14,2,2)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(1,0,3)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(10,0,0)
$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(0,0,3)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(11,2,2)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(4,0,1)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(13,3,1)
$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(6,2,2)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(5,1,1)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(8,0,3)	$\begin{bmatrix}1&0\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(9,2,3)
$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(8,2,0)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(11,0,3)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(4,2,2)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(5,3,2)
$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(12,2,2)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(7,1,1)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(10,2,1)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(3,2,3)
$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(9,0,0)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(0,2,0)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(13,1,2)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(6,0,3)
$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(15,3,1)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(14,0,3)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(1,2,0)	$\begin{bmatrix}1&0\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(2,2,1)
$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(15,0,0)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(14,1,2)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(11,1,2)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(8,3,3)
$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(3,3,2)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(10,3,0)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(13,2,1)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(6,1,2)
$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(12,3,1)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(7,2,0)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(0,3,3)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(9,1,3)
$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(2,3,0)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(1,3,3)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(4,3,1)	$\begin{bmatrix}1&0\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(5,0,1)
$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(6,3,1)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(5,2,0)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(2,1,3)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(3,1,1)
$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(10,1,3)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(1,1,2)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(4,1,0)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(13,0,0)
$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(7,0,3)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(14,3,1)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(11,3,1)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(0,1,2)
$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(9,3,2)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(8,1,2)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(15,2,3)	$\begin{bmatrix}1&0\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(12,1,0)
$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(13,2,3)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(12,3,3)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(9,1,1)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(10,3,2)
$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(1,3,1)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(8,3,1)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(15,0,2)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(4,3,3)
$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(14,1,0)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(5,0,3)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(2,3,2)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(11,1,0)
$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(0,3,1)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(3,3,0)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(6,1,0)	$\begin{bmatrix}1&1\\0&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(7,2,2)
$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(14,1,1)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(13,2,0)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(10,3,3)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(11,1,1)
$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(2,3,3)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(9,1,2)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(12,3,0)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(5,0,0)
$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(15,0,3)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(6,1,1)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(3,3,1)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(8,3,2)
$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(1,3,2)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(0,3,2)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(7,2,3)	$\begin{bmatrix}1&1\\0&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(4,3,0)
$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(11,0,2)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(10,2,0)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(7,1,0)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(4,2,1)
$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(15,3,0)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(6,0,2)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(9,0,3)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(2,2,0)
$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(8,2,3)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(3,2,2)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(12,2,1)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(5,3,1)
$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(14,0,2)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(13,1,1)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(0,2,3)	$\begin{bmatrix}1&1\\1&0\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(1,2,3)
$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&0\end{bmatrix}$	(10,2,3)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\0&1\end{bmatrix}$	(9,0,2)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&0\end{bmatrix}$	(6,0,1)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&0\\1&1\end{bmatrix}$	(7,1,3)
$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&0\end{bmatrix}$	(14,0,1)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\0&1\end{bmatrix}$	(5,3,0)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&0\end{bmatrix}$	(8,2,2)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}0&1\\1&1\end{bmatrix}$	(1,2,2)
$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&0\end{bmatrix}$	(11,0,1)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\0&1\end{bmatrix}$	(2,2,3)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&0\end{bmatrix}$	(15,3,3)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&0\\1&1\end{bmatrix}$	(4,2,0)
$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&0\end{bmatrix}$	(13,1,0)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\0&1\end{bmatrix}$	(12,2,0)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&0\end{bmatrix}$	(3,2,1)	$\begin{bmatrix}1&1\\1&1\end{bmatrix}\times\begin{bmatrix}1&1\\1&1\end{bmatrix}$	(0,2,2)

higher dimensions have been achieved, resulting in just three of those shapes, such as N-simplex, N-hypercube and N-orthoplex.

Moreover, some background on trees has been examined, starting from binary ones and extending them all the way to M-ary ones. Besides, graph theory has been reviewed, pointing out the difference between directed and undirected graphs, the importance of Hamiltonian and Eulerian cycles, the interesting features of the geometrical graphs, along with the basics of the construction of de Bruijn graphs, which have been carried out both graphically and algorithmically.

Furthermore, with respect to mathematical topology, toroidal surfaces have been introduced, along with some characteristics which in turn have been associated with network designs. Moreover, de Bruijn tori have been studied as an extension of de Bruijn graphs in two dimensions, which have been further enlarged with an extra dimension to form de Bruijn hypertori in three dimensions, where some instances haven been shown in all cases.

Additionally, all these mathematical structures presented in this chapter may be used in order to optimize the DC designs, specifically regarding the network interconnections among hosts. This way, each DC topology being introduced in chapter 4 is going to be related to one of the aforementioned structures, which in turn will be used in chapter 5 to obtain models for those DC, where each one may take the most appropriate instance out of such structures.

Chapter 4

Common network topologies for Data Centres

Prior to obtain mathematical models for the topologies proposed, each of them is going to be reviewed in order to present their main features, and furthermore, such topologies may be assigned to appropriate geometrical shapes, chosen from those described in the previous chapter, in order to better carry out the models later on. Among those topologies, partial mesh ones are going to be studied more in depth, as they are the most usual topologies employed in real implementations.

The hosts present in the topologies may be interconnected by a number of switches, those being linked together according to a given topology. However, it is not the case of one topology being the best, but the performance achieved in different circumstances might differ depending on the designs, as each one has its own benefits and drawbacks. Hence, any of them might be the best option in a given situation, but not in others.

As stated before, it is to be reminded that most of the switching interconnection topologies being used in production networks may be classified between two broad categories, such as tree-like and graph-like designs.

On the one hand, the former ones are hierarchical, in a sense that not all switches play the same role, which brings different features for the switches standing in different layers. This way, hosts are only hanging on switches being located in the lower layer, also being referred to as end switches, whereas the rest of the layers play interconnection roles, along with aggregation tasks.

On the other hand, the latter ones do not have any implicit hierarchy, hence all switches may have hosts hanging on them, meaning that all of them are end switches. Additionally, more freedom of design is permitted, thus opening a full range of possibilities, although geometrical shapes may be the most popular because of their different symmetries and easy designs, even though performance may be worse.

Furthermore, as stated before, fog domains keep far less users and traffic than their cloud counterparts, therefore, complex topological designs are going to be avoided as the amount of resources being necessary to get them working do not compensate the gains in performance rates. Therefore, the following topologies may be a selection of candidate designs for implementing DCs with small to medium size aimed at fog computing environments.

4.1 Partial mesh: tree-like designs

Regarding tree-like designs, it may be noted that back in the 50s, telephony traffic was ever growing and the design of Clos networks was suggested so as to forward switch telephone calls efficiently [45]. The key point was to employ equipment with multiple interconnection stages to get the calls ready. This way, redundant paths between source and destination devices were available, hence permitting an ongoing phone call not to be blocked by other incoming calls.

Next, in the 90s, Ethernet switches were catching on, taking the place of hubs and repeaters, and the idea of Clos networks was extended in order to attain cost-effective deployments with low operational complexity [191]. Basically, the idea was the creation of multistage topologies being built up with commodity switches, also known as commercial-of-the-shelf (COTS) hardware, thus reducing both capital expenditure (CAPEX), as acquiring switches might be cheaper, and operational expenditure (OPEX), as maintaining them might be easier.

Then, in the 21st century, DC facilities still keep those ideas under different approaches, such as two-stage layouts, three-stage designs, or even more, depending on the needs of users and throughput [209]. Cloud computing deployments may need complex designs in order to achieve great performance with a huge amount of traffic, although that is not the case for fog scenarios, where the throughput is far less, thus restricting the number of switches and their interconnection topologies.

Therefore, two main topologies are going to be selected for tree-like designs, as they are the ones being the most popular in small to medium-sized designs. One of them is fat tree, which accounts for three layers of switches, whereas the other one is leaf and spine, which does it for two layers of switches.

4.1.1 Fat tree

Fat tree is a tree-like design, with *three layers of switches*, where the lower one is called *edge*, the middle one is called *aggregation* and the upper one is called *core*. Furthermore, hosts are only connected to the lower layer, as it happens with the rest of tree-like topologies. The reason for name of this topology is because of its design being like an inverted tree, having a reduced number of links on the top and a great number of them on the bottom. Besides, each top-layer item plays the role of a root as it grows downwards, thus reinforcing the view of a tree being fat [101].

Fat tree designs are deeply influenced by parameter K, and that is why the topology is often times called as *K-ary fat tree*. Parameter K is an even number and it defines the whole structure, such as the number of ports in all switches, the number of hosts connected to each lower layer switch, the number of pods defined, the number of switches within each pod, the overall number of devices located at each layer and the number of links among those devices [4].

Fat tree offers full mesh connectivity within each *pod*, which may be defined as a bunch of lower layer switches and middle layer switches logically grouped, along with the correpondant hosts being connected to those lower layer switches [210]. It is to be noted that there are $^{K}/_{2}$ switches of each type within a pod, and as there are $^{K}/_{2}$ hosts per each lower layer switch, it accounts for an amount of $^{K^2}/_{4}$ hosts per pod.

That matches with the number of ports per switch, which happens to be K, forcing lower and middle layer switches to have half of their ports looking upwards, whereas the other half are looking downwards. Obviously, upper layer switches have all their ports looking downwards, as there is no other item above them. On the other hand, each host have just one port, which gets it connected to a lower layer switch, that being its gateway to reach the rest of the hosts deployed within the topology.

Therefore, this design provides full mesh connectivity between hosts hanging on different switches within the same pod, resulting in $K/2$ redundant paths for intrapod commmunications. Nevertheless, hosts connected to the same switch have only 1 path, as each host is connected to a lower layer switch with just one link, hence there is no redundancy for intraswitch commmunications.

Additionally, fat tree also offers full mesh connectivity among pods, meaning that each pod has a path to the rest of them, as there is always an interconnection between a given switch located in the middle layer of a certain pod and each one of the upper layer switches, which in turn may lead to the other pods. However, this fact results in partial mesh connectivity between any two hosts belonging to different pods, as not all middle layer switches are linked to all upper layer switches. Anyway, the number of redundant paths available for interpod communications grows up to $(K/2)^2 = K^2/4$.

In summary, the distance between hosts depends whether they are connected to the same switch, being just 1 hop away, or whether they belong to the same pod, thus being just 2 hops away, or otherwise, whether they are held in different pods, hence being just 3 hops away.

An example of a fat tree network architecture is shown in Figure 4.1, where parameter $K = 4$ and oversubscription ratio is $1 : 1$, meaning that all of the links being expected in the design are available, thus none of them are left out. In this sense, it is to be said that not all the possible links are included in some implementations in order to save costs, and in such cases, the oversubscription ratio is the ratio between the number of all possible links according to the topology design and the number of links actually implemented in the deployment, which may or may not be the same.

To sum it all up, Table 4.1 shows the most relevant values for fat tree topologies, all of them related to parameter K, whereas Table 4.2 exhibits the number of redundant paths between hosts depending on their number of hops away, which may be found out throughout the expression $(K/2)^{hops-1}$.

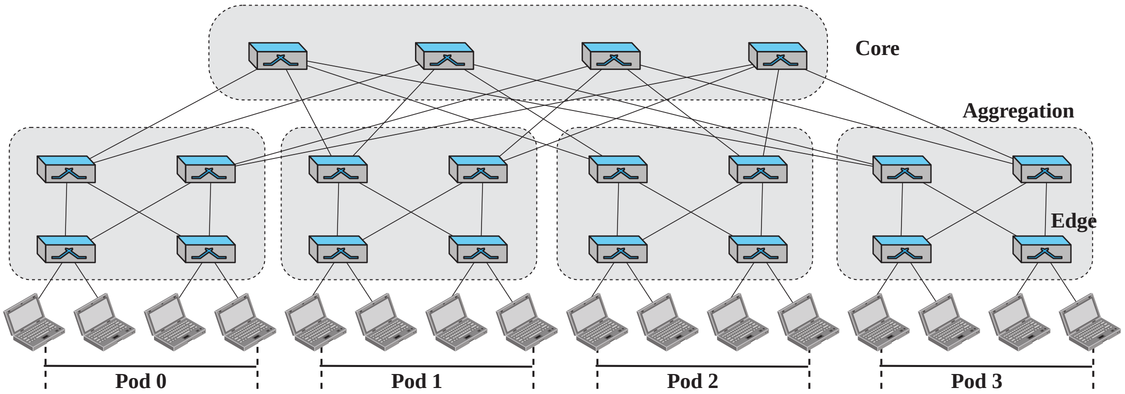


Figure 4.1: Fat tree architecture for $K = 4$ and oversubscription ratio 1:1.

Regarding the first table, the expressions for a generic K value are presented, as well as the values for the most common fat tree designs, such as $K = 4$ and $K = 8$.

Table 4.1: Fat tree most relevant values.

	K-ary	**K = 4**	**K = 8**
Number of pods	K	4	8
Number of switches in a pod	K	4	8
Number of ports per switch	K	4	8
Number of Hosts per edge switch	$K/2$	2	4
Num. aggregation switches within a pod	$K/2$	2	4
Num. edge switches within a pod	$K/2$	2	4
Num. hosts within a pod	$K^2/4$	4	16
Number of links within a pod	$K^2/2$	8	32
Total core switches in a topology	$K^2/4$	4	16
Total aggregation switches in a topology	$K^2/2$	8	32
Total edge switches in a topology	$K^2/2$	8	32
Total switches in a topology	$5K^2/4$	20	80
Total hosts in a topology	$K^3/4$	16	128
Total links in a topology	$3K^3/4$	48	384

4.1.2 Leaf and spine

Leaf and spine is another tree-like design with just *two layers of switches*, where the lower one is called *leaf* and the upper one is called *spine*. Furthermore, hosts are

Table 4.2: Number of redundant paths in fat tree among hosts.

	$(\mathbf{K/2})^{\mathbf{hops-1}}$	$\mathbf{K = 4}$	$\mathbf{K = 8}$
Paths between two hosts being 1-hop away	$(K/2)^0$	1	1
Paths between two hosts being 2-hop away	$(K/2)^1$	2	4
Paths between two hosts being 3-hop away	$(K/2)^2$	4	16

only connected to the lower layer, as it happens with the rest of tree-like topologies [140]. This topology does not depend on any parameter such as K, thus the designs have more degrees of freedom.

The main feature of leaf and spine is the full mesh connectivity between both layers of switches, thus offering as many redundant paths between hosts located in different switches as the number of upper layer switches [7]. In such a case, those hosts are considered to be 2 hops away. On the other hand, hosts being connected to the same switch just have 1 single link between them, considering them to be just 1 hop away.

This kind of topology fits better the requirements of modern DCs, such as steadier values for latency and jitter, as the number of hops away is just 1 or 2, although the drawback yields in scalability, as this design is not fit if the number of hosts is very high, needing an extra third layer of switches to cope with it, called superspine [227].

Those features allow leaf and spine designs to enhance east-west traffic, meaning data flows among devices allocated inside the DC, which is ever growing due to the extensive use of virtualization and hyperconverged infrastructures. It is to be said that traditional Clos-based DC was mainly focused on north-south traffic, meaning traffic flows getting in and out of the DC looking for foreign devices. In this sense, the term used to expand the latter is called scaling up, whilst the expansion of the former is called scaling out.

However, as virtualization techniques were catching on, traffic within the DC significatively increased and leaf and spine seemed the appropriate solution to deal with it. Despite of this, it is to be reminded that fat tree may offer a more scalable solution as the number of hosts grows, even though losing out in performance as such a number shrinks, hence a comparison may need to be undertaken to assess which one better fits the requirements expected in each particular case.

An example of a leaf and spine architecture is shown in Figure 4.2, with 8 switches located in the lower layer and other 4 switches in the upper layer, all of them being interconnected in a full mesh fashion. If all switches were to have the same number of ports, each lower layer switch will have four ports to connect hosts, with a total of 32 hosts. However, this is not usually the case, allowing for many other possibilities when undertaking a design of this topology.

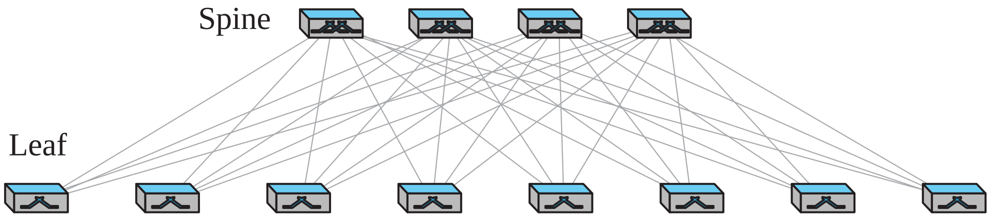


Figure 4.2: Leaf and spine architecture with 8 leaves and 4 spines.

4.2 Partial mesh: graph-like designs

As stated before, graphs may account for multiple types of topologies, as basically any design may be put in place. However, focusing on network topologies, it may be interesting to look for graphs with some specific characteristics, such as undirected, connected and cyclic structures with symmetry, which allow for redundant paths and easy forwarding expressions among nodes.

There may be different sort of shapes meeting these requirements, although the ones being more appropriate are the regular geometric shapes. In this context, synthetic geometry applies, as just vertices and edges of such shapes are considered, hence not considering any characteristics out of analytic geometry because no system of coordinates needs to be adopted.

On the other hand, trees might be considered to be graphs with further restrictions, in a way that a tree may be seen as a specific subset of graphs with some particular features, such as having the concept of root nodes or leaf nodes, as well as not having cyclical links. In fact, trees may connect nodes in a hierarchical way, whereas graphs may connect nodes in any possible way. It is to be remarked that, according to graph theory, a graph may be contemplated as a formal abstract way to represent a network, thus being a collection of objects being interconnected in a certain manner.

Anyway, no matter whether a topology is tree-like or graph-like, it is clear that any pair of hosts being connected to a topology belonging to any such categories, as exposed herein, will get at least a continuous path from a source host to a destination host, as all nodes are connected. Hence, communication between them both may obviously happen, provided that their connecting links are up and running.

Focusing on interconnections, different options arise, even though partial mesh topologies include a larger amount of options, as severe restrictions are not imposed on designs, that being the case of full mesh or quasi full mesh topologies. However, some degrees of symmetry may be interesting in order to achieve redundant paths with easy path forwarding policies, which may well apply for routing in overlay networks.

Therefore, two related topologies are going to be selected for graph-like designs regarding partial mesh, as they may perfectly fit in small to medium-sized designs, achieving a tradeoff between complexity of the network and effectiveness in path forwarding. First, plain N-hypercube is seen, and then, folded N-hypercube is shown.

4.2.1 N-hypercube

N-hypercube is a graph-like topology, following the shape of a hypercube of dimension N, also referred to as Q_n or plain N-hypercube. In that graph, there is a switch located in each vertex of the hypercube, where the interconnections among switches are given by the edges forming the hypercube and the hosts are connected to any of the switches involved.

The fact that this topology is a graph-like design makes all switches within the topology work as end switches, such that all of them may carry hosts connected, thus reducing the distance among them [87]. On the other hand, maintenance and complexity may get higher than their counterparts within tree-like designs, even though it might not be the case for small deployments.

The number of nodes and edges among them in an N-hypercube topology is given by the dimension N, such that there must be 2^N nodes being equally distributed and interconnected by $N \times 2^{N-1}$ edges [60]. Those expressions offer the same results as those obtained back in (3.16) and in Table 3.12 for $k = 0$ (nodes) and $k = 1$ (edges). As per the number of links from each node to directly connected nodes, there are N. On the other hand, the number of links to directed connected hosts depend on how many hosts hang on each switch, which depends of the implementation.

The dimension N selected may be the lowest being able to hold the number of switches required in a given deployment, even though it is strongly recommended that the number of nodes is a power of 2 in order to optimize the structure, thus being able to use all potential redundant paths offered by the topology. Figure 4.3 shows pictures of N-hypercube designs for an N value ranging from 0 all the way to 4, although any higher value may also be permitted.

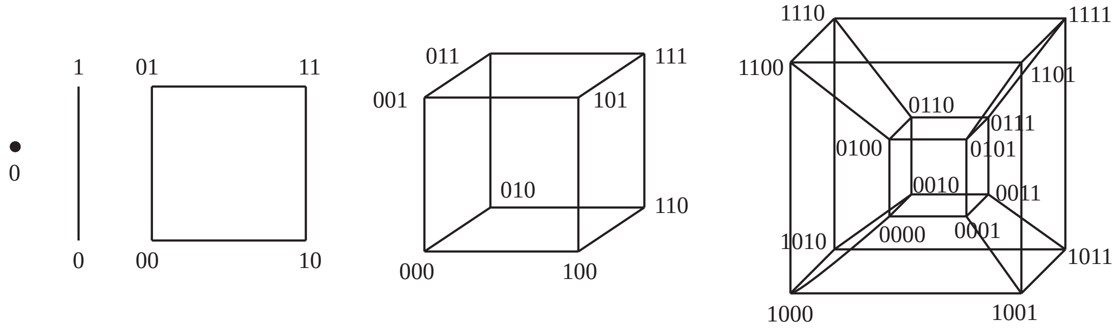


Figure 4.3: N-hypercube architectures for $N \in \{0 \cdots 4\}$.

According to that picture, every node is identified by a binary number of N bits, where there is an initial node with all bits being $0s$. Moreover, moving along a particular i-th dimension implies swapping the i-th bit of the source identifier, which covers the case where a message from a given node moves along to a neighbouring node in that particular i-th dimension. It is to be noted that the N dimensions are identified within the range going from 0 to $N - 1$ in decimal format.

Regarding the distance between any two different nodes, an easy way to find it out is to apply the logical operator *XOR* between the binary identifiers of the source node and the destination node, where the position of $1s$ found in that operation may point out the different dimensions between source to destination dimensions, also known as mismatching dimensions, whilst the overall amount of $1s$ may indicate the number of such mismatching dimensions between source to destination, meaning the distance between them both, measured in nodes away.

Hence, in order to travel such a distance from source node to destination node, a message in the source switch may move through all those dimensions to get to the destination switch, regardless the order in which such dimensions are traversed. This fact may allow for different redundant paths, accounting for the factorial of such a distance, thus making possible to apply different load balancing policies.

Alternatively to the binary *XOR*, that distance may also be found out by working with the decimal identifiers of source and destination nodes. This procedure relays on extracting the value of each bit position in them both by means of the appropriate arithmetic operations in both nodes in decimal format, and in turn, checking them out in order to see whether the same bits within each node match or otherwise, where the number of mismatches account for the distance between that couple of nodes.

Furthermore, the maximum distance between any pair of given nodes is N, which accounts for the case where there is no coincidence at all between source and destination binary identifiers. This case only happens with opposite nodes, meaning that each node only have one opposite node, being at a distance N, where the rest of the nodes are nearer. In order to calculate the *opposite node* of a given node i, it may be done by performing its 1's complement in binary form, as well as by applying the expression $(2^N - 1 - i)$ in decimal form.

Additionally, it is possible to calculate how many nodes are at a given distance of one particular node. To start with, there is only one node at a distance zero of node i, which is obviously itself. Hence, the rest of the nodes may be at a variable distance k, being greater than 0 and up to N. In this context, the number of nodes being at a distance k is given by the value of the cell found when crossing the N-th row and the k-th column of the *Pascal's triangle*, as shown in Figure 4.4.

It is to be reminded that each value on the Pascal's triangle may be obtained by adding up the two values located just above, this is, the value on the N-th row and the k-th column is the result of adding up the value on the $(N-1)$-th row and the $(k-1)$-th column along with the value on the $(N-1)$-th row and the k-th column, whenever those values are depicted in the triangle. Alternatively, each value may

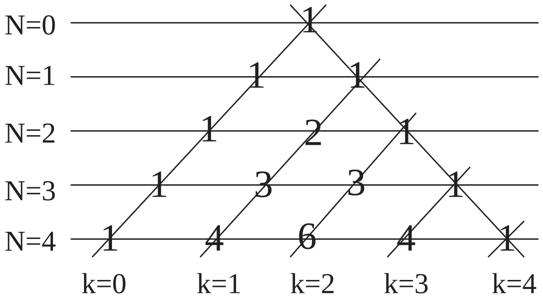


Figure 4.4: Pascal's triangle for the first rows.

also be obtained by means of applying the binomial coefficient as in (4.1), where each value relates to the number of nodes being at a distance k.

$$W_{k=0}^{N}\binom{N}{k} = W_{k=0}^{N}\frac{N!}{k! \times (N-k)!} = \binom{N}{0};\binom{N}{1};\binom{N}{2};\cdots;\binom{N}{N-1};\binom{N}{N} \quad (4.1)$$

As an example, a node i in a square ($N = 2$) has 1 node at a distance zero (itself), 2 nodes at a distance one (its direct neighbours) and 1 node at a distance two (its opposite node), whereas the sequence given by the 2nd row of Pascal's triangle is just $1; 2; 1$. Likewise, a node i in a cube ($N = 3$) has 1 node at a distance zero, 3 nodes at a distance one, 3 nodes at a distance two and 1 node at a distance three, whilst the sequence stated on the 3rd row of Pascal's triangle is just $1; 3; 3; 1$.

4.2.2 Folded N-hypercube

Folded N-hypercube is an extension of N-hypercube, also referred to as FQ_n, where all opposite nodes are joint together by extra edges [240]. Opposite nodes are easy to find out, as one is the 1's complement of the other. Those extra links make reduce the distance between opposite nodes to just 1, leaving the maximum distance between any two nodes to $\lceil N/2 \rceil$, which accounts for the successive integer from $N/2$, in case that value is not integer.

On the other hand, it is to be said that if $N/2$ is *not an integer*, then $\lfloor N/2 \rfloor$ means its predecessor integer, whilst $\lceil N/2 \rceil$ refers to its successor integer. In other words, under that condition, $\lfloor N/2 \rfloor$ may be seen like asking for the integer part of $N/2$, whereas $\lceil N/2 \rceil$ may be viewed like asking for the integer part of a value, and in turn, adding up 1.

Else, when $^N/_2$ is indeed *an integer*, then those three expressions match, such that $^N/_2 = \lfloor ^N/_2 \rfloor = \lceil ^N/_2 \rceil$.

Furthermore, $x_{|n}$ is the equivalent of asking for the remainder of the integer division between x and n, just like asking for x *modulo* n, also known as x *mod* n, which indicates the application of the arithmetic operation called congruence modulo n.

This reduction on distance among nodes, allowing also for more redundant paths as a new route through the opposite node is available, helps improve performance, although at the cost of adding more links and some complexity [56]. Figure 4.5 exhibits how to convert a plain N-hypercube into a folded N-hypercube, where it may be seen a link between any pair of opposite nodes, where the different nodes are being named according to the convention explained for the plain one, this is, in binary format using N bits, as depicted in Figure 4.6.

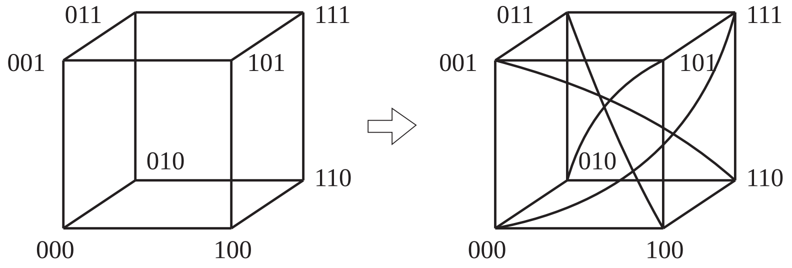


Figure 4.5: Converting a plain N-hypercube into a folded N-hypercube.

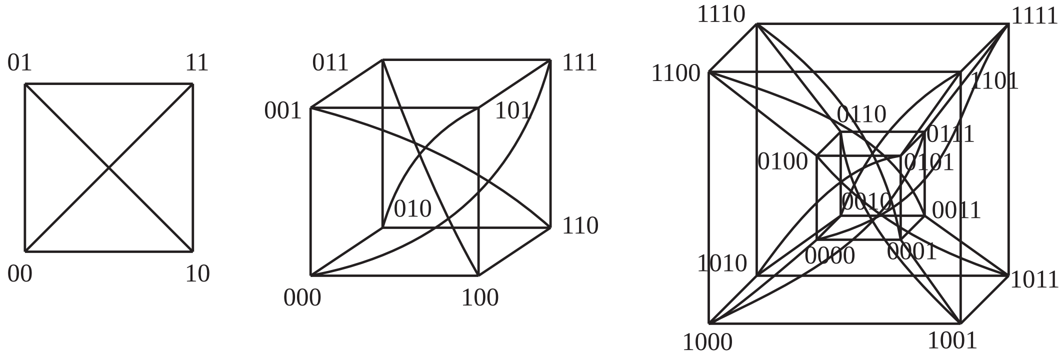


Figure 4.6: Folded N-hypercube architectures for $N \in \{2 \cdots 4\}$.

As a matter of fact, the number of nodes is the same as in plain N-hypercube, that being 2^N equally distributed nodes, which are now interconnected by $(N+1) \times 2^{N-1}$

edges. Regarding the number of links from each node to directly connected nodes, there are $N+1$, whilst the number of links to directed connected hosts depends on the implementation. Therefore, the number of k-facets present in a folded N-hypercube is just the same as in a plain N-hypercube, which was already exposed in Table 3.12, except the number of edges, which now accounts for $(N+1) \times 2^{N-1} = 2^{N-1} \times \binom{N+1}{1}$.

With respect to the distance between any two diverse nodes, two strategies must be confronted, such as that not using the path through the opposite node (called the opposite link), or otherwise, using it. Hence, the proper way may be to assess the distance with both and select the one offering a shorter path, or both in case of a tie.

In that sense, the former may be done by applying the logical operator *XOR* between source and destination identifiers in binary format, where the number of $1s$ gives the amount of mismatching dimensions, thus making up for the distance between them when the opposite link is not employed.

Otherwise, the latter may be done by also applying the same logical operator *XOR*, where the number of $0s$ gives the matching dimensions, and an extra unit must be added so as to count up for the link going to the opposite node, hence making for the distance between them when the opposite link is indeed employed.

Anyway, it may be clear that node i has a node at a distance zero (itself), and the rest of nodes may be at a distance in the range between 1 and $\lceil N/2 \rceil$. In order to find the number of nodes being at any given distance, it may be obtained by three different procedures.

The first method involves the value found in the cross of the N-th row and the k-th column of the folded Pascal's triangle, which is a variation of the Pascal's triangle, where the first column ($k=0$) is left as it is and the right side of the triangle is bent over the left, such that the last column ($k=N$) gets over the second column ($k=1$).

This operation provokes that cells in the columns from $k=1$ to $k=\lfloor N/2 \rfloor$ accumulate two values in each one, which may be added up, resulting that column $k=1$ holds the result of the addition of its own value along with the value on column $k=N$, whilst column $k=2$ holds the sum of its own value and that of column $k=N-1$, and so on. However, it may be considered that if N is odd, the column $\lceil N/2 \rceil$ is left alone, whereas if N is even, then $\lfloor N/2 \rfloor = \lceil N/2 \rceil = N/2$ and this issue does not happen.

The process of folding the Pascal's triangle is depicted in Figure 4.7, where each cell shows either the result of its corresponding addition or its original value.

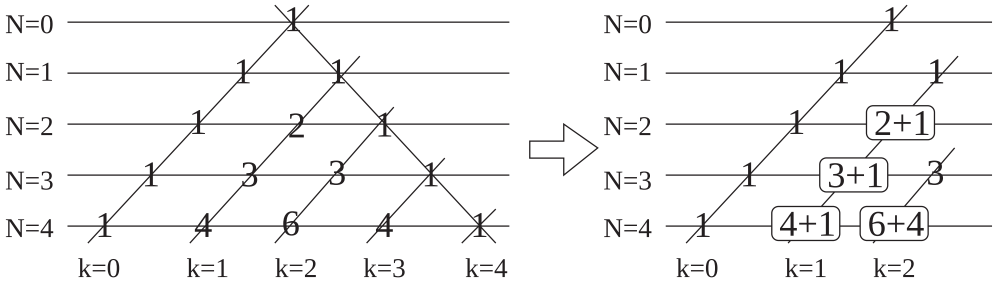


Figure 4.7: Converting a Pascal's triangle into a folded Pascal's triangle.

Therefore, the number of nodes standing at a distance k of any particular node out of N is found by looking for the value located right at the cross of the N-th row and the k-th column, as depicted in Figure 4.8.

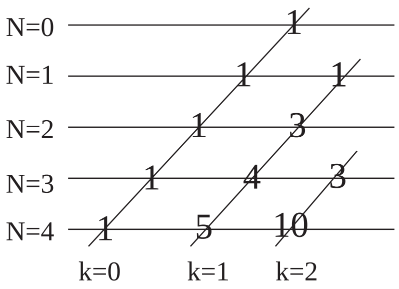


Figure 4.8: Folded Pascal's triangle for the first rows.

Regarding the features of the folded Pascal's triangle, it is clear that it does maintain a sequence of 1s in its column $k = 0$, each cell keeps the sum of its two upper cells and the sum of each row is still a power of 2, but it does neither contain a sequence of natural numbers in column $k = 1$, such as $\{a_n = N\}$, nor a sequence of triangular numbers in $k = 2$, such as $\{a_n = {}^{N\times(N-1)}/_{2}\}$, nor a sequence of tetrahedral numbers in $k = 3$, such as $\{a_n = {}^{N\times(N+1)\times(N+2)}/_{6}\}$, nor a sequence of Fibonacci numbers in the diagonal sums, such as $\{a_n = a_{n-1} + a_{n-2}\}$. Nonetheless, it fits the task of giving out how many nodes stand at a distance k in a folded N-hypercube of dimension N.

As an example, a node i in a folded square ($N = 2$) has 1 node at a distance zero (itself) and 3 nodes at a distance one (its direct neighbours), whilst the sequence given by the 2nd row of folded Pascal's triangle is just $1; 3$. Likewise, a node i in a folded cube ($N = 3$) has 1 node at a distance zero, 4 nodes at a distance one and 3 nodes at a distance two, whereas the sequence stated on the 3rd row of folded Pascal's triangle is just $1; 4; 3$.

The second method involves the use of an expression related to binomial coefficients. In this context, it is to be reminded that the mathematical definition of factorial, being $n! = n \times (n-1)!$, may yield to $(n-1)! = n!/n$. Hence, it is easy to see that $0! = 1!/1 = 1$, which brings to $(-1)! = 0!/0$, thus resulting in an indeterminate form in $\mathbb{R}$. It is to be said that in case the Gamma function is not applied, the former result may not be considered, hence making possible for it to be skipped.

Additionally, a correction factor needs to be applied in rows where N is odd, as the column $k = \lceil N/2 \rceil$ is singled out after the triangle left bending process. Therefore, that corrective coefficient may only be active in such cases, where just $k = N - k + 1$, whilst being inactive in the rest of cases. This may be achieved by dividing every term by $\lfloor \frac{i}{N-i+1} + 1 \rfloor$, or multiplying each term by $2^{-\lfloor \frac{i}{N-i+1} \rfloor}$, as both halve the result just in the case of interest, while keeping neutral on the other cases, as shown in (4.2).

$$W_{k=0}^{\lceil N/2 \rceil} \left(\binom{N}{k} + \binom{N}{N-k+1} \right) \times 2^{-\lfloor \frac{k}{N-k+1} \rfloor} \tag{4.2}$$

For instance, here they are the first values obtained with the aforesaid expression, which are just the same as those attained with the folded Pascal triangle.

- $N = 2 \rightarrow w_k = \left(\binom{2}{0} + \cancel{\binom{2}{3}} \right) \times 1; \left(\binom{2}{1} + \binom{2}{2} \right) \times 1 \rightarrow w_k = 1; 3$

- $N = 3 \rightarrow w_k = \left(\binom{3}{0} + \cancel{\binom{3}{4}} \right) \times 1; \left(\binom{3}{1} + \binom{3}{3} \right) \times 1; \left(\binom{3}{2} + \binom{3}{2} \right) \times 1/2 \rightarrow w_k = 1; 4; 3$

- $N = 4 \rightarrow w_k = \left(\binom{4}{0} + \cancel{\binom{4}{5}} \right) \times 1; \left(\binom{4}{1} + \binom{4}{4} \right) \times 1; \left(\binom{4}{2} + \binom{4}{3} \right) \times 1 \rightarrow w_k = 1; 5; 10$

The third method involves an alternative expression, which ends up with diverse values depending on the parity of N, and it is shown in (4.3).

$$W_k = \begin{cases} \text{for } 0 \leq k < \lceil N/2 \rceil \,, & \dbinom{N+1}{k} \\ \text{for } k = \lceil N/2 \rceil \,, & \begin{cases} \text{for even } N, & \dbinom{N+1}{N/2+1} \\ \text{for odd } N, & \dbinom{N}{\lceil N/2 \rceil} \end{cases} \end{cases} \tag{4.3}$$

For example, here they go the first values attained with the expression exposed, which are just the same as in the previous methods.

- $N = 2 \rightarrow w_k = \binom{3}{0}; \binom{3}{2} \rightarrow w_k = 1; 3$
- $N = 3 \rightarrow w_k = \binom{4}{0}; \binom{4}{1}; \binom{3}{2} \rightarrow w_k = 1; 4; 3$
- $N = 4 \rightarrow w_k = \binom{5}{0}; \binom{5}{1}; \binom{5}{3} \rightarrow w_k = 1; 5; 10$

Eventually, N-hypercube topology might be considered as plain N-hypercube, for having just that shape. Then, folded N-hypercube might be considered as a variation of it, but it is obviously not the only possible one. There are many other variations available such as multiple twisted N-hypercube or double looped N-hypercube, although just the folded one will be accounted here, as those may offer better performances at the cost of making them harder to manage.

4.3 Comparing tree-like -vs- graph-like designs

It may be interesting to compare the number of nodes and edges for all four topologies described above. First of all, a precondition must be established in order to be able to compare them in a fair way, such that the number of hosts all over the topology must be the same in all cases.

In this context, it is necessary to clarify some definitions, such as *end switches* for those switches where hosts may be connected to, *source switch* for the particular switch connected to a *source host*, and *destination switch* for the given switch connected to a *destination host*. Obviously, the concepts or node and switch are interchangeable, as it is the case for edge and link.

Therefore, all switches in a graph-like design are end switches, whereas only the switches located on the lower layer in a tree-like design may be labelled this way. The easiest way to achieve the same number of hosts within all topologies appears to be setting the same number of end switches, where each of them may have the same number of hosts hanging on.

However, there is an issue with fat tree, as parameter K imposes all values, including both the number of end switches, that being $K^2/2$, and the number of hosts per end switch, that being $K/2$, making an overall amount of $K^3/4$ hosts within the whole topology. Hence, as fat tree is the most restrictive topology, the aforesaid values may have to be applied to the other three topologies so as to compare them.

It is to be mentioned that leaf and spine does not have restrictions with neither the number of end switches nor the number of hosts per switch, whereas the hypercube topologies fix the number of end switches to a power of two, as 2^N is the number of nodes, but the number of hosts per switch is free.

Putting the focus on fat tree, it is to be remarked that the key values for $K = 4$, $K = 8$ and *generic* K have already been shown back in Table 4.1. Nevertheless, Table 4.3 exposes just the most relevant values stated above, along with the overall amounts of items for all those K values cited.

Table 4.3: Relevant number of items for the scenarios proposed (fat tree).

Item description	**K-ary**	**K = 4**	**K = 8**
Number of hosts per switch	$K/2$	2	4
Number of end switches	$K^2/2$	8	32
Number of hosts overall	$K^3/4$	16	128
Number of switches overall	$5K^2/4$	20	80
Number of links overall	$3K^3/4$	48	384

With regards to leaf and spine, it may be defined a parameter called $L = log_2(K)$ in order to build up a leaf and spine topology being related to parameter K of fat tree so as to achieve the same values for the number of overall hosts and number of end switches. In that sense, the number of leaf switches may be sufficient to interconnect all hosts, evenly distributed along all switches, and the number of spine switches may be enough to interconnect to all leaf switches. Table 4.4 presents the relevant number of items for the scenarios proposed.

Table 4.4: Relevant number of items for the scenarios proposed (leaf and spine).

	K-ary	**K = 4**	**K = 8**
Equivalent L value respect to K	**L**	**L = 2**	**L = 3**
Number of ports per leaf switch	2^L	4	8
Number of ports per spine switch	2^{2L-1}	8	32
Number of hosts per switch	2^{L-1}	2	4
Number of hosts overall	2^{3L-2}	16	128
Number of leaf switches (end switches)	2^{2L-1}	8	32
Number of spine switches	2^{2L-3}	2	8
Number of switches overall	$5 \times 2^{2L-3}$	10	40
Number of links overall	2^{3L-1}	32	192

Referring to plain N-hypercube, parameter N is the one organizing the infrastructure, thus it may be used in order to get the same values for number of end switches and number of hosts per end switch obtained for fat tree. Table 4.5 exhibits the relevant number of items for the scenarios proposed.

Table 4.5: Relevant number of items for the scenarios proposed (N-hypercube).

	K-ary	**K = 4**	**K = 8**
Equivalent N value respect to K	**N**	**N = 3**	**N = 5**
Number of links to other switches	N	3	5
Number of ports per switch	$2N - 1$	5	9
Number of hosts per switch	$N - 1$	2	4
Number of hosts overall	$(N - 1) \times 2^N$	16	128
Number of switches overall	2^N	8	32
Number of links overall	$N \times 2^{N-1}$	12	80

With respect to folded N-hypercube, parameter N is also the organizer of the infrastructure, hence it happens the same as in N-hypercube because they both share the number of nodes, in spite of having more links among them. Table 4.6 exposes the relevant number of items for the scenarios proposed.

Table 4.6: Relevant number of items for the scenarios proposed (folded N-Hyp).

	K-ary	$\mathbf{K = 4}$	$\mathbf{K = 8}$
Equivalent N value respect to K	**N**	$\mathbf{N = 3}$	$\mathbf{N = 5}$
Number of links to other switches	$N + 1$	4	6
Number of ports per switch	$2N$	6	10
Number of hosts per switch	$N - 1$	2	4
Number of hosts overall	$(N - 1) \times 2^N$	16	128
Number of switches overall	2^N	8	32
Number of links overall	$(N + 1) \times 2^{N-1}$	16	96

At this point, it is important to distinguish two different distances, such as between end switches and between hosts, where the former is given by the number of links between two switches and the latter is achieved by adding up 2 extra units to the distance between their corresponding end switches, being the switches where those pair of hosts are hanging on, as an extra link is needed to interconnect each of both hosts to their proper end switches.

Otherwise, it is to be remarked that in tree-like designs, 1-hop away implies 1 layer of switches between any two hosts, meaning that the path from source host to destination host goes through just 1 switch in between those two hosts. Therefore, there are 2 links between hosts being 1-hop away, one of them going from source host to a port of that switch and another one going from destination host to another port of such a switch.

However, 2-hops away implies 2 layers of switches, meaning that the path is composed by 3 switches, such as the lower layer switch where the source host is hanging on, the lower layer switch where the destination host is connected, and a middle layer switch interconnecting both lower layer switches. Therefore, there are 4 links between hosts being 2-hops away.

Likewise, 3-hops away implies 3 layers of switches, meaning that the path is composed by 5 switches, following the same reasoning. This is, a lower layer switch for source host and another lower layer switch for destination host, then, a middle layer switch connecting to the source lower layer switch and another middle layer switch connecting to the destination lower layer switch, and eventually, an upper layer switch connecting both middle layer switches. Therefore, there are 6 links between hosts being 3-hops away.

On the other hand, graph-like design paths are composed by all switches between source switch and destination switch. This way, it may be said that there is a set of n hops-away paths containing all available combinations of n switches forming a string of length n, being located in a sequential order and taken through the shortest routes, starting from the next hop off the source switch (which might differ in each string) and finishing with the destination switch (which obviously must end each string).

Furthermore, in order to calculate the number of links between any pair of hosts, the amount of hops between their connecting switches may be incremented by 2, so as to consider the links going from the source switch to the source host and from the destination switch to the destination host.

Therefore, putting together all of the above and considering the proposed topologies on tables from 4.3, 4.4, 4.5 and 4.6, it is possible to undertake some comparisons among them all for the same number of end switches and hosts.

4.3.1 Average distance in links among hosts

The results obtained regarding the average distance are counted in number of links (λ) among any given pair of hosts (η), which may hang on any particular pair of end switches (σ). Those results are given for all four topologies regarding a generic parameter, and the equivalent values for $K = 4$ and $K = 8$ in each of those topologies.

Focusing on *fat tree*, a given host is obviously at a distance zero of itself, whereas it is to be reminded that there are ${}^{K}\!/_{2}$ hosts per edge switch, as well as ${}^{K^2}\!/_{4}$ hosts per pod, whilst there are ${}^{K^3}\!/_{4}$ all over the topology (4.4).

$$\frac{\cancel{1\,\eta \times 0\,\lambda} + ({}^{K}\!/_{2} - 1)\,\eta \times 2\,\lambda + ({}^{K^2}\!/_{4} - {}^{K}\!/_{2})\,\eta \times 4\,\lambda + ({}^{K^3}\!/_{4} - {}^{K^2}\!/_{4})\,\eta \times 6\,\lambda}{({}^{K^3}\!/_{4})\,\eta} \tag{4.4}$$

Focusing on *leaf and spine*, a particular host is clearly at a distance zero of itself, whereas it is to be remarked that there are 2^{L-1} hosts per leaf switch, whilst there are 2^{3L-2} all over the topology (4.5).

$$\frac{\cancel{1\,\eta \times 0\,\lambda} + (2^{L-1} - 1)\,\eta \times 2\,\lambda + (2^{3L-2} - 2^{L-1})\,\eta \times 4\,\lambda}{2^{3L-2}\,\eta} \tag{4.5}$$

Focusing on N-*hypercube*, a single host is just at a distance zero of itself, whilst it is to be pointed out that there are $N-1$ hosts per switch, with an overall amount of 2^N switches, taking into account that there are as many switches at a certain distance as stated by the coefficients of the Pascal's triangle for row N (4.6).

$$\frac{\left(\cancel{1\,\eta \times 0\,\lambda} + (N-2)\,\eta \times 2\,\lambda\right) \times 1\,\sigma + (N-1)\,\eta \times \left(\sum_{i=1}^{N} \binom{N}{i}\,\sigma \times (i+2)\,\lambda\right)}{(N-1)\,\eta \times 2^N\,\sigma} \tag{4.6}$$

Focusing on *folded* N-*hypercube*, a given host is obviously at a distance zero of itself, whereas it is to be said that there are $N-1$ hosts per switch, with 2^N overall, and taking into consideration that there are as many switches at a certain distance as stated by the coefficients of the folded Pascal's triangle for row N (4.7).

$$\frac{\left(\cancel{1\,\eta{\times}0\,\lambda}{+}(N{-}2)\,\eta{\times}2\,\lambda\right){\times}1\,\sigma{+}(N{-}1)\,\eta{\times}\left(\sum_{i=1}^{\lceil N/2\rceil}\left(\binom{N}{i}{+}\binom{N}{N-i+1}\right){\times}2^{-\left\lfloor\frac{i}{N-i+1}\right\rfloor}\,\sigma{\times}(i{+}2)\,\lambda\right)}{(N{-}1)\eta{\times}2^N\,\sigma} \tag{4.7}$$

4.3.2 Average distance in links among end switches

At this point, let us remake all calculations again by moving the focus from hosts to end switches in order to calculate the average number of links between any pair of given end switches.

Focusing on *fat tree*, a given edge switch is obviously at a distance zero of itself, whilst it is to be noted that there are $K/2$ edge switches per pod, as well as a total amount of $K^2/2$ throughout the whole topology (4.8).

$$\frac{\cancel{1\,\eta \times 0\,\lambda} + (K/2 - 1)\,\sigma \times 2\,\lambda + (K^2/2 - K/2)\,\sigma \times 4\,\lambda}{(K^2/2)\,\sigma} \tag{4.8}$$

Focusing on *leaf and spine*, a particular leaf switch is clearly at a distance zero of itself, whereas it is to be reminded that there are 2^{2L-1} leaf switches being located all over this topology (4.9).

$$\frac{\cancel{1\,\sigma \times 0\,\lambda +}(2^{2L-1} - 1)\,\sigma \times 2\,\lambda}{2^{2L-1}\,\sigma} \tag{4.9}$$

Focusing on N-*hypercube*, a single end switch is just at a distance zero of itself, whilst there are as many switches at a certain distance as given by the coefficients of the Pascal's triangle for row N, where the overall amount of switches is 2^N (4.10).

$$\frac{\sum_{i=0}^{N} \binom{N}{i} \, \sigma \times i \, \lambda}{2^N \, \sigma} \tag{4.10}$$

Focusing on *folded* N-*hypercube*, any end switch is obviously at a distance zero of itself, whereas the number of switches at some distance is given by the coefficients of the folded Pascal's triangle for row N, where 2^N accounts for all switches (4.11).

$$\frac{\left(\sum_{i=0}^{\lceil N/2 \rceil} \left(\binom{N}{i} + \binom{N}{N-i+1} \right) \times 2^{-\left\lfloor \frac{i}{N-i+1} \right\rfloor} \sigma \times i \, \lambda \right)}{2^N \, \sigma} \tag{4.11}$$

4.3.3 Results and analysis

To sum of it all up, the aforementioned expressions have been used to calculate the four topologies with the values quoted in tables 4.3, 4.4, 4.5 and 4.6. In fact, Table 4.7 presents a summary of the results obtained for the average distances in links (λ) among hosts, whereas Table 4.8 does it for the average distances in links (λ) among end switches. This way, the number of hosts and end switches may be seen as equivalent to those attained with a fat tree topology for $K = 4$ and $K = 8$.

Table 4.7: Average distance in links (λ) among hosts.

	Fat tree	**Leaf & spine**	**N-hypercube**	**Folded N-hyp.**
$\mathbf{K = 4}$	5.13 $(K = 4)$	3.62 $(L = 2)$	3.38 $(N = 3)$	3.13 $(N = 3)$
$\mathbf{K = 8}$	5.67 $(K = 8)$	3.92 $(L = 3)$	4.48 $(N = 5)$	4.05 $(N = 5)$

According to the data obtained in this couple of tables, it might seem that graph-like designs perform better than tree-like designs. Likewise, within the former, folded N-hypercube does it better than plain N-hypercube, whereas within the latter, leaf and spine does it better than fat tree.

Table 4.8: Average distance in links (λ) among end switches.

	Fat tree	**Leaf & spine**	**N-hypercube**	**Folded N-hyp.**
K = 4	3.25 $(K = 4)$	1.75 $(L = 2)$	1.50 $(N = 3)$	1.25 $(N = 3)$
K = 8	3.69 $(K = 8)$	1.94 $(L = 3)$	2.50 $(N = 5)$	1.59 $(N = 5)$

It is to be noted that the scenarios proposed were set up in order to export the number of hosts of a fat tree architecture to the other ones, as that is the design imposing fixed specifications according to parameter K, and that way, the performance for all topologies might be compared for the same number of hosts.

However, it is to be said that the number of hosts per end switch usually depends on the deployment in all topologies, except for fat tree. Besides, such a number might not be constant throughout all end switches. Regardless the case, any variation with the amount of hosts may change the values shown in the aforementioned tables.

On the other hand, this factor is usually not the most important one, as some other factors may well be taken into account. For example, the values of latency and jitter must be steady when dealing with VoIP and video streaming, or otherwise, the easiness of management requires an internetworking system as simple as possible, or even scalability might be an important issue when dealing with growing workloads.

Therefore, when dealing with convergent networks, meaning those carrying data traffic along with voice and video all together, it may be more interesting to implement tree-like designs, as the number of hops away is steady, not being greater than 3 in case of fat tree, or even 2 in case of leaf and spine, whereas in graph-like designs, the number of hops away grows as it does the dimension of the topology.

Furthermore, regarding easiness of management for graph-like designs, N-hypercube might be more interesting than folded N-hypercube, as extra links makes things harder regarding routing, packet forwarding or managing the whole infrastructure. As per the tree-like designs, it depends on the situation, as leaf and spine contains less switches and links, but requires full mesh connectivity and fat tree do not.

Eventually, with respect to scalability, graph-like designs get more complicated as the number of switches grows. Otherwise, tree-like designs scale better, but fat tree does it finer than leaf and spine, as the former has partial mesh interconnection, thus maintaining the number of switches and links steadier as the network grows, whereas the latter needs full mesh interconnection, making the number of links increase faster as network needs grow. On the other hand, it needs to be noted that the latter displays better performances up to a certain network size, thus being a highly interesting option in small to medium deployments.

In summary, none of the four topologies may be considered as the best of all, because there are many factors affecting performance, which may well be a key player, but in many occasions there may be others as important, such as the need of steady values for latency and jitter, the easiness of management, and the possible scalability issues. Hence, each implementation may have its better choice regarding the interconnection topology depending on the required conditions.

4.4 Other mesh and star topologies

The topologies presented above do not have the shortest distances between end switches, thus getting worse performance measurements when dealing with real time streams. In order to cope with this, full mesh and quasi full mesh topologies come into play, with the advantage of having much direct interconnections among switches, but at the expense of having manageability and scalability issues, hence being a great option for small deployments, although not so good as the number of nodes grow.

4.4.1 Full mesh

The particularity of this topology is that each node within this shape has a direct interconnection with the rest of its counterparts, thus making the distance between any pair of switches equal to 1 link, whilst the distance between any pair of hosts hanging on different nodes is just 3 links. Obviously, the distance between any two hosts tied to the same switch is 2 links, regardless the switching topology involved.

This sort of topology may be built up with different designs for N switches, such as an N-*simplex*, where all nodes are directly interconnected, also a folded N-orthoplex, where the missing link between opposite nodes is established, or even a convex regular n-polygon (having n sides located in $2D$), where all diagonals are depicted.

The main benefit of this topology is clearly that all nodes are at the same distance, thus improving the performance, whereas the main drawback is the scalability. Anyway, Figure 4.9 exhibits a full mesh topology with six nodes.

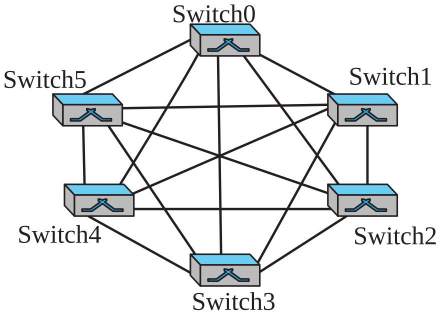


Figure 4.9: Full mesh topology.

4.4.2 Quasi full mesh

This topology is based on the N-*orthoplex* shape, where there is a direct link between one node and the rest of them, except for its opposite node. This way, the distance between any pair of switches is 1 link, except for opposite nodes, which rises to 2 links. Therefore, when dealing with the number of links between any pair of hosts, there may be obviously 2 if they both hang on the same switch, rising up to 3 if they both hang on different switches not being opposite, and reach up to 4 if that condition arises.

The advantage of this design respect to the previous one is the saving of some links, which accounts for some improvement in scalability, whilst still having all nodes at a close distance, hence this design might be seen as a commitment between manageability and proximity of nodes. Anyway, Figure 4.10 exposes a quasi full mesh topology with six nodes.

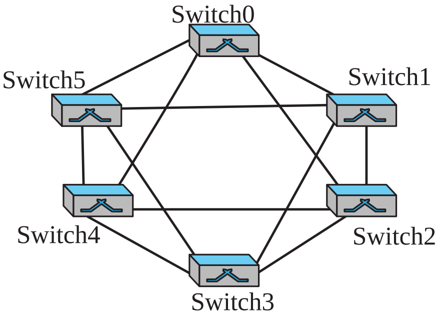


Figure 4.10: Quasi full mesh topology.

4.4.3 Star

This type of topology, also known as hub & spoke, may be seen on the borderline between tree-like and graph-like designs. On the one hand, it may be considered as a tree due to its hierarchical infrastructure, this is, the hub has a different role than the spokes, as the latter get hosts connected and the former does not, just serving as the interconnection among all the latter. In this sense, it might be viewed as if the spokes had been extended all around the hub, covering the whole 360 degrees, where this is located in the centre of the infrastructure.

On the other hand, it may be seen as a graph due to its geometric shape, which might be seen as a wheel, where the hub is right at the centre and the spokes are equally distributed just over the circumference, with the links just playing the part of radius within the whole circle. Likewise, a hub and spoke topology may also be viewed as a regular n-polygon, where n is related to the number of spokes available, and with a single hub just in the middle of such a shape. This view may provide a set of undirected connected cyclic graphs with some symmetry axes for all spokes, accounting for multiple redundant paths between spokes, which are the ones connected to hosts, taking into account that the hub is only connected to spokes.

Anyway, traditional star topologies, typically known as hub and spoke, usually have one single device acting as a hub, and with as many spokes as necessary. In this sense, if there are n spokes, it might also be associated with an n regular polygon wheel. Figure 4.11 depicts a typical hub and spoke design with six spokes, where all communications among them go through the central hub. It may be appreciated that all spokes, which are the end switches, are at a distance of 2 links of each other, whereas any pair of hosts hanging on different switches are at a distance of 4 links.

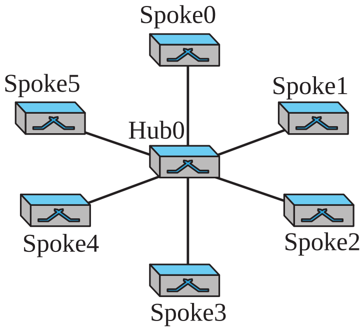


Figure 4.11: Hub and Spoke topology.

Regarding DC designs, where redundancy is a key factor, it is strongly recommended to have a couple of hubs in order to get redundancy. This way, fault-tolerant designs are achieved, whilst not needing a routing protocol if the network operates at layer 2, or otherwise, using simple static routing if the network operates at layer 3.

Alternatively, it might be considered the option of having a physical device connected to all spokes, having two virtual devices acting as redundant hubs. In that case, there is no physical redundancy, although there is service redundancy, as well as link redundancy if aggregate links are implemented and power redundancy if power lines coming from different sources are deployed. On the other hand, the option of using a single virtual device with a snapshot ready to load in case the hub gets corrupted might be considered as a single hub with a fast recovery time ready to go.

However, despite of the fact that both alternative options described might do the job of a redundant hub, it may be noted that the option with two redundant physical devices acting as a hub brings the best performance, as it may offer better full redundancy, and hence, that is the most safe and secure option.

The topology with two redundant hubs is a typical solution being put in place in many production environments, as stated before, where there might be even more than two in certain cases. There are different network protocols to deal with such a situation, which permits two different approaches, such as active-standby and active-active. The former allows for one device always dealing with data traffic, whereas the other device is just ready to go if the other one fails. Otherwise, the latter permits both devices to deal with certain data traffic simultaneously, where diverse virtual contexts may be assigned for each of them to deal with different types of traffic.

On the other hand, the former does not usually involve any direct link between both hubs, as control communication between them is forwarded through the spokes, and the lack of such messages on the other device implies that the other hub is down, thus the role of active is assigned to the hub being up, hence needing some small time interval to build up the forwarding table before being fully operational.

Regarding the latter, there is usually a direct link between both hubs, where control communication between them takes place. This extra control link is used for control messages, as well as to make backup copies on a regular basis, thus making possible a faster recovery time in case of one hub going down. Figure 4.12 shows such a design.

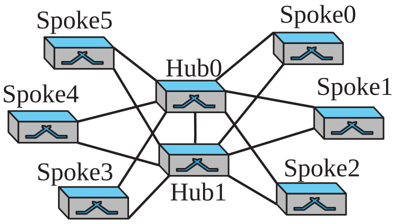


Figure 4.12: Redundant Hub and Spoke with a direct communication link.

4.5 Toroidal grid topologies

Those kinds of topologies may be embedded on an N-dimensional torus without any crosses, meaning that the items along each particular dimension are interconnected in a circular fashion. Only one and two dimensions are going to be considered, resulting in single rings and toroidal rings, respectively, leaving aside higher dimensions, as 3D hypertoroidal rings are aimed at large-sized networks, those being far too much for the needs of small-to-medium network deployments.

4.5.1 Redundant ring

A single ring may be seen as a toroidal grid design along just 1 dimension, where each node has only 2 neighbours, those being its predecessor and its successor. In other words, a single ring may be seen as a toroidal grid composed by a number of nodes n located in only 1 circular row, whereas a peer-to-peer topology may be seen as a single ring with only two nodes.

It is to be said that the easiest way to arrange the nodes may seem as a sequence of natural numbers, this is, going from left to right for simplicity purposes, starting out at zero and moving sequentially. Other node distributions may be applied, such as a de Bruijn sequence, even though the topology remains the same in any case. Hence, the former case has been considered.

This design is going to be presented with two redundant links in each direction, as it is sometimes used in production environments for some critical networks with double links between the nodes for redundancy purposes, where each node has a double link pointing to both left and right neighbours, that being the reason why the topology proposed is called redundant ring.

4.5.2 Toroidal ring

If the previous design is extended up to 2 dimensions, the result is a toroidal ring, where the nodes are organized in circular rows and circular columns, respectively. The most common arrangement has the same number of rows and columns, that being n, resulting in a squared mesh where the items on the edges are wrapped around. Hence, the dimensions of such a squared mesh are $n \times n$, where each node has just 4 neighbours, those being predecessor and successor in both its row and its column.

It is to be noted that the more straightforward way to arrange the nodes may appear to be as an incremental sequence of natural numbers for each row, from left to right, and then, from bottom to top, for simplicity purposes. However, it is to be reminded that other node distributions might be applied, such as a de Bruijn torus, although the topology stays the same in all cases, where each node have four single links pointing to its directly connected neighbours, such as up, down, left and right.

As a side note, it is to be said that, when there are just 2 nodes per dimension in a toroidal network, the resulting design is that of an hypercube. Specifically, if the toroidal grid has 1 dimension, the design obtained is that of a segment, whereas if it has 2 dimensions, a square design is attained, whilst if it has 3 dimensions, a cube design is achieved, and so on.

4.6 Directed-graph topologies

In the context of directed graphs, two sorts of topologies are going to be built up following the de Bruijn graphs presented before, such as that based on its original version and a reverse one. Anyway, it is to be remarked that they are both based on directed graphs, meaning that all edges have a unique compulsory direction.

4.6.1 De Bruijn graph

A design based on a de Bruijn graph of binary n-length words may be considered as a deployment where the bits identifying the destination node are injected as a left bit shifting, one at a time [233]. In this case, the destination is reached with a maximum of n hops. The way to go through it has been described in the previous section, which may be further clarified in Figure 4.13 for a Hamiltonian path on $B(2, 3)$, and also in Figure 4.14 for a Eulerian path on $B(2, 4)$.

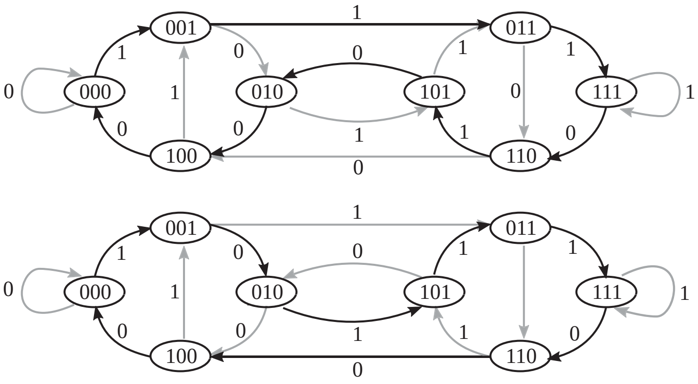


Figure 4.13: Hamiltonian paths for $B(2, 3)$.

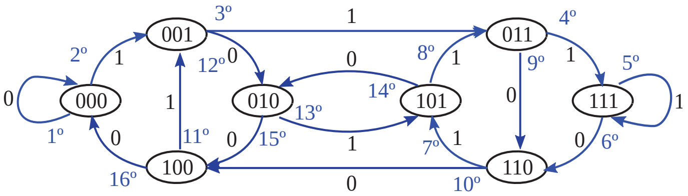


Figure 4.14: Eulerian path for $B(2,4)$.

By looking at the aforesaid binary de Bruijn graph, it may be clear that each node only has 2 possible paths inwards and other 2 possible paths outwards, due to the core feature as a directed graph with a binary alphabet. Therefore, the way to traverse the graph is to get into a node through any of the permitted entry paths, while getting off through any of the exit paths, as depicted in Figure 4.15.

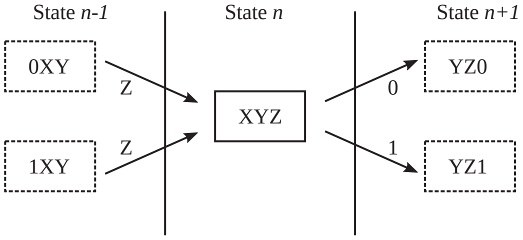


Figure 4.15: Paths in and out of a node in a binary de Bruijn graph.

The aforementioned example $B(2,3)$ is quite simple, as $k = 2$ and $n = 3$, but the complexity increases as both values may rise, even though the principle of use remains the same. For example, a ternary alphabet may be established with a word length of 2, and then, the $B(3,2)$ graph may be obtained, which is exhibited in Figure 4.16, having a trefoil shape. This example may shows that the rise in k leads to much more states, which may further complicate the graph, as both the number of nodes increases and each node may have k entry paths and other k exit paths.

Regarding the expression to get to the next nodes in a de Bruijn graph, departing from a particular node i, when using a binary alphabet ($k = 2$) with a word length n, is given by (4.12).

$$(2i + j) \bmod 2^n, \text{ where } j = \{0, 1\} \tag{4.12}$$

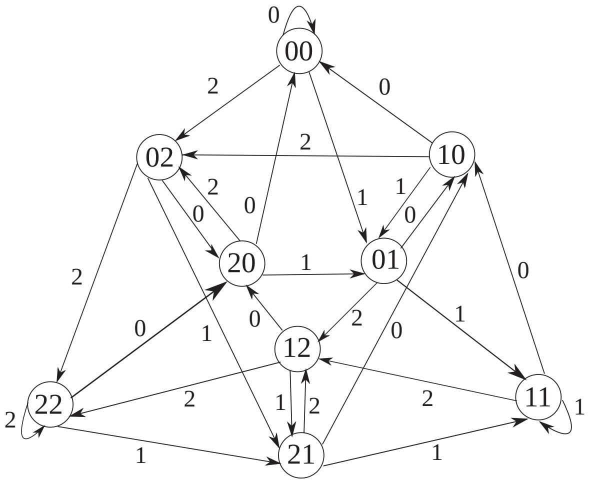


Figure 4.16: De Bruijn graph B(3,2).

On the other hand, it is pretty straightforward to adapt the aforesaid expression for a k-ary alphabet, as it only implies to exchange the appearances of number 2 in that expression with k, which may stand for the expression given in (4.13).

$$(k \times i + j) \mod k^n, \text{ where } j = \{0, 1, \cdots, k-1\} \tag{4.13}$$

4.6.2 De Bruijn reverse graph

The way it works the aforesaid de Bruijn design may be flipped around so as to achieve that the bits identifying the destination node are injected as a right bit shifting, one at a time. The result would be the de Bruijn reverse graph, shown in Figure 4.17, which is obtained in a similar way as explained for the de Bruijn graph, where a 3-length Hamiltonian cycle may be found straightforward by going from left to right and coming back home, and a 4-length Eulerian cycle may be found by setting the edge weight found when getting off a node in front of that node identifier, and in turn, making the same path as described in the previous section.

In order not to get confused with the original de Bruijn graph $B(k,n)$, the reverse de Bruijn graph is going to be named $\overline{B}(k,n)$, where the underscored values in the corresponding picture are located in a different node from where they are in the original one, whilst the rest of nodes and all edge weights remain at the same place.

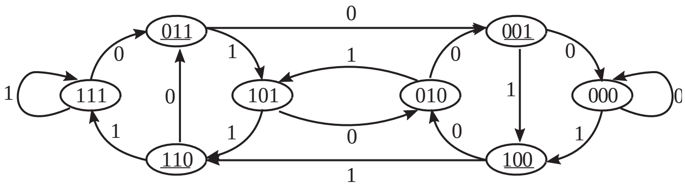


Figure 4.17: De Bruijn reverse graph: Hamiltonian $\overline{B}(2,3)$ and Eulerian $\overline{B}(2,4)$.

As stated in the aforementioned case, the complexity rises as n or k grow. In this context, the expression to get from node i to the next nodes one-hop away when using a binary alphabet is given by (4.14).

$$(int(^{i}/_{2}) + j \times 2^{n-1}) \bmod 2^n, \text{ where } j = \{0,1\} \tag{4.14}$$

On the other hand, that expression may be adapted for using a k-ary alphabet by just swapping all appearances of number 2 with k, which may result in (4.15).

$$(int(^{i}/_{k}) + j \times k^{n-1}) \bmod k^n, \text{ where } j = \{0,1,\cdots,k-1\} \tag{4.15}$$

4.7 Undirected-graph topologies

In the context of undirected graphs, a couple of topologies are going to be constructed where the edges between nodes are only present if their n-length binary identifiers meet some specific requirements. On the one hand, a binary grid interconnects a node with the other nodes having just one mismatching bit between their node identifiers. On the other hand, this behaviour may be reversed, in a way that a link may take place when two nodes identifiers differ in just $n-1$ bits, and also joining each pair of opposite nodes, which makes for a specific type of Hamming graph.

In both cases, it is to be remarked that for n even, a binary square $n \times n$ toroidal grid is achieved, whilst if n is odd, just a binary flat representation is attained, where the all 0s and 1s values are left alone at opposite ends, and the proper groups of n elements are allocated in between, with their appropriate connections according to the established conditions.

Moreover, just for clarification purposes, bit zero is the least significant bit, whereas bit $n-1$ is the most significant one, thus the bit positions grow from right to left.

Additionally, a couple of topologies are going to be proposed related to cage graphs, whose main features are having a specific regular degree and girth, being the designs with the minimum amount of vertices to achieve such a girth. It is to be noted that in those cases, nodes are named in a clockwise manner.

4.7.1 Binary grid

This topology may be seen as a deployment of k^n nodes, where each one is identified by means of a unique k-ary n-length word, where k is a binary alphabet and n is a small natural number. When n happens to be even, it might be viewed as if those nodes are interconnected as a grid in a toroidal array, having the shape of a k-ary n-cube, even though it is not to be confused with neither a de Bruijn torus, because there are no unique toroidal subarrays involved, nor a de Bruijn graph, as that is a directed graph whilst this is an undirected one. Otherwise, when n occurs to be odd, it might be observed a grid as a flat array, although two extra elements are left alone located on opposite sides, those containing all equal bits, one with all zeros and another one with all ones. In any case, the degree of each node is n, as there is just a different flipping bit for each neighbour node.

Therefore, an edge between a pair of nodes involves only one binary symbol shifting to get from one to the other. Figure 4.18 exhibits the relationships among all nodes in a binary grid layout for $n = 2$, $n = 3$ and $n = 4$.

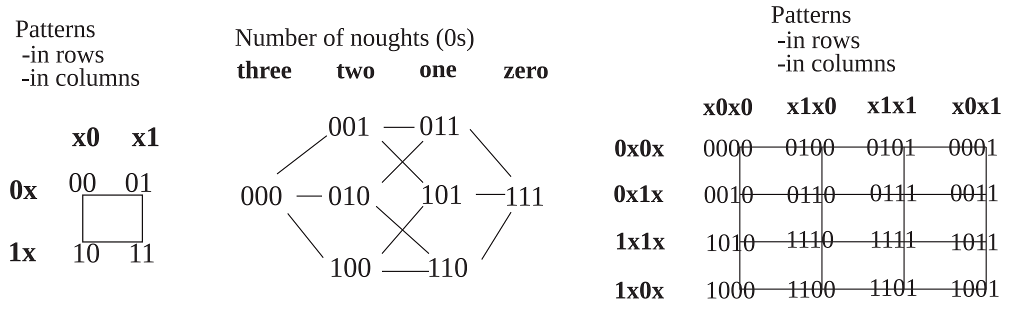


Figure 4.18: Binary grid for $n = 2$, $n = 3$, $n = 4$.

Regarding the case when n is even, it is to be observed that all rows and columns shown follow different definite patterns containing 0s and 1s, along with undetermined values expressed by x (whose value might be either 0 or 1). Therefore, this regularity may make easy to work out an algorithm to move from a source to a destination by just shifting the discordant bits. Furthermore, by crossing a horizontal and a vertical patterns, any given number may be easily spotted, as well as the discordant bits between a source node and a destination node. On the other hand, some regularity related to the number of bits being 0 may also be established in case n is odd.

Hence, departing from a node i, the path to the destination node may be calculated by first spotting the discordant bits j between them, meaning the different values located in the same bit positions of both neighbour nodes. This may be done with the help of logical operations, by applying the *XOR* operator to the binary identifiers of both nodes, or otherwise, by applying an appropriate arithmetic expression to their decimal identifiers.

Once those discordant bits have been found out, a logical approach may be used by swapping the discordant bits j on the source node, or otherwise, an arithmetic approach may be imposed by applying the expression given in (4.16). Anyway, each discordant bit accounts for a movement to an intermediate node on the way towards the destination node. Furthermore, if the count of j discordant bits is greater than one, then every possible combinational strings using all elements of j make up for as many redundant paths as the factorial of such a number of discordant bits, where each path contains all of the movements, but ordered in a unique manner, making up for as many different paths from source node to destination node.

$$i + \sum_{j} \left((-1)^{int(i/2^j) \bmod 2} \times 2^j \right), \text{ where } j = \{discordant\ bits\} \tag{4.16}$$

Additionally, the grid may be extended to any k-alphabet, which may enlarge the links for each single node, and in that case, the expression may be adapted by exchanging k for the number 2 in the appropriate algorithm, and considering that j now stands for the discordant k-ary digits. Hence, the appropriate expression may be obtained by (4.17).

$$i + \sum_{j} \left((-1)^{int(i/k^j) \bmod k} \times k^j \right), \text{ where } j = \{discordant\ k - ary\ digits\} \tag{4.17}$$

4.7.2 Hamming graph

If the concept of binary grid is flipped around, then a kind of reverse binary grid may appear to be the topology, where there is an edge between any pair of nodes whose distance is exactly $n - 1$, in a way that all bits minus one are just the binary shifting of each other. In this context, the distance d is measured as the number of different binary digits identifying a pair of nodes, obviously comparing the bits in the same positions.

That case may also be viewed as a subgroup within the *Hamming graphs*, those being *undirected* graphs where the distance between any two vertices is at least a certain value d [57], even though other definitions may be found in the literature, such as the distance being just one [102], which would make for the binary grid case describe above. However, the former definition may be applied herein.

Hence, in that context, a Hamming graph may have a nomenclature such as $H_k(n, d)$, where k is the length of the alphabet, n is the length of the words and d is the minimum distance between nodes. Therefore, a Hamming graph for distance $n - 1$, such as $H_k(n, n - 1)$, may connect all nodes having a distance $n - 1$ among them, as well as those at a distance n, which may account for opposite nodes. Those graphs are also known as Hamming-distance graphs and they are all Hamiltonian, regular, simple and connected graphs [88], making them quite feasible to work with.

Figure 4.19 depicts the relationships among nodes in this particular type of Hamming graph layout using a binary alphabet, such as $k = 2$, resulting in $H_2(n, n - 1)$ for $n = 3$ and $n = 4$, as the picture for $n = 2$ remains the same as the one exhibited for the binary grid. In any case, the degree of each node is $n + 1$, making up for the n neighbours at a distance $n - 1$ and the extra neighbour at a distance n, meaning the opposite node. On the other hand, Hamming graphs need not to be confused with Hamming codes, which are being used for detection and correction of errors in data transmissions [86], even though they both share the concept of distance.

Those figures are similar to the ones obtained in the binary grid case for small natural numbers n, but the layout of the nodes and the interconnections among them are not the same. Regarding the case where n is odd, the number of overall zeros within

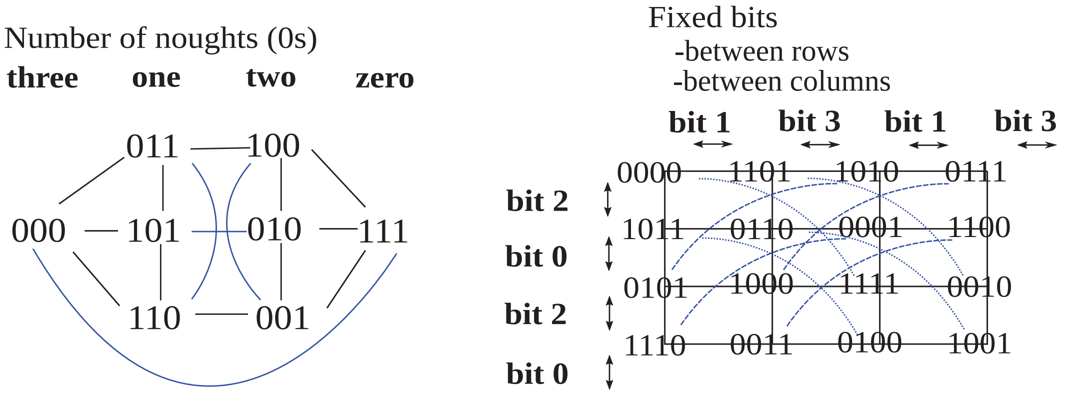


Figure 4.19: Hamming graph $H_2(n, n-1)$ for $n = 3$ and $n = 4$.

each block does not follow the same sequence described above. Otherwise, when n is even, a specific bit is fixed between every pair of neighbour rows or columns, being bits 1 or 3 for the former and bits 2 or 0 for the latter, where different combinations of bit positions for the fixed value may be taken between a source node and a destination node, whilst the rest of bits are shifted. Additionally, it is to be reminded that a direct link exists between opposite nodes, no matter the parity of n.

Therefore, departing from a node i, the path through the destination node may be found out in two stages. As per the first round, the number of discordant bits j are counted up so as to check whether it is equal or greater than $n-1$. If that is the case, if it is equal to $n-1$, there may only be a *matching bit* (regardless of its value), which may be referred to as $j*$, being located in the position $j*$ inside the binary string, whilst there may be no matching bits if it is equal to n.

According to that result, the second round takes places, such that if the aforesaid condition is met, then the destination node is a direct neighbour, or otherwise. Hence, in the first case, the expression to get the unique path to the destination node would be given by expression (4.18), which accounts for all discordant bits.

$$i + \sum_{\substack{j=0 \\ j \neq j*}}^{n-1} (-1)^{int(i/2^j) \bmod 2} \times 2^j \tag{4.18}$$

Otherwise, if there are less than $n-1$ discordant bits, then there may be some redundant paths, in a way that each path may go through an intermediate neighbour by fixing one of the discordant bits, where each of those paths may be calculated by using a similar expression, but using $j**$ to express *discordant bits*, and dealing with them all, such as in (4.19). Besides, if $max(j**) = {}^{n}\!/_{2}$, then the link to the opposite node may also be taken into account, as being an equal distance path.

$$i + \sum_{j**}^{max(j**)} \Big(\sum_{\substack{j=0 \\ j \neq j**}}^{n-1} (-1)^{int(i/2^j) \bmod 2} \times 2^j \Big) \tag{4.19}$$

On the other hand, if there is just one *discordant bit*, then the path may go through the opposite node, where it may be calculated by means of (4.20), and then the destination will be just a direct neighbour of the opposite node. Alternatively, an opposite node may be found out by applying the expression $2^n - 1 - i$.

$$i + \sum_{j=0}^{n-1} (-1)^{int(i/2^j) \bmod 2} \times 2^j \tag{4.20}$$

All those steps involve arithmetic operations, although they might also be performed by means of logical operations, this is, by applying the *XOR* operator to find out the discordant and matching bits between the binary identifiers of the source node and the destination node, where the former yield 1 and the latter do 0 after the *XOR* operation, and in turn, by swapping the discordant bits j accordingly.

Additionally, the Hamming graphs may also work with any k-alphabet, which may allow for more links for each single node, and therefore, the expression may be adapted by substituting k for the number 2 in the aforementioned algorithms, and taking into account that j, $j*$ and $j**$ have the same meaning as before, but the term *bit* may be changed by k-ary digit or symbol.

Therefore, the appropriate expression for reaching a direct neighbour is given by (4.21), the one to get to an intermediate neighbour is shown in (4.22) and the one needed to achieve the opposite neighbour is provided in (4.23).

$$i + \sum_{\substack{j=0 \\ j \neq j*}}^{n-1} (-1)^{int(i/k^j) \bmod k} \times k^j \tag{4.21}$$

$$i + \sum_{j**}^{max(j**)} \Big(\sum_{\substack{j=0 \\ j \neq j**}}^{n-1} (-1)^{int(i/k^j) \bmod k} \times k^j \Big) \tag{4.22}$$

$$i + \sum_{j=0}^{n-1} (-1)^{int(i/k^j) \bmod k} \times k^j \tag{4.23}$$

4.7.3 Cage graphs

A (k, g)-graph may be defined as a cage graph being k-regular, as each node has degree k, and girth g, meaning the length of its shortest cycle. Within that type of graphs, a (k, g)-cage is the one with the smallest possible number of nodes [59]. In this context, if $k = 2$, the only available cages are the regular polygons, and $g < 3$ does not form a cycle, thus $k \geq 2$ and $g \geq 3$.

It is to be said that not all combinations of k and g make for a cage, furthermore considering that a $(k, 3)$-cage makes for a complete graph, meaning a full mesh one, and $(k, 4)$-cage is a complete bipartite graph, where nodes may be divided into two groups where there is a full mesh connectivity between each node belonging to one set and all the nodes located in the other set.

Focusing on $k = 3$, there have been found $(3, g)$-cages from $5 \leq g \leq 12$, where Table 4.9 shows its main features. However, values from 5 to 8 may account for the most well-known, along with 12.

Regarding small-to-medium network deployments, the most interesting ones may be the $(3, 5)$-cage and the $(3, 6)$-cage, namely Petersen and Heawood graph, respectively, as they both assure a maximum internode distance, also known as *diameter*, of 2 and 3, respectively. Likewise, the former has a chromatic number of 2 and a chromatic index of 3, whereas the latter has 3 and 4, respectively. Neither of them offer equal length paths to a given destination, thus redundant paths may not be equally long, even though all links might be doubled as an alternative to achieve that.

Table 4.9: Known $(3, g)$-cages.

Name of the cage graph	Nomenclature	Girth	Nodes (Order)
Petersen	$(3, 5)$-cage	5	10
Heawood	$(3, 6)$-cage	6	14
McGee	$(3, 7)$-cage	7	24
Tutte-Coxeter	$(3, 8)$-cage	8	30
-	$(3, 9)$-cage	9	58
-	$(3, 10)$-cage	10	70
Balaban	$(3, 11)$-cage	11	112
Benson	$(3, 12)$-cage	12	126

4.7.3.1 Petersen graph

Focusing on $(3, 5)$-cage (Petersen graph), it may be interesting to identify the nodes belonging to the outer circle first (going from 0 to 4), and then, those of the inner circle (going from 5 to 9), where in both cases the lowest value is awarded to the topmost node, and from those points on, going in a clockwise fashion. This way, the forwarding strategy may be arranged just by distinguishing combinations between both groups. Figure 4.20 shows an implementation of a Petersen graph.

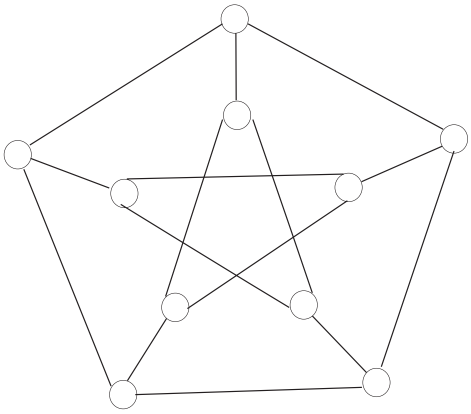


Figure 4.20: Petersen graph.

4.7.3.2 Heawood graph

With respect to the $(3, 6)$-cage (Heawood graph), it may be interesting to identify the nodes in a clock manner as well (going from 0 to 13), assigning node 0 to the topmost node with the middle link going clockwise. This way, all even nodes have their middle link clockwise, whereas all odd nodes have it counterclockwise, which may simplify the forwarding expressions. Figure 4.21 shows an implementation of a Heawood graph.

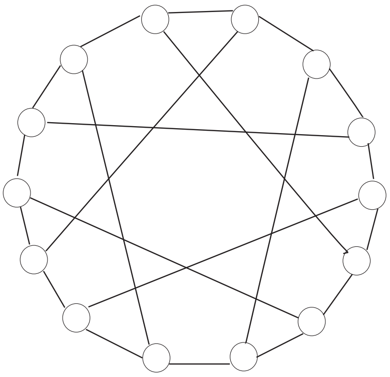


Figure 4.21: Heawood graph.

4.8 Summary

In summary, the most relevant features of the topologies proposed are going to be cited in Table 4.10. First of all, tree-like, graph-like or star-like designs are going to be distinguished, and in turn, the *girth* is quoted, meaning the shortest cycle, where ∞ stands for no cycles, as well as the *diameter*, meaning the largest distance between any pair of switches or any pair of hosts, and the *degree* in graphs, stating the number of interswitch links in all switches.

On the one hand, tree-like designs have been represented by fat tree, which accounts for three layers of switches, and leaf and spine, accounting for just two layers. Otherwise, star-like designs have been exposed as a single hub and spoke and a redundant hub and spoke. On the other hand, the rest of topologies shown are considered as graph-like designs, where all of them share some common characteristics among them, such as being regular, connected, simple and Hamiltonian.

After having described the main characteristics of all those topologies, their corresponding models are going to be designed in the following chapter.

Table 4.10: Topologies proposed along with some relevant features.

Topology name	**Type**	**Girth**	**Switch diameter**	**Host diameter**	**Degree (just to switches)**
Fat tree	tree-like	∞	4	4+2	end switches: $K/2$ other switches: K
Leaf & spine	tree-like	∞	2	2+2	up to the designer
N-hypercube	graph-like	4	N	$N+2$	N
Folded N-hypercube	graph-like	4	$\lceil N/2 \rceil$	$\lceil N/2 \rceil + 2$	$N+1$
Full mesh	graph-like	3	1	1+2	N
Quasi full mesh	graph-like	3	2	2+2	N
Hub & Spoke	star-like	∞	2	2+2	end switches: 2 hub: $2N$
Redundant Hub & Spoke	star-like	∞	2	2+2	end switches: 2 hub: $N+1$
Redundant ring	graph-like	N	$N-1$	$N-1+2$	4
Toroidal ring	graph-like	4	$2 \times \lfloor N/2 \rfloor$	$2 \times \lfloor N/2 \rfloor + 2$	4
De Bruijn graph	graph-like	2	2^N	$2^N + 2$	indegree: N outdegree: N
De Bruijn reverse graph	graph-like	2	2^N	$2^N + 2$	indegree: N outdegree: N
Binary grid	graph-like	4	$2 \times \lceil N/2 \rceil$	$2 \times \lceil N/2 \rceil + 2$	N
Hamming graph	graph-like	$N \leq 3 = 3$ $N \geq 4 = 4$	$\lceil N/2 \rceil$	$\lceil N/2 \rceil + 2$	$N+1$
Petersen graph	graph-like	5	2	2+2	3
Heawood graph	graph-like	6	3	2+2	3

Chapter 5

Modelling network Data Centre topologies with ACP

After having presented the topologies selected for small to medium DC implementations, along with their mathematical bases, those structures are going to be modelled with ACP. In this sense, a common structure is going to be followed in the study for each of the models proposed:

- *Preamble*, where a quick description is offered.
- *Layout*, where nodes and edges are identified for being used in the model.
- *Flow chart*, where a diagram depicts how the model behaves so as to find the paths with the minimal distance between a given pair of devices [219], where the metric considered is the number of links between them.
- *Pseudocode model*, where arithmetic and logical approaches carry the behaviour.
- *Algebraic model*, where ACP is used to describe the behaviour.
- *Redundant paths*, where an algorithm is proposed to list all devices (and ports, if at all possible) within each minimal path available.
- *Verification*, where an ACP verification takes place in order to verify the behaviour of the model proposed against that of the real system.

5.1 Hosts, switches and ports

First of all, it is to be reminded that an *end switch* is a switch having some hosts hanging on it, each of them holding a determined number of hosts. For simplicity purposes, such a number has been set to M in all cases presented herein, with the exception of fat tree, where there must be $K/2$ hosts per end switch. Taking that aside, most of the designs presented admit any value of M, except for the N-hypercube and folded N-hypercube designs herein, which in order to keep things simple, it is required a value of M to be a power of 2, such as $M = 2^W$, where W may be any exponent.

On the other hand, it is to be said that each end switch may have a unique identifier in the form of a natural number i, going from left to right as a general rule, thus starting from 0 and increasing rightwards. Likewise, the downlink ports for each end switch will also go from port 0 to port $M-1$, in a way that each host is connected to its proper end switch through the correspondant port, where a given port may be generically named as p.

This way, each host will be uniquely identified by variable h, according to the following expression (5.1), which depend on i, M and p.

$$h = i \times M + p \tag{5.1}$$

Therefore, the group of hosts connected to switch i will be identified within the range from $i \times M$ to $i \times (M + 1) - 1$, as it may be shown in Figure 5.1. Moreover, it has been considered that hosts only have one port, which is referred to as port 0. Furthermore, the overall number of switches within a certain topology is denoted by Z, where any given switch i may go from 0 to $Z - 1$, whose order may be adapted through each particular design.

This way, a message may be forwarded on from a source host to a destination host by means of the outcome of certain arithmetic operations to be calculated with the destination host identifier, although sometimes logical operations may also apply.

To start with, source and destination switches may be determined, and in turn, all possible paths between them, leading to the discovery of all possible switches making up each of those ways.

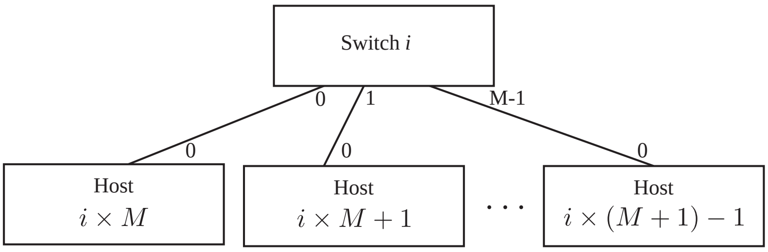


Figure 5.1: Model for a generic switch i, with its connected hosts.

For instance, let us suppose that there is a communication where source host is a and destination host is b, assuming that each switch have the same amount of M hosts hanging on. In this case, the source switch may be found out by applying $\lfloor {}^{a}/_{M} \rfloor$, which accounts for the result of the *integer division*, alternatively indicated by the expression $int({}^{a}/_{M})$. Taking that into consideration, the destination switch may be learnt by applying $\lfloor {}^{b}/_{M} \rfloor$, which is obviously equivalent to $int({}^{b}/_{M})$.

Besides, the port on the source switch connected to the source host is given by $a_{|M}$, which accounts for the result of calculating *a congruence modulo M*, also known as the remainder of the integer division of both variables, alternatively indicated by the expression $a\ mod\ M$. Taking this into consideration, the port on the destination switch where the destination host is connected is given by $b_{|M}$, or alternatively, $b\ mod\ M$. Otherwise, certain operations may help find the following switches available on the way to the destination switch, depending on the topology.

To wrap it all up, Table 5.1 summarizes all of this nomenclature and the arithmetic operations for communicating from source host a to destination host b.

5.2 Modelling hosts

Prior to undertaking the modelling of the different topologies proposed, it is necessary to model the behaviour of the hosts, as they may appear in all topology models as end devices. Therefore, after having explained how hosts are being identified, the focus is going to be set on their real behaviour regarding communications, and in turn, a couple of different models may be exposed to express it.

Table 5.1: Relevant variables and operations to be applied in the models.

Item	Expression
Total amount of switches	Z
Range of switch identifiers	$[0 \cdots Z-1]$
Any given switch	i
Total amount of hosts per switch	M
Range of downlink ports per switch	$[0 \cdots M-1]$
Any given port of a switch	p
Any given host	$h = i \times M + p$
Total amount of hosts overall	$max(h) = M \times Z$
Range of host identifiers	$[0 \cdots M \times Z - 1]$
Source host	a
Destination host	b
Source switch	$\lfloor a/M \rfloor \longleftrightarrow int(a/M)$
Destination switch	$\lfloor b/M \rfloor \longleftrightarrow int(b/M)$
Source port in the source switch	$a_{\mid M} \longleftrightarrow a\ mod\ M$
Destination port in the destination switch	$b_{\mid M} \longleftrightarrow b\ mod\ M$

A host may behave in two ways, depending on whether a given VM associated to a particular user u, that being expressed by $VM(u)$, is located within that particular host. If this is the case, then such a VM may be sent over to another host at any time, whereas on the contrary, such a VM may be received in from another host at any time.

As per the arguments involved in such actions, there may be three of them, such as source host a, destination host b, and the remote computing resource being moved $VM(u)$, although all those arguments may also be generically expressed as just d for simplicity purposes. In the following models, any of those solutions may be employed indistinctly in the models being presented herein.

Therefore, the model for a given host may require to check all hosts being part of the system, whose amount is given by $max(h)$, identified from 0 to $max(h) - 1$, and within each one, to search for all users within the system, whose amount is given by $max(u)$, identified from 0 to $max(u) - 1$, in order to inspect whether the VM associated to each user is indeed inside that host. If that is the case, such a host may get ready to send that VM belonging to that particular user off at any given time through its port 0, or otherwise, it may get ready to receive it in at any given moment through its port 0.

Eventually, it is to be remarked that hosts are the end devices within all switching topologies, thus from the point of view of network communication within Fog environments, the function of hosts is merely to send a VM to the switching topology, or otherwise, to receive a VM from the switching topology, leaving aside the computing processing being carried out on hosts. Hence, the verification of those switching topologies may include the hosts starting and finishing the VM transactions on them.

As per the *layout*, every host h is supposed to have only a single port, identified by 0, as depicted in Figure 5.2. It is to be reminded that there may be M hosts per switch, whereas there may be $max(h) = M \times Z$ hosts, being equally distributed through all end switches.

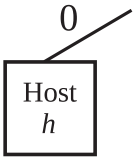


Figure 5.2: Model for a generic host h.

Two *flow charts* are going to be exposed for a given host h. First, a simplified way, supposing that there may be only one user within the system, as it is exhibited in Figure 5.3. This case scenario may make things easier to follow.

Afterwards, a generic way, considering that many users may be within the system, as it is depicted in Figure 5.4. This case scenario may show a more complete scenario.

Regarding the *pseudocode model* to express the generic behaviour of a particular host, it is to be reminded that *send* and *receive* functions take three arguments, such as the source host sending the VM, the destination host receiving the VM and the VM itself, which is associated with a user u. Besides, both functions take two parameters, such as the device identifier in action and the port being in use.

Therefore, algorithm 1 (shown in the Appendix 10.1) exhibits the *pseudocode model* regarding the behaviour of a given host h for the generic way, thus considering many users within the system. In this case, functions *send* and *receive* are expressed with

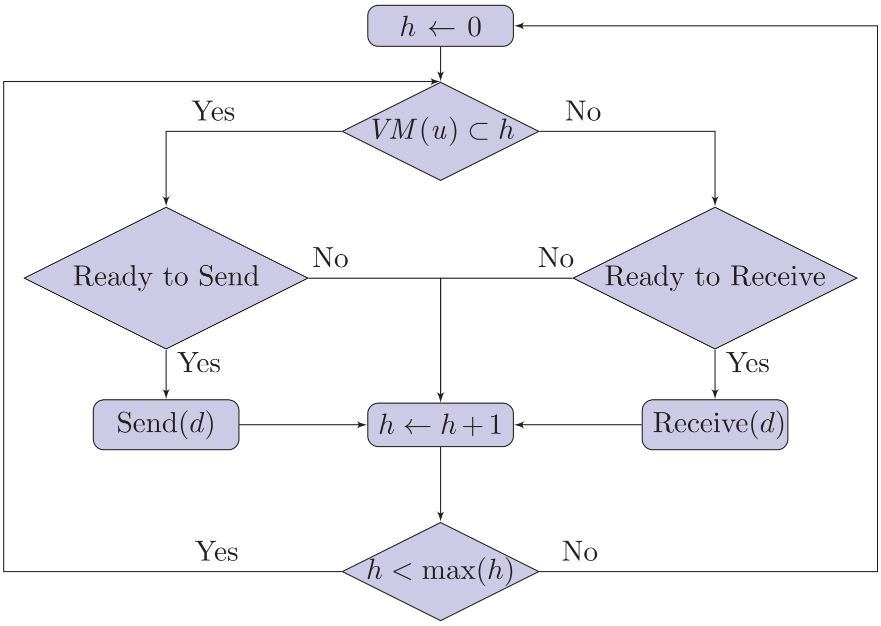


Figure 5.3: Flow chart for hosts, with a unique user.

their three arguments exposed before, such as source host as the first one, destination host as the second one and the associated VM to a given user as the third one.

With respect to the *algebraic model*, the following recursive expression exhibits the behaviour of a particular host by means of ACP, where the three arguments explained above are consolidated as just one, resulting in d, such as sending is expressed as $s(d)$ and receiving is exhibited as $r(d)$, whereas the empty set ($\emptyset$) represents the case where none of them occurs. On the other hand, the two parameters quoted above still apply, such as host H_h (in order to distinguish the host identified by h, which is the one performing a given action, such as send or receive) and its port 0 (that being the unique port in every host). Additionally, it is to be said that sequential and alternate operators are being used in the expression, rendered by the product and the addition signs, respectively, whereas the guarded linear recursion implies that the action never ends. The process described is checked out for all hosts and all users in (5.2).

$$H_h = \sum_{h=0}^{max(h)-1} \Bigg(\sum_{u=0}^{max(u)-1} \Big(s_{H_h,0}(d) + r_{H_h,0}(d) + \emptyset \Big) \Bigg) \cdot H_h \tag{5.2}$$

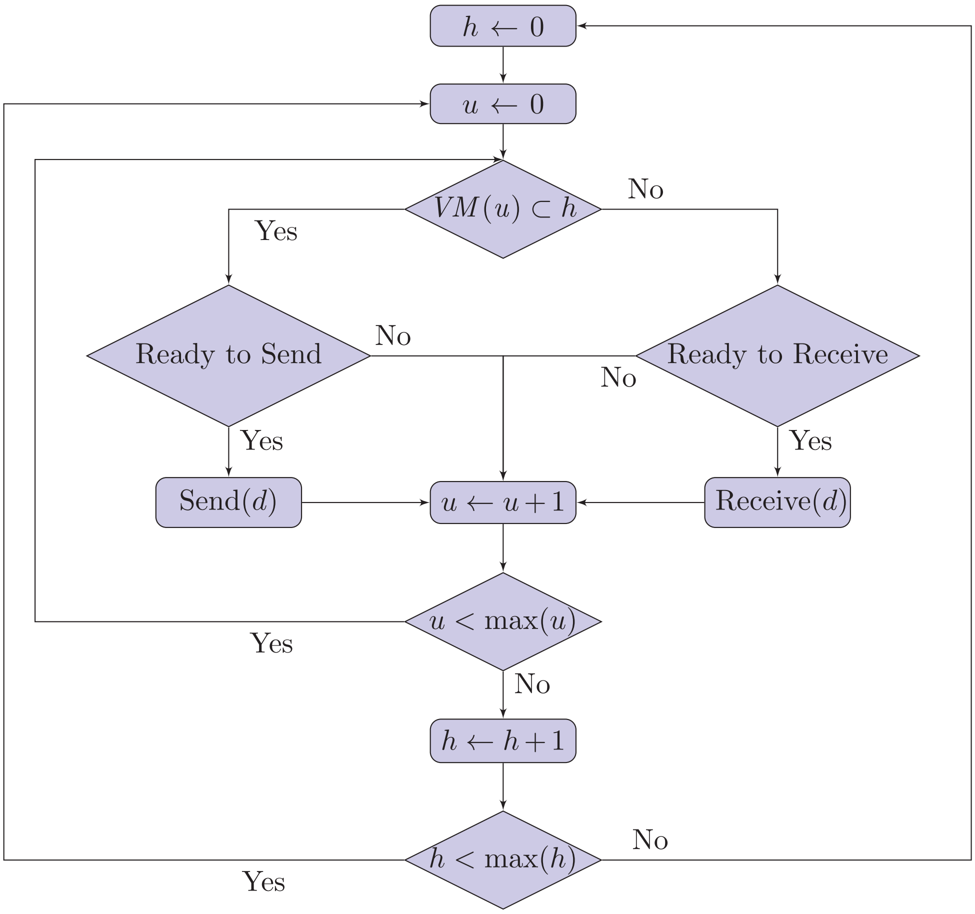


Figure 5.4: Flow chart for hosts, with several users.

5.3 Fat tree

As stated before, fat tree has 3 layers of switches, where the switches standing on each layer are enumerated from 0 on, starting from the left hand side and going all the way to the right hand side, where the last member of each kind is identified by the integer predecessor of the overall values for each layer, which have already been stated in Table 4.1.

In this context, the lower layer switches are identified as E_i, as that layer is called edge and i is the device identifier. Likewise, the middle layer is named as A_j, as this layer is named aggregation and j is the item identifier. And finally, the upper layer is referred to as C_l, as such as layer is called core and l is the object identifier.

As per the *layout*, Figure 5.5 depicts the models for each type, along with their corresponding upper and lower ports. Additionally, it is to be reminded that each end switch, meaning the edge switches, connect to $K/2$ hosts, accounting for $K^2/4$ per pod, which makes an overall amount of $K^3/4$ hosts included into the topology.

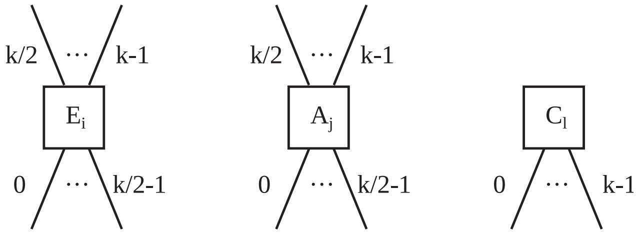


Figure 5.5: Model for switches and their ports in fat tree.

Furthermore, Figure 5.6 exhibits the whole structure for a fat tree deployment with $K = 4$ and oversubscription rate 1:1.

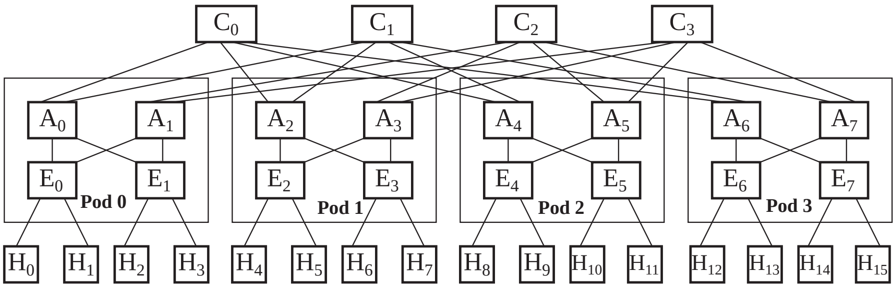


Figure 5.6: Fat tree device nomenclature for $K = 4$.

5.3.1 Edge switches

Basically, going along all edge switches (from 0 to $K^2/2$), all ports are checked in order to see if a particular VM is received, and if that is the case, it is verified whether the destination host is hanging on that switch. If this is it, the VM is sent to the corresponding destination host, or otherwise, that VM is sent to all upper ports, for redundancy purposes.

The arithmetic condition to verify whether a given edge switch i is the destination switch of a VM is given in (5.3), meaning that both integer parts are the same.

Likewise, the condition to verify that a source and a destination hosts are hanging on the same edge switch is given in (5.4).

On the other hand, if an edge switch happens to be the destination switch, the way to forward a VM through the proper port to the destination host is given by the expression (5.5).

Those expressions just take advantage of the port layout, where the lower half of ports (from 0 to ${}^{K}\!/2 - 1$) look downwards to the hosts, whereas the upper half (${}^{K}\!/2$ to $K - 1$) look upwards to the middle layer switches. In this sense, any of the ports are identified by variable p, whilst just any of the upper ports are spotted by variable q.

$$int({}^{i}\!/({}^{K}\!/2)) = int({}^{b}\!/({}^{K}\!/2)) \longleftrightarrow \left\lfloor \frac{i}{{}^{K}\!/2} \right\rfloor = \left\lfloor \frac{b}{{}^{K}\!/2} \right\rfloor \tag{5.3}$$

$$int({}^{a}\!/({}^{K}\!/2)) = int({}^{b}\!/({}^{K}\!/2)) \longleftrightarrow \left\lfloor \frac{a}{{}^{K}\!/2} \right\rfloor = \left\lfloor \frac{b}{{}^{K}\!/2} \right\rfloor \tag{5.4}$$

$$b \;\; mod \;\; {}^{K}\!/2 \longleftrightarrow b \;\; modulo \;\; {}^{K}\!/2 \longleftrightarrow b_{|{}^{K}\!/2} \tag{5.5}$$

Considering that, Figure 5.7 exposes the *flow chart* explaining the behaviour of a particular edge switch i.

Moreover, algorithm 2 (shown in the Appendix 10.2) exposes the *pseudocode model* regarding the behaviour of a given edge switch i.

With respect to the *algebraic model*, the ACP expression representing the behaviour of an edge switch is exhibited in (5.6). ACP does not take time into account, hence the first action may always be the reception of a message through any of its ports, thus the initial conditional clause may be skipped.

$$E_i = \sum_{i=0}^{\frac{K^2}{2}-1} \left(\sum_{p=0}^{K-1} \left(r_{E_i,p}(d) \cdot \left(s_{E_i,b_{|K/2}}(d) \triangleleft \left\lfloor \frac{i}{{}^{K}\!/2} \right\rfloor = \left\lfloor \frac{b}{{}^{K}\!/2} \right\rfloor \triangleright \sum_{q={}^{K}\!/2}^{K-1} s_{E_i,q}(d) \right) \right) \right) \cdot E_i \tag{5.6}$$

5.3.2 Aggregation switches

Basically, going through all aggregation switches (from 0 to ${}^{K^2}\!/2$), all ports are verified so as to find out if a given VM is received, and in that case, it is checked whether the destination host belongs to the same pod. In case they do, that VM is

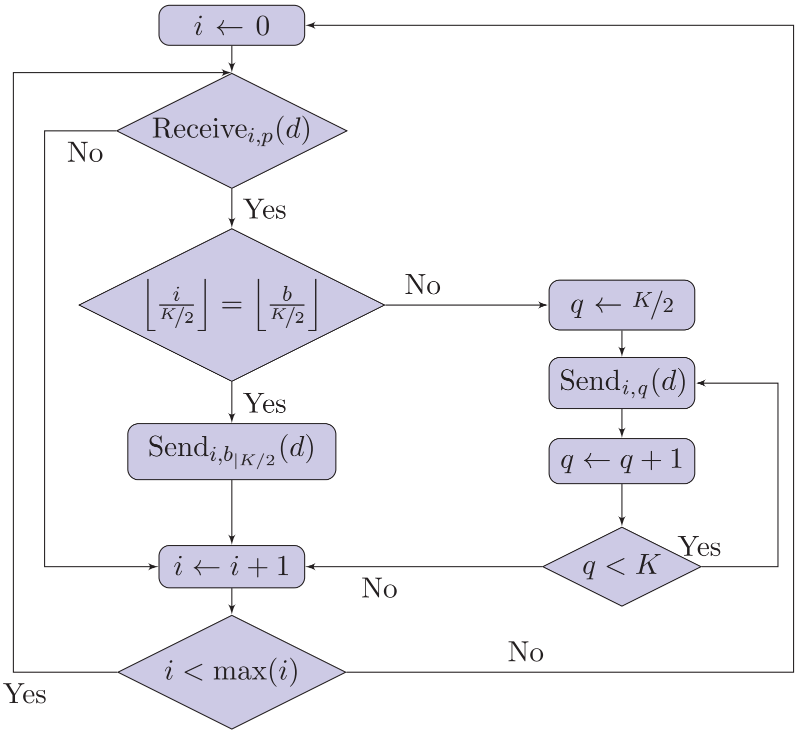


Figure 5.7: Flow chart for the behaviour of an edge switch (fat tree).

forwarded to the edge switch where the destination host is hanging on, or otherwise, that VM is forwarded through all upper ports, for redundancy purposes.

The arithmetic condition to establish whether a particular aggregation switch j belongs to the same pod as the destination switch of a VM is given in (5.7), meaning that both integer parts are the same. Likewise, the condition to check whether a source and a destination hosts belong to the same pod is given in (5.8).

On the other hand, the way to forward a VM through the proper port to the edge switch where the destination host hangs on is given by the expression (5.9).

Those expressions just take advantage of the port layout, where the lower half of ports (from 0 to $K/2 - 1$) look downwards to the hosts, whereas the upper half of ports ($K/2$ to $K - 1$) look upwards to the middle layer switches, where the whole set of ports are identified by variable p, whilst the set of upper ports is branded as q.

$$int(i/({}^{K}\!/_{2})^2) = int(b/({}^{K}\!/_{2})^2) \longleftrightarrow \left\lfloor \frac{i}{({}^{K}\!/_{2})^2} \right\rfloor = \left\lfloor \frac{b}{({}^{K}\!/_{2})^2} \right\rfloor \tag{5.7}$$

$$int(a/({}^{K}\!/_{2})^2) = int(b/({}^{K}\!/_{2})^2) \longleftrightarrow \left\lfloor \frac{a}{({}^{K}\!/_{2})^2} \right\rfloor = \left\lfloor \frac{b}{({}^{K}\!/_{2})^2} \right\rfloor \tag{5.8}$$

$$int(b/({}^{K}\!/_{2}))\ \ mod\ \ {}^{K}\!/_{2} \longleftrightarrow \left\lfloor \frac{b}{{}^{K}\!/_{2}} \right\rfloor_{|{}^{K}\!/_{2}} \tag{5.9}$$

Considering this, Figure 5.8 exposes the *flow chart* showing the behaviour of an aggregation switch j.

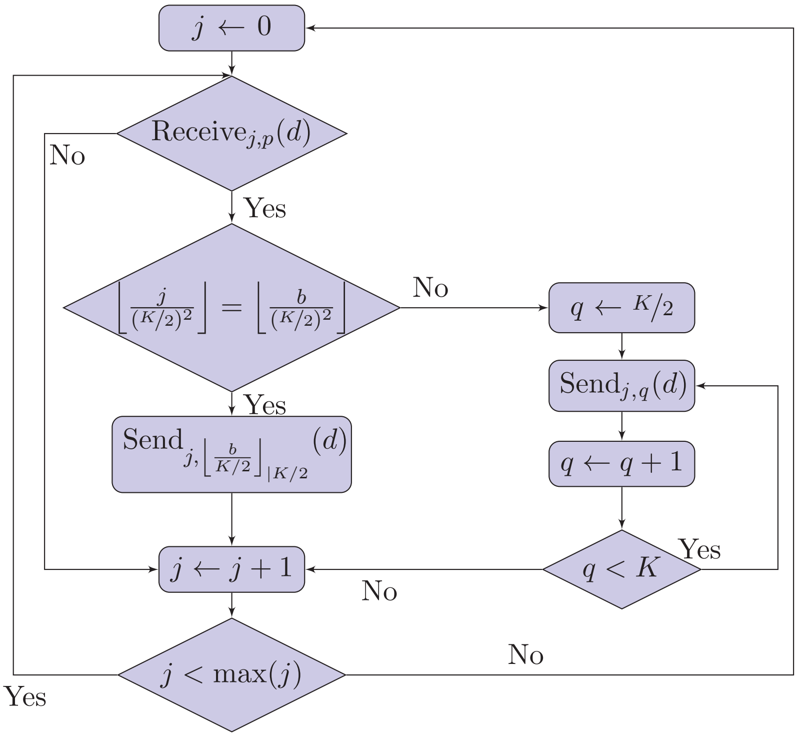

$j \leftarrow 0$
Receive$_{j,p}(d)$
No
Yes
$\left\lfloor \frac{j}{(K/2)^2} \right\rfloor = \left\lfloor \frac{b}{(K/2)^2} \right\rfloor$
No
$q \leftarrow K/2$
Send$_{j,q}(d)$
Yes
Send$_{j,\left\lfloor \frac{b}{K/2} \right\rfloor_{|K/2}}(d)$
$q \leftarrow q+1$
Yes
$j \leftarrow j+1$
$q < K$
No
No
$j < \max(j)$
Yes

Figure 5.8: Flow chart for the behaviour of an aggregation switch (fat tree).

Besides, algorithm 3 (shown in the Appendix 10.3) exhibits the *pseudocode model* regarding the behaviour of a given aggregate switch j.

With respect to the *algebraic model*, the ACP expression representing the behaviour of an aggregation switch is exhibited in (5.10), where the conditional in the receive function has been dropped as ACP does not take the time into consideration.

$$A_j = \sum_{j=0}^{\frac{K^2}{2}-1} \left(\sum_{p=0}^{K-1} \left(r_{A_j,p}(d) \cdot \left(s_{A_j, \lfloor \frac{b}{K/2} \rfloor_{|K/2}}(d) \triangleleft \left\lfloor \frac{i}{(K/2)^2} \right\rfloor = \left\lfloor \frac{b}{(K/2)^2} \right\rfloor \triangleright \sum_{q=K/2}^{K-1} s_{A_j,q}(d) \right) \right) \right) \cdot A_j \tag{5.10}$$

5.3.3 Core switches

Basically, core switches are used in order to jump from a source pod to a destination pod, where the condition to check whether a source host and a destination host are located in different pods is given in (5.11).

In this context, going core switch by core switch (from 0 to ${}^{K^2}\!/_{4} - 1$), if a given port receives a VM in, then it is forwarded on to the port pointing to an aggregation switch located within the pod where the destination host belongs to, meaning the destination pod, and such a port is given by (5.12).

That arithmetic expression just takes advantage of the port layout, where all ports look downwards (from 0 to $K-1$), all of them being identified by variable p. It happens that each of those ports just connects to a single aggregation switch in each pod, where each pod identifier matches the proper port identifier in each core switch.

This way, full mesh is achieved among all pods, as any core has connections to each pod, although partial mesh is obtained among all hosts, as all core switches do not have connectivity with all aggregation switches, but just one in each pod.

$$int(a/({}^{K}\!/_{2})^2) \neq int(b/({}^{K}\!/_{2})^2) \longleftrightarrow \left\lfloor \frac{a}{({}^{K}\!/_{2})^2} \right\rfloor \neq \left\lfloor \frac{b}{({}^{K}\!/_{2})^2} \right\rfloor \tag{5.11}$$

$$int(b/({}^{K}\!/_{2})^2) \longleftrightarrow \lfloor {}^{b}\!/_{(K/2)^2} \rfloor \tag{5.12}$$

With that in mind, Figure 5.9 exposes the *flow chart* exposing the behaviour of a core switch l.

Furthermore, algorithm 4 (shown in the Appendix 10.4) exhibits the *pseudocode model* regarding the behaviour of a given core switch l.

With regards to the *algebraic model*, the ACP expression representing the behaviour of a core switch is shown in (5.13), where the conditional in the receive function has been skipped as ACP does not consider the time.

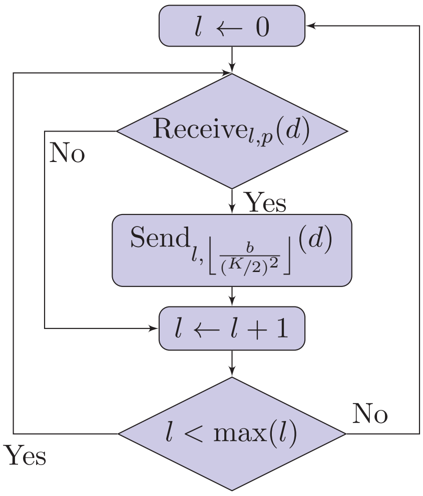

$l \leftarrow 0$
$\text{Receive}_{l,p}(d)$
No
Yes
$\text{Send}_{l,\left\lfloor \frac{b}{(K/2)^2} \right\rfloor}(d)$
$l \leftarrow l+1$
$l < \max(l)$
No
Yes

Figure 5.9: Flow chart for the behaviour of a core switch (fat tree).

$$C_l = \sum_{l=0}^{\frac{K^2}{4}-1} \left(\sum_{p=0}^{K-1} \left(r_{C_l,p}(d) \cdot s_{C_l,\left\lfloor \frac{b}{(K/2)^2} \right\rfloor}(d) \right) \right) \cdot C_l \tag{5.13}$$

5.3.4 Consolidating fat tree

Focusing on *redundant paths* within the model, it may be possible to find out all devices being within the path from source host a to destination host b by means of applying the appropriate arithmetic expressions to both variables. Furthermore, all relevant ports being traversed may also be found out with other arithmetic expressions, thanks to the nomenclature previously assigned to both devices and ports.

Otherwise, in the cases where two different edge switches are used, those may be identified by i the first one and i' the second one, whereas two different aggregated switches may be distinguished by j the first one and j' the second one.

In order to better clarify the items and ports taking part in each communication, the items are going to be enclosed into curly brackets, whereas the ports will be situated facing to each other, showing an arrow between both ports. This fact forces the inversion of the receiving end from the way it has been shown in the previous algorithms, such that the port is shown first, whilst the device is shown afterward. Therefore, a link between a pair of devices is expressed as in (5.14).

$$\{SourceItem\}SourcePort \longmapsto DestinationPort\{DestinationItem\} \tag{5.14}$$

This way, in the path between H_h and E_i, the communication between the source host and the source switch is $\{H_a\}, 0 \longmapsto a_{|K/2}, \{E_i\}$, whereas the communication between that edge switch and the destination host is $\{E_i\}, b_{|K/2} \longmapsto 0, \{H_b\}$, leaving aside the links to upper layers of switches.

It is to be reminded that, in the case that source host and destination host are just 1 hop away, then $i = \left\lfloor \frac{a}{K/2} \right\rfloor = \left\lfloor \frac{b}{K/2} \right\rfloor$, as source switch and destination switch happen to be the same item, hence there is only an edge switch, leaving the rest of them in an idle state. Therefore, the only relevant items herein are source host, edge switch and destination host, thus making the rest of items go deadlock.

Likewise, in the path between E_i and A_j, the communication between the source switch and one of the aggregate switches j inside its own pod is given by the expression $\{E_i\}, \frac{K}{2} + j_{|K/2} \longmapsto \left\lfloor \frac{a}{K/2} \right\rfloor_{|K/2}, \{A_j\}$, whereas the communication between any of those aggregation switches j and the destination switch is exposed by the expression $\{A_j\}, \left\lfloor \frac{b}{K/2} \right\rfloor_{|K/2} \longmapsto \frac{K}{2} + j_{|K/2}, \{E_{i'}\}$, putting aside the devices located in other layers.

It is to be considered that, in the case that source switch and destination switch belong to the same pod, then $pod = \left\lfloor \frac{a}{(K/2)^2} \right\rfloor = \left\lfloor \frac{b}{(K/2)^2} \right\rfloor$. With regards to the source switch, it is given by $i = \left\lfloor \frac{a}{K/2} \right\rfloor$, whereas the destination switch is attained by $i' = \left\lfloor \frac{b}{K/2} \right\rfloor$, taking the rest of edge switches to deadlock. With respect to the aggregation switches, they are all located in the same pod where both source and destination hosts are located, being those going from $\frac{K}{2} \times \left\lfloor \frac{b}{(K/2)^2} \right\rfloor$ to $\frac{K}{2} \times \left\lfloor \frac{b}{(K/2)^2} \right\rfloor + \frac{K}{2} - 1$, thus bringing the rest to deadlock.

Regarding the upper links in the edge layer, being those connecting to the aggregation layer, hence going from $^{K}/_{2}$ to $K-1$, it is to be mentioned that all of them are involved in communication, thus allowing the set up of $^{K}/_{2}$ redundant paths. Those ports may be easily identified by the expression $\frac{K}{2} + j_{|K/2}$, meaning that each port is identified thanks to that expression related to the aggregation switch j being directly connected, and that occurs in both upwards and downwards communications.

With respect to the lower links in the aggregation layer, being the ones making a connection to the edge layer, only two ports are being used, such as the one receiving traffic from the source switch, which is identified by $\left\lfloor \frac{a}{K/2} \right\rfloor_{|K/2}$, and the one sending traffic to the destination switch, which is identified by $\left\lfloor \frac{b}{K/2} \right\rfloor_{|K/2}$.

Otherwise, in the path between A_j and C_l, the communication between source pod and destination pod goes through all core switches, where the uplink may be expressed as $\{A_j\}, \frac{K}{2} + l_{|K/2} \longmapsto \left\lfloor \frac{a}{(K/2)^2} \right\rfloor, \{C_l\}$, considering that all ports of core switches l communicate with aggregate switches j, as opposed to what happens in aggregate switches, where only the upper half of the ports are used for that. On the other hand, the downlink may be seen as $\{C_l\}, \left\lfloor \frac{b}{(K/2)^2} \right\rfloor \longmapsto \frac{K}{2} + l_{|K/2}, \{A_{j'}\}$.

It is to be recalled that items belonging to other layers go deadlock, as well as aggregation switches not belonging to source or destination pods, where the former are those going from $\frac{K}{2} \times \left\lfloor \frac{a}{(K/2)^2} \right\rfloor$ to $\frac{K}{2} \times \left\lfloor \frac{a}{(K/2)^2} \right\rfloor + \frac{K}{2} - 1$ and the latter are from $\frac{K}{2} \times \left\lfloor \frac{b}{(K/2)^2} \right\rfloor$ to $\frac{K}{2} \times \left\lfloor \frac{b}{(K/2)^2} \right\rfloor + \frac{K}{2} - 1$. Otherwise, an aggregation switch j gets connected to $\frac{K}{2}$ core switches, being those from $l = j \times \frac{K}{2}$ to $l = j \times \frac{K}{2} + \frac{K}{2} - 1$.

With regards to the ports being involved, aggregation switches uses ports from ${}^{K}\!/_{2}$ to $K - 1$, which may be identified with the expression $\frac{K}{2} + l_{|K/2}$, which relates to the core switch l where they connect. Referring to core switches, all of them take part in communication, where the port receiving the traffic from the source pod is $\left\lfloor \frac{a}{({}^{K}\!/_{2})^2} \right\rfloor$, whilst the port sending the traffic to the destination pod is $\left\lfloor \frac{b}{({}^{K}\!/_{2})^2} \right\rfloor$.

Therefore, following the communications between the different layers of devices, it is possible to obtain a *list of devices* making up the redundant paths from source host to destination host, where their relevant ports are also shown as well. To wrap it all up, Table 5.2 summarizes all relevant arithmetic expression presented to build up the list of devices and their ports for hosts being one hop away (where $\left\lfloor \frac{a}{{}^{K}\!/_{2}} \right\rfloor = \left\lfloor \frac{b}{{}^{K}\!/_{2}} \right\rfloor$), whilst Table 5.3 does it for hosts being two hops away (where $\left\lfloor \frac{a}{({}^{K}\!/_{2})^2} \right\rfloor = \left\lfloor \frac{b}{({}^{K}\!/_{2})^2} \right\rfloor$) and Table 5.4 does it for hosts being three hops away (where $\left\lfloor \frac{a}{({}^{K}\!/_{2})^2} \right\rfloor \neq \left\lfloor \frac{b}{({}^{K}\!/_{2})^2} \right\rfloor$), taking into account that switch identifiers have been substituted by their correspondent references to source and destination hosts.

As per the results exposed in those tables, it is to be remarked that in the 1 hop away case, there is only 1 available path, as both hosts share the same edge switch, whereas in the 2 hops away case, there are ${}^{K}\!/_{2}$ available paths, as both hosts share the same pod, whilst in the 3 hops away case, there are up to $({}^{K}\!/_{2})^2$ available paths, as they share nothing.

Table 5.2: List of devices and ports between 1-hop away hosts in fat tree.

Communication	**Direction**	**Expression**
Hosts and edges	Uplink	$\{H_a\}, 0 \longmapsto a_{\lvert K/2}, \{E_{\lfloor \frac{a}{K/2} \rfloor}\}$
Edges and hosts	Downlink	$\{E_{\lfloor \frac{b}{K/2} \rfloor}\}, b_{\lvert K/2} \longmapsto 0, \{H_b\}$

Table 5.3: List of devices and ports between 2-hops away hosts in fat tree.

Communication	**Direction**	**Expression**
Hosts and edges	Uplink	$\{H_a\}, 0 \longmapsto a_{\lvert K/2}, \{E_{\lfloor \frac{a}{K/2} \rfloor}\}$
Edges and aggr.	Uplink	$\sum_{t=0}^{\frac{K}{2}-1} \Big(\{E_{\lfloor \frac{a}{K/2} \rfloor}\}, (\frac{K}{2}+t) \longmapsto$ $\longmapsto \left\lfloor \frac{a}{K/2} \right\rfloor_{\lvert K/2}, \{A_{\frac{K}{2} \cdot \lfloor \frac{a}{(K/2)^2} \rfloor + t}\}\Big)$
Aggr. and edges	Downlink	$\sum_{t=0}^{\frac{K}{2}-1} \Big(A_{\frac{K}{2} \cdot \lfloor \frac{b}{(K/2)^2} \rfloor + t}\}, \left\lfloor \frac{b}{K/2} \right\rfloor_{\lvert K/2} \longmapsto$ $\longmapsto (\frac{K}{2}+t), \{E_{\lfloor \frac{b}{K/2} \rfloor}\}\Big)$
Edges and hosts	Downlink	$\{E_{\lfloor \frac{b}{K/2} \rfloor}\}, b_{\lvert K/2} \longmapsto 0, \{H_b\}$

Eventually, taking into consideration the ACP models for each component, and the previous discussion about paths between neighbouring items, it may be possible to establish a verification framework for the whole model. To start with, let us put all devices in a concurrent manner by means of the merge operator, $\|$, and in turn, let us apply the *encapsulation operator* to the whole set of items, which accounts for a combination of many terms according to expression (1), mentioned before.

The actions of the encapsulation operator are twofold. On the one hand, it combines atomic actions taking place in the same internal channel, such as sending and receiving at opposite ends of an internal link, thus allowing communication in such a channel, whilst discarding those atomic actions. On the other hand, the atomic actions not resulting in internal communications go deadlock, hence they go discarded, whereas atomic actions related to external actions are invariant.

Table 5.4: List of devices and ports between 3-hops away hosts in fat tree.

Communication	**Direction**	**Expression**
Hosts and edges	Uplink	$\{H_a\}, 0 \longmapsto a_{\mid K/2}, \{E_{\left\lfloor \frac{a}{K/2} \right\rfloor}\}$
Edges and aggr.	Uplink	$\sum_{t=0}^{\frac{K}{2}-1} \Big(\{E_{\left\lfloor \frac{a}{K/2} \right\rfloor}\}, (\frac{K}{2}+t) \longmapsto$ $\longmapsto \left\lfloor \frac{a}{K/2} \right\rfloor_{\mid K/2}, \{A_{\frac{K}{2} \cdot \left\lfloor \frac{a}{(K/2)^2} \right\rfloor + t}\}\Big)$
Aggr. and core	Uplink	$\sum_{t=0}^{\frac{K}{2}-1} \Bigg(\sum_{u=0}^{\frac{K}{2}-1} \Big(\{A_{\frac{K}{2} \cdot \left\lfloor \frac{a}{(K/2)^2} \right\rfloor + t}\}, (\frac{K}{2}+u) \longmapsto$ $\longmapsto \left\lfloor \frac{a}{(K/2)^2} \right\rfloor, \{C_{t \cdot \frac{K}{2}+u}\}\Big)\Bigg)$
Core and aggr.	Downlink	$\sum_{t=0}^{\frac{K}{2}-1} \Bigg(\sum_{u=0}^{\frac{K}{2}-1} \Big(\{C_{t \cdot \frac{K}{2}+u}\}, \left\lfloor \frac{b}{(K/2)^2} \right\rfloor \longmapsto$ $\longmapsto (\frac{K}{2}+u), \{A_{\frac{K}{2} \cdot \left\lfloor \frac{b}{(K/2)^2} \right\rfloor + t}\}\Big)\Bigg)$
Aggr. and edges	Downlink	$\sum_{t=0}^{\frac{K}{2}-1} \Big(A_{\frac{K}{2} \cdot \left\lfloor \frac{b}{(K/2)^2} \right\rfloor + t}\}, \left\lfloor \frac{b}{K/2} \right\rfloor_{\mid K/2} \longmapsto$ $\longmapsto (\frac{K}{2}+t), \{E_{\left\lfloor \frac{b}{K/2} \right\rfloor}\}\Big)$
Edges and hosts	Downlink	$\{E_{\left\lfloor \frac{b}{K/2} \right\rfloor}\}, b_{\mid K/2} \longmapsto 0, \{H_b\}$

In other words, it might be said that only the internal communications prevail from the application of the merge operator, whereas external atomic actions are not affected. It is to be said that all internal atomic actions define a subset H out of set A containing all atomic actions, and that is why the encapsulation operator on the term $(x \parallel y)$ is represented by $\partial_H(x \parallel y)$, which may only execute the possible communication between them both as long as it is not deadlock (δ).

Additionally, it is to be considered that a source host a and a destination host b are supposed to be set, making the rest of hosts inactive for that transaction, as well as the rest of the switches not taking part in it, thus going deadlock. Therefore, the encapsulation operator only keeps the relevant internal links for a particular communication, although not applying to atomic actions related to external communications.

In this sense, communications between hosts and edge switches are exposed in (5.15), as well as between edge and aggregation switches in (5.16), and between aggregation and core switches in (5.17). Besides, it is to be noted that every pair of non-neighbouring layers do not directly interact to each other, hence their interactions will go deadlock, thus getting discarded (5.18).

$$\begin{aligned}
\Big|\Big|_{h=0}^{\frac{K^3}{4}-1}\Big|\Big|_{i=0}^{\frac{K^2}{2}-1}\partial_H(H_h \parallel E_i) = \Bigg(&c_{\{H_a\},0\longmapsto a_{|K/2},\{E_i\}}(d)\cdot \\
\bigg(c_{\{E_i\},b_{|K/2}\longmapsto 0,\{H_b\}}(d) \lhd \left\lfloor \frac{i}{K/2} \right\rfloor = \left\lfloor \frac{b}{K/2} \right\rfloor \rhd \sum_{t=K/2}^{K-1} s_{E_i,t}(d)\bigg)\Bigg) &\cdot (H_h \parallel E_i) \qquad (5.15) \\
\Big|\Big|_{i=0}^{\frac{K^2}{2}-1}\Big|\Big|_{j=0}^{\frac{K^2}{2}-1}\partial_H(E_i \parallel A_j) = \sum_{t=0}^{\frac{K}{2}-1}\Bigg(&c_{\{E_i\},\frac{K}{2}+t\longmapsto \left\lfloor \frac{a}{K/2}\right\rfloor_{|K/2},\{A_j\}}(d)\cdot \\
\bigg(c_{\{A_j\},\left\lfloor \frac{b}{K/2}\right\rfloor_{|K/2}\longmapsto \frac{K}{2}+t,\{E_i\}}(d) \lhd \left\lfloor \frac{i}{(K/2)^2} \right\rfloor = \left\lfloor \frac{b}{(K/2)^2} \right\rfloor \rhd \sum_{u=K/2}^{K-1} s_{A_j,u}(d)\bigg)\Bigg) &\cdot (E_i \parallel A_j) \\
&\qquad (5.16) \\
\Big|\Big|_{j=0}^{\frac{K^2}{2}-1}\Big|\Big|_{l=0}^{\frac{K^2}{4}-1}\partial_H(A_j \parallel C_l) = \sum_{u=0}^{\frac{K}{2}-1}\Bigg(&c_{\{A_j\},\frac{K}{2}+u\longmapsto \left\lfloor \frac{a}{(K/2)^2}\right\rfloor,\{C_l\}}(d)\cdot \\
&c_{\{C_l\},\left\lfloor \frac{b}{(K/2)^2}\right\rfloor\longmapsto \frac{K}{2}+u,\{A_j\}}(d)\Bigg) \cdot (A_j \parallel C_l) \qquad (5.17) \\
\Big|\Big|_{h=0}^{\frac{K^3}{4}-1}\Big|\Big|_{j=0}^{\frac{K^2}{2}-1}\partial_H(H_h \parallel A_j) &= \Big|\Big|_{h=0}^{\frac{K^3}{4}-1}\Big|\Big|_{l=0}^{\frac{K^2}{4}-1}\partial_H(H_h \parallel C_l) \\
&= \Big|\Big|_{i=0}^{\frac{K^2}{2}-1}\Big|\Big|_{l=0}^{\frac{K^2}{4}-1}\partial_H(E_i \parallel C_l) = \delta \qquad (5.18)
\end{aligned}$$

After having applied the encapsulation operator, it is the moment to apply the *abstraction operator* in order to mask internal communications, as well as silent steps, meaning abstraction from the internal actions, whereas external communications prevail. After that, the external behaviour of the model may show up, and in turn, it may be compared to that of the real system.

It is to be said that all internal communications define a subset I out of set A containing all atomic actions, and this is why the abstraction operator on the term (v) is represented by $\tau_I(v)$, which may account for v if, and only if, $v \notin I$, while going to silent step (τ) if, and only if, $v \in I$, whereas it does not have any effect on deadlock, such that $\tau_I(\delta) = \delta$. Therefore, internal communications and internal actions go to silent step, thus getting discarded, whilst external communications make it through.

At this point, two approaches may be followed, such as including the end hosts and all the switches within the model, or otherwise, only consider the switching infrastructure, thus leaving the end hosts out of the system being modelled. The former may account for expression (5.19), which in fact shows the external behaviour of a closed system, as no external communication would exist therein. Likewise, in the previous reasoning it has been seen that all results obtained show that all communications are internal, which will be masked by the abstraction operator, resulting in the empty set $\varnothing$, thus reiterating the fact that no external behaviour is shown. As a side note, it is to be said that $\emptyset$ symbol has been used in conditional operators in case no statement needs to be carried out, which is not to be confused with $\varnothing$.

$$\begin{aligned}
\Big|\Big|_{h=0}^{\frac{K^3}{4}-1}\Big|\Big|_{i=0}^{\frac{K^2}{2}-1}\Big|\Big|_{j=0}^{\frac{K^2}{2}-1}\Big|\Big|_{l=0}^{\frac{K^2}{4}-1}\tau_I\Bigg(\partial_H\Big(H_h \parallel E_i \parallel A_j \parallel C_l\Big)\Bigg) = \tau_I\Bigg(& c_{\{H_a\},0\longmapsto a_{|K/2},\{E_i\}}(d)\cdot \\
\Bigg(\sum_{t=0}^{\frac{K}{2}-1} c_{\{E_i\},\frac{K}{2}+t\longmapsto \left\lfloor \frac{a}{K/2}\right\rfloor_{|K/2},\{A_j\}}(d)\cdot\Bigg(\sum_{u=0}^{\frac{K}{2}-1} & c_{\{A_j\},\frac{K}{2}+u\longmapsto \left\lfloor \frac{a}{(K/2)^2}\right\rfloor,\{C_l\}}(d)\cdot \\
c_{\{C_l\},\left\lfloor \frac{b}{(K/2)^2}\right\rfloor\longmapsto \frac{K}{2}+u,\{A_j\}}(d) \lhd & \left\lfloor \frac{i}{(K/2)^2}\right\rfloor \neq \left\lfloor \frac{b}{(K/2)^2}\right\rfloor \rhd \emptyset\Bigg)\cdot \\
c_{\{A_j\},\left\lfloor \frac{b}{K/2}\right\rfloor_{|K/2}\longmapsto \frac{K}{2}+t,\{E_i\}}(d) \lhd & \left\lfloor \frac{i}{K/2}\right\rfloor \neq \left\lfloor \frac{b}{K/2}\right\rfloor \rhd \emptyset\Bigg)\cdot \\
c_{\{E_i\},b_{|K/2}\longmapsto 0,\{H_b\}}(d)\Bigg)\cdot\tau_I\Bigg(\partial_H\Big(H_h \parallel E_i \parallel A_j \parallel C_l\Big)\Bigg) & = \varnothing
\end{aligned} \tag{5.19}$$

However, the latter may account for expression (5.20), which actually shows the behaviour of the switching topology being modelled, where links from and to hosts are labelled as external communications, thus being considered as an open system. Likewise, in the previous subsection it has been shown that communications between hosts and edge switches are internal, whereas this case scenario may not consider hosts within the system being modelled, but just foreign items whose target is to interact with the system by means of a message d. Hence, such communications may not be included now into the model, thus the abstraction operator may not affect the links going out of the system.

$$\begin{gathered}
||_{i=0}^{\frac{K^2}{2}-1}||_{j=0}^{\frac{K^2}{2}-1}||_{l=0}^{\frac{K^2}{4}-1}\tau_I\Big(\partial_H\Big(E_i \parallel A_j \parallel C_l\Big)\Big) = \tau_I\Bigg(r_{\{E_i\},a_{|K/2}}(d)\cdot \\
\Bigg(\sum_{t=0}^{\frac{K}{2}-1} c_{\{E_i\},\frac{K}{2}+t\longmapsto\left\lfloor\frac{a}{K/2}\right\rfloor_{|K/2},\{A_j\}}(d)\cdot\Bigg(\sum_{u=0}^{\frac{K}{2}-1} c_{\{A_j\},\frac{K}{2}+u\longmapsto\left\lfloor\frac{a}{(K/2)^2}\right\rfloor,\{C_l\}}(d)\cdot \\
c_{\{C_l\},\left\lfloor\frac{b}{(K/2)^2}\right\rfloor\longmapsto\frac{K}{2}+u,\{A_j\}}(d)\lhd\left\lfloor\frac{i}{(K/2)^2}\right\rfloor\neq\left\lfloor\frac{b}{(K/2)^2}\right\rfloor\rhd\emptyset\Bigg)\cdot \\
c_{\{A_j\},\left\lfloor\frac{b}{K/2}\right\rfloor_{|K/2}\longmapsto\frac{K}{2}+t,\{E_i\}}(d)\lhd\left\lfloor\frac{i}{K/2}\right\rfloor\neq\left\lfloor\frac{b}{K/2}\right\rfloor\rhd\emptyset\Bigg)s_{\{E_i\},b_{|K/2}}(d)\Bigg)\cdot \\
\tau_I\Big(\partial_H\Big(E_i \parallel A_j \parallel C_l\Big)\Big) = r_{\{E_i\},a_{|K/2}}(d)\cdot s_{\{E_i\},b_{|K/2}}(d)\cdot\tau_I\Big(\partial_H\Big(E_i \parallel A_j \parallel C_l\Big)\Big)
\end{gathered} \tag{5.20}$$

On the other hand, expression (5.21) exhibits the external behaviour of the real system X, where a message d gets out of a source host and enters into the switching topology through a source switch i via its port p, which then forwards the message over the appropriate links to reach a destination switch i', which in turn delivers it to a destination host via its port p', thus completing the communication flow.

$$\tau_I\big(X\big) = r_{\{E_i\},p}\big(d\big)\cdot s_{\{E_{i'}\},p'}\big(d\big)\cdot\tau_I\big(X\big) \tag{5.21}$$

Therefore, it is clear that both recursive expressions are multiplied by the same factors, considering that the entry port into the system is $p = a_{|K/2}$ and the exit port out of the system is $p' = b_{|K/2}$. This fact makes them *rooted branching bisimilar*, because they both have the same string of actions, and furthermore, they both share the same branching structure, thus allowing the application of expression (5.22).

$$X \longleftrightarrow \tau_I\Big(\partial_H\big(E_i \parallel A_j \parallel C_l\big)\Big) \tag{5.22}$$

In conclusion, this is a sufficient condition to get a model verified, hence, the model proposed for a fat tree architecture has been duly verified.

5.4 Leaf and spine

After fat tree, let us go for leaf and spine. The study is going to be performed in a similar fashion as above, as they are both tree-like designs. However, switches belong to just 2 types, namely, lower layer switches called leaf switches (F_i), thus making them the end switches, and upper layer switches named spine switches (G_j). Each category is going to be enumerated from 0 on, starting from the left and going all the way to the right, where the last member of each kind is identified by the predecessor of the overall values stated in Table 4.2.

In this context, variable i identifies the leaf switches and j does it for the spine switches, where hosts are named as explained in a previous section. It is to be reminded that the main point in a leaf & spine topology is to achieve a full mesh connectivity among all leaf switches and all spine switches. Additionally, there is not a parameter fixing all design details, as parameter K does it in fat tree deployments.

Therefore, there are some degrees of freedom when choosing the number of leaf switches, as well as the number of spine switches, and their respective ports. Furthermore, the number of hosts per end switch depends on the design, although a generic number M is going to be considered herein, as explained above.

As per the *layout*, Figure 5.10 exhibits the models for each type, along with their corresponding upper and lower ports. It has been considered that variable t denotes both the number of lower layer switches and the overall number of ports in the upper layer switches, whereas u does it for both the number of upper layer switches and the number of uplink ports in the lower layer switches, hence building up the full mesh connectivity between both layers. Besides, the number of downlink ports in the lower layer switches is given by M, as stated before, which are identified from 0 to $M - 1$. This makes an overall amount of $t \times M$ hosts included in the whole topology.

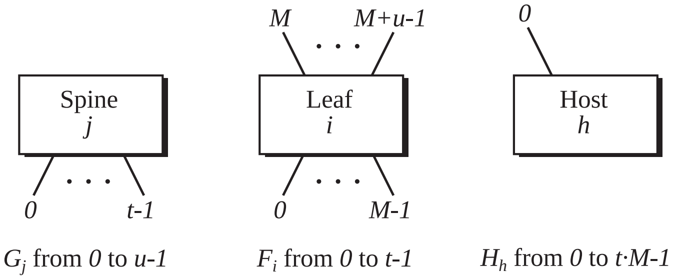


Figure 5.10: Model for switches and their ports in leaf and spine.

Moreover, Figure 5.11 depicts the whole structure for a leaf and spine deployment, with $t = 8$ leaf switches and $u = 4$ spine switches, where items on each layer are being sequentially identified from left to right, along with an oversubscription rate 1:1, where the full mesh topology between both layers may be observed.

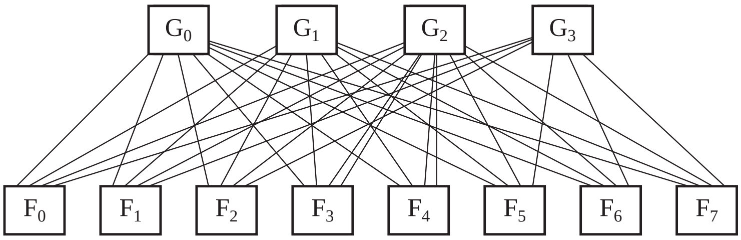


Figure 5.11: Leaf and spine device nomenclature.

5.4.1 Leaf switches

Basically, the way they work is analogous to the edge switches, but taking the M lower ports first, and then the $u-1$ upper ports, whilst switches go from 0 to $t-1$.

The arithmetic condition to verify whether a given leaf switch i is the destination switch of a VM is similar to that quoted for edge switches and it is given in (5.23). Likewise, the condition to verify that a source and a destination hosts are hanging on the same leaf switch is given in (5.24).

Besides, if a leaf switch happens to be the destination switch, the way to forward a VM through the proper port to the destination host is given by the expression (5.25).

Those expressions just take advantage of the port layout, where the lower half of ports (from 0 to $M-1$) look downwards to the hosts, whereas the upper half (M to $M+u-1$) look upwards to the upper layer switches, where all ports are identified by variable p, whilst upper ports are named by variable q.

$$int({}^{i}\!/_{M}) = int({}^{b}\!/_{M}) \longleftrightarrow \left\lfloor \frac{i}{M} \right\rfloor = \left\lfloor \frac{b}{M} \right\rfloor \tag{5.23}$$

$$int({}^{a}\!/_{M}) = int({}^{b}\!/_{M}) \longleftrightarrow \left\lfloor \frac{a}{M} \right\rfloor = \left\lfloor \frac{b}{M} \right\rfloor \tag{5.24}$$

$$b \;\; mod \;\; M \longleftrightarrow b \;\; modulo \;\; M \longleftrightarrow b_{|M} \tag{5.25}$$

With this all in mind, Figure 5.12 exposes the *flow chart* explaining the behaviour of an leaf switch i.

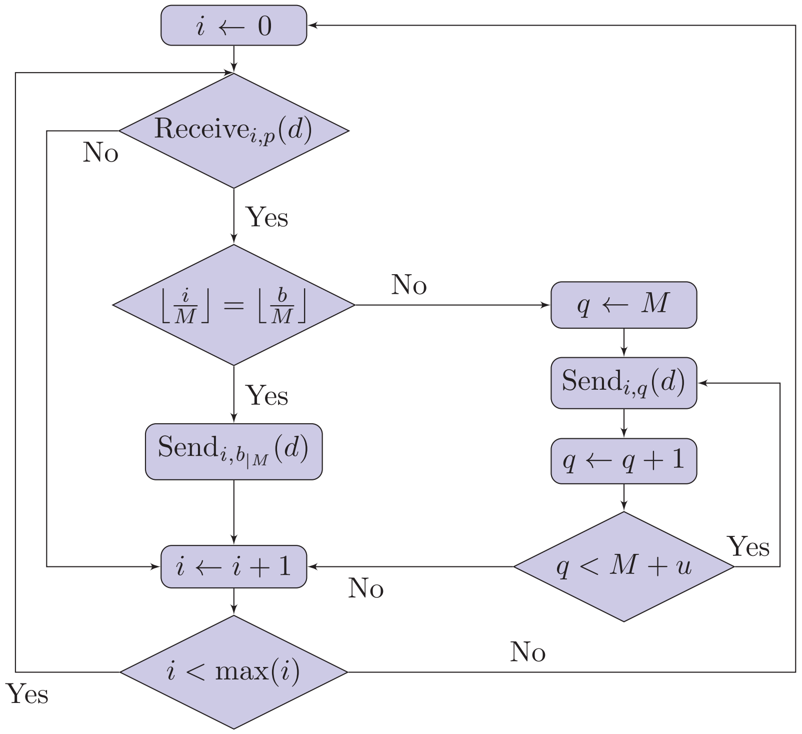


Figure 5.12: Flow chart for the behaviour of an leaf switch (leaf and spine).

Moreover, algorithm 5 (shown in the Appendix 10.5) exposes the *pseudocode model* regarding the behaviour of a given leaf switch i.

With regards to the *algebraic model*, the ACP expression representing the behaviour of a leaf switch is exhibited in (5.26). ACP does not take time into account, thus the initial conditional clause may be skipped, hence the first action may always be the reception of a message through any of its ports.

$$F_i = \sum_{i=0}^{t-1} \left(\sum_{p=0}^{M+u-1} \left(r_{F_i,p}(d) \cdot \left(s_{F_i,b_{|M}}(d) \lhd \left\lfloor \frac{i}{M} \right\rfloor = \left\lfloor \frac{b}{M} \right\rfloor \rhd \sum_{q=M}^{M+u-1} s_{F_i,q}(d) \right) \right) \right) \cdot F_i \tag{5.26}$$

5.4.2 Spine switches

Basically, the way they work is analogous to the core switches, where the condition to check whether a source host and a destination host are hanging on different leaf switches is given in (5.27). On the other hand, if a given port receives a VM in, then it is forwarded on to the port pointing to the leaf switch where the destination host is connected to, and such a port is given by (5.28).

Those expressions just take advantage of the port layout, where all ports look downwards (from 0 to $t-1$), all of them being identified by variable p. It occurs that each of those ports just connect to a single leaf switch, where each identifier match the port identifier from all spine switches. Moreover, full mesh is achieved among all leaf switches, as all of them connect to all spine switches, those going from 0 to $u-1$.

$$int(a/M) \neq int(b/M) \longleftrightarrow \left\lfloor \frac{a}{M} \right\rfloor \neq \left\lfloor \frac{b}{M} \right\rfloor \tag{5.27}$$

$$int(b/M) \longleftrightarrow \left\lfloor \frac{b}{M} \right\rfloor \tag{5.28}$$

With this all in mind, Figure 5.13 exhibits the *flow chart* showing the behaviour of a spine switch j.

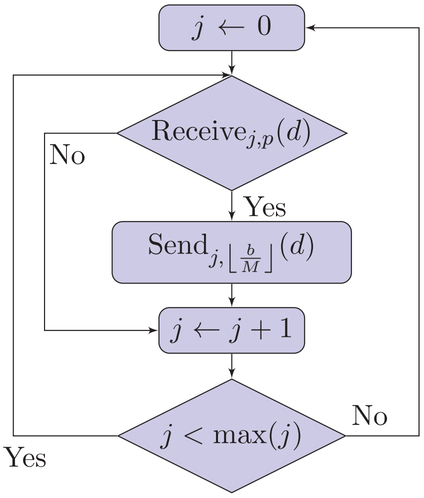


Figure 5.13: Flow chart for the behaviour of a spine switch (leaf and spine).

Besides, algorithm 6 (shown in the Appendix 10.6) exhibits the *pseudocode model* regarding the behaviour of a given spine switch j.

Concerning the *algebraic model*, the ACP expression representing how a spine switch behaves is depicted in (77), where the conditional clause in the receive function has been dropped as ACP does not consider the time.

$$G_j = \sum_{j=0}^{u-1} \left(\sum_{p=0}^{t-1} \left(r_{G_j,p}(d) \cdot s_{G_j, \lfloor \frac{b}{M} \rfloor}(d) \right) \right) \cdot G_j \tag{5.29}$$

5.4.3 Consolidating leaf and spine

Focusing on *redundant paths* within the model, and following the discussion undertaken in the previous section regarding redundant paths, it is to be noted that the path between H_h and F_i is analogous to that between hosts and edge switches, such as exposed in Table 5.5 for the case where source and destination hosts are hanging on the same end switch, thus being 1 hop away. Therefore, the arithmetic expressions to build up the list of devices and their ports for hosts being 1-hop away in leaf and spine may meet this requirement: $i = \lfloor \frac{a}{M} \rfloor = \lfloor \frac{b}{M} \rfloor$.

Table 5.5: List of devices and ports between 1-hop away hosts in leaf and spine.

Communication	**Direction**	**Expression**
Hosts and leaves	Uplink	$\{H_a\}, 0 \longmapsto a_{\vert M}, \{F_{\lfloor \frac{a}{M} \rfloor}\}$
Leaves and hosts	Downlink	$\{F_{\lfloor \frac{b}{M} \rfloor}\}, b_{\vert M} \longmapsto 0, \{H_b\}$

On the other hand, communication between F_i and G_j has to be full mesh between both layers, where the uplinks may be expressed as $\{F_i\}, M + j \longmapsto \lfloor \frac{a}{M} \rfloor, \{G_j\}$, and the downlinks may be seen as $\{G_j\}, \lfloor \frac{b}{M} \rfloor \longmapsto M + j, \{F_{i'}\}$. Hence, in this case, hosts are 2-hops away, as they are both connected to different end switches, which allow for the establishment of u available paths, as it is shown in Table 5.6, where the arithmetic expressions to build up the list of devices and their ports for hosts being 2-hop away in leaf and spine meet the following requirement: $\lfloor \frac{a}{M} \rfloor \neq \lfloor \frac{b}{M} \rfloor$.

Regarding leaf switches, it is to be reminded that $i = \lfloor \frac{a}{M} \rfloor$, whereas $i' = \lfloor \frac{b}{M} \rfloor$, which in this case result in $i \neq i'$. Furthermore, spine switches j go from 0 to $u - 1$, as there are u overall, where the uplink ports in the leaves arriving to those spines are identified by $M + j$, in a way that both are related for traffic going up and down.

Eventually, taking the discussion in the previous section regarding *verification*, the same reasoning may applied herein. In that context, the encapsulation operator is applied to communications between hosts and leaf switches, as shown in (5.30),

Table 5.6: List of devices and ports between 2-hops away hosts in leaf and spine.

Communication	Direction	Expression
Hosts and leaves	Uplink	$\{H_a\}, 0 \longmapsto a_{\vert M}, \{F_{\lfloor \frac{a}{M} \rfloor}\}$
Leaves and spines	Uplink	$\sum_{q=0}^{u-1} \Big(\{F_{\lfloor \frac{a}{M} \rfloor}\}, (M+q) \longmapsto$ $\longmapsto \left\lfloor \frac{a}{M} \right\rfloor, \{G_q\}\Big)$
Spines and leaves	Downlink	$\sum_{q=0}^{u-1} \Big(\{G_q\}, \left\lfloor \frac{b}{M} \right\rfloor \longmapsto$ $\longmapsto (M+q), \{F_{\lfloor \frac{b}{M} \rfloor}\}\Big)$
Leaves and hosts	Downlink	$\{F_{\lfloor \frac{b}{M} \rfloor}\}, b_{\vert M} \longmapsto 0, \{H_b\}$

whereas it is also applied to those between leaf and spine switches, as seen in (5.31). Moreover, it is to be noted that there is not direct interaction between hosts and spine switches, thus after applying the encapsulation operator, they go deadlock, thus getting discarded (5.32).

$$||_{h=0}^{t\times M-1} ||_{i=0}^{t-1} \partial_H(H_h \parallel F_i) = \Bigg(c_{\{H_a\},0\longmapsto a_{\vert M},\{F_i\}}(d) \cdot \Bigg(c_{\{F_i\},b_{\vert M}\longmapsto 0,\{H_b\}}(d)$$

$$\lhd \left\lfloor \frac{i}{M} \right\rfloor = \left\lfloor \frac{b}{M} \right\rfloor \rhd \sum_{q=M}^{u-1} s_{F_i,q}(d)\Bigg)\Bigg) \cdot (H_h \parallel F_i) \tag{5.30}$$

$$||_{i=0}^{t-1} ||_{j=0}^{u-1} \partial_H(F_i \parallel G_j) = \sum_{q=0}^{u-1} \Bigg(c_{\{F_i\},M+q\longmapsto \lfloor \frac{a}{M} \rfloor,\{G_j\}}(d) \cdot$$

$$c_{\{G_j\},\lfloor \frac{b}{M} \rfloor\longmapsto M+q,\{G_j\}}(d)\Bigg) \cdot (F_i \parallel G_j) \tag{5.31}$$

$$||_{h=0}^{t\times M-1} ||_{j=0}^{u-1} \partial_H(H_h \parallel G_j) = \delta \tag{5.32}$$

At this point, the abstraction operator is applied to both the whole set of switches and hosts. Expression (5.33) proves that the whole set becomes a closed system, as discussed in the previous section, whereas expression (5.34) states that the set involving only the switches is an open system, thus revealing its external behaviour.

$$||_{h=0}^{t\times M-1} ||_{i=0}^{t-1} ||_{j=0}^{u-1} \tau_I\Bigg(\partial_H\Big(H_h \parallel F_i \parallel G_j\Big)\Bigg) = \tau_I\Bigg(c_{\{H_a\},0\longmapsto a_{\vert M},\{F_i\}}(d) \cdot$$

$$\Bigg(\sum_{q=0}^{u-1} c_{\{F_i\},M+q\longmapsto \lfloor \frac{a}{M} \rfloor,\{G_j\}}(d) \cdot c_{\{G_j\},\lfloor \frac{b}{M} \rfloor\longmapsto M+q,\{F_i\}}(d)$$

$$\lhd \left\lfloor \frac{i}{M} \right\rfloor \neq \left\lfloor \frac{b}{M} \right\rfloor \rhd \emptyset\Bigg) \cdot c_{\{F_i\},b_{\vert M}\longmapsto 0,\{H_b\}}(d)\Bigg) \cdot \tau_I\Bigg(\partial_H\Big(H_h \parallel F_i \parallel G_j\Big)\Bigg) = \varnothing \tag{5.33}$$

$$
\begin{aligned}
||_{i=0}^{t-1}||_{j=0}^{u-1}\tau_I\Big(\partial_H\Big(F_i \parallel G_j\Big)\Big) = \tau_I\Bigg(r_{\{F_i\},a_{|M}}(d)\cdot \\
\Bigg(\sum_{q=0}^{u-1} c_{\{F_i\},M+q\longmapsto\lfloor\frac{a}{M}\rfloor,\{G_j\}}(d)\cdot c_{\{G_j\},\lfloor\frac{b}{M}\rfloor\longmapsto M+q,\{F_i\}}(d) \\
\lhd \left\lfloor\frac{i}{M}\right\rfloor \neq \left\lfloor\frac{b}{M}\right\rfloor \rhd \emptyset\Bigg)\cdot s_{\{F_i\},b_{|M}}(d)\Bigg)\cdot\tau_I\Big(\partial_H\Big(F_i \parallel G_j\Big)\Big) \\
= r_{\{F_i\},a_{|M}}(d)\cdot s_{\{F_i\},b_{|M}}(d)\cdot\tau_I\Big(\partial_H\Big(F_i \parallel G_j\Big)\Big) \qquad (5.34)
\end{aligned}
$$

On the other hand, expression (5.35) exposes the external behaviour of the real system X, such that a message d comes from a source host into the switching infrastructure through a particular source switch i via its port p, then it gets forwarded towards the appropriate destination switch i' throughout the switching topology, and eventually, it goes off along port p' to a given destination host.

$$\tau_I\big(X\big) = r_{\{F_i\},p}\big(d\big)\cdot s_{\{F_{i'}\},p'}\big(d\big)\cdot\tau_I\big(X\big) \qquad (5.35)$$

Hence, it seems obvious that both recursive expressions are multiplied by the same factors, taking into consideration that the incoming port into the system is $p = a_{|M}$ and outgoing port off the system is $p' = b_{|M}$. This means that they may both be considered as being *rooted branching bisimilar*, as they have the same string of actions, and also, they share the same branching structure, hence permitting the application of expression (5.36).

$$X \longleftrightarrow \tau_I\Big(\partial_H\big(F_i \parallel G_j\big)\Big) \qquad (5.36)$$

In conclusion, this is a sufficient condition to get a model verified, hence, the model proposed for a leaf and spine architecture has been duly verified.

5.5 N-hypercube

At this point, it is time to talk about N-hypercube. As said before, it is a graph-like design, making the modelling easier, as all switches are considered to be end switches, with hosts connected to them. Therefore, the only items to be modelled are hosts (H_h) and end switches, also known as nodes or vertices (V_i). Additionally, the overall figures for this topology are quoted in Table 4.3.

First of all, an initial node is chosen, identified by 0 in decimal format, or otherwise in binary format, with as many bits 0 as the dimensions of the N-hypercube. From that node, if a given node has its j-th bit set to 0, it may mean that it shares the same value for that dimension j as the initial node, whereas if that j-th bit is set to 1, its value for that dimension j may differ. This way, each node will get a unique identifier of N bits, called *NodeID*.

With respect to the hosts, it is assumed that there is a number of M hosts connected to each node, where such a number is a power of 2, for simplicity purposes. That accounts for $M = 2^W$ hosts per node, where each host gets an identifier called *HostID* by concatenating a *NodeID* with a binary string going from 0 to $M - 1$, leading to a unique identifier of $N + W$ bits. Overall, the total amount of hosts may account for 2^{N+W}.

Eventually, if a VM is received in a port of a given node i, it is to be checked whether that node is the destination node, expressed by $\left\lfloor \frac{b}{M} \right\rfloor$, or likewise, $\left\lfloor \frac{b}{2^W} \right\rfloor$. If that is the case, then such a VM may be forwarded on straight to the destination host, or otherwise, the mismatching dimensions between both nodes may be search for, and in turn, that VM may be sent through all of them, thus covering all redundant paths.

The values for every dimension j may be compared by operating with both nodes identifiers in decimal or in binary format. The former makes use of arithmetic operations, such as applying the operator congruence modulo 2 to integer divisions, according to expression (5.37), whereas the latter does it with the logical *XOR* operator, where matching bits result in 0 and mismatching bits goes to 1, as shown in expression (5.38), where the corresponding truth table may be seen.

$$\left\lfloor \frac{i}{2^j} \right\rfloor_{|2} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2} \quad \text{for } j = \{0 \cdots N-1\} \tag{5.37}$$

$$\begin{cases} 0 \text{ XOR } 0 & \rightarrow & 0 \\ 0 \text{ XOR } 1 & \rightarrow & 1 \\ 1 \text{ XOR } 0 & \rightarrow & 1 \\ 1 \text{ XOR } 1 & \rightarrow & 0 \end{cases} \tag{5.38}$$

As per the *layout*, regarding the ports of each node, there is a total amount of $N + M$ ports. On the one hand, the first M ports are connecting to its hosts and are identified the same way as its hosts, this is, from left to right, starting by 0 and finishing by $M-1$. On the other hand, the last N ports are identified by the different dimensions where its neighbour nodes are located, such that dimension 0 is reached through port M, dimension 1 is attached to port $M+1$, all the way to dimension $N-1$, being attached to port $M+N-1$. Figure 5.14 exhibits the port identifiers of a given switch.

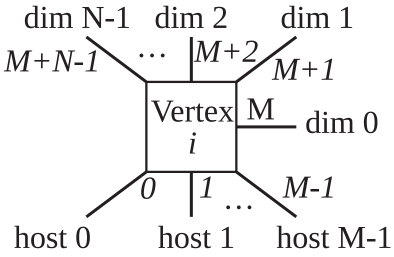


Figure 5.14: Model for switches and their ports in N-hypercube.

Otherwise, with respect to the node setup, Figure 5.15 depicts the node setup in binary format, whereas Figure 5.16 exposes its decimal counterpart.

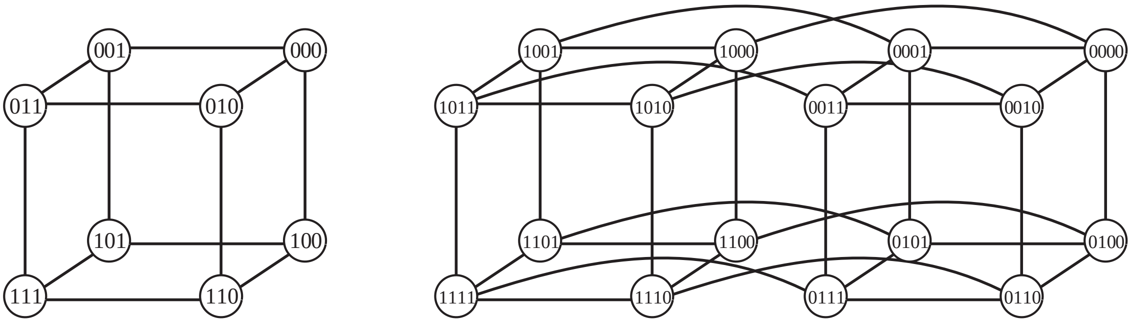


Figure 5.15: Node nomenclature using N binary values: 3 (left) and 4 (right).

As an example, Figure 5.17 present an alternative way to setup the nodes based on the decimal identifier, as opposed that based on the the binary identifiers. Basically, a node 0 is located right in the origin of the Cartesian coordinates of N dimensions, whereas the identifier of its neighbouring nodes in a certain dimension are calculated by adding up its own identifier and 2 to the power of the predecessor of that dimension, where that procedure is repeated until all nodes are registered.

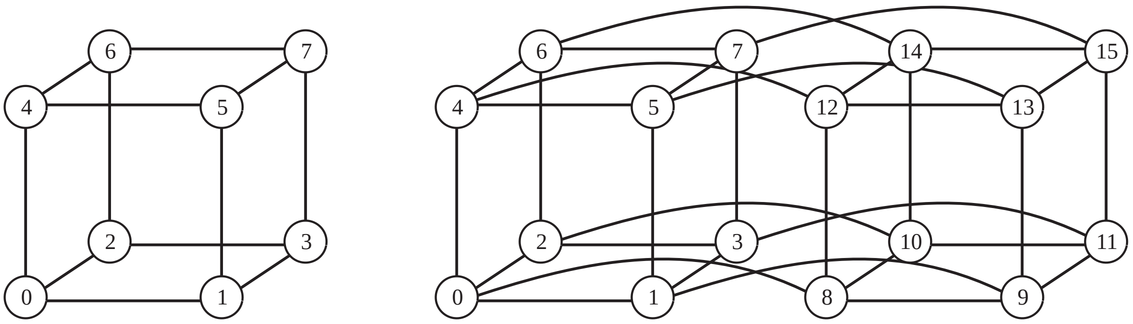


Figure 5.16: Node nomenclature translated into decimal values: 3 (left) and 4 (right).

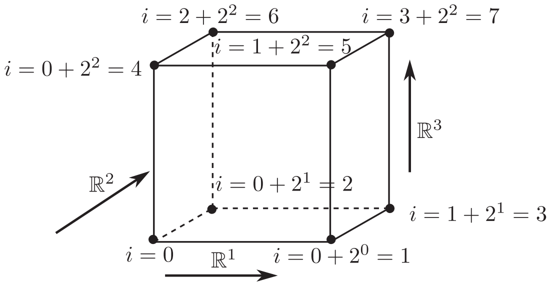


Figure 5.17: Model the interconnected switches i for 3-hypercube.

Taking all that into consideration, Figure 5.18 exhibits the *flow chart* showing the behaviour of a switch i.

Furthermore, the *pseudocode model* related to the behaviour of a given switch i is shown in algorithm 7 (shown in the Appendix 10.7), where the mismatching dimensions are spotted by means of arithmetic operations.

Alternatively, Table 5.7 presents the code snippet for finding out the mismatching bits by means of logical operations, which may be compared to the *else* clause in the previous algorithm.

Respecting to the *algebraic model*, the ACP expression to represent how a given switch behaves is exhibited in (5.39), where the conditional clause in the receive function has been dropped as ACP does not consider the time.

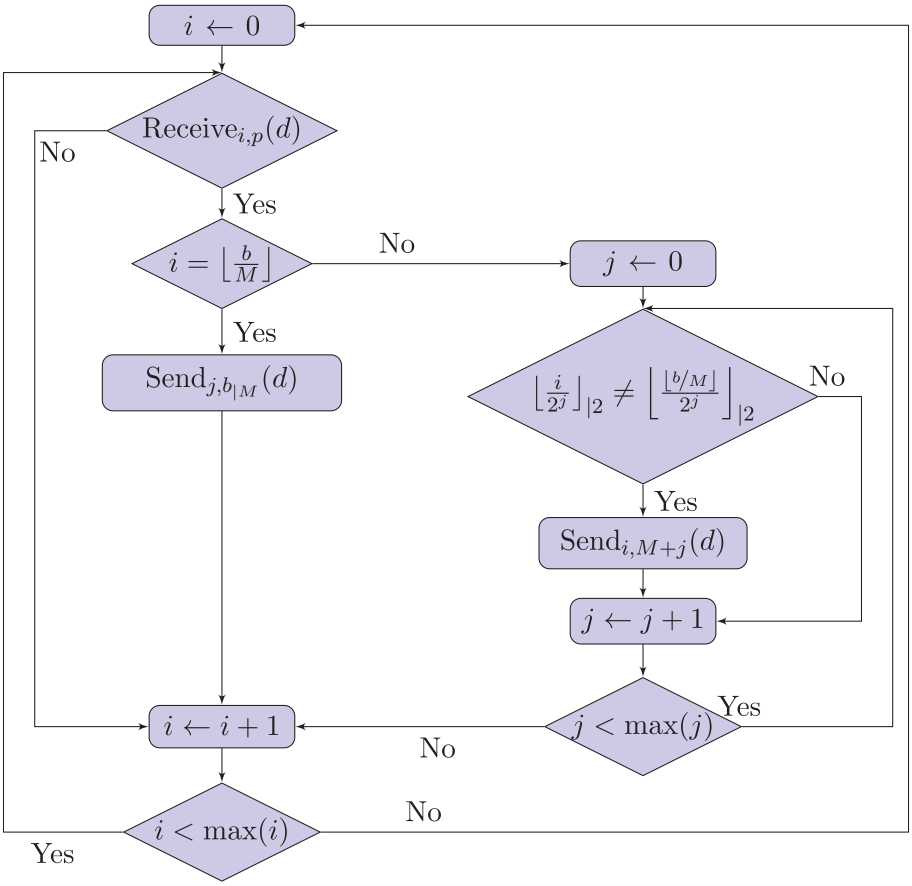

$i \leftarrow 0$
$\text{Receive}_{i,p}(d)$
No
Yes
$i = \lfloor \frac{b}{M} \rfloor$
No
$j \leftarrow 0$
Yes
$\text{Send}_{j,b_{|M}}(d)$
$\left\lfloor \frac{i}{2^j} \right\rfloor_{|2} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2}$
No
Yes
$\text{Send}_{i,M+j}(d)$
$j \leftarrow j + 1$
$j < \max(j)$
Yes
$i \leftarrow i + 1$
No
$i < \max(i)$
No
Yes

Figure 5.18: Flow chart for the behaviour of a given switch (N-hypercube).

Table 5.7: Searching mismatching bits by arithmetic and logical means.

Mode	**Code snippet**

Arithmetic	else
	for (j = 0; j < N; j++)
	if ((int(i/2^{j})) mod 2) ≠ ((int(int(b/M)/2^{j})) mod 2)
	send $_{\text{NODE\{i\},PORT\{M+j\}}}$(a,h,VM(u));

Logical	else
	logicalXOR = DecimalToBinary(i) XOR DecimalToBinary(int(b/M));
	for (j = 0; j < N; j++)
	if (logicalXOR[j] = 1)
	send $_{\text{NODE\{i\},PORT\{M+j\}}}$(a,h,VM(u));

$$V_i = \sum_{i=0}^{2^N-1} \left(\sum_{p=0}^{M+N-1} \left(r_{V_i,p}(d) \cdot \left(s_{V_i,b_{|M}}(d) \lhd i = \left\lfloor \frac{b}{M} \right\rfloor \rhd \right.\right.\right.$$
$$\left.\left.\left.(\sum_{j=0}^{N-1} s_{V_i,M+j}(d) \lhd \left\lfloor \frac{i}{2^j} \right\rfloor_{|2} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2} \rhd \emptyset) \right) \right) \right) \cdot V_i \tag{5.39}$$

Focusing on *redundant paths* within the model, the points discussed in the previous sections on this issue prevail, as well as the establishment of the source switch by means of $\lfloor \frac{a}{M} \rfloor$ and the destination switch by means of $\lfloor \frac{b}{M} \rfloor$. However, some redundant paths may be available between both end switches, related to the distance between them. Specifically, if variable f is the distance between those end switches, then there may be up to $f!$ redundant paths.

Therefore, in case both end switches are the same item, the complete path is presented in Table 5.8, such that the arithmetic expressions to build up the list of devices and their ports for hosts being 1-hop away in N-hypercube fulfill this requirement: $\lfloor \frac{a}{M} \rfloor = \lfloor \frac{b}{M} \rfloor$.

Table 5.8: List of devices and ports between 1-hop away hosts in N-hypercube.

Communication	Direction	Expression
Hosts and end switches	Uplink	$\{H_a\}, 0 \longmapsto a_{\vert M}, \{V_{\lfloor \frac{a}{M} \rfloor}\}$
End switches and hosts	Downlink	$\{V_{\lfloor \frac{b}{M} \rfloor}\}, b_{\vert M} \longmapsto 0, \{H_b\}$

On the other hand, a set of movements are necessary to go from the source switch to the destination switch, which may be combined in any order to get there in $f!$ different ways, resulting in as many different redundant paths containing all those movements in a unique order. It is to be noted that each movement involves a certain dimension and a certain direction, where the former involves one of the mismatched dimensions between both end switches and the latter points out the way to get to the destination one.

Hence, it may be considered that variable *dimension* is given by $x \subset j$, where $j = \{0 \cdots N-1\}$, as shown in (5.40), whilst variable *direction* attached to dimension x is calculated as in (5.41). Furthermore, it is to be said that each *movement x* available is a combination of both variables, according to expression (5.42).

Eventually, the expression to get from a current node to the *next node* is exposed in (5.43). It is to be noted that such an expression may reveal the components of a given path by applying all movements in a sequential manner, whereas all redundant paths may be obtained by working out with all combinations of the available movements. It is to be remarked that there may be up to f movements, which accounts for the distance between source node and destination node, thus making up for $f!$ redundant paths with f switch hops.

$$\text{Dimension}[x] = x, \qquad \text{for mismatched dimensions } x \subset \{0 \cdots N-1\} \tag{5.40}$$

$$\text{Direction}[x] = (-1)^g, \text{ where } g = \left\lfloor \frac{\lfloor a/M \rfloor}{2^x} \right\rfloor_{|2} \qquad \text{for } x \subset \{0 \cdots N-1\} \tag{5.41}$$

$$\text{Movement}[x] = \text{Direction}[x] \times 2^{\text{Dimension}[x]} \qquad \text{for } x \subset \{0 \cdots N-1\} \tag{5.42}$$

$$\text{nextNode} = \text{oldNode} + \text{Movement}[x] \qquad \text{for } x \subset \{0 \cdots N-1\} \tag{5.43}$$

Therefore, Table 5.9 is proposed to achieve a list of movements to go from a given source switch to a given destination switch (supposing $\lfloor \frac{a}{M} \rfloor \neq \lfloor \frac{b}{M} \rfloor$), where the details of each redundant path is not going to be quoted, but just the necessary movements to get there. In order to attain the detailed list of devices to go across each of the redundant paths available, the different permutations of such movements may be worked out, where each permutation results in a different combination of the intermediate nodes being traversed in each of those paths when applying all the movements according to the order established by such a permutation.

This is exposed by the function *permutations(f,c)*, where f is related to the number of movements and c does it to the identifier of each combination of those movements. Additionally, the ports involved may be easily spotted by following the mismatching dimension between every pair of nodes within each redundant path.

Eventually, a *verification* of the model proposed may be carried out. In that sense, by applying the principles discussed in the previous sections, and taking into consideration that all switches are end switches, it may seem clear that the encapsulation operator only allows internal communications to take place, thus bringing the atomic internal actions to deadlock, whereas atomic external actions still prevail. In this case, it may appear that all communications within neighbouring nodes belonging

Table 5.9: Permutations from source switch to destination switch in N-hypercube.

Code snippet

```
$f = 0$
for (j = 0; j < N; j++)
 if ((int(int(a/M)/$2^{j}$)) mod 2) ≠ ((int(int(b/M)/$2^{j}$)) mod 2) {
  dimension = j;
  direction = $(-1)^{(\mathrm{int(int(a/M)/}2^{j}\mathrm{))\ mod\ 2}}$;
  movement[f] = direction × $2^{\mathrm{dimension}}$;
  $f = f + 1$;
 }
print(”Movements: ” + f + ”, with ” + f! + ” permutations available.”);
for (c = 0; c < f!; c++){
 y = permutations(f,c);
 for (k = 0; k < length(y); k++)
  $V_{\mathrm{next}} = V_{\mathrm{current}} + \mathrm{movement}[k]$;
}
```

to the hypercube structure are internal, hence they prevail, whereas communications with non-neighbouring nodes do not take place, thus the related atomic actions may go deadlock.

As per the expression for the application of the encapsulation operator, it is to be said that paths between any pair of hosts in tree-like topologies are clearly stated because of its hierarchical design. However, it does not happen with graph-like designs, hence permutations may be performed including all necessary moves to get from the source node to the destination node. Therefore, there may be not a unique expression, but it may depend on the identifiers of source and destination nodes.

Furthermore, the abstraction operator masks internal communications, hence revealing the external behaviour of the model. In this case, all communications among nodes get masked, whilst atomic actions related to external hosts are still available, thus being ready to receive messages from them, as well as to send messages to them. In summary, the external behaviour of the model proposed is shown in (5.44) if end hosts are taken into consideration, which leads to ∅ for the external behaviour of this model, as that is a closed system, whereas in (5.45) end hosts are not taken into account, thus leading to an expression revealing the external behaviour of that model, as that is an open system. Anyway, the source node may be calculated with $\lfloor {}^{a}/_{M} \rfloor$ and destination node may be done with $\lfloor {}^{b}/_{M} \rfloor$, due to the layout described above.

$$\Big|\Big|_{h=0}^{2^{N+W}-1} \Big|\Big|_{i=0}^{2^N-1} \tau_I\Big(\partial_H\Big(H_h \parallel V_i\Big)\Big) = \tau_I\Big(c_{\{H_a\},0 \longmapsto a_{|M},\{V_i\}}(d)\cdot \Big(\text{permutations(f,c)} \lhd \left\lfloor \frac{i}{M} \right\rfloor \neq \left\lfloor \frac{b}{M} \right\rfloor \rhd \emptyset\Big) \cdot c_{\{V_i\},b_{|M} \longmapsto 0,\{H_b\}}(d)\Big) \cdot (H_h \parallel V_i) = \varnothing \tag{5.44}$$

$$\Big|\Big|_{i=0}^{2^N-1} \tau_I\Big(\partial_H\big(V_i\big)\Big) = r_{\{V_{\lfloor \frac{a}{M} \rfloor}\},a_{|M}}(d) \cdot s_{\{V_{\lfloor \frac{b}{M} \rfloor}\},b_{|M}}(d) \cdot \tau_I\Big(\partial_H\big(V_i\big)\Big) \tag{5.45}$$

Otherwise, expression (5.46) exposes the external behaviour of the real system X.

$$\tau_I\big(X\big) = r_{\{V_i\},p}\big(d\big) \cdot s_{\{V_{i'}\},p'}\big(d\big) \cdot \tau_I\big(X\big) \tag{5.46}$$

Therefore, it appears to be obvious that both recursive expressions are multiplied by the same factors, considering that the incoming port into the system is $p = a_{|M}$ and outgoing port out of the system is $p' = b_{|M}$. This means that they may be considered as *rooted branching bisimilar*, as they both have the same string of actions, and also, they both share the same branching structure, hence permitting the application of expression (5.47).

$$X \longleftrightarrow \tau_I\Big(\partial_H\big(V_i\big)\Big) \tag{5.47}$$

In conclusion, this is a sufficient condition to get a model verified, hence, the model proposed for an N-hypercube architecture has been duly verified.

5.6 Folded N-hypercube

At this stage, it is time to talk about folded N-hypercube. As said before, it is an extension of N-hypercube, where extra links are available over every pair of opposite nodes, whereas the rest of characteristics mentioned for N-hypercube remain the same, such as the number of nodes and faces. It is to be reminded that Table 4.4 specifies relevant data regarding this topology.

Focusing on the role of the link between opposite nodes, it permits to reach much faster the farthest nodes, thus reducing the diameter of the switching network to only $\lceil {}^{N}/_{2} \rceil$. Therefore, when dealing with mismatching dimensions, an extra verification may be done to check whether the opposite link provides a shorter path over an ordinary path, or the other way around, or even a tie is reached.

Hence, three different scenarios may arise, such as the *regular path* (the one not using any opposite node to go from source to destination) is shorter than the one using the *opposite path* (the one using any opposite node), or quite the contrary, or otherwise, both alternatives account for the same length. The proper decision to be made is to always choose the shortest path, and if both paths are just as long, then both alternatives may be used.

Regarding the opposite node of a given node i, it is worth reminding that this may be calculated in *binary form* by applying the $1's$ complement to i, meaning that all its bits will be swapped, such that all $0s$ will become $1s$, and otherwise, all $1s$ will turn into $0s$. Still, the opposite node may be calculated in *decimal form*, by applying the expression $2^N - 1 - i$ to the current node i.

When dealing with opposite nodes, an extra unit of length must be added up to account for that extra link going from any node to its opposite node. This means that if the distance through neighbouring nodes is the value of variable *distance*, then the distance through an opposite node is $N + 1 - distance$.

Therefore, the first part is to find out which path is the shortest one, this is, the one through only regular neighbours, or otherwise, the one through an opposite link along with regular neighbours, or even if they are both just as long, and afterwards, take that path, or equal cost redundant paths, to get from source to destination. It is to be remarked that regular paths make use of all mismatching dimensions, whereas opposite paths do use of all matching dimensions along with the opposite link.

As per the *layout*, it is to be noted that the extra link directed to the opposite node is identified as port $M + N$, this is, the last one, accounting for a total amount of $M + N + 1$ ports. Specifically, the first M ports go to hosts according to the hosts identification scheme exposed in the previous section, then, the following N ports go to other nodes according to the different dimensions, and the last one goes to the opposite node. Regarding the hosts, there are no changes with respect to the hypercube design, resulting in $M = 2^W$ per node, with a total amount of 2^{N+W}. Putting it all together, Figure 5.19 exhibits the port identifiers of a given switch.

Likewise, Figure 5.20 exposes the node setup in binary format, whilst Figure 5.21 does it in decimal format.

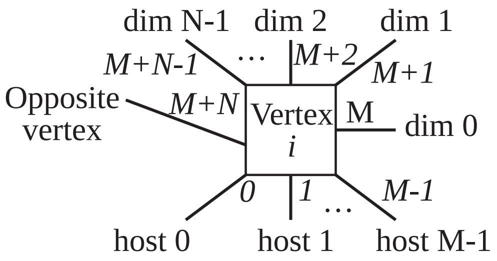


Figure 5.19: Model for switches, along with its ports (folded N-hypercube).

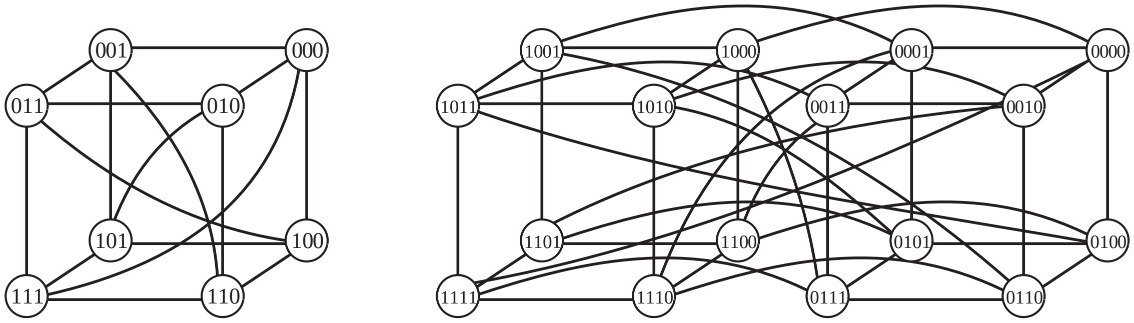


Figure 5.20: Node nomenclature employing N binary values: 3 (left) and 4 (right).

Taking all that into consideration, Figure 5.22 depicts the *flow chart* exhibiting the behaviour of a switch i.

Furthermore, the *pseudocode model* referred to the behaviour of a given switch i is shown in algorithm 8 (shown in the Appendix 10.8), where the mismatching and matching dimensions are found out by means of arithmetic operations.

Alternatively, Table 5.10 presents the code snippet for looking for the shortest distance and spotting the mismatching and matching bits by means of logical operations, which happens in the *else* clause in the previous algorithm.

With regards to the *algebraic model*, the ACP expression to represent the behaviour of a switch in presented in (5.50), where two auxiliary functions called *neighboursPath* and *oppositePath* have been defined in order to measure the distance in switches between the current node i and the destination node, which are shown in expressions (5.48) and (5.49).

In both functions, the initial value of the variable for distance, called *dist*, is first initialized ($\sim$*dist*), and in turn, the mismatching dimensions are counted up (*dist*++)

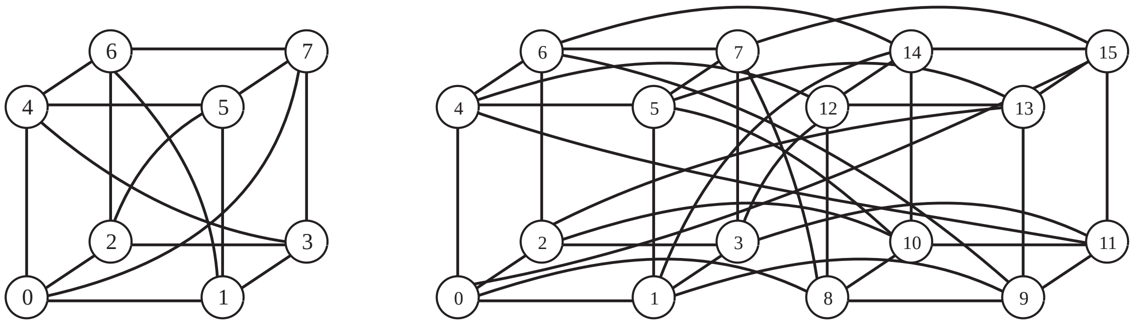


Figure 5.21: Node nomenclature converted into decimal values: 3 (left) and 4 (right).

Table 5.10: Searching the shortest path and its next hops by using of logical means.

Mode	Code snippet

Logical	else
	distance = 0;
	logicalXOR = DecimalToBinary(i) XOR DecimalToBinary(int(b/M));
	for (j = 0; j < N; j++)
	if (logicalXOR[j] = 1){
	index[distance] = j;
	distance = distance + 1;
	}
	if (distance $\leq$ N + 1 - distance){
	x = 0;
	for(y=0; y < N; y++)
	if(index[x] == y){
	send $_{\text{NODE\{i\},PORT\{M+y\}}}$(a,h,VM(u));
	x = x + 1;
	}
	}
	if (distance $\geq$ N + 1 - distance){
	x = 0;
	for(y=0; y < N; y++){
	if(index[x] $\neq$ y)
	send $_{\text{NODE\{i\},PORT\{M+y\}}}$(a,h,VM(u));
	else
	x = x + 1;
	}
	send $_{\text{NODE\{i\},PORT\{M+N\}}}$(a,h,VM(u));
	}

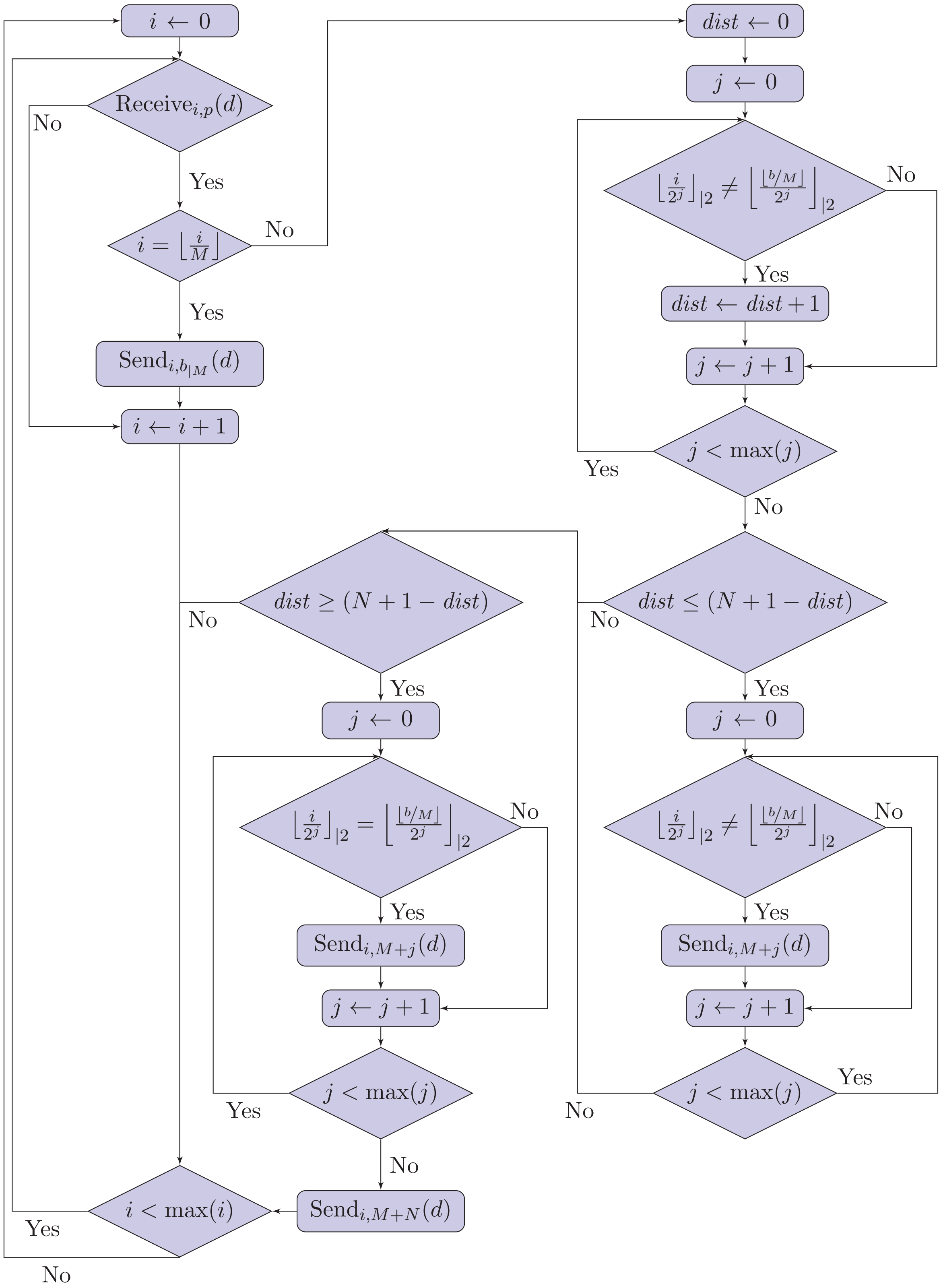

$i \leftarrow 0$
$\text{Receive}_{i,p}(d)$
No
Yes
$i = \left\lfloor \frac{i}{M} \right\rfloor$
No
Yes
$\text{Send}_{i,b_{|M}}(d)$
$i \leftarrow i+1$
$dist \leftarrow 0$
$j \leftarrow 0$
$\left\lfloor \frac{i}{2^j} \right\rfloor_{|2} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2}$
No
Yes
$dist \leftarrow dist + 1$
$j \leftarrow j+1$
$j < \max(j)$
Yes
No
$dist \geq (N+1-dist)$
No
Yes
$dist \leq (N+1-dist)$
No
Yes
$j \leftarrow 0$
$\left\lfloor \frac{i}{2^j} \right\rfloor_{|2} = \left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2}$
No
Yes
$\text{Send}_{i,M+j}(d)$
$j \leftarrow j+1$
$j < \max(j)$
Yes
No
$j \leftarrow 0$
$\left\lfloor \frac{i}{2^j} \right\rfloor_{|2} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2}$
No
Yes
$\text{Send}_{i,M+j}(d)$
$j \leftarrow j+1$
$j < \max(j)$
Yes
No
$\text{Send}_{i,M+N}(d)$
$i < \max(i)$
Yes
No

Figure 5.22: Flow chart for the behaviour of a given switch (folded N-hypercube).

in the neighboursPath, whilst in the oppositePath, the matching dimensions are also counted up, bearing in mind that an additional unit must be added up to take the opposite link into account.

The results of both functions are being applied in the V_i model in order to select the shortest path, and in turn, to apply the proper operations in order to send the message through the appropriate ports. Furthermore, the conditional clause in the receive function has been dropped as ACP does not consider the time.

$$neighboursPath = \big({\sim}dist\big) \cdot \sum_{j=0}^{N-1} \Bigg(\big(dist{+}{+}\big) \lhd \left\lfloor \frac{i}{2^j} \right\rfloor_{|2} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2} \rhd \emptyset \Bigg) \tag{5.48}$$

$$oppositePath = \big({\sim}dist\big) \cdot \sum_{j=0}^{N-1} \Bigg(\big(dist{+}{+}\big) \lhd \left\lfloor \frac{i}{2^j} \right\rfloor_{|2} = \left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2} \rhd \emptyset \Bigg) \cdot \big(dist{+}{+}\big) \tag{5.49}$$

$$\begin{aligned} V_i = \sum_{i=0}^{2^N-1} \Bigg(\sum_{p=0}^{M+N} \Bigg(& r_{V_i,p}(d) \cdot \Bigg(s_{V_i,b_{|M}}(d) \lhd i = \left\lfloor \frac{b}{M} \right\rfloor \rhd \\ & \Bigg(\Bigg(\Bigg(\sum_{j=0}^{N-1} s_{V_i,M+j}(d) \lhd \left\lfloor \frac{i}{2^j} \right\rfloor_{|2} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2} \rhd \emptyset \Bigg) \\ & \lhd neighboursPath <= oppositePath \rhd \emptyset \Bigg) \\ & \cdot \Bigg(\Bigg(\Bigg(\sum_{j=0}^{N-1} s_{V_i,M+j}(d) \lhd \left\lfloor \frac{i}{2^j} \right\rfloor_{|2} = \left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2} \rhd \emptyset \Bigg) + s_{V_i,M+N}(d) \Bigg) \\ & \lhd neighboursPath >= oppositePath \rhd \emptyset \Bigg) \Bigg) \Bigg) \Bigg) \Bigg) \Bigg) \cdot V_i \end{aligned} \tag{5.50}$$

Focusing on *redundant paths* within the model, the discussion for N-hypercube regarding the redundant paths may apply in this case, even though some modifications may be done in order to portray the selection between the only-neighbours path or the opposite path.

First of all, Table 5.11 shows the arithmetic expressions to build up the list of devices and their ports for hosts being 1-hop away in folded N-hypercube, resulting in the path to be carried out to interconnect two hosts hanging on the same switch, which is right the same as in the N-hypercube, this is, the case where $\lfloor \frac{a}{M} \rfloor = \lfloor \frac{b}{M} \rfloor$.

On the other hand, hosts hanging on different switches take part of the cases where $\lfloor \frac{a}{M} \rfloor \neq \lfloor \frac{b}{M} \rfloor$. In such instances, the terms explained for this type of situation in the

Table 5.11: List of devices and ports between 1-hop away hosts in folded N-hyp.

Communication	Direction	Expression
Hosts and end switches	Uplink	$\{H_a\}, 0 \longmapsto a_{\|M}, \{V_{\lfloor \frac{a}{M} \rfloor}\}$
End switches and hosts	Downlink	$\{V_{\lfloor \frac{b}{M} \rfloor}\}, b_{\|M} \longmapsto 0, \{H_b\}$

N-hypercube applies, but adapting them to the specific features explained so far for this topology. In this sense, Table 5.12 exhibits the necessary movements to be carried out to get to the destination node through the shortest path.

Eventually, it is to be noted that all switches in this topology are end switches, therefore, the *verification* procedure explained for the N-hypercube applies in this case, as the extra links added up to this model are just internal to the infrastructure. Therefore, the external behaviour of a folded N-hypercube is the same as the one exposed for N-hypercube, thus the model proposed gets duly verified.

5.7 Full mesh

This topology may be seen as an N-simplex structure, which accounts for $N+1$ nodes, where each one has N paths to the other switches. In order not to flood the topology with an excess of redundant messages, only the source node $\lfloor \frac{a}{M} \rfloor$ sends messages to all nodes, whereas the rest just forward messages on to the destination node $\lfloor \frac{b}{M} \rfloor$. Moreover, it is to be noted that there are $(N+1) \times M$ hosts overall.

As per the *layout*, regarding the port identifiers for each switch, it is to be reminded that the first M ports are dedicated to hosts (going from 0 to $M-1$), whilst there are N uplink ports being in use. In order to achieve easier expressions, those uplink ports are numerated from M to $M+N$, according to the expression $M+i$, where i is the identifier of the switch on the other end of each link, going from 0 to N. Hence, the port related to its own i value is skipped out, as it would do a loopback link.

Figure 5.23 depicts the uplink ports of a single switch i, whereas Figure 5.24 presents the nomenclature of switches by selecting one of them as a reference, called 0, and quoting the real dimension where each switch is moved from such a reference.

Table 5.12: Permutations from source switch to destination switch in folded N-hyp.

Code snippet

```
f = 0;
distance = 0;
 for (j = 0; j < N; j++)
  if ((int(int(a/M)/$2^j$)) mod 2) $\neq$ ((int(int(b/M)/$2^j$)) mod 2)
   distance = distance + 1;
if (distance $\leq$ (N + 1 - distance)){
 for (j = 0; j < N; j++)
  if ((int(int(a/M)/$2^j$)) mod 2) $\neq$ ((int(int(b/M)/$2^j$)) mod 2){
   dimension = j;
   direction = $(-1)^{(\mathrm{int}(\mathrm{int}(a/M)/2^j)) \bmod 2}$;
   movement[f] = direction $\times$ $2^{\mathrm{dimension}}$;
   $f = f + 1$;
  }
}
if (distance $\geq$ (N + 1 - distance)){
 for (j = 0; j < N; j++)
  if ((int(int(a/M)/$2^j$)) mod 2) = ((int(int(b/M)/$2^j$)) mod 2){
   dimension = j;
   direction = $(-1)^{(\mathrm{int}(\mathrm{int}(a/M)/2^j)) \bmod 2}$;
   movement[f] = direction $\times$ $2^{\mathrm{dimension}}$;
   $f = f + 1$;
  }
  dimension = $2^N$ - 1 - a;
  direction = $(-1)^{(\mathrm{int}(\mathrm{int}(a/M))/(N/2))}$;
  movement[f] = direction $\times$ $2^{\mathrm{dimension}}$;
  $f = f + 1$;
}
print("Movements: " + f + ", with " + f! + " permutations available.");
for (c = 0; c < f!; c++){
 y = permutations(f,c);
 for (k = 0; k < y; k++)
  $V_{\mathrm{next}} = V_{\mathrm{current}}$ + movement[$k$];
}
```

With that in mind, Figure 5.25 depicts the *flow chart* exhibiting the behaviour of a switch i.

Besides, the *pseudocode model* related to the behaviour of a given switch i is shown in algorithm 9 (shown in the Appendix 10.9).

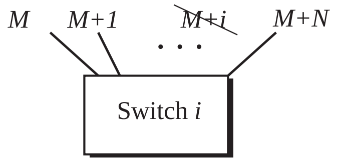


Figure 5.23: Model for the uplink ports in a full mesh switch (N-simplex).

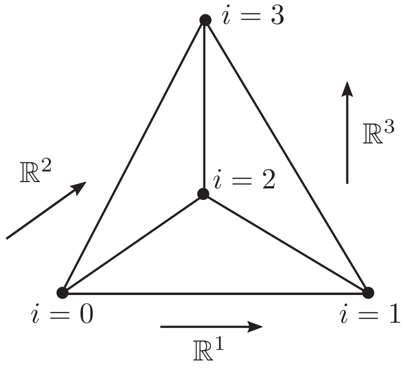


Figure 5.24: Model for the interconnected switches in a 3-simplex.

Concerning the *algebraic model*, the ACP expression to represent the behaviour of a particular switch i is given in (5.51), where the conditional clause in the receive function has been dropped as ACP does not consider the time.

$$V_i = \sum_{i=0}^{N} \Bigg(\sum_{p=0}^{M+N} \Big(r_{V_i,p}(d) \cdot \Big(s_{V_i,b_{|M}}(d) \lhd i = \left\lfloor \frac{b}{M} \right\rfloor \rhd \\ \Big(\Big(\sum_{q=0}^{N} s_{V_i,M+q}(d) \lhd q \neq i \rhd \emptyset \Big) \lhd i = \left\lfloor \frac{a}{M} \right\rfloor \rhd s_{V_i,M+\lfloor b/M \rfloor}(d) \Big) \Big) \Big) \Bigg) \cdot V_i \qquad (5.51)$$

Focusing on *redundant paths* within the model, it is to be said that Table 5.13 exposes the redundant paths for hosts being one switch away (where $\lfloor \frac{a}{M} \rfloor = \lfloor \frac{b}{M} \rfloor$), or otherwise, Table 5.14 exhibits the those being two switches away (where $\lfloor \frac{a}{M} \rfloor \neq \lfloor \frac{b}{M} \rfloor$).

Table 5.13: List of devices and ports between 1-hop away hosts in full mesh.

Communication	**Direction**	**Expression**
Hosts and end switches	Uplink	$\{H_a\}, 0 \longmapsto a_{\vert M}, \{V_{\lfloor \frac{a}{M} \rfloor}\}$
End switches and hosts	Downlink	$\{V_{\lfloor \frac{b}{M} \rfloor}\}, b_{\vert M} \longmapsto 0, \{H_b\}$

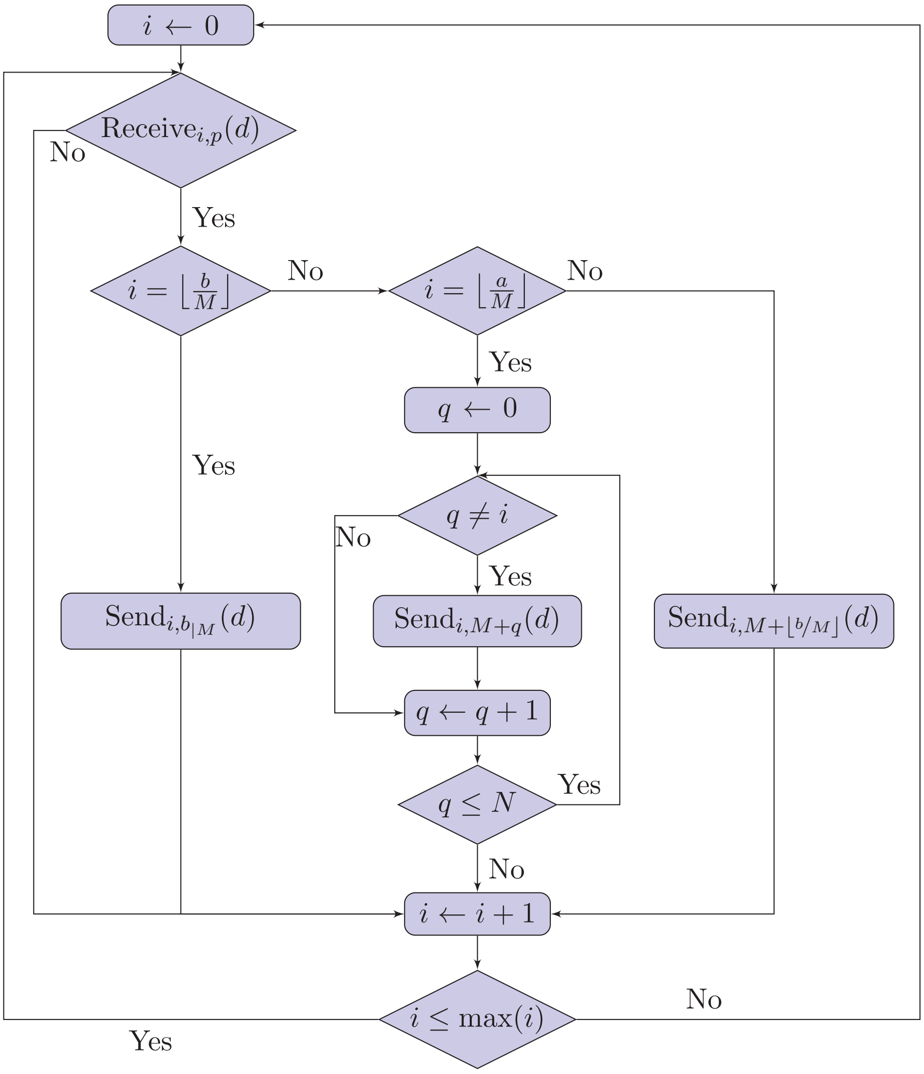

$i \leftarrow 0$
$\text{Receive}_{i,p}(d)$
No
Yes
$i = \lfloor \frac{b}{M} \rfloor$
No
$i = \lfloor \frac{a}{M} \rfloor$
No
Yes
$q \leftarrow 0$
Yes
$q \neq i$
No
Yes
$\text{Send}_{i,b_{|M}}(d)$
$\text{Send}_{i,M+q}(d)$
$\text{Send}_{i,M+\lfloor b/M \rfloor}(d)$
$q \leftarrow q+1$
$q \leq N$
Yes
No
$i \leftarrow i+1$
No
$i \leq \max(i)$
Yes

Figure 5.25: Flow chart for the behaviour of a given switch (N-simplex).

Table 5.14: List of devices and ports between 2-hops away hosts in full mesh.

Communication	Direction	Expression
Hosts and source switches	Uplink	$\{H_a\}, 0 \longmapsto a_{\vert M}, \{V_{\lfloor \frac{a}{M} \rfloor}\}$
End switches and other switches	Uplink	$\sum_{\substack{0 \leq q \leq N \\ q \neq i}} \Big(\{V_{\lfloor \frac{a}{M} \rfloor}\}, (M+q) \longmapsto$ $\longmapsto \left\lfloor \frac{a}{M} \right\rfloor, \{V_q\} \Big)$
Other switches and dest. switches	Downlink	$\sum_{\substack{0 \leq q \leq N \\ q \neq i}} \Big(\{V_q\}, \left\lfloor \frac{b}{M} \right\rfloor \longmapsto$ $\longmapsto (M+q), \{V_{\lfloor \frac{b}{M} \rfloor}\} \Big)$
Dest. switches and hosts	Downlink	$\{V_{\lfloor \frac{b}{M} \rfloor}\}, b_{\vert M} \longmapsto 0, \{H_b\}$

Eventually, it is to be said that, as all switches in this topology are end switches and all links among switches are considered to be internal communications, whereas links going to hosts are considered to be external ones, then the *verification* procedure explained for the N-hypercube applies in this case. This is, the external behaviour of the model is rooted branching bisimilar to that of the real system, hence the model gets duly verified.

5.8 Quasi full mesh

This topology may be seen as an N-orthoplex structure, where each individual switch has $2 \times (N-1)$ paths to the other switches, meaning to all the others but its opposite one. As in the full mesh topology, in order not to flood the topology with too many redundant messages, only the source node $\left\lfloor \frac{a}{M} \right\rfloor$ sends messages to other switches, whilst in turn, those just forward them on to the destination node $\left\lfloor \frac{b}{M} \right\rfloor$. Also, it is to be noted that there are $2N$ switches overall, accounting for $2N \times M$ hosts overall.

As per the *layout*, with respect to the port identifiers for each switch, the first M ports are connected to hosts (going from 0 to $M-1$), whilst there are up to $2(N-1)$ uplink ports being in use. In order to attain easier expressions, the numeration of them goes from M to $M+2N-1$, according to the formula $M+i$, where i is the identifier of the switch on the other end of each link, where switches go from 0 to $2N - 1$. Hence, the port related to its own i value is skipped out, as so is the port going to its opposite node, given by the expression $(i + N)\ mod\ 2N$.

To round it all up, Figure 5.26 exhibits the uplink ports of a single switch i, whilst Figure 5.27 depicts the nomenclature of switches by selecting one of them as a reference, called 0, and then, citing the real dimension where up to $N - 1$ nodes are moved from that reference, whereas at this point, adding up N to all items deployed so far to reach their corresponding opposite nodes.

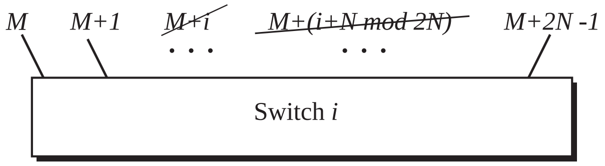


Figure 5.26: Model for the uplink ports in a quasi full mesh switch (N-orthoplex).

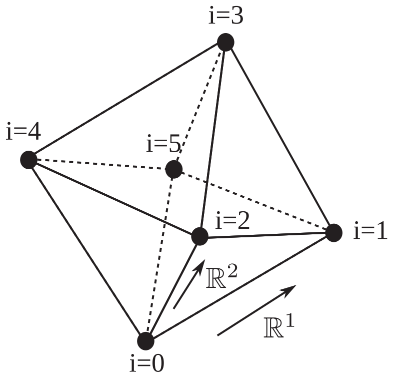


Figure 5.27: Model for the interconnected switches in a 3-orthoplex.

With that in mind, Figure 5.28 exhibits the *flow chart* exhibiting the behaviour of a switch i.

Moreover, the *pseudocode model* referred to the behaviour of a given switch i is shown in algorithm 10 (shown in the Appendix 10.10).

Regarding the *algebraic model*, the ACP expression to represent the behaviour of a particular switch i is exposed in (5.52), where the conditional clause in the receive function has been dropped as ACP does not consider the time.

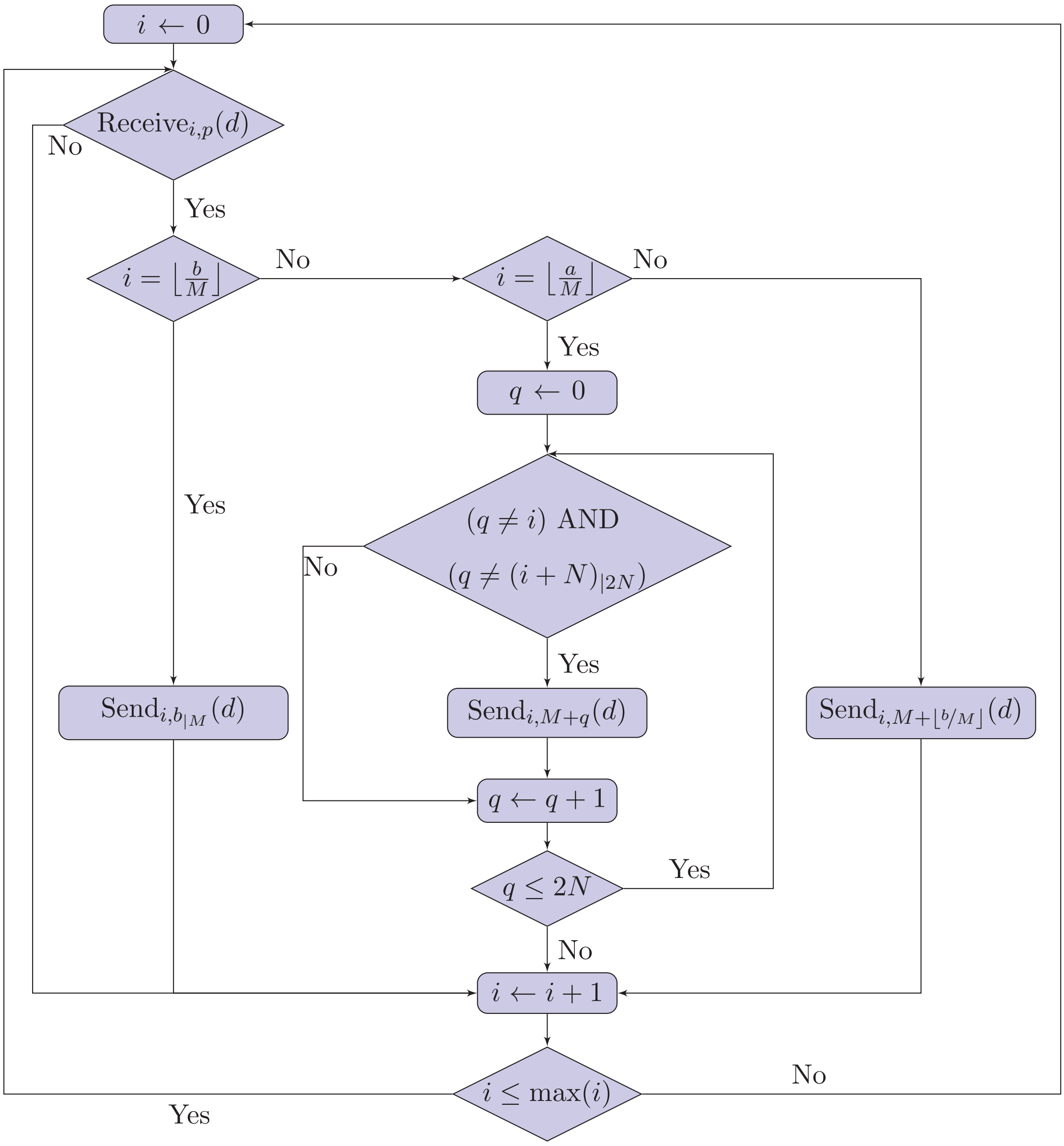


Figure 5.28: Flow chart for the behaviour of a given switch (N-orthoplex).

$$V_i = \sum_{i=0}^{2N-1} \left(\sum_{p=0}^{M+2N-1} \Bigg(r_{V_i,p}(d) \cdot \Bigg(s_{V_i,b_{|M}}(d) \lhd i = \left\lfloor \frac{b}{M} \right\rfloor \rhd \\ \Bigg(\Bigg(\sum_{q=0}^{2N-1} s_{V_i,M+q}(d) \lhd q \neq i \;\; AND \;\; q \neq (i+N)_{|2N} \rhd \emptyset \Bigg) \lhd i = \left\lfloor \frac{a}{M} \right\rfloor \rhd s_{V_i,M+\lfloor b/M \rfloor}(d) \Bigg) \Bigg) \Bigg) \right) \cdot V_i \tag{5.52}$$

Focusing on *redundant paths* within the model, it is to be noted that Table 5.15 shows the redundant paths for hosts being one switch away (where $\lfloor \frac{a}{M} \rfloor = \lfloor \frac{b}{M} \rfloor$), whilst Table 5.16 presents those being two switches away (where $\lfloor \frac{a}{M} \rfloor \neq \lfloor \frac{b}{M} \rfloor$).

Table 5.15: List of devices and ports between 1-hop away hosts in quasi full mesh.

Communication	**Direction**	**Expression**
Hosts and end switches	Uplink	$\{H_a\}, 0 \longmapsto a_{\|M}, \{V_{\lfloor \frac{a}{M} \rfloor}\}$
End switches and hosts	Downlink	$\{V_{\lfloor \frac{b}{M} \rfloor}\}, b_{\|M} \longmapsto 0, \{H_b\}$

Table 5.16: List of devices and ports between 2-hops away hosts in quasi full mesh.

Communication	**Direction**	**Expression**
Hosts and source switches	Uplink	$\{H_a\}, 0 \longmapsto a_{\|M}, \{V_{\lfloor \frac{a}{M} \rfloor}\}$
End switches and other switches	Uplink	$\sum_{\substack{0 \leq q \leq N \\ q \neq i \\ q \neq (i+N)_{\|2N}}} \Big(\{V_{\lfloor \frac{a}{M} \rfloor}\}, (M+q) \longmapsto \longmapsto \lfloor \frac{a}{M} \rfloor, \{V_q\} \Big)$
Other switches and dest. switches	Downlink	$\sum_{\substack{0 \leq q \leq N \\ q \neq i \\ q \neq (i+N)_{\|2N}}} \Big(\{V_q\}, \lfloor \frac{b}{M} \rfloor \longmapsto \longmapsto (M+q), \{V_{\lfloor \frac{b}{M} \rfloor}\} \Big)$
Dest. switches and hosts	Downlink	$\{V_{\lfloor \frac{b}{M} \rfloor}\}, b_{\|M} \longmapsto 0, \{H_b\}$

Eventually, it is to be said that the reasoning regarding the *verification* of the model is analogous to that exposed in the N-hypercube, as all switches are end switches.

5.9 Redundant star

This topology contains two hubs in order to achieve redundant connections with all spokes. It may be implemented in different ways and running protocols, although that is irrelevant for this model, as it only takes into account the redundant paths. If it is considered that there are n spokes, their disposition might be associated with an n-wheel polygon. However, the hubs are usually located right in the middle, although sometimes they might also be seen as the root of a tree, where the spokes are the leaves of such a tree.

It is to be acknowledged that such a topology is quite commonly deployed in production environments, because it avoids the use of routing protocols if implemented as layer 2 environments, or otherwise, routing is pretty easy if done as layer 3 ones. Any way, the distance between any pair of switches gets reduced to just 2.

As per the *layout*, Figure 5.29 exhibits the models for hubs and for spokes, along with their corresponding ports. Regarding the hubs, 2 of them are considered, both with n downlink ports going to the spokes and 1 uplink port going to the other hub, hence accounting for $n+1$ ports overall. With respect to the spokes, n are considered, all with M downlink ports going to hosts and 2 uplink port going to the hubs, thus accounting for $M+2$ ports overall. Regarding hosts, there are M per switch, leading to a total amount of $M \times n$. It is to be noted that the link among hubs is meant for control purposes, hence it does not take part into forwarding tasks among spokes.

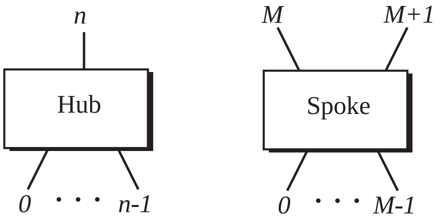


Figure 5.29: Model for switches, along with its ports (redundant hub and spoke).

On the other hand, Figure 5.30 exposes the nomenclature used in this model for hubs and spokes, where both types are numerated clockwise.

5.9.1 Spoke switches

To start with, Figure 5.31 exhibits the *flow chart* exhibiting the behaviour of a spoke i.

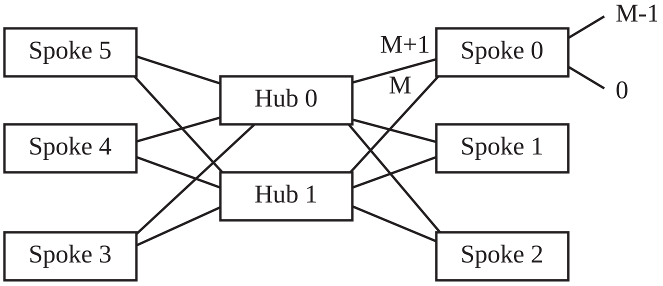


Figure 5.30: Nomenclature of switches in redundant hub and spoke.

Moreover, algorithm 11 (shown in the Appendix 10.11) presents the *pseudocode model* regarding the behaviour of a given spoke switch i.

With respect to the *algebraic model*, the ACP expression to represent the behaviour of a spoke switch is exhibited in (5.53), where spokes are identified with variable C. ACP does not take time into account, hence the initial conditional clause may be skipped, thus the first action may always be the reception of a message through any of its ports.

$$C_i = \sum_{i=0}^{n-1} \left(\sum_{p=0}^{M+1} \left(r_{C_i,p}(d) \cdot \left(s_{C_i,b_{|M}}(d) \triangleleft i = \left\lfloor \frac{b}{M} \right\rfloor \triangleright \sum_{q=M}^{M+1} s_{C_i,q}(d) \right) \right) \right) \cdot C_i \quad (5.53)$$

5.9.2 Hub switches

To begin with, Figure 5.32 depicts the *flow chart* exhibiting the behaviour of a hub j.

Besides, algorithm 12 (shown in the Appendix 10.12) exhibits the *pseudocode model* regarding the behaviour of a given hub switch j.

In relation to the *algebraic model*, the ACP expression to represent how a hub switch behaves is presented in (5.54), where hubs are identified with variable D. Besides, the conditional clause in the receive function has been dropped as ACP does not consider the time.

$$D_j = \sum_{j=0}^{1} \left(\sum_{p=0}^{n} \left(r_{D_j,p}(d) \cdot s_{D_j,\left\lfloor \frac{b}{M} \right\rfloor}(d) \right) \right) \cdot D_j \quad (5.54)$$

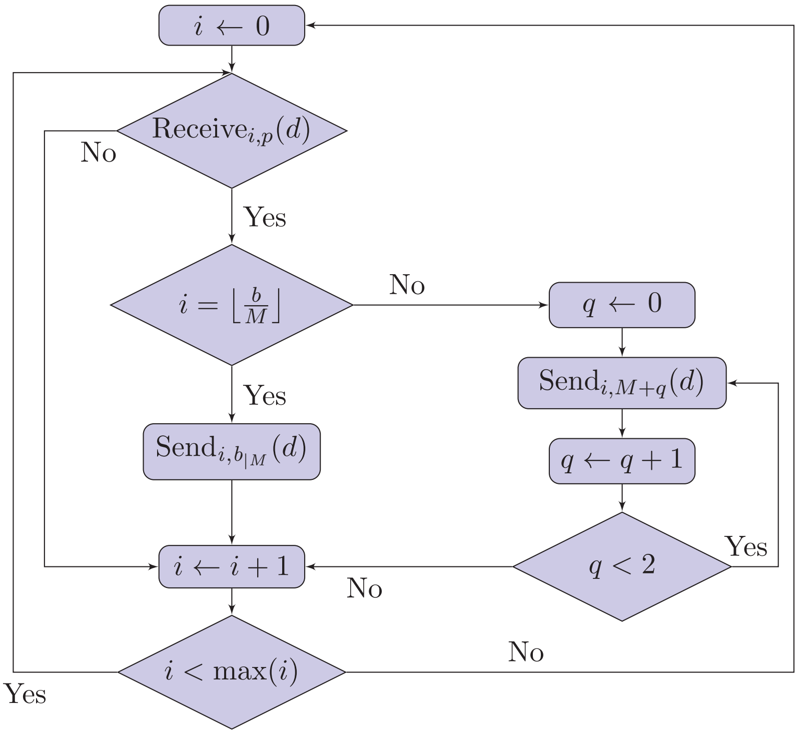


Figure 5.31: Flow chart for the behaviour of a spoke (redundant hub and spoke).

5.9.3 Consolidating hub and spoke

Focusing on *redundant paths* within the model, it is to be said that Table 5.17 exposes the path for hosts being one hop away (where $\lfloor \frac{a}{M} \rfloor = \lfloor \frac{b}{M} \rfloor$), thus hosts hanging on the same spoke switch, whilst Table 5.18 exhibits the redundant paths being connected to different spoke switches (where $\lfloor \frac{a}{M} \rfloor \neq \lfloor \frac{b}{M} \rfloor$), hence being two hops away.

Table 5.17: List of devices and ports between 1-hop away hosts in R. hub & spoke.

Communication	**Direction**	**Expression**
Hosts and spokes	Uplink	$\{H_a\}, 0 \longmapsto a_{\vert M}, \{C_{\lfloor \frac{a}{M} \rfloor}\}$
Spokes and hosts	Downlink	$\{C_{\lfloor \frac{b}{M} \rfloor}\}, b_{\vert M} \longmapsto 0, \{H_b\}$

Eventually, it is to be considered that the reasoning regarding the *verification* of the model is analogous to that exposed in a previous section for the leaf and spine

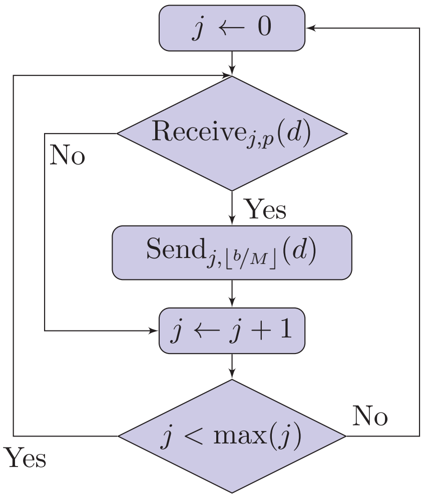


Figure 5.32: Flow chart for the behaviour of a hub (redundant hub and spoke).

Table 5.18: List of devices and ports between 2-hops away hosts in R. hub & spoke.

Communication	**Direction**	**Expression**
Hosts and spokes	Uplink	$\{H_a\}, 0 \longmapsto a_{\mid M}, \{C_{\lfloor \frac{a}{M} \rfloor}\}$
Spokes and hubs	Uplink	$\sum_{q=0}^{1} \left(\{C_{\lfloor \frac{a}{M} \rfloor}\}, (M+q) \longmapsto \right.$ $\left. \longmapsto \left\lfloor \frac{a}{M} \right\rfloor, \{D_q\} \right)$
Hubs and spokes	Downlink	$\sum_{q=0}^{1} \left(\{D_q\}, \left\lfloor \frac{b}{M} \right\rfloor \longmapsto \right.$ $\left. \longmapsto (M+q), \{C_{\lfloor \frac{b}{M} \rfloor}\} \right)$
Spoke and hosts	Downlink	$\{C_{\lfloor \frac{b}{M} \rfloor}\}, b_{\mid M} \longmapsto 0, \{H_b\}$

topology, as both might be considered to form a two-layer hierarchy. In this sense, expression (5.55) presents the encapsulation operator being applied to communications between spokes and hubs.

$$||_{i=0}^{n-1} ||_{j=0}^{1} \partial_H (C_i \parallel D_j) = \Bigg(r_{\{C_i\}, a_{\mid M}}(d) \cdot \Bigg(\sum_{q=0}^{1} c_{\{C_i\}, M+q \longmapsto a_{\mid M}, \{D_q\}}(d) \cdot$$
$$c_{\{D_q\}, b_{\mid M} \longmapsto M+q, \{C_i\}}(d) \lhd i \neq \left\lfloor \frac{b}{M} \right\rfloor \rhd \emptyset \Bigg) \cdot s_{\{C_i\}, b_{\mid M}}(d) \Bigg) \cdot (C_i \parallel D_j) \qquad (5.55)$$

Afterwards, the abstraction operator is applied to the whole set of spokes and hubs in expression (5.56), therefore, the switching infrastructure may get masked, thus showing up the external behaviour of the model.

$$
\begin{aligned}
&||_{i=0}^{n-1}||_{j=0}^{1}\tau_I\Big(\partial_H\big(C_i \parallel D_j\big)\Big) = \tau_I\Big(r_{\{C_i\},a_{|M}}(d)\cdot \\
&\Big(\sum_{q=0}^{1} c_{\{C_i\},M+q\longmapsto a_{|M},\{D_q\}}(d)\cdot c_{\{D_q\},b_{|M}\longmapsto M+q,\{C_i\}}(d) \\
&\lhd i \neq \left\lfloor \frac{b}{M} \right\rfloor \rhd \emptyset\Big)\cdot s_{\{C_i\},b_{|M}}(d)\Big)\cdot \tau_I\Big(\partial_H\Big(C_i \parallel D_j\Big)\Big) = \\
&r_{\{C_i\},a_{|M}}(d)\cdot s_{\{C_i\},b_{|M}}(d)\cdot \tau_I\Big(\partial_H\big(C_i \parallel D_j\big)\Big) \qquad (5.56)
\end{aligned}
$$

On the other hand, expression (5.57) depicts the external behaviour of the real system X, stating that a message d gets into the switching infrastructure through a given source switch i, and eventually, gets off through a given destination switch i'.

$$
\tau_I\big(X\big) = r_{\{C_i\},p}\big(d\big)\cdot s_{\{C_{i'}\},p'}\big(d\big)\cdot \tau_I\big(X\big) \qquad (5.57)
$$

Therefore, it appears to be obvious that both recursive expressions are multiplied by the same factors, where the incoming port into the system is $p = a_{|M}$ and outgoing port off the system is $p' = b_{|M}$. This means that they may be considered as *rooted branching bisimilar*, as they both have the same string of actions and they both share the same branching structure, hence allowing the application of expression (5.58).

$$
X \longleftrightarrow \tau_I\Big(\partial_H\big(C_i \parallel D_j\big)\Big) \qquad (5.58)
$$

In conclusion, this is a sufficient condition to get a model verified, hence, the model proposed for a leaf and spine architecture has been duly verified.

5.10 Redundant ring

This topology may be seen like a linear torus, where double links are deployed for redundancy purposes. This design may be fit for critical networks such as core interconnections. Supposing a number of n nodes, each one represented by one switch having M hosts hanging on, the point here is to check for the shortest path from source node to destination node, being leftwards or rightwards, where a distance of $\frac{n}{2}$ stands for load balancing paths in both directions, taking into account the wraparound step if necessary.

In order to achieve that, the difference between nodes $\lfloor \frac{b}{M} \rfloor$ and i is the key, such as if that value is both lower than 0 and greater or equal than $-\frac{n}{2}$, or otherwise, greater than $\frac{n}{2}$, then the resulting path is through the left neighbour, whereas if such a value is both greater than 0 and lower or equal than $\frac{n}{2}$, or otherwise, lower than $-\frac{n}{2}$, then the resulting path is through the right neighbour.

As per the *layout*, regarding the n node identifiers, they go from 0 all the way to $n-1$, from left to right. With respect to the ports of each one, the M lower ports (going from 0 to $M-1$) are connected to hosts, whereas ports M and $M+1$ point to the node on the left hand side, whereas ports $M+2$ and $M+3$ do it to the right hand side. Figure 5.33 exhibits the model with the port nomenclature of a given node, whilst Figure 5.34 depicts the setup of the n nodes in a linear torus way.

Therefore, if $\big(-\frac{n}{2} \leq (\lfloor\frac{b}{M}\rfloor - i) < 0\big)$ OR $\big((\lfloor\frac{b}{M}\rfloor - i) > \frac{n}{2}\big)$, then the path towards destination goes through the left links, being ports M and $M+1$, whereas if $\big(0 < (\lfloor\frac{b}{M}\rfloor - i) \leq \frac{n}{2}\big)$ OR $\big((\lfloor\frac{b}{M}\rfloor - i) < -\frac{n}{2}\big)$, then that path goes through the right links, being ports $M+2$ and $M+3$.

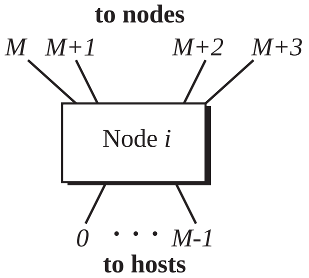


Figure 5.33: Model for switches, along with its ports (redundant ring).

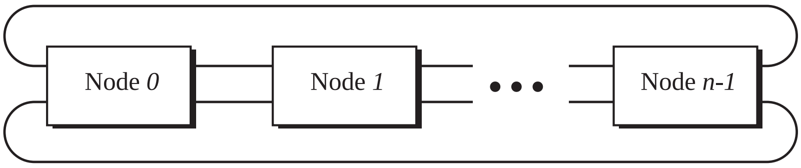


Figure 5.34: Node setup for a redundant ring topology.

Taking this into account, Figure 5.35 exposes the *flow chart* exhibiting the behaviour of a switch i.

Besides, algorithm 13 (shown in the Appendix 10.13) presents the *pseudocode model* related to the behaviour of a given switch i.

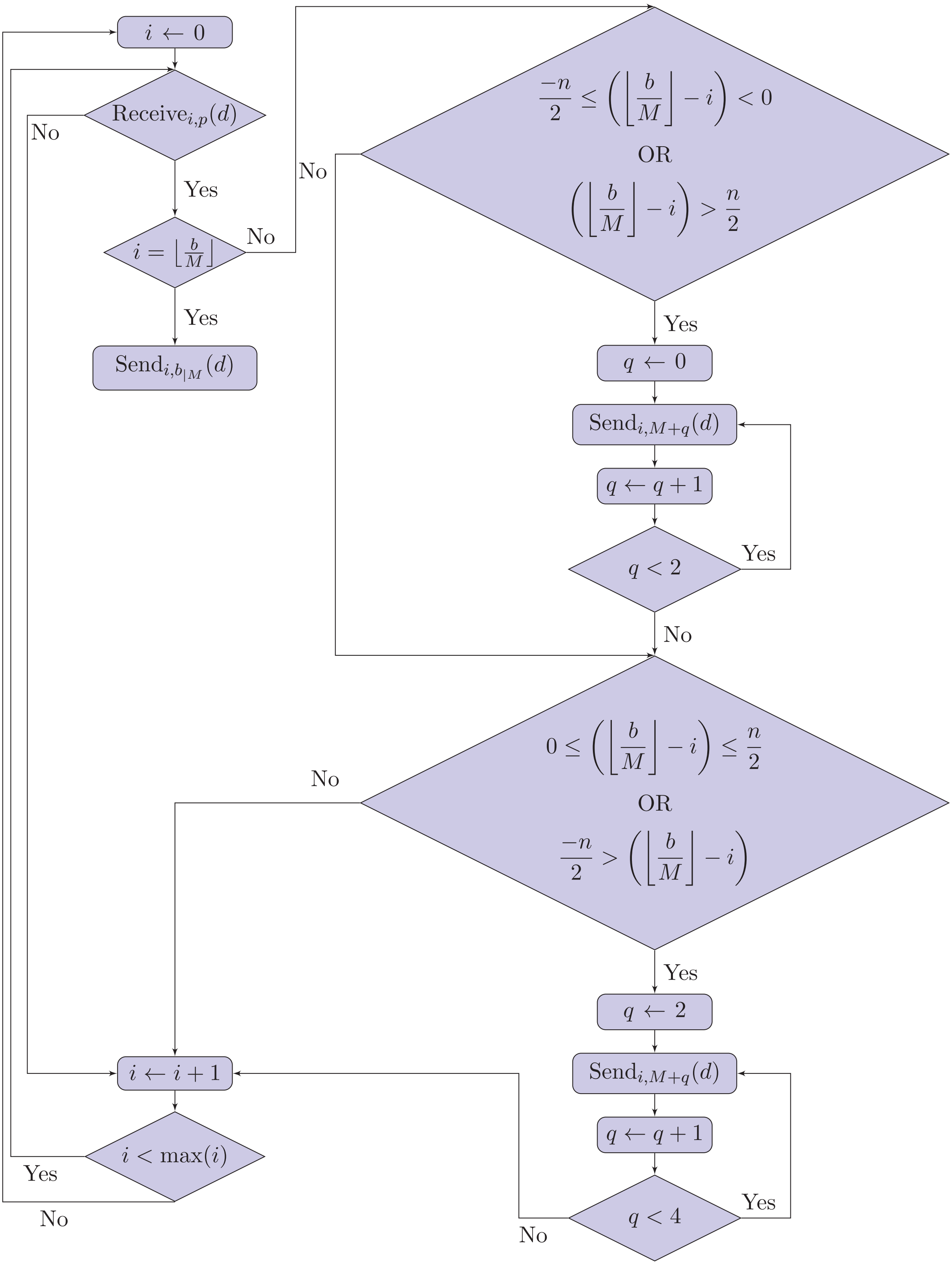

$i \leftarrow 0$
Receive$_{i,p}(d)$
No
Yes
$i = \lfloor \frac{b}{M} \rfloor$
No
Yes
Send$_{i,b_{|M}}(d)$
$\frac{-n}{2} \leq \left(\left\lfloor \frac{b}{M} \right\rfloor - i\right) < 0$
OR
$\left(\left\lfloor \frac{b}{M} \right\rfloor - i\right) > \frac{n}{2}$
No
Yes
$q \leftarrow 0$
Send$_{i,M+q}(d)$
$q \leftarrow q + 1$
$q < 2$
Yes
No
$0 \leq \left(\left\lfloor \frac{b}{M} \right\rfloor - i\right) \leq \frac{n}{2}$
OR
$\frac{-n}{2} > \left(\left\lfloor \frac{b}{M} \right\rfloor - i\right)$
No
Yes
$q \leftarrow 2$
Send$_{i,M+q}(d)$
$q \leftarrow q + 1$
$q < 4$
Yes
No
$i \leftarrow i + 1$
$i < \max(i)$
Yes
No

Figure 5.35: Flow chart for the behaviour of a given switch (redundant ring).

With respect to the *algebraic model*, the ACP expression which represents the behaviour of an particular switch i is exhibited in (5.59). ACP does not take time into account, hence the initial conditional clause may be skipped, thus the first action may always be the reception of a message through any of its ports.

$$\begin{gathered}V_i = \sum_{i=0}^{n-1} \Bigg(\sum_{p=0}^{M+3} \Bigg(r_{V_i,p}(d) \cdot \Big(s_{V_i,b_{|M}}(d) \triangleleft i = \left\lfloor \frac{b}{M} \right\rfloor \triangleright \\ \bigg(\Big(\sum_{q=M}^{M+1} s_{V_i,M+q}(d) \triangleleft \big(\frac{-n}{2} \leq (\left\lfloor \frac{b}{M} \right\rfloor - i) < 0 \big) \text{ OR } \big((\left\lfloor \frac{b}{M} \right\rfloor - i) > \frac{n}{2} \big) \triangleright \emptyset \Big) \cdot \\ \Big(\sum_{q=M+2}^{M+3} s_{V_i,M+q}(d) \triangleleft \big(0 < (\left\lfloor \frac{b}{M} \right\rfloor - i) \leq \frac{n}{2} \big) \text{ OR } \big(\frac{-n}{2} > (\left\lfloor \frac{b}{M} \right\rfloor - i) \big) \triangleright \emptyset \Big) \bigg) \Big) \Bigg) \Bigg) \cdot V_i\end{gathered} \tag{5.59}$$

Focusing on *redundant paths* within the model, it is to be reminded that graph-like designs may exhibit a variety of them depending on the distance between end switches, as explained for the N-hypercube case.

Eventually, it is to be reminded that this topology is a graph-like one, where all switches are end switches. Hence, its *verification* procedure is analogous to that stated for the N-hypercube scenario.

5.11 Toroidal ring

This topology may be viewed as a ring torus, in a squared setup, meaning that nodes are connected forming a square with side n, having wraparound links between the external nodes. This way, supposing a number of nodes n^2 equally distributed in the square, each one represented by one switch having M hosts hanging on, the point is to check for the shortest path from source node to destination node. It is to be considered that a distance of $\frac{n}{2}$ in a single direction may allow for 2 load balancing paths, taking into account a wraparound step if necessary. Otherwise, if distances occur in different directions, the permutations of all steps to reach destination may lead to a variety of load balancing paths, growing with the number of those steps.

In order to calculate that distance between source and destination nodes, one approach may be to spot whether they are located in different rows, which is done by comparing $\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor$ and $\left\lfloor \frac{i}{n} \right\rfloor$. Otherwise, an alternative approach may be to check whether both nodes are in different columns, which is obtained by comparing $\left\lfloor \frac{b}{M} \right\rfloor_{|n}$ and $i_{|n}$. In case any of those apply, then the procedure to go from source to destination may be similar to that proposed for the redundant ring case, where the path in a single direction may not be greater than $\frac{n}{2}$, and the maximum path length may never be greater than n, accounting both dimensions.

That way, if source and destination nodes are in different rows ($\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor \neq \left\lfloor \frac{i}{n} \right\rfloor$), then if $\left\lfloor \frac{b}{M} \right\rfloor_{|n} - i_{|n}$ is both lower than 0 and greater or equal than $-\frac{n}{2}$, or otherwise, greater than $\frac{n}{2}$, then the resulting path is through the neighbour below, whilst if such a value is both greater than 0 and lower or equal than $\frac{n}{2}$, or otherwise, lower than $-\frac{n}{2}$, then the resulting path is through the neighbour above.

Likewise, if source and destination are in different columns ($\left\lfloor \frac{b}{M} \right\rfloor_{|n} \neq i_{|n}$), then if $\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor - \left\lfloor \frac{i}{n} \right\rfloor$ is both lower than 0 and greater or equal than $-\frac{n}{2}$, or otherwise, greater than $\frac{n}{2}$, then the resulting path is through the left neighbour, whereas if such a value is both greater than 0 and lower or equal than $\frac{n}{2}$, or otherwise, lower than $-\frac{n}{2}$, then the resulting path is through the right neighbour.

As per the *layout*, regarding the n^2 node identifiers, they go from 0 all the way to n^2-1, starting from the bottom left corner and completing each row from left to right with n items, where the first row goes from 0 to $n-1$ and the rest of rows get filled in with the natural number sequence, such as shown in Figure 5.36. With regards to the ports of each node, the M lower ports (going from 0 to $M-1$) are connected to hosts, whereas port M looks upwards and the rest of the ports go clockwise, thus, port $M+1$ looks rightwards, port $M+2$ looks downwards, and port $M+3$ looks leftwards, as it is depicted in Figure 5.37.

Taking this into account, Figure 5.38 exhibits the *flow chart* showing the behaviour of a switch i.

Furthermore, algorithm 14 (shown in the Appendix 10.14) exhibits the *pseudocode model* referred to the behaviour of a given switch i.

With regards to the *algebraic model*, the ACP expression representing the behaviour of a given switch i is exhibited in (5.60). It is to be reminded that ACP does

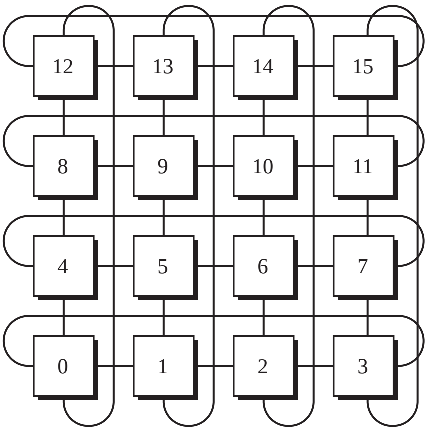


Figure 5.36: Node setup for a toroidal ring topology for $n = 4$.

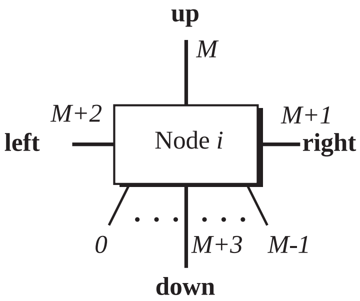


Figure 5.37: Model for switches, along with its ports (toroidal ring).

not take time into account, thus the initial conditional clause may be skipped, hence the first action may always be the reception of a message through any of its ports.

$$
\begin{aligned}
V_i = \sum_{i=0}^{n-1} \Biggl(\sum_{p=0}^{M+3} \Biggl(r_{V_i,p}(d) \cdot \Bigl(s_{V_i,b_{|M}}(d) \triangleleft\ i = \left\lfloor \frac{b}{M} \right\rfloor \triangleright \\
\Biggl(\Biggl(\Bigl(s_{V_i,M+2}(d) \triangleleft \bigl(\frac{-n}{2} \leq (\left\lfloor \frac{b}{M} \right\rfloor_{|n} - i_{|n}) < 0 \bigr) \;\; OR \;\; \bigl((\left\lfloor \frac{b}{M} \right\rfloor_{|n} - i_{|n}) > \frac{n}{2} \bigr) \triangleright \emptyset \Bigr) \cdot \\
\Bigl(s_{V_i,M}(d) \triangleleft \bigl(0 < (\left\lfloor \frac{b}{M} \right\rfloor_{|n} - i_{|n}) \leq \frac{n}{2} \bigr) \;\; OR \;\; \bigl(\frac{-n}{2} > (\left\lfloor \frac{b}{M} \right\rfloor_{|n} - i_{|n}) \bigr) \triangleright \emptyset \Bigr) \\
\triangleleft \left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor \neq \left\lfloor \frac{i}{n} \right\rfloor \triangleright \emptyset \Biggr) \cdot \\
\Biggl(\Bigl(s_{V_i,M+3}(d) \triangleleft \bigl(\frac{-n}{2} \leq (\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor - \left\lfloor \frac{i}{n} \right\rfloor) < 0 \bigr) \;\; OR \;\; \bigl((\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor - \left\lfloor \frac{i}{n} \right\rfloor) > \frac{n}{2} \bigr) \triangleright \emptyset \Bigr) \cdot \\
\Bigl(s_{V_i,M+1}(d) \triangleleft \bigl(0 < (\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor - \left\lfloor \frac{i}{n} \right\rfloor) \leq \frac{n}{2} \bigr) \;\; OR \;\; \bigl(\frac{-n}{2} > (\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor - \left\lfloor \frac{i}{n} \right\rfloor) \bigr) \triangleright \emptyset \Bigr) \\
\triangleleft \left\lfloor \frac{b}{M} \right\rfloor_{|n} \neq i_{|n} \triangleright \emptyset \Biggr)\Biggr)\Bigr)\Biggr)\Biggr) \cdot V_i
\end{aligned}
\tag{5.60}
$$

Focusing on *redundant paths* within the model, it is to be remarked that graph-like designs may present a broad range of them, which depends on the distance between the end switches, as exposed for the N-hypercube.

Eventually, regarding verification, it is to be recalled that this topology is a graph-like one, meaning that all switches are end switches. Hence, its *verification* process is similar to that stated for the N-hypercube.

5.12 De Bruijn graph

De Bruijn graphs are a type of directed graph, which has been studied above. Considering a generic k-alphabet, even though the binary alphabet is the most commonly used, along with a word length n, this design may account for k^n switches, where each of those has just k inwards links and just k outwards links.

Therefore, those values may be tailored to match different requirements, where the expressions to calculate the neighbour nodes to be visited through the outwards links of the nodes were given in (4.12) for binary alphabets (where obviously $k = 2$), and in (4.13) for generic k-ary alphabets, where the former works in bits, whilst the latter does it in k-symbols.

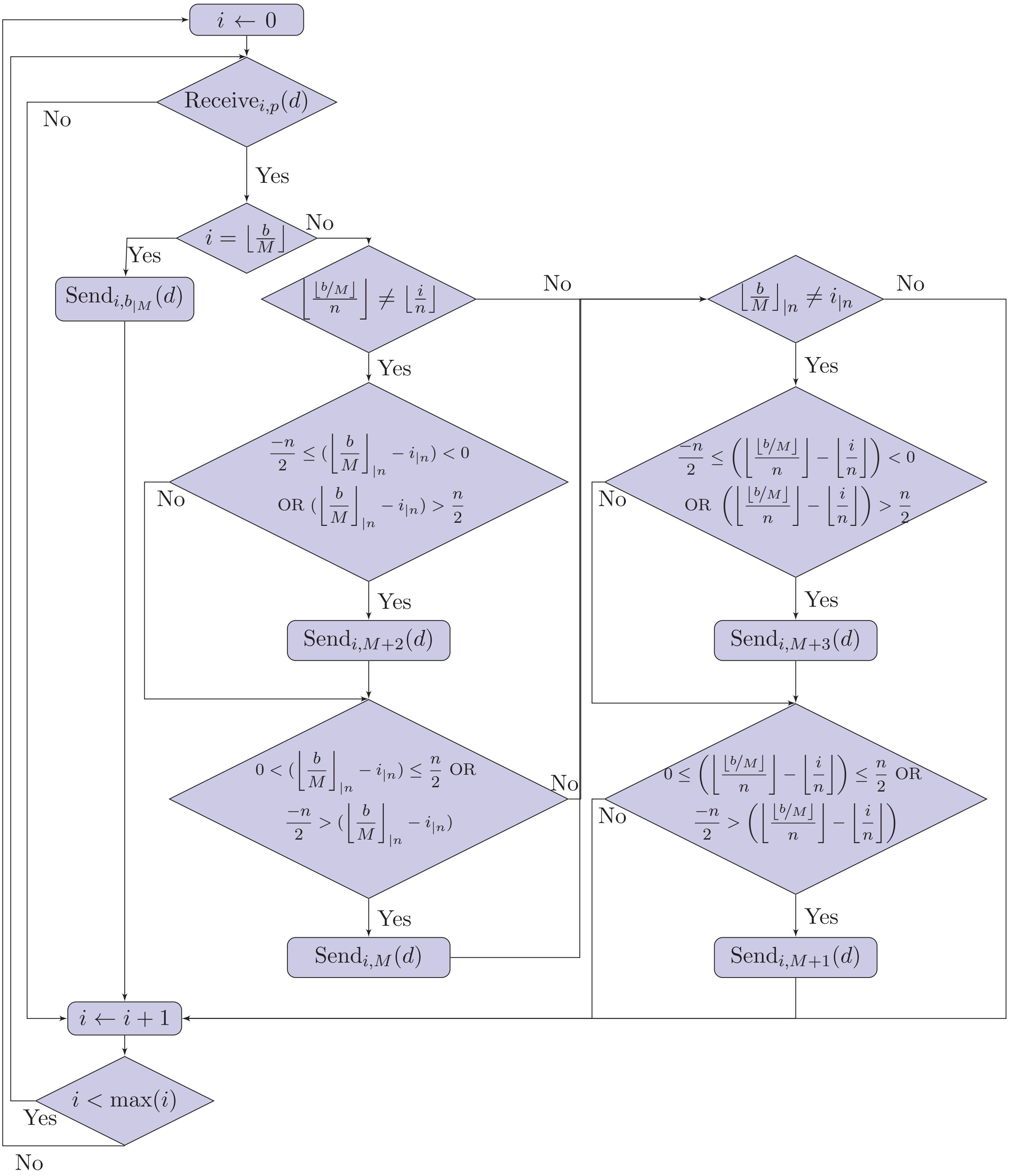

$i \leftarrow 0$
Receive$_{i,p}(d)$
No
Yes
$i = \left\lfloor \frac{b}{M} \right\rfloor$
No
Yes
Send$_{i,b_{|M}}(d)$
$\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor \neq \left\lfloor \frac{i}{n} \right\rfloor$
No
$\left\lfloor \frac{b}{M} \right\rfloor_{|n} \neq i_{|n}$
No
Yes
Yes
$\frac{-n}{2} \leq (\left\lfloor \frac{b}{M} \right\rfloor_{|n} - i_{|n}) < 0$ OR $(\left\lfloor \frac{b}{M} \right\rfloor_{|n} - i_{|n}) > \frac{n}{2}$
No
$\frac{-n}{2} \leq \left(\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor - \left\lfloor \frac{i}{n} \right\rfloor\right) < 0$ OR $\left(\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor - \left\lfloor \frac{i}{n} \right\rfloor\right) > \frac{n}{2}$
No
Yes
Send$_{i,M+2}(d)$
Yes
Send$_{i,M+3}(d)$
$0 < (\left\lfloor \frac{b}{M} \right\rfloor_{|n} - i_{|n}) \leq \frac{n}{2}$ OR $\frac{-n}{2} > (\left\lfloor \frac{b}{M} \right\rfloor_{|n} - i_{|n})$
No
$0 \leq \left(\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor - \left\lfloor \frac{i}{n} \right\rfloor\right) \leq \frac{n}{2}$ OR $\frac{-n}{2} > \left(\left\lfloor \frac{\lfloor b/M \rfloor}{n} \right\rfloor - \left\lfloor \frac{i}{n} \right\rfloor\right)$
No
Yes
Send$_{i,M}(d)$
Yes
Send$_{i,M+1}(d)$
$i \leftarrow i + 1$
$i < \max(i)$
Yes
No

Figure 5.38: Flow chart for the behaviour of a given switch (toroidal ring).

Sticking to binary digits ($k = 2$), it is to be reminded that each new hop is equivalent to inserting a new bit on the right of the previous node identifier, meaning the least significant bit, which accounts for performing a left-shifting bit in the binary identifier of the previous node.

The way to proceed is to compare incremental groups with the rightmost bits in the source node and the leftmost bits in the destination node, starting from just one bit (meaning the least significant bit of the source node with the most significant bit of the destination node), and if they match, then take a group of two bits (meaning the two least significant bits of the former with the two most significant bits of the latter), and in turn, going all the way until finding the minimum group length where both groups do not match.

Getting back to a general k-alphabet, the aforesaid procedure may be done in an *arithmetic way* by means of looking for a mismatch between $\left\lfloor \frac{i}{k^{top-j}} \right\rfloor_{|k}$ and $\left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-j}} \right\rfloor_{|k}$, where variable *top* grows sequentially and j loops it, until those values do not match, and at that point, the appropriate k-symbol to be added to the right of the source node is given by the expression $\left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-top}} \right\rfloor_{|k}$. Furthermore, the outgoing port may be $M + \left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-top}} \right\rfloor_{|k}$, which joins node i with neighbour node $\left(k \times i + \left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-top}} \right\rfloor_{|k} \right)_{|k^n}$.

Apart from the aforesaid arithmetic way, a *logical way* may also be applied in this case, which just performs bitwise operations to insert bits on the right hand side of the source node given in binary format (considering $k = 2$), or alternatively k-symbols in case of a generic k-alphabet, in order to perform the transition from such a given source node to the expected particular destination node.

As per the *layout*, regarding the k^n node identifiers, they are identified from 0 to $k^n - 1$, although the setup of their interswitching links depends on the value of variables k and n, as shown previously. On the other hand, each node has M ports connected to hosts (going from 0 to $M - 1$), whereas ports from M to $M + k - 1$ stand for the outwards links, whilst ports from $M + k$ to $M + 2k - 1$ do it for the inwards links, as shown in Figure 5.39.

Considering this, Figure 5.40 exhibits the *flow chart* exposing the behaviour of a particular switch i.

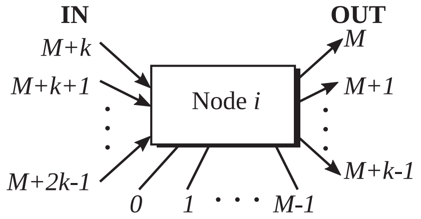


Figure 5.39: Model for a given node i in a de Bruijn graph, along with its ports.

Besides, algorithm 15 (shown in the Appendix 10.15) presents the *pseudocode model* regarding the behaviour of a given switch i.

With respect to the *algebraic model*, the ACP expression representing the behaviour of a particular switch i is exposed in (5.61). ACP does not take time into account, thus the initial conditional clause may be skipped, hence the first action may always be the reception of a message through any of its ports.

It is to be said that all ports are waiting for the reception of any messages, even though the outgoing ports (those going from M to $M+k-1$) may never get any, as de Bruijn graphs are directed, hence those ports may only send messages.

Additionally, variable *flag* is initialized by means of (~*flag*) and is incremented by one unit with (*flag*++), whereas variable *top* does it the same way. Besides, a new variable *temp* is introduced to perform the role of *top* within the *while* loop, which is incremented with a summatory from 0 to $n-1$. In fact, *temp* acts as an incremental upper bound for the looping variable j, such that it will get a wider range for looping at each iteration, having the target of finding the shortest value of *temp* meeting the inequality $\left\lfloor \frac{i}{k^{temp-j}} \right\rfloor_{|k} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-j}} \right\rfloor_{|k}$ with any value of j, and at that point, *top* takes the value of *temp*, which will be used to calculate the outgoing port $M + \left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-top}} \right\rfloor_{|k}$.

Therefore, if source and destination nodes are not the same item, the sequence of events states that as long as variable *flag* holds a zero, variable *top* gets incremented following the value of variable *temp* if, and only if, the aforementioned inequality does not hold. However, once it does hold, variable *flag* gets a one, which may avoid the inequality to be further examined, thus preventing *top* from getting any other value. Eventually, the value of *top* is used to obtain the outgoing port.

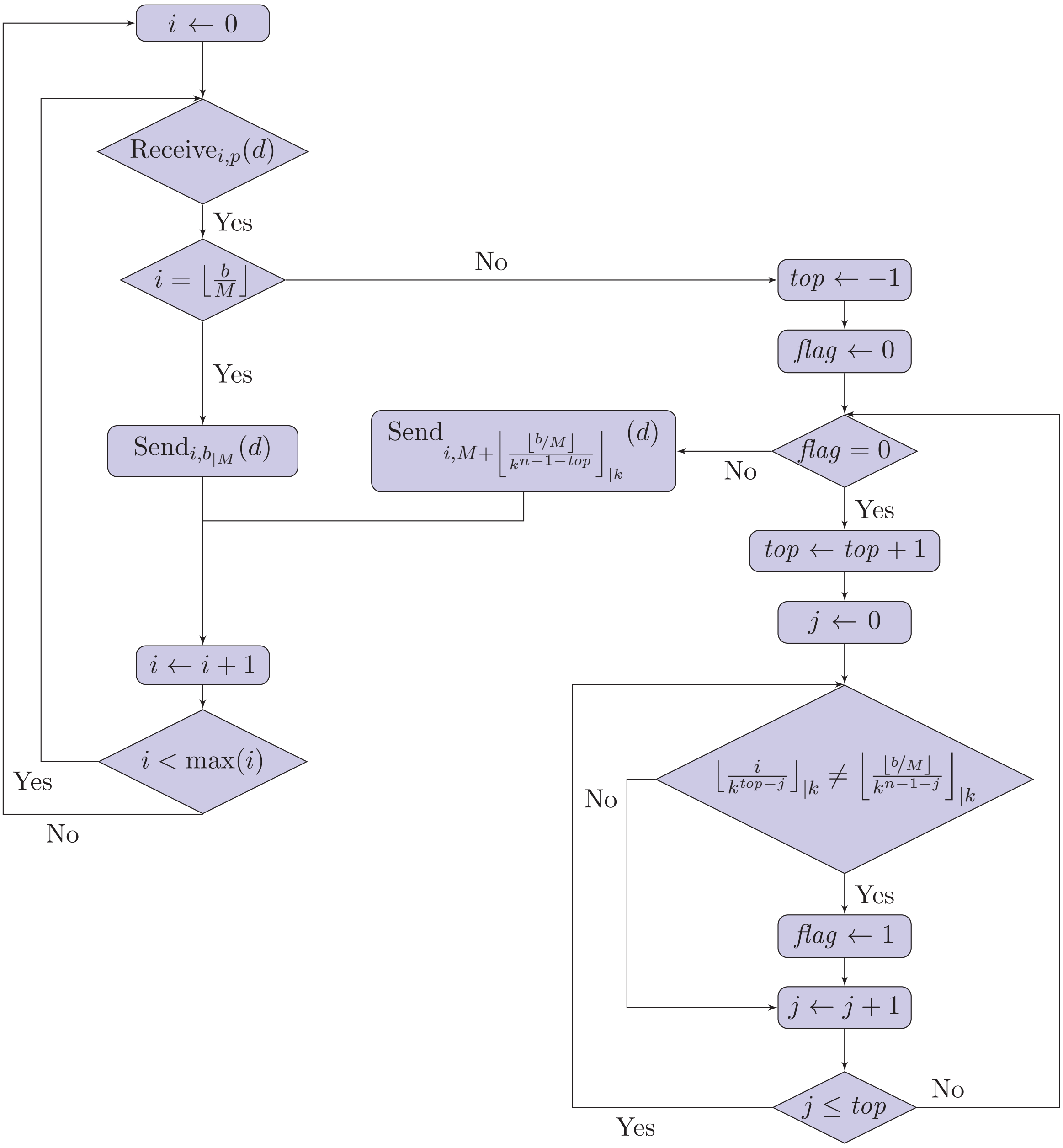

$i \leftarrow 0$
$\text{Receive}_{i,p}(d)$
Yes
$i = \lfloor \frac{b}{M} \rfloor$
No
Yes
$\text{Send}_{i,b_{|M}}(d)$
$\text{Send}_{i,M+\left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-top}} \right\rfloor_{|k}}(d)$
$i \leftarrow i+1$
$i < \max(i)$
Yes
No
$top \leftarrow -1$
$flag \leftarrow 0$
$flag = 0$
No
Yes
$top \leftarrow top + 1$
$j \leftarrow 0$
$\left\lfloor \frac{i}{k^{top-j}} \right\rfloor_{|k} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-j}} \right\rfloor_{|k}$
No
Yes
$flag \leftarrow 1$
$j \leftarrow j+1$
$j \leq top$
No
Yes

Figure 5.40: Flow chart for the behaviour of a switch (de Bruijn graph).

$$
\begin{aligned}
V_i = \sum_{i=0}^{k^n-1} \Bigg(\sum_{p=0}^{M+2k-1} \Bigg(r_{V_i,p}(d) \cdot \Bigg(s_{V_i,b_{|M}}(d) \lhd i = \left\lfloor \frac{b}{M} \right\rfloor \rhd \Bigg((\sim\!flag) \cdot (\sim\!top) \cdot \\
\Bigg(\sum_{temp=0}^{n-1} \Big(\sum_{j=0}^{temp} \Big((flag{+}{+}) \lhd \left\lfloor \frac{i}{k^{temp-j}} \right\rfloor_{|k} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-j}} \right\rfloor_{|k} \rhd \emptyset \Big) \Big) \cdot \Big((top{+}{+}) \lhd flag = 0 \rhd \emptyset \Big) \\
\lhd flag = 0 \rhd \emptyset \Bigg) \cdot s_{V_i, M + \left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-top}} \right\rfloor_{|k}}(d) \Bigg) \Bigg) \Bigg) \Bigg) \cdot V_i
\end{aligned}
\tag{5.61}
$$

Focusing on *redundant paths* within the model, it is to be noted that graph-like designs may present a bunch of them, depending on the distance between the end switches, as exposed for the N-hypercube.

However, Table 5.19 presents a code snippet so as to predict the string of hops to get from source node to destination node.

Table 5.19: Searching the hops from source to destination node (de Bruijn graph).

Mode	**Code snippet**

	NODE{ i }
	top = -1;
	flag = 0;
	while (flag == 0) {
	top = top + 1;
	for (j = 0; j $\leq$ top; j++) {
	if ($\left\lfloor \frac{i}{k^{top-j}} \right\rfloor_{\mid k} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-j}} \right\rfloor_{\mid k}$) {
	flag = 1;
	}
	}
	}
	for (m = top; m < n; m++) {
	NODE{ $\left(k \cdot i + \left\lfloor \frac{\lfloor b/M \rfloor}{k^{n-1-m}} \right\rfloor_{\mid k} \right)_{\mid k^n}$ };
	}

Eventually, it is to be noted that this topology is a graph-like one, as all switches are end switches. Hence, its *verification* procedure is similar to that stated for the N-hypercube scenario.

5.13 De Bruijn reverse graph

This type of directed graph may be seen as a variation of the original de Bruijn graphs, where all argumentation exposed therein applies. However, the key difference herein is that each new hop is equivalent to the insertion of a new bit on the left of the previous node (supposing $k = 2$), meaning the most significant bit, which may be seen as performing a right-shifting bit in the binary identifier of the previous node.

Therefore, the approach is the opposite of that given in the previous section, taking into account that the expressions to calculate the neighbour nodes through the outward links were already given in (4.14) for binary alphabets (where obviously $k = 2$), and in (4.15) for generic k-ary alphabets, where the former works in bits and the latter does it in k-symbols.

In that sense, taking the binary case as an example ($k = 2$), the way to go is to compare incremental groups with the leftmost bits in the source node and the rightmost bits in the destination node, starting from a group of just one bit (just the most significant bit of the source node with the least significant bit of the destination node), and if they happen to match, then increment the size of the group to include two bits (taking the two most significant bits of the former with the two least significant bits of the latter), and keep incrementing the group size until spotting the minimum group length where both groups do not occur to match.

As in the previous section, this task may be done in an *arithmetic way* by means of searching for a mismatch between $\left\lfloor \frac{i}{k^{n-1-j}} \right\rfloor_{|k}$ and $\left\lfloor \frac{\lfloor b/M \rfloor}{k^{top-j}} \right\rfloor_{|k}$, where variable *top* grows in a sequential manner, being looped as j in each iteration, until both values do not match, and then, the appropriate bit to be added to the right hand side of the source node is given by the expression $\left\lfloor \frac{\lfloor b/M \rfloor}{k^{top}} \right\rfloor_{|k}$. Furthermore, the outgoing port is given by $M + \left\lfloor \frac{\lfloor b/M \rfloor}{k^{top}} \right\rfloor_{|k}$, which joins node i with neighbour node $\left(i/k + \left\lfloor \frac{\lfloor b/M \rfloor}{k^{top}} \right\rfloor_{|k} \times k^{n-1}\right)_{|k^n}$.

Apart from the aforesaid arithmetic way, a *logical way* may be applied in this case as well, by performing bitwise operations to insert bits on the left hand side of the source node, given in binary format (considering $k = 2$), or alternatively k-symbols in case of a generic k-alphabet, in order to undertake the transition from the source node to the given destination node.

As per the *layout*, with regards to the de Bruijn reverse graph case, there are k^n node identifiers, going from 0 to $k^n - 1$, with the same port configuration as explained therein and as exhibited in Figure 5.39.

Considering this, Figure 5.41 shows the *flow chart* exhibiting the behaviour of a switch i.

Moreover, algorithm 16 (shown in the Appendix 10.16) presents the *pseudocode model* related to the behaviour of a given switch i.

Respecting the *algebraic model*, the ACP expression representing the behaviour of a particular switch i is exposed in (5.62). ACP does not take time into account, thus the initial conditional clause may be skipped, hence the first action may always be the reception of a message through any of its ports.

It is to be said that the explanation regarding the behaviour of the variables involved given in the previous case applies for this case as well, even though the expressions for the inequality terms and the outgoing port obviously differ.

$$\begin{aligned} V_i = \sum_{i=0}^{k^n-1} \Bigg(\sum_{p=0}^{M+2k-1} \Bigg(r_{V_i,p}(d) \cdot \Bigg(s_{V_i,b_{|M}}(d) \triangleleft i = \left\lfloor \frac{b}{M} \right\rfloor \triangleright \Bigg((\sim\!\mathit{flag}) \cdot (\sim\!\mathit{top}) \cdot \\ \Bigg(\sum_{temp=0}^{n-1} \Big(\sum_{j=0}^{temp} \Bigg((\mathit{flag}{+}{+}) \triangleleft \left\lfloor \frac{i}{k^{n-1-j}} \right\rfloor_{|k} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{k^{temp-j}} \right\rfloor_{|k} \triangleright \emptyset \Bigg) \Bigg) \cdot \Bigg((\mathit{top}{+}{+}) \triangleleft \mathit{flag} = 0 \triangleright \emptyset \Bigg) \\ \triangleleft \mathit{flag} = 0 \triangleright \emptyset \Bigg) \cdot s_{V_i, M + \left\lfloor \frac{\lfloor b/M \rfloor}{k^{top}} \right\rfloor_{|k}}(d) \Bigg) \Bigg) \Bigg) \Bigg) \cdot V_i \end{aligned} \tag{5.62}$$

Focusing on *redundant paths* within the model, it is to be said that graph-like designs may present a handful of those, depending on the distance between the end switches, as exposed for the N-hypercube.

However, Table 5.20 presents a code snippet in order to predict the string of hops to get from the source node to the destination node.

Eventually, it is to be considered that this topology is a graph-like one, where all switches are end switches. Hence, its *verification* is similar to that stated for the N-hypercube scenario.

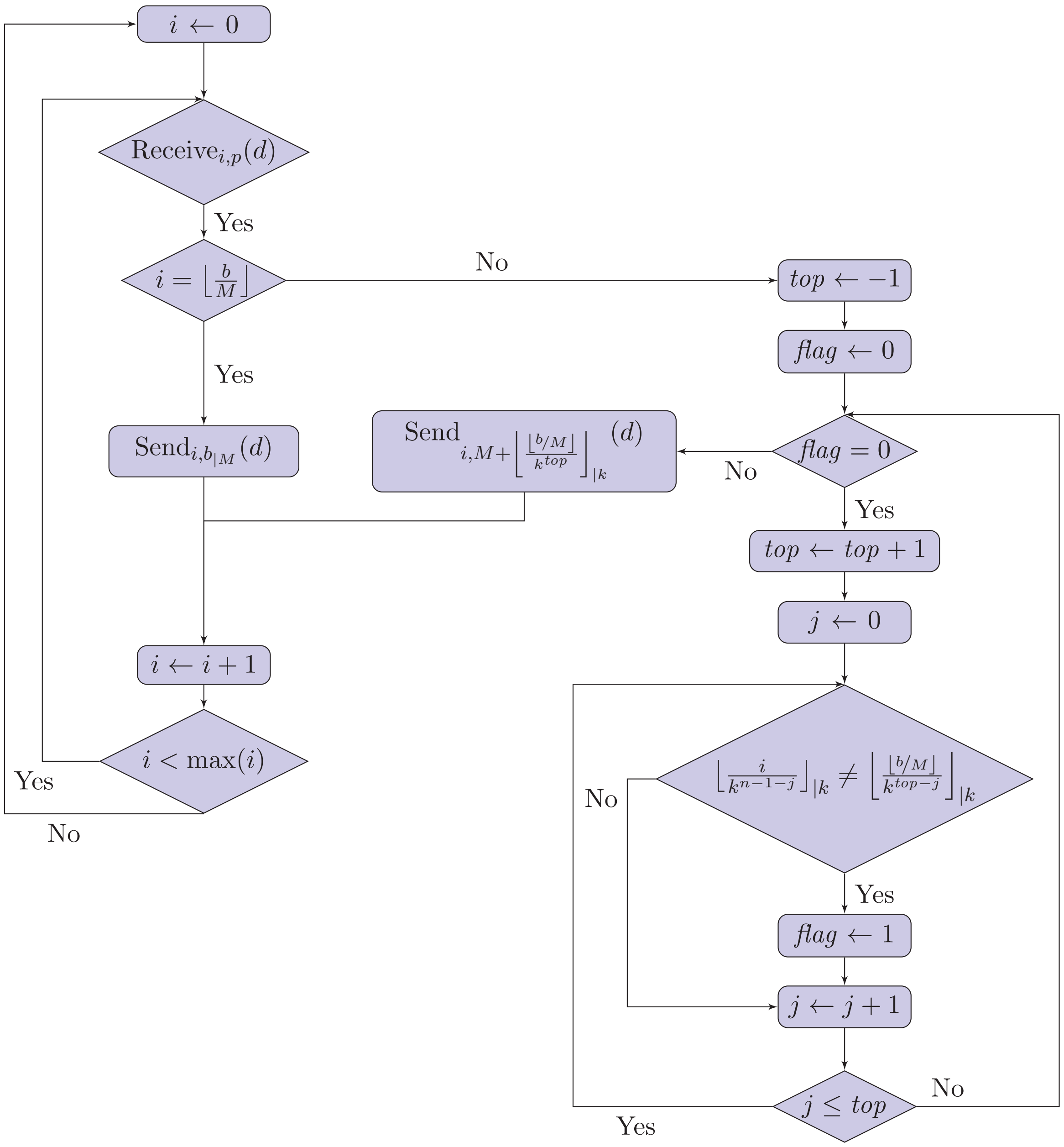


Figure 5.41: Flow chart for the behaviour of a switch (de Bruijn reverse graph).

Table 5.20: Searching the hops from source to destination node (de Bruijn rev.).

Mode	**Code snippet**

	NODE{ i }
	top = -1;
	flag = 0;
	while (flag = 0) {
	top = top + 1;
	for (j = 0; j $\leq$ top; j++) {
	if ($\left\lfloor \frac{i}{k^{n-1-j}} \right\rfloor_{\|k} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{k^{top-j}} \right\rfloor_{\|k}$) {
	flag = 1;
	}
	}
	}
	for (m = top; m < n; m++) {
	NODE{ $\left(i/k + \left\lfloor \frac{\lfloor b/M \rfloor}{k^{m}} \right\rfloor_{\|k} \right)_{\|k^n}$ };
	}

5.14 Binary grid

This sort of undirected graph works with node identifiers in binary form (k-ary form for a generic k), in a way that neighbour nodes just swap one bit from one another (supposing $k = 2$). Therefore, the procedure for a node i to identify a neighbour may be comparing all bits between them both in order to spot the discordant bits. On the other hand, the expression to calculate the possible nodes on the way through the destination has been already cited in (4.16) for nodes identified in binary form, whereas (4.17) extends that expression to nodes identified in a generic k-ary form, where k may be any natural number.

It is to be noted that, although the expressions herein are cited with a generic k, the usual way to make it work as a ”binary” grid may imply the use of binary notation, where variable k may well be substituted with the value $k = 2$ on every appearance in the following expressions.

Regarding the expressions to compare the symbols (bits if $k = 2$) being part of a node identifier, it may be done by means of arithmetic of logical operations. As per the *arithmetic way*, the j-th symbol of node i is obtained by the expression $\left\lfloor \frac{i}{k^j} \right\rfloor_{|k}$, whereas the same symbol of destination node is attained by the expression $\left\lfloor \frac{\lfloor b/M \rfloor}{k^j} \right\rfloor_{|k}$, where variable j may go from 0 to $n-1$ to go through each symbol being part of both identifiers.

Therefore, in the cases where both expressions mismatch, then variable j portrays a discordant bit, which may be used to get to a possible next hop on the way to destination node. The internode port to send that message may be $M+j$, which joins node i with neighbour node $\left(i+k^j\right)_{|k^n}$.

Alternatively, a *logical way* may be put in place by comparing bit by bit (supposing $k=2$, this is, identifiers are quoted in binary format) by means of XOR logical operator, such that it gives 1 for a particular bit in case of a mismatch in the values of that bit j, or it gives 0 otherwise, where the former triggers the forwarding through port $M+j$.

As per the *layout*, there are k^n node identifiers, those being identified from 0 to k^n-1, following the setup exposed in Figure 4.18, where each particular node has M ports connected to hosts (going from 0 to $M-1$), whereas ports from M to $M+n-1$ are internode ports. The latter interconnect with other nodes, and it may be interesting to assign the value of j in each $M+j$ port to the bit being discordant between the two nodes sharing that link. Figure 5.42 depicts the proposed port layout.

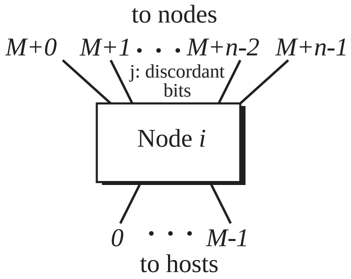


Figure 5.42: Model for a given node i in a Binary grid, along with its ports.

With that in mind, Figure 5.43 shows the *flow chart* exhibiting the behaviour of a switch i.

Besides, algorithm 17 (shown in the Appendix 10.17) exposes the *pseudocode model* related to the behaviour of a given switch i.

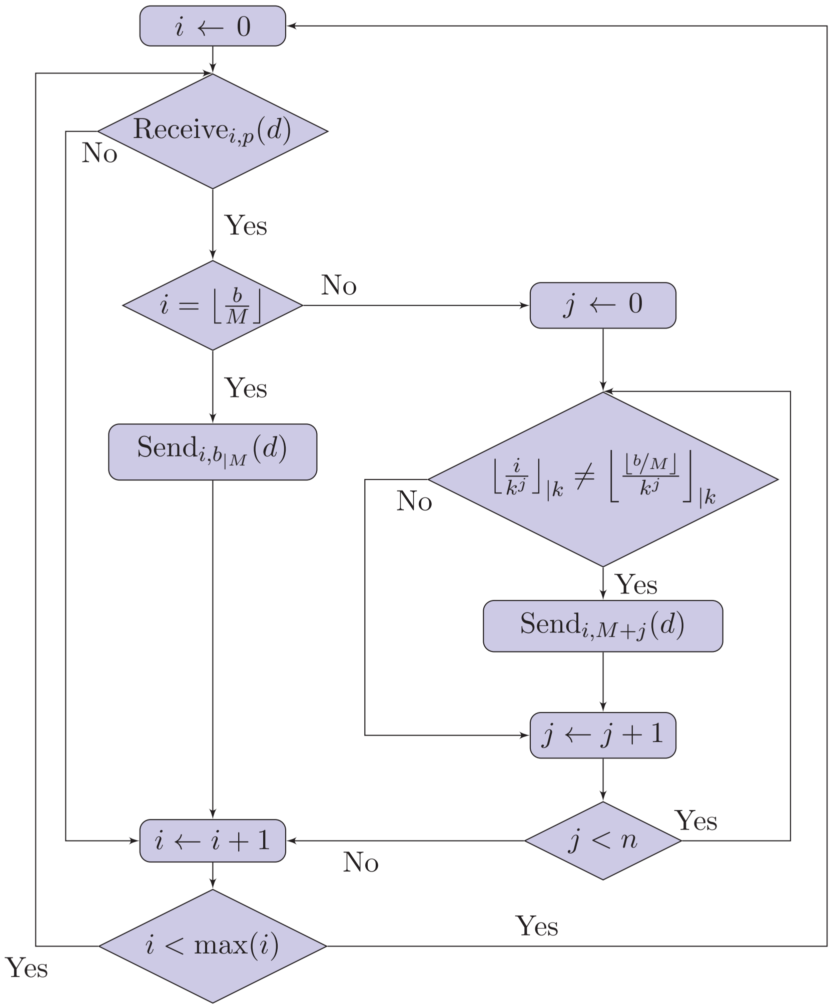


Figure 5.43: Flow chart for the behaviour of a given switch (binary grid).

Concerning the *algebraic model*, the ACP expression representing the behaviour of a particular switch i is exposed in (5.63). ACP does not take time into account, thus the initial conditional clause may be skipped, hence the first action may always be the reception of a message through any of its ports.

$$V_i = \sum_{i=0}^{k^n-1} \left(\sum_{p=0}^{M+3} \left(r_{V_i,p}(d) \cdot \left(s_{V_i,b_{|M}}(d) \triangleleft i = \left\lfloor \frac{b}{M} \right\rfloor \triangleright \right.\right.\right.$$

$$\left.\left.\left.\left(\sum_{j=0}^{n-1} s_{V_i,M+j}(d) \triangleleft \left\lfloor \frac{i}{k^j} \right\rfloor_{|k} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{k^j} \right\rfloor_{|k} \triangleright \emptyset \right) \right) \right) \right) \cdot V_i \tag{5.63}$$

Focusing on *redundant paths* within the model, it is to be reminded that graph-like designs may present a group of them, which depends on the distance between the end switches, as exposed for the N-hypercube.

Eventually, it is to be said that this topology is a graph-like one, as all its switches are end switches. Hence, its *verification* is similar to that stated for the N-hypercube.

5.15 Hamming graph

Among all possible Hamming graphs, only a particular kind is selected, such as $H_k(n, n-1)$, where k is the length of the alphabet (which is binary if $k = 2$), n is the length of the word and $n-1$ is minimum distance between nodes. It is to be noted that this topology may be seen as the reverse of the binary grid (if $k = 2$), but including a new link for opposite nodes, this is, those being the 1's complement of each other.

Hence, for each node identifier of length n, cited in k-ary format, there may be a link towards any other node whose corresponding digits has a difference of at least $n-1$ among its own identifier, thus becoming neighbour nodes. On the other hand, the expression to calculate the possible nodes on the way to the destination has been already cited in (4.18), (4.19) and (4.20), depending on the distance, for nodes identified in binary form, whereas (4.21), (4.22) and (4.23) extend those expressions to nodes identified in a generic k-ary form, where k may be any natural number.

As stated for the binary grid, it is to be noted that, although the expressions herein are cited with a generic k, the usual way to make it work as an extension of a reverse

”binary” grid may imply the use of binary notation, where variable k may well be substituted with the value $k = 2$ on every appearance in the following expressions. This way, $H_k(n, n-1)$ may be converted into $H_2(n, n-1)$.

The way to proceed with a binary alphabet may be to first evaluate the distance between node i and destination node $\lfloor b/M \rfloor$, measured in number of different binary digits between their node identifiers, which may be referred to as variable *diff*. Once that value is achieved, some possibilities arise. To start with, if all digits are discordant, such that $\mathit{diff} = n$, then the link to the opposite node may be used. Likewise, if more than half of the digits are discordant, such as $\mathit{diff} > n/2$, then only the appropriate links to non-opposite nodes are taken, this is, links with just one matching bit. In case $\mathit{diff} = n-1$, just the appropriate direct link towards that node is employed.

Otherwise, if less than half of the digits are discordant, such that $\mathit{diff} < n/2$, then the opposite link is taken, because it makes for the shortest path in that case. Additionally, if just half of the digits are discordant, such as $\mathit{diff} = n/2$, then all links are used in order to take advantage of all load-balancing equal cost paths.

As in the previous section, the bit checking (supposing k=2) may be done through arithmetic means by using the same expressions quoted above, such as $\left\lfloor \frac{i}{2^j} \right\rfloor_{|2}$ for the node i and $\left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2}$ for the destination node in binary format. Alternatively, the k-symbol checking may involve the use of $\left\lfloor \frac{i}{k^j} \right\rfloor_{|k}$ for the node i and $\left\lfloor \frac{\lfloor b/M \rfloor}{k^j} \right\rfloor_{|k}$ for the destination node for a generic k-ary format. Additionally, that checking may also be done through logical means by using the *XOR* logical operator as cited above.

As per the *layout*, there are k^n node identifiers, going from 0 to $k^n - 1$, according to the setup depicted in Figure 4.19. Every particular node has M ports connecting to hosts (those going from 0 to $M-1$), whilst ports from M to $M+n$ are internode ports, which interconnect with other nodes. It may be interesting to assign the value of j in each $M+j$ port to the matching bit between the two nodes sharing that link. This way, as each node is identified by n digits, going from 0 to $n-1$, then those may be the values for j reserved for the cases where distance is $n-1$, whilst links going to the opposite node, where distance is n, are identified by $M+n$. Figure 5.44 exhibits the proposed port layout.

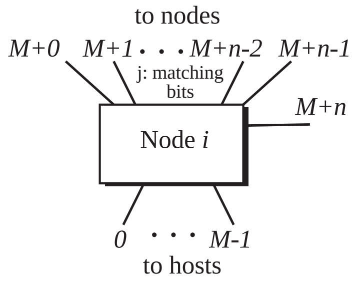


Figure 5.44: Model for a node i in a Hamming graph $H_k(n, n-1)$ and its ports.

With this in mind, Figure 5.45 shows the *flow chart* exhibiting the behaviour of a switch i.

Moreover, the *pseudocode model* regarding the behaviour a given switch i is shown in algorithm 18 (shown in the Appendix 10.18), where the mismatching and matching bits are found out by means of arithmetic operations.

With regards to the *algebraic model*, the ACP expression representing the behaviour of a switch in presented in (5.65), where one auxiliary function called *mismatchingBits* has been defined in order to measure the difference in bits between the current switch i and the destination node, which is shown in expression (5.64).

In that function, the initial value of the variable accouting for bit difference, called *diff*, is first initialized (~*diff*), and in turn, the mismatching bits are counted up (*diff*++) in the function *mismatchingBits*.

The results of this function is being applied in the V_i model to select the appropriate option, and in turn, to apply the proper operations in order to send the message through the proper ports. Furthermore, the conditional clause in the receive function has been dropped as ACP does not consider the time. Additionally, it is worth reminding that k accounts for a generic length of the alphabet being use, although binary format (where $k = 2$) is the usual way to get a kind of reverse binary grid.

$$mismatchingBits = {\sim}diff \cdot \sum_{j=0}^{n-1} \left(\left(diff{+}{+} \right) \lhd \left\lfloor \frac{i}{k^j} \right\rfloor_{|k} \neq \left\lfloor \frac{\lfloor b/M \rfloor}{k^j} \right\rfloor_{|k} \rhd \emptyset \right) \tag{5.64}$$

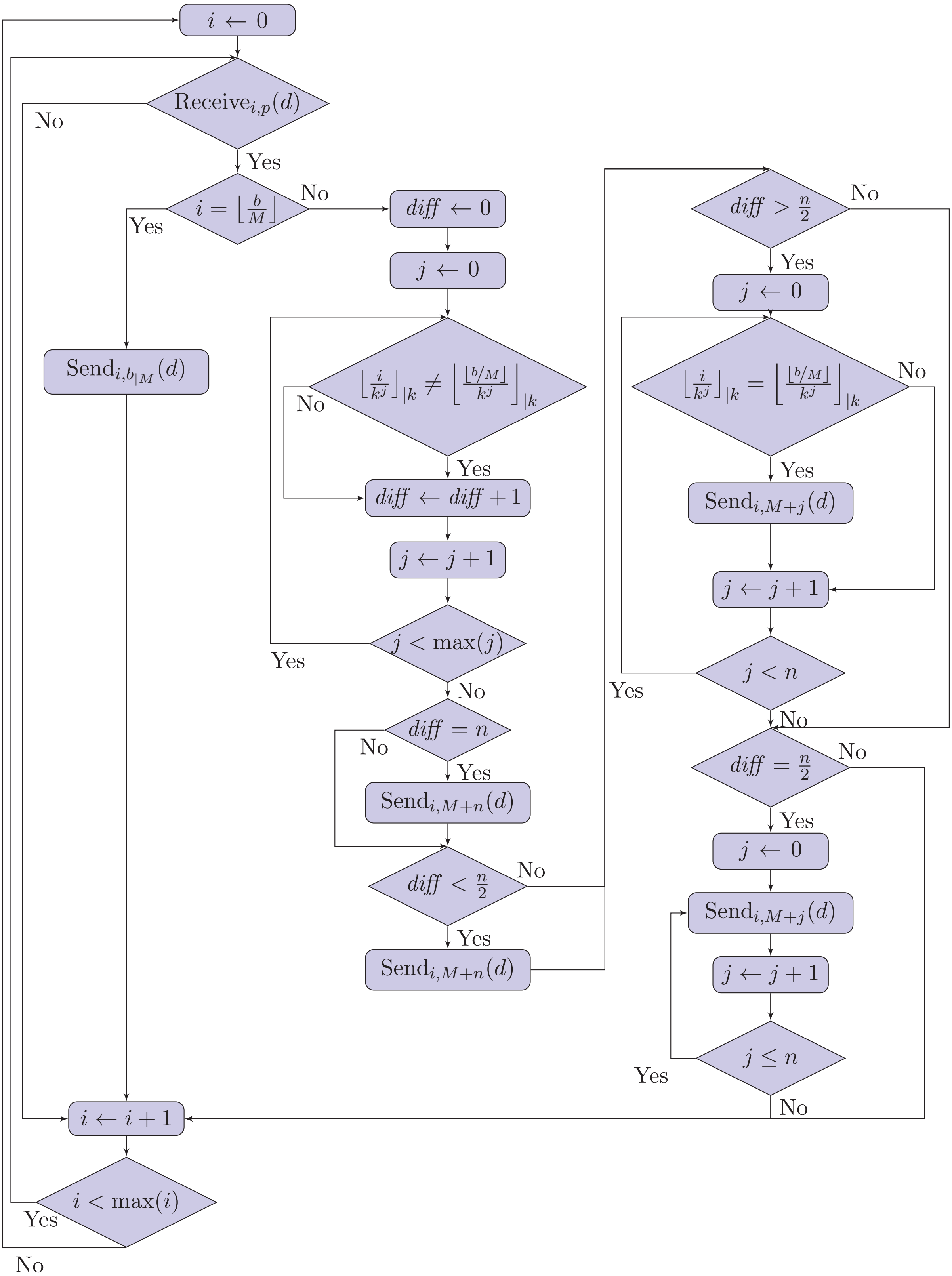

$i \leftarrow 0$
Receive$_{i,p}(d)$
No
Yes
$i = \lfloor \frac{b}{M} \rfloor$
No
Yes
Send$_{i,b_{|M}}(d)$
$\mathit{diff} \leftarrow 0$
$j \leftarrow 0$
$\lfloor \frac{i}{k^j} \rfloor_{|k} \neq \lfloor \frac{\lfloor b/M \rfloor}{k^j} \rfloor_{|k}$
No
Yes
$\mathit{diff} \leftarrow \mathit{diff} + 1$
$j \leftarrow j + 1$
$j < \max(j)$
Yes
No
$\mathit{diff} = n$
No
Yes
Send$_{i,M+n}(d)$
$\mathit{diff} < \frac{n}{2}$
No
Yes
Send$_{i,M+n}(d)$
$\mathit{diff} > \frac{n}{2}$
No
Yes
$j \leftarrow 0$
$\lfloor \frac{i}{k^j} \rfloor_{|k} = \lfloor \frac{\lfloor b/M \rfloor}{k^j} \rfloor_{|k}$
No
Yes
Send$_{i,M+j}(d)$
$j \leftarrow j + 1$
$j < n$
Yes
No
$\mathit{diff} = \frac{n}{2}$
No
Yes
$j \leftarrow 0$
Send$_{i,M+j}(d)$
$j \leftarrow j + 1$
$j \leq n$
Yes
No
$i \leftarrow i + 1$
$i < \max(i)$
Yes
No

Figure 5.45: Flow chart for the behaviour of a switch (Hamming graph $H_k(n, n-1)$).

$$
\begin{aligned}
V_i = \sum_{i=0}^{k^n-1} \Bigg(\sum_{p=0}^{M+n} \Bigg(r_{V_i,p}(d) \cdot \Bigg(s_{V_i,b_{|M}}(d) \lhd i = \left\lfloor \frac{b}{M} \right\rfloor \rhd \\
\Bigg(\Bigg(s_{V_i,M+n}(d) \lhd \mathit{mismatchingBits} = n \rhd \emptyset \Bigg) \cdot \\
\Bigg(s_{V_i,M+n}(d) \lhd \mathit{mismatchingBits} < \frac{n}{2} \rhd \emptyset \Bigg) \cdot \\
\Bigg(\Bigg(\sum_{j=0}^{n-1} s_{V_i,M+j}(d) \lhd \left\lfloor \frac{i}{2^j} \right\rfloor_{|2} = \left\lfloor \frac{\lfloor b/M \rfloor}{2^j} \right\rfloor_{|2} \rhd \emptyset \Bigg) \lhd \mathit{mismatchingBits} > \frac{n}{2} \rhd \emptyset \Bigg) \cdot \\
\Bigg(\sum_{j=0}^{n} s_{V_i,M+j}(d) \lhd \mathit{mismatchingBits} = \frac{n}{2} \rhd \emptyset \Bigg) \Bigg) \Bigg) \Bigg) \Bigg) \cdot V_i
\end{aligned}
\tag{5.65}
$$

Focusing on *redundant paths* within the model, it is to be noted that graph-like designs may present some of them, which depends on the distance between the end switches, as exposed for the N-hypercube.

Eventually, it is to be reminded that this topology is a graph-like one, meaning that all of its switches are end switches. Hence, its *verification* procedure is analogous to that cited for the N-hypercube.

5.16 Petersen graph

Regarding this kind of undirected graph, it is also known as $(3, 5)$-cage graph, where its 10 nodes are equally distributed into two concentric rings. Those nodes are identified in decimal format, where the outer ones are joined together forming a pentagon and the inner ones are forming a star pentagon, hence the former may be numerated clockwise from 0 to 4, and so may the latter from 5 to 9, where there are additional connections between each pair of nodes being congruent modulo 5, such that any node has a direct link to its modulo 5 counterpart within the opposite ring.

This distribution formed by 10 nodes and 15 edges presents a diameter of 2, hence any pair of nodes are at most 2 hops away, as it may be appreciated in Figure 5.46. Additionally, it may be seen that each node has degree 3, as that is the number of links pointing towards other nodes, whereas the shortest loops need to run through 5 links around the topology.

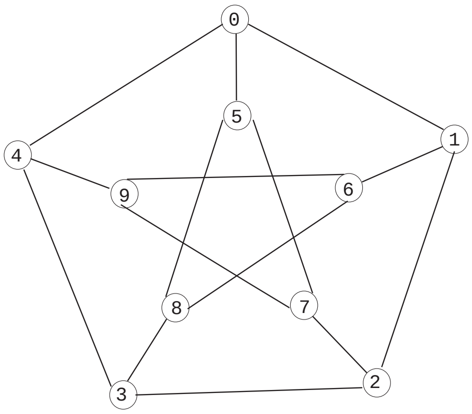


Figure 5.46: Node distribution in a Petersen graph.

That way, when getting from an outer node towards an inner node, it must be checked whether the destination node $\left\lfloor \frac{b}{M} \right\rfloor$ is congruent modulo 5 with the current node i, or alternatively, with $i + 2$, or even with $i + 3$, and if that is the case, the next hop may be towards the opposite node in the inner ring, hence adding five units. On the other hand, the next hop may be towards the successor node modulo 5 if the destination node is congruent modulo 5 with $i + 1$, or otherwise, towards the predecessor node modulo 5 if the destination node is congruent modulo 5 with $i + 4$, where both next hops are part of the outer ring.

Moreover, when getting from an inner node towards an outer node, it must be verified whether the destination node $\left\lfloor \frac{b}{M} \right\rfloor$ is congruent modulo 5 with the current node i, or alternatively, with $i + 1$, or even with $i + 4$, and if this is the case, the next hop may be towards the opposite node in the outer ring, hence subtracting five units. On the contrary, the next hop may be towards the successor node modulo 5 if the destination node is congruent modulo 5 with $i + 3$, or otherwise, towards the predecessor node modulo 5 if the destination node is congruent modulo 5 with $i + 2$, where both next hops are part of the inner ring.

Besides, when going between any pair of outer nodes, it may be as easy as running around the polygon, thus the next hop may be moving to the successor node modulo 5, or to the predecessor node modulo 5, both within the pentagon. Therefore, the former is like adding up one unit modulo 5 if the distance modulo 5 between the destination node $\left\lfloor \frac{b}{M} \right\rfloor$ and the current node i is less than $\frac{N}{2}$, which may be expressed by $(\left\lfloor \frac{b}{M} \right\rfloor - i)_{|5} < \frac{N}{2}$, thus covering the case of $i+1$, and in turn $i+2$, or otherwise, the latter is like subtracting one unit modulo 5 if such a distance modulo 5 is higher than $\frac{N}{2}$, hence covering the case of $i+4$, and in turn $i+3$. It is to be said that the current node i needs to be taken as a reference for the addition or subtraction operations described.

Furthermore, when running between any pair of inner nodes, it may be seen as going around the star pentagon, thus the next hop may be moving to the successor node modulo 5 with offset 5 or to the predecessor node modulo 5 with offset 5, both within the star. Hence, the former is like adding up three units modulo 5 with offset 5 and it may be used if the destination node $\left\lfloor \frac{b}{M} \right\rfloor$ has the opposite parity as the current node i, thus covering the case of $i+3$, and in turn $i+1$, where the latter is like adding up two units modulo 5 and it may be employed if the destination node has the same parity as the current node, hence covering the case of $i+2$, and in turn $i+4$. It is to be said that the current node i needs to be taken as a reference for the parity measurements.

Eventually, it is to be noted that any node i located in the outer ring results in $\left\lfloor \frac{i}{5} \right\rfloor = 0$, whereas in the inner ring does it in $\left\lfloor \frac{i}{5} \right\rfloor = 1$. Therefore, by comparing the values of $\left\lfloor \frac{i}{5} \right\rfloor$ and $\left\lfloor \frac{\lfloor b/M \rfloor}{5} \right\rfloor$, it may be possible to distinguish the four scenarios described related to movements among nodes, either being located in the outer ring or the inner ring.

As per the *layout*, the 10 nodes available are identified going from 0 to 9, as stated above. Furthermore, each given node has M ports connecting to hosts (these going from 0 to $M-1$), whilst ports from M to $M+2$ are interconnecting with other nodes.

Specifically, port M is aimed to the predecessor node modulo 5, regardless whether the ring referred to is the outer ring (a regular pentagon) or the inner ring (a star pentagon), which may be calculated by adding (-1 modulo 5, which is equivalent to +4 modulo 5) for the nodes in the outer ring, or otherwise, by adding (+2 modulo 5) for nodes in the inner ring. Likewise, port $M+1$ is going to the successor node modulo 5, which may be found by adding (+1 modulo 5) for the nodes in the outer ring, or otherwise, by adding (+3 modulo 5) for nodes in the inner ring. Furthermore, port

$M+2$ is a bridge to the node being congruent modulo 5, also known as opposite node, thus being located just the opposite within the other ring, which may be determined by adding (+5 modulo 10) for nodes located in any of both rings.

Figure 5.47 exhibits the proposed port layout, where the three upper links are branded as *successor*, *predecessor* and *opposite*.

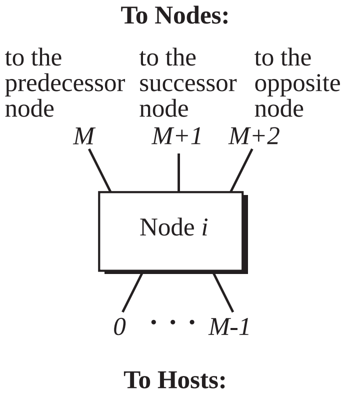


Figure 5.47: Model for a given node i in a Petersen graph, along with its ports.

Taking that into consideration, Figure 5.48 shows the *flow chart* exhibiting the behaviour of a switch i.

Furthermore, the *pseudocode model* regarding the behaviour of a given switch i is shown in algorithm 19 (shown in the Appendix 10.19).

The ACP expression modelling the behaviour of a particular switch i is exposed in (5.66). ACP does not take time into account, thus the initial conditional clause may be skipped, hence the first action may always be the reception of a message through any of its ports.

$$\begin{aligned}
V_i = &\sum_{i=0}^{9} \left(\sum_{p=0}^{M+2} \left(r_{V_i,p}(d) \cdot \left(s_{V_i,b_{|M}}(d) \lhd i = \left\lfloor \frac{b}{M} \right\rfloor \rhd \right.\right.\right. \\
&\left(\left(\left(\left(s_{V_i,M+2}(d) \lhd \left\lfloor \frac{b}{M} \right\rfloor_{|5} \in \{ i_{|5}, (i+2)_{|5}, (i+3)_{|5} \} \rhd \emptyset \right) \cdot \right.\right.\right. \\
&\left.\left(s_{V_i,M+1}(d) \lhd \left\lfloor \frac{b}{M} \right\rfloor_{|5} = (i+1)_{|5} \rhd \emptyset \right) \cdot \left(s_{V_i,M}(d) \lhd \left\lfloor \frac{b}{M} \right\rfloor_{|5} = (i+4)_{|5} \rhd \emptyset \right)\right) \\
&\left. \lhd \left\lfloor \frac{i}{5} \right\rfloor < \left\lfloor \frac{\lfloor b/M \rfloor}{5} \right\rfloor \rhd \emptyset \right) \cdot \\
&\left(\left(\left(s_{V_i,M+2}(d) \lhd \left\lfloor \frac{b}{M} \right\rfloor_{|5} \in \{ i_{|5}, (i+1)_{|5}, (i+4)_{|5} \} \rhd \emptyset \right) \cdot \right.\right. \\
&\left.\left(s_{V_i,M+1}(d) \lhd \left\lfloor \frac{b}{M} \right\rfloor_{|5} = (i+3)_{|5} \rhd \emptyset \right) \cdot \left(s_{V_i,M}(d) \lhd \left\lfloor \frac{b}{M} \right\rfloor_{|5} = (i+2)_{|5} \rhd \emptyset \right)\right) \\
&\left. \lhd \left\lfloor \frac{i}{5} \right\rfloor > \left\lfloor \frac{\lfloor b/M \rfloor}{5} \right\rfloor \rhd \emptyset \right) \cdot \\
&\left(\left(s_{V_i,M+1}(d) \lhd (\left\lfloor \frac{b}{M} \right\rfloor - i)_{|5} < N/2 \rhd s_{V_i,M}(d) \right)\right. \\
&\left. \lhd \left\lfloor \frac{i}{5} \right\rfloor = \left\lfloor \frac{\lfloor b/M \rfloor}{5} \right\rfloor = 0 \rhd \emptyset \right) \cdot \\
&\left(\left(s_{V_i,M+1}(d) \lhd i_{|2} \neq \left\lfloor \frac{b}{M} \right\rfloor_{|2} \rhd s_{V_i,M}(d) \right)\right. \\
&\left.\left.\left.\left.\left. \lhd \left\lfloor \frac{i}{5} \right\rfloor = \left\lfloor \frac{\lfloor b/M \rfloor}{5} \right\rfloor = 1 \rhd \emptyset \right)\right)\right)\right)\right) \cdot V_i
\end{aligned} \tag{5.66}$$

Focusing on *redundant paths* within the model, it is to be reminded that graph-like designs may present a range of them, depending on the distance between the end switches, as exposed for the N-hypercube.

Eventually, it is to be considered that this topology is a graph-like one, meaning that all switches are end switches. Hence, its *verification* procedure is similar to that quoted for the N-hypercube.

5.17 Heawood graph

This type of undirected graph is also known as $(3, 6)$-cage graph, where its 14 nodes are equally distributed along a single ring. Those nodes are identified in decimal

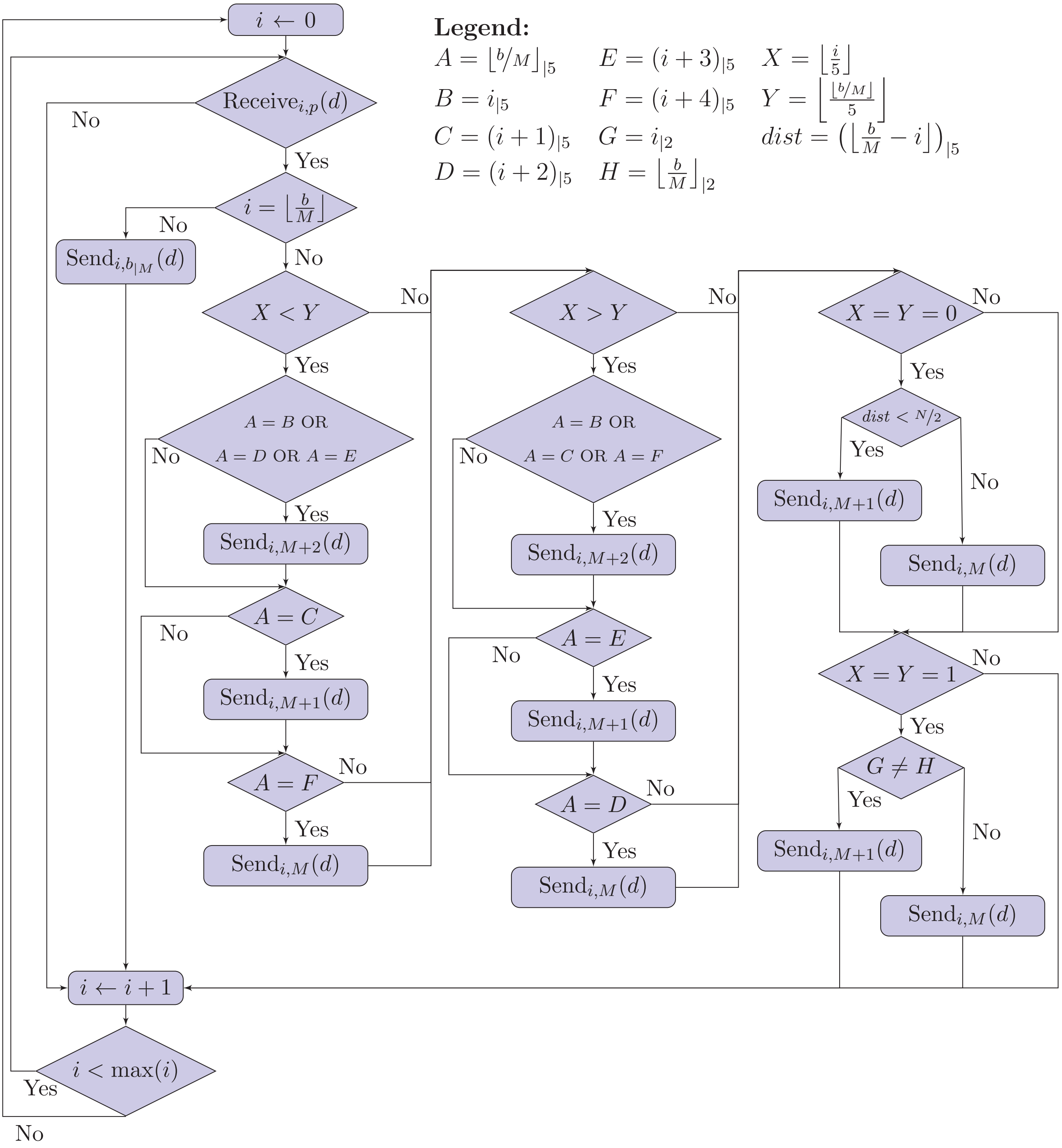


Figure 5.48: Flow chart for the behaviour of a given switch (Petersen graph).

format, where each one gets a number in a clockwise manner (from 0 to 13). Each particular node i is linked together with its *successor* $(i+1)$ and its *predecessor* $(i-1)$ along the ring, both taken in modulo 14 notation, and furthermore, with a *remote* node, being the result of adding up i to the expression $(-1)^i \times 5$, also taken in modulo 14 notation.

Furthermore, this distribution formed by 14 nodes and 21 edges has a diameter of 3, thus any pair of nodes are at most 3 hops away. Additionally, it is to be noted that each node has a degree 3, as that is the number of internode links for all nodes, whilst the shortest loops need to run through 6 links around the topology. This facts may be appreciated in Figure 5.49.

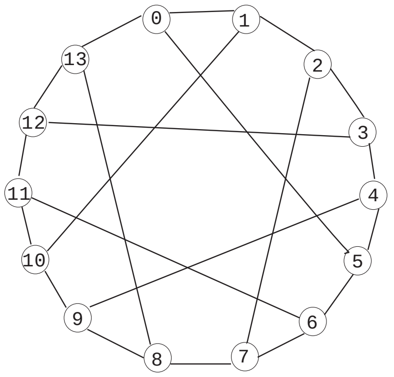


Figure 5.49: Node distribution in a Heawood graph.

Focusing on the link to the remote node, the aforesaid expression $(i+(-1)^i \times 5)_{|14}$ results in any even node i having a link going to node $i+5$ modulo 14 (thus moving up to 5 nodes clockwise), whilst any odd node i doing it to node $i-5$ modulo 14 (hence moving up to 5 nodes counterclockwise).

On the other hand, if *distance* between the current node i and the destination node $\lfloor \frac{b}{M} \rfloor$ is defined as $dist = \lfloor \frac{b}{M} \rfloor - i$, and taking into account that distances higher than $|7|$ (regardless the sign) may get a shorter way going the other way around by applying the expression $distance = 14 - distance$ (for being in a closed loop of 14 nodes, where obviously the movements $i+7$ and $i-7$ are equivalent), then all case studies regarding the movement between any pair of nodes get reduced to just 5 different scenarios, where the departing point is supposed to be node i.

On the one hand, the first one is pretty straightforward, as the movement to nodes $i+1$, $i+2$ and $i+3$ (all of them taken in modulo 14) only takes the successor link in all intermediate nodes.

Likewise, the second one is also pretty clear, as the movement to nodes $i-1$, $i-2$ and $i-3$ (all of them taken in modulo 14) only needs the predecessor link in all intermediate nodes.

Regarding the rest of the cases, it may be interesting to distinguish between even and odd values of i. At this point, supposing that i has an *even* value, then the movement to nodes $i+4$, $i+5$, $i+6$ and $i+7$ (all of them taken in modulo 14) is done through the remote link of i, as such a link makes the direct move to $i+5$, and from there on, it is possible to move to $i+4$ through its predecessor link, or otherwise, it is also possible to move to $i+6$ through its successor link, and from there, moving to $i+7$ through its successor link.

However, the movement to node $i-4$, and then to node $i-5$ (both taken in modulo 14), requires to first move to $i+1$ through the successor link of i, and then, to take its remote link to reach $i-4$, and from there on, it is possible to get to $i-5$ through its predecessor link.

On the other hand, the movement to node $i-6$ (taken in modulo 14) requires to first move to $i-1$ through the predecessor link of i, and in turn, to take its remote link to reach $i-6$. In summary, it is possible to reach any node from a given even node with at most three hops.

At this stage, supposing that i has an *odd* value, the movements are analogous to those exposed for the previous case, but changing all signs exposed above. This way, the move to nodes $i-4$, $i-5$, $i-6$ and $i+7$ (all of them taken in modulo 14) by taking the remote link of i, which goes to $i-5$, and from there on, the move to $i-4$ is done through its successor link, or otherwise, the move to $i-6$ and $i+7$ is done through its correspondent predecessor links.

Instead, the move to node $i+4$, and then to node $i+5$ (both taken in modulo 14), requires to first moving to $i-1$ through the predecessor link of i, and in turn, to take its remote link to reach $i+4$, and from there on, to get to $i+5$ through its successor link.

On the contrary, the movement to node $i+6$ (taken in modulo 14) requires to first move to $i+1$ through the successor link of i, and then, to take its remote link to reach $i+6$. In summary, it is possible to reach any node from a particular odd node with at most three hops.

Regarding the implementation of all those cases, it is to be noted that if i is even, then $(-1)^i = +1$, whereas if i is odd, then $(-1)^i = -1$. Furthermore, it is to be said that if i is even, then $(-1)^{i+1} = -1$, whilst if i is odd, then $(-1)^{i+1} = +1$. These expressions may help express movements in both directions to get to the rest of possible destination nodes from a given node i.

Besides, the port number $M+1$ for even values of i may be obtained with the expression $M + (1+(-1)^i)^{1+(-1)^{i+1}}$, which at the same time gives port number M for odd values of i, whereas the expression $M + (1+(-1)^{i+1})^{1+(-1)^i}$ assigns the port number M for even values and $M+1$ for odd values. Those expressions are to be applied where distance is greater than 3 in absolute value, whilst values of shorter distances may need the expression $M + (1+\frac{dist}{|dist|})^{1-\frac{dist}{|dist|}}$ to relate to port $M+1$ regarding positive distances and port M for negative ones.

Additionally, it is to be noted that the aforesaid movements are enough to get to all nodes, departing from any given source node i. Nonetheless, apart for those movements described, it is possible to establish alternative redundant paths to reach some destinations by completing up to 3 steps in each available path. In that sense, it is to be considered up to 4 further moves through the perimeter and other 4 further moves through the diagonals.

With respect to the former, it is possible to go from $i+(-1)^i \times 4$ to $i+(-1)^i \times 3$, from $i+(-1)^{i+1} \times 4$ to $i+(-1)^{i+1} \times 3$, from $i+(-1)^{i+1} \times 6$ to $i+(-1)^{i+1} \times 5$ or from $i+(-1)^{i+1} \times 6$ to $i+7$, all of them applying modulo 14.

With regards to the latter, it is possible to go from $i+(-1)^i \times 2$ to $i+7$, from $i+(-1)^{i+1} \times 2$ to $i+(-1)^i \times 3$, from $i+(-1)^i \times 6$ to $i+(-1)^{i+1} \times 3$ or from $i+(-1)^i \times 4$ to $i+(-1)^{i+1} \times 5$, all of them modulo 14.

However, in order to keep things as simple as possible, neither of those 8 redundant paths are considered in the following representations, as they may only be taken if, and only if, the number of hops away from the source node i is less than 3, which in some cases may only be achieved by means of an additional register to count up the number of steps so far.

As per the *layout*, it is to be mentioned that every given node has M ports connecting to hosts, those going from 0 to $M-1$, whereas there are 3 internode ports, such that M, $M+1$ and $M+2$. As stated before, port M is directed to the predecessor node modulo 14, port $M+1$ goes to the successor node modulo 14, and port $M+2$ does it to a remote node, which moves 5 nodes clockwise if i is even, or otherwise, 5 nodes anticlockwise if i is odd.

Figure 5.50 exhibits the proposed port layout, where the three upper links are branded as *successor*, *predecessor* and *remote*.

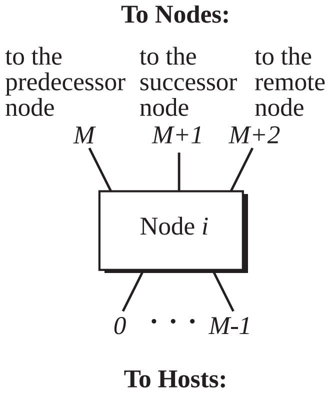


Figure 5.50: Model for a given node i in a Heawood graph, along with its ports.

Taking this into account, Figure 5.51 shows the *flow chart* exhibiting the behaviour of a switch i.

Moreover, the *pseudocode model* related to the behaviour of a given switch i is shown in algorithm 20 (shown in the Appendix 10.20).

The ACP expression to model the behaviour of a switch in presented in (5.68), where one auxiliary function called *Distance* has been defined in order to measure the shortest distance measured in nodes away between the current switch i and the destination node $\lfloor \frac{b}{M} \rfloor$, which is shown in expression (5.67).

In this function, the initial value of the variable citing the distance, called *dist*, is first initialized ($\sim$*dist*), and in turn, it is assigned to the subtraction of the destination node minus the current switch, which may be further adjusted so as to reflect the circular nature of the graph, thus accounting for values not exceeding 7 in absolute value in any case.

The result of this function needs to be applied in the V_i model to select the pertinent option, and in turn, to apply the appropriate operators in order to send the message through the correspondent ports. Furthermore, the conditional clause in the receive function has been dropped as ACP does not consider the time.

$$\begin{aligned} Distance = {\sim}dist \cdot \left(dist = \left\lfloor \frac{b}{M} \right\rfloor - i \right) \cdot \\ \left(dist = -(14 - dist) \lhd dist > 7 \rhd \left(dist = -(-14 - dist) \lhd dist \leq -7 \rhd \emptyset \right) \right) \end{aligned} \tag{5.67}$$

$$\begin{aligned} V_i = \sum_{i=0}^{13} \left(\sum_{p=0}^{M+2} \left(r_{V_i,p}(d) \cdot \left(s_{V_i,b_{|M}}(d) \lhd i = \left\lfloor \frac{b}{M} \right\rfloor \rhd \right.\right.\right. \\ \left(\left(s_{V_i,M+(1+\frac{dist}{|dist|})^{1-\frac{dist}{|dist|}}}(d) \lhd Distance \in \{ \frac{dist}{|dist|} \times 1, \frac{dist}{|dist|} \times 2, \frac{dist}{|dist|} \times 3 \} \rhd \emptyset \right) \cdot \right. \\ \left(s_{V_i,M+2}(d) \lhd Distance \in \{ (-1)^i \times 4, (-1)^i \times 5, (-1)^i \times 6, +7 \} \rhd \emptyset \right) \cdot \\ \left(s_{V_i,M+(1+(-1)^i)^{1+(-1)^{i+1}}}(d) \lhd Distance \in \{ (-1)^{i+1} \times 4, (-1)^{i+1} \times 5 \} \rhd \emptyset \right) \cdot \\ \left.\left.\left.\left. \left(s_{V_i,M+(1+(-1)^{i+1})^{1+(-1)^i}}(d) \lhd Distance \in \{ (-1)^{i+1} \times 6 \} \rhd \emptyset \right) \right) \right) \right) \right) \cdot V_i \end{aligned} \tag{5.68}$$

Focusing on *redundant paths* within the model, it is to be noted that graph-like designs may present a group of them, depending on the distance between the end switches, as exposed for the N-hypercube.

Eventually, it is to be noted that this topology is a graph-like one, meaning that all its switches are end switches. Hence, its *verification* procedure is analogous to that explained for the N-hypercube.

5.18 Summary

In this chapter, each of the 15 DC topologies presented in the previous one have been modelled. To begin with, the nomenclature to be used herein has been depicted,

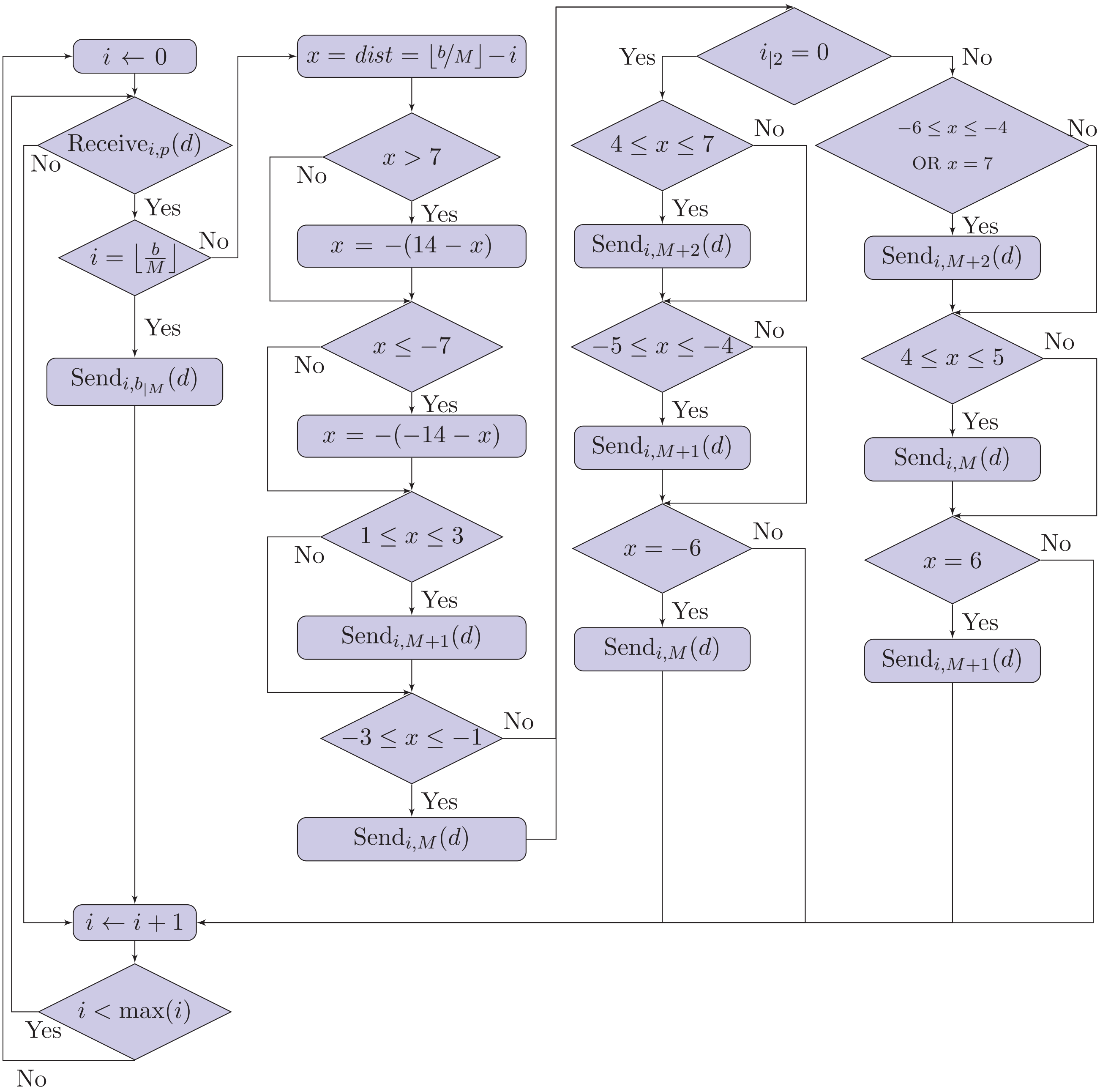


Figure 5.51: Flow chart for the behaviour of a given switch (Heawood graph).

as well as the model to identify each host within the topology. Afterwards, for each of the aforementioned DC topologies, the following scheme has been applied: first of all, a short preface has been introduced so as to explain its main features, then, the layout of the model is exhibited with the identification of nodes and edges, next, a flow chart stating how the model behaves is proposed, after that, a pseudocode model based on that flow chart is referred to in the Appendix chapter.

At that point, an algebraic model by means of ACP is proposed, which is further used to deduce the redundant paths between a given source host and a particular destination host, and eventually, a verification of that model is performed by applying the ACP axioms in order to obtain a rooted branching bisimulation between the aforesaid model and the behaviour of the real system, meaning that both expressions share the same string of actions and the same branching structure, which may imply that the model gets verified, meaning that for any input in the model, the expected output is achieved.

In the following chapter, the DC topology is going to be abstracted away, thus allowing any of the aforementioned topologies to be employed, which may be part of a generic fog environment with a cloud acting as a backup system.

Chapter 6

Generalization of Data Centre models with ACP

Once the different DC topology models proposed have been developed, it is time to abstract away from them and consider the DC as a black box within a bigger system, which is the fog environment, which may well be considered as an IoT/fog ecosystem. Therefore, let us present the building blocks of a fog domain in order to obtain an appropriate algebraic model.

6.1 Topology framework for a generic fog model

To start with, it is to be reminded that the scenario being modelled is a fog computing environment, where moving IoT devices may be running around [189]. It is also to be remarked that, when an IoT device moves around, its associated computing assets might need to be allocated in different hosts within the fog domain being as close as possible to the position occupied by that device at any time, hence performing a VM migration between the source host and a destination host with enough resources, whilst having a cloud system being used as a backup solution.

The framework proposed is an informal model composed by 5 layers, where each one has a different role within the topology, although they work all together for the system to behave the way it is meant to be. It is to be noted that the set of all moving

IoT devices are not being considered as an independent layer, because they are the external factors triggering the different behaviours offered by the fog environment. Figure 6.1 exhibits the organization of such layers.

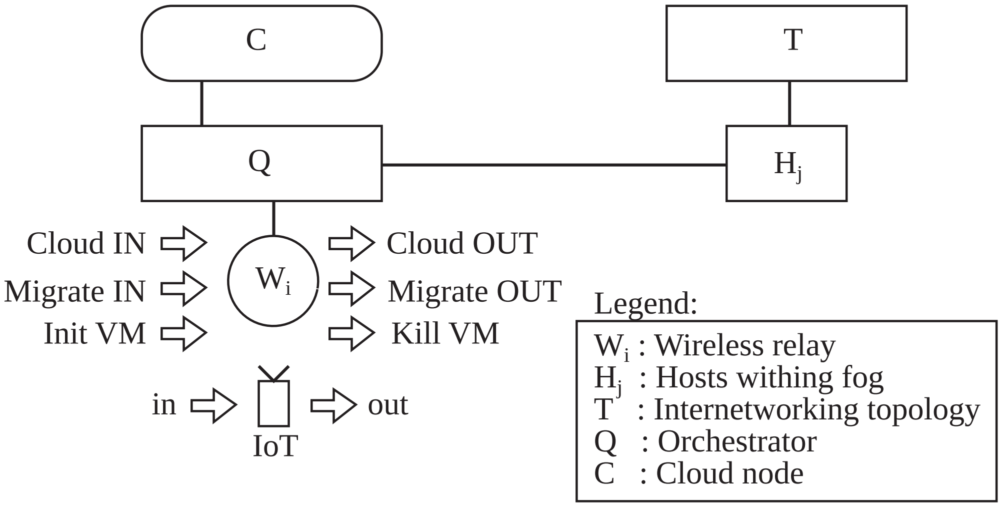


Figure 6.1: Generic topology scheme for an IoT/fog model.

First of all, *wireless layer* is the lower one, and that is where each moving IoT device gets connected to the fog environment, hence being the interface between the device and the fog domain. Specifically, when a moving device gets into the coverage area of a given wireless relay i and gets connected to it, being expressed as W_i, such a moving device gets access to its associated VM through the fog infrastructure. Otherwise, when a moving devices gets out of the coverage area of i and gets disconnected to it, such a VM gets out of the system.

After that, *orchestration layer* is the key one, as its role is managing the behaviour of the whole system, by means of selecting the host where each VM is located, where a VM is migrated, creating a new VM, terminating an existing VM, and even interacting with the cloud so as to use it as a backup system for VMs. In the scheme proposed herein, only one orchestrator is assigned, which is identified by Q, although more than one physical device may act as a virtual device by applying high availability policies.

Next, the *host layer* is where hosts are located, whose role is to allocate the different associated VMs. Each host may be identified by j, being expressed as H_j. It is important to note that hosts are ubicated all around the coverage area of the fog environment, even though a particular host might be the closest to a group of wireless relays, and in such a case, it would not be necessary to carry out any migration whatsoever when moving within the coverage areas of those wireless relays, thus the orchestration layer may not manage any migration in those cases.

Then, the *topology layer* is related to the interconnection network connecting all hosts together within a given topology in order to communicate to each other. Many different interconnection topologies have been studied in the previous chapter, even though in this case, an abstract topology is considered, being identified by T, as any topology may do the job. It is to be pointed out that the migration decisions are taken by the orchestrator, regarding the destination host and the appropriate paths to get there from the source host, which depends on the topology considered.

Eventually, the *cloud layer* is used as a backup solution for the fog environment, which is identified by C. It is to be remarked that the cloud is not managed by the orchestrator, as that is done by a cloud service provider. This way, the orchestrator just undertakes the transmision of VMs from fog to cloud, and the other way around, according to the resources available in the fog, as the cloud is acting as a backup service, which is independently run from the fog system.

On the other hand, it is important to define the *communication channels* among those layers. In this sense, a specific nomenclature is used to identify those channels and their directions, in a way that each channel may be named by the source layer, followed by the destination layer, thus defining each channel as unidirectional. In this context, Table 6.1 presents all communication channels being used in this model.

It is to be noted that the orchestrator is the main element, as it is the central point of the fog system, and it makes all decisions, which in turn, are communicated to the other layers to execute them. This way, all actions are triggered at wireless layer, then, those actions are planned at orchestration layer, and finally, they are run at the corresponding layers according to the orchestrator instructions.

Apart from the internal channels among these layers, which take part of the generic fog computing environment presented herein, there are other 2 external wireless physical channels. One is labelled as *in*, which it is related to the way for a moving device to get into the fog domain by connecting through a wireless relay, whereas the other one is labelled *out*, which it is referred to the way for a moving device to get out of the fog domain by disconnecting from a wireless relay.

Table 6.1: Communication channels in the model.

Channel ID	**Direction**	**Meaning**
W_iQ	Incoming	Channel from the wireless relay W_i to orchestrator Q
QW_i	Outgoing	Channel from orchestrator Q to the wireless relay W_i
QH_j	Incoming	Channel from orchestrator Q to the host H_j
H_jQ	Outgoing	Channel from the host H_j to orchestrator Q
H_jT	Incoming	Channel from the host H_j to switching topology T
TH_j	Outgoing	Channel from switching topology T to the host H_j
QC	Incoming	Channel from orchestrator Q to the cloud C
CQ	Outgoing	Channel from the cloud C to orchestrator Q
in	Inwards	Wireless physical channel coming into wireless relay W_i
out	Outwards	Wireless physical channel going out of wireless relay W_i

6.2 Modelling VM actions within a fog environment with ACP

The 6 *actions* occurring at *wireless layer*, as exhibited in the previous figure 6.1, are all triggered when a moving IoT device comes in or goes out of a wireless relay, this is, getting associated or dissociated to it. This kind of traffic may be related to control actions, which belong to the control plane in a conceptual manner, as they are related to how the VMs are managed by the fog system.

However, this type of traffic does not embrace neither the user data being forwarded through the system, which belongs to the data plane, nor the way to access and manage the devices being part of the system in order to configure, monitor or maintain them, which belongs to the management plane.

Those 6 *actions* may be divided into two categories, hence distinguishing between those when a moving IoT device gets associated to a particular wireless relay W_i, or otherwise, those when a moving IoT device gets disassociated from a given wireless relay W_i. Looking back to the figure 6.1, the actions shown on the left hand side belong to the former, whereas those on the right hand side belong to the latter.

Furthermore, algebraic models are being built up for each action by means of ACP, which may eventually be verified. Regarding the switching topology, it is to be reminded that all models presented in the previous chapter have been verified by means of their corresponding ACP models, which implies that any of them might be applied to this generic model, without loosing any generality related to the results obtained in this chapter.

6.2.1 Action *cloud in*

It takes place when a moving IoT device gets into the fog domain through a wireless physical channel in, coming from a cloud and getting connected to a wireless relay W_i. In such a case, the IoT device has already an associated VM located up in the cloud C, hence, the point is to migrate that VM allocated in the cloud domain into one of the hosts within the fog environment, called host H_j.

The cloud where that VM is located has an identifier k, and the proper VM inside that cloud is identified by m. Therefore, the combination of the cloud environment (k) where the VM is ubicated at a given point and the VM itself (m) may be identified by the expression k_m. Likewise, if that VM is located in a host within the fog domain (j), that VM may be identified by j_m.

First of all, an IoT device associated with a VM in the cloud gets connected to a wireless relay W_i, which is expressed by receiving (r_{in}) an IoT with a parameter showing the VM identifier and the environment where it is located, which in this case is $IoT(k_m)$. Then, W_i sends a control message to the orchestrator Q so as to find a host H_j having enough available resources and located as close as possible to W_i.

When the orchestrator Q has carried out the selection of the appropriate host candidate H_j within the fog domain, then, it sends a control message with the host identifier j and the associated VM identifier k_m straight to cloud C in order to migrate that VM from the cloud C to the selected host H_j, which will result in that VM getting now identified by j_m.

As per the model of this action, it is to be noted that the control message to prepare this type of migration is labelled as ctr_1, as this action is considered as action 1. Furthermore, some internal processes are used to explain the actions taking place in each layer within the system, such as $'Select'$ to choose host j, $'MoveOut'$ to start the migration from cloud to fog in the former, $'CloudIn'$ to carry out the migration from cloud to destination host within the fog environment, $'MoveIn'$ to finish that migration from cloud to fog in the latter, $'Unbind'$ to take out the old identifier of the associated VM in the IoT device and $'Bind'$ to include the new identifier of that VM in the IoT device. Eventually, this is the modelling of action *cloud in*, where W_i is quoted in (6.1), Q is cited in (6.2), C is referred to in (6.3) and H_j is done in (6.4).

$$\begin{aligned} W_i = \Big(& r_{in}\Big(IoT(k_m)\Big) \cdot s_{W_iQ}\Big(ctr_1(k_m)\Big) \cdot r_{QW_i}\Big(j_m\Big) \cdot Unbind\Big(IoT(k_m)\Big) \cdot \\ & Bind\Big(IoT(j_m)\Big)\Big) \cdot W_i \end{aligned} \tag{6.1}$$

$$\begin{aligned} Q = \Big(& r_{W_iQ}\Big(ctr_1(k_m)\Big) \cdot Select(k_m) \cdot s_{QC}\Big(ctr_1(k_m, j_m)\Big) \cdot r_{CQ}\Big(CloudIn(k_m, j_m)\Big) \cdot \\ & s_{QH_j}\Big(CloudIn(k_m, j_m)\Big) \cdot r_{H_jQ}\Big(ACK, j_m\Big) \cdot s_{QW_i}\Big(j_m\Big)\Big) \cdot Q \end{aligned} \tag{6.2}$$

$$C = \Big(r_{CQ}\Big(ctr_1(k_m, j_m)\Big) \cdot MoveOut(k_m, j_m) \cdot s_{CQ}\Big(CloudIn(k_m, j_m)\Big)\Big) \cdot C \tag{6.3}$$

$$H_j = \Big(r_{QH_j}\Big(CloudIn(k_m, j_m)\Big) \cdot MoveIn(k_m, j_m) \cdot s_{H_jQ}\Big(ACK, j_m\Big)\Big) \cdot H_j \tag{6.4}$$

At this stage, all processes may be run concurrently, and after applying the encapsulation operator, it is obtained the sequence of events in a timely manner, triggered by the moving IoT device getting into the fog domain and asking for a *cloud in* action, as shown in (6.5).

$$\begin{aligned} \partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j\Big) = \Big(& r_{in}\Big(IoT(k_m)\Big) \cdot c_{W_iQ}\Big(ctr_1(k_m)\Big) \cdot Select(k_m) \cdot \\ & c_{QC}\Big(ctr_1(k_m, j_m)\Big) \cdot MoveOut(k_m, j_m) \cdot c_{CQ}\Big(CloudIn(k_m, j_m)\Big) \cdot \\ & c_{QH_j}\Big(CloudIn(k_m, j_m)\Big) \cdot MoveIn(k_m, j_m) \cdot c_{H_jQ}\Big(ACK, j_m\Big) \cdot c_{QW_i}\Big(j_m\Big) \cdot \\ & Unbind\Big(IoT(k_m)\Big) \cdot Bind\Big(IoT(j_m)\Big)\Big) \cdot \partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j\Big) \end{aligned} \tag{6.5}$$

Finally, the abstraction operator may be applied so as to mask both internal processes and internal communications, hence maintaining only the external behaviour of the model, where all layers may be included, regardless whether they take part in this action or otherwise, as those do not change the overall outcome. This may be displayed in (6.6).

$$\tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big) = r_{in}\Big(IoT(k_m)\Big) \cdot \tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big) \tag{6.6}$$

6.2.2 Action *migrate in*

It takes place when a moving IoT device gets connected to a wireless relay W_i through a wireless physical channel *in*, even though it comes from another wireless relay W_{i*} within the fog environment. In that case, the moving IoT device has already an associated VM located in a host H_j, hence, two options may be available, such as either migrate that VM from its current host H_j to another one $H_{j'}$ being closer to the moving IoT device, or otherwise, to leave that VM in the same host H_j as there is no closer host available within the fog domain to the new wireless relay W_i.

First of all, a moving IoT device associated with a VM located in a particular host gets to wireless relay W_i. Then, W_i sends a control message to the orchestrator Q so as to determine whether there is an available host $H_{j'}$ with enough available resources and being located closer than the current H_j.

When receiving that message, the orchestrator Q undertakes the processing to find out whether there is any new host available, and if so, it is selected as the proper host candidate j'. Thus, two things may happen: if there is a possible candidate, then the orchestrator sends a control message to host H_j with the identifier of the VM j_m, which shows the current host where the VM is standing, and the identifier which will brand that VM after the migration has been carried out from host j to host j', that being VM j'_m. Otherwise, if there is no candidate, then just an acknowledge receipt message is sent back to the IoT device, stating that no VM migration needs to be carried out.

As per the model of this action, it is to be said that the control message to prepare this type of migration is called ctr_2, as this action is going to be considered as action 2. In such a case, it is clear from its description that two couple of entities are involved, this is, a pair of wireless relays W_i* and W_i and another pair of hosts H_j

and $H_{j'}$. Regarding the former pair, i represents the wireless relay where the IoT has just arrived, whilst $i*$ does the one where it came from. With respect to the latter pair, j represents the host where the VM is located on arriving to the new wireless relay i, whereas j' does the possible new host if VM migration takes place. So this model just includes W_i, H_j and $H_{j'}$, as W_{i*} is not relevant.

As a side note, it is to be noted that migration within the fog domain may be described from two opposite points of view, such as from the source wireless relay or from the destination one. The former accounts for action 5, meaning the action called *migration out*, whereas the latter accounts for action 2, called *migration in*, which is the action currently developed.

Additionally, some processes defined for the previous action has been employed, as well as three new ones named $'MigrateOut'$ to carry out the migration from source host to destination host, from the point of view of the former, $'MigrateIn'$ to do the same, but from the point of view of the latter, and $'Networking'$ to embrace the whole set of redundant paths being available between source host and destination host, which depend on the topology selected for the switches where hosts are hanging on. Eventually, this is the modelling of action *migrate in*, such that W_i is quoted in (6.7), Q is cited in (6.8), H_j is referred to in (6.9), H_j' is portrayed in (6.10) and T is specified in (6.11).

$$\begin{aligned} W_i = \Bigg(& r_{in}\Big(IoT(j_m)\Big) \cdot s_{W_iQ}\Big(ctr_2(j_m)\Big) \cdot \Bigg(r_{QW_i}\Big(j'_m\Big) \cdot Unbind\Big(IoT(j_m)\Big) \cdot Bind\Big(IoT(j'_m)\Big) + \\ & r_{QW_i}\Big(ACK\Big)\Bigg)\Bigg) \cdot W_i \end{aligned} \tag{6.7}$$

$$\begin{aligned} Q = \Bigg(& r_{W_iQ}\Big(ctr_2(j_m)\Big) \cdot \Bigg(s_{QH_j}\Big(ctr_2(j_m, j'_m)\Big) \cdot r_{H_{j'}Q}\Big(ACK, j'_m\Big) \cdot s_{QW_i}\Big(j'_m\Big) \\ & \lhd\ Select(j_m) \rhd s_{QW_i}\Big(ACK\Big)\Bigg)\Bigg) \cdot Q \end{aligned} \tag{6.8}$$

$$H_j = \Bigg(r_{QH_j}\Big(ctr_2(j_m, j'_m)\Big) \cdot MoveOut(j_m, j'_m) \cdot s_{H_jT}\Big(MigrateOut(j_m, j'_m)\Big)\Bigg) \cdot H_j \tag{6.9}$$

$$H_{j'} = \Bigg(r_{TH_{j'}}\Big(MigrateIn(j_m, j'_m)\Big) \cdot MoveIn(j_m, j'_m) \cdot s_{H_{j'}Q}\Big(ACK, j'_m\Big)\Bigg) \cdot H_{j'} \tag{6.10}$$

$$T = \Big(r_{H_jT}\Big(\mathit{MigrateOut}(j_m, j'_m)\Big) \cdot \mathit{Networking}(j_m, j'_m) \cdot s_{TH_{j'}}\Big(\mathit{MigrateIn}(j_m, j'_m)\Big)\Big) \cdot T \tag{6.11}$$

At this point, all processes may be executed concurrently, and after applying the encapsulation operator, it is achieved the sequence of events in a timely manner, triggered by the moving IoT device being already into the fog environment and asking for a *migrate in* action, as seen in (6.12).

$$\begin{gathered}
\partial_H\Big(W_i \mid\mid Q \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big) = \Big(r_{in}\Big(IoT(j_m)\Big) \cdot c_{W_iQ}\Big(ctr_2(k_m)\Big) \cdot \\
\Big(c_{QH_j}\Big(ctr_2(j_m, j'_m)\Big) \cdot \mathit{MoveOut}(j_m, j'_m) \cdot c_{H_jT}\Big(\mathit{MigrateOut}(j_m, j'_m)\Big) \cdot \\
\mathit{Networking}(j_m, j'_m) \cdot c_{TH_{j'}}\Big(\mathit{MigrateIn}(j_m, j'_m)\Big) \cdot \mathit{MoveIn}(j_m, j'_m) \cdot c_{H_{j'}Q}\Big(ACK, j'_m\Big) \cdot \\
c_{QW_i}\Big(j'_m\Big) \cdot \mathit{Unbind}\Big(IoT(j_m)\Big) \cdot \mathit{Bind}\Big(IoT(j'_m)\Big) \lhd \mathit{Select}(j_m) \rhd c_{QW_i}\Big(ACK\Big)\Big)\Big) \cdot \\
\partial_H\Big(W_i \mid\mid Q \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big)
\end{gathered} \tag{6.12}$$

Finally, the abstraction operator may be applied in order to mask both internal processes and internal communications, thus maintaining just the external behaviour of the model, where all layers are to be included, considering that those not taking part do not change the overall outcome. That may be observed in (6.13).

$$\begin{gathered}
\tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big)\Big) = \\
r_{in}\Big(IoT(j_m)\Big) \cdot \tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big)\Big)
\end{gathered} \tag{6.13}$$

6.2.3 Action *init VM*

It takes place when a moving IoT device gets into the fog domain through a wireless physical channel *in*, and it does not have any associated VM in the cloud. Hence, it is the case of an IoT device getting first registered in the fog environment to get its associated VM, therefore, the point is to create and initialize that VM into the closest available host within the fog domain.

First of all, the IoT gets connected to the wireless relay W_i without any associated VM, which is shown by the parameter $-$. Next, W_i sends a control message to the orchestrator Q in order to find a host H_j with enough resources being situated the closest possible to W_i. After that, when Q finds out an available host H_j, then a VM is created in there, and afterwards, the identifier is sent back to the IoT device.

Regarding the model of this action, it is to be said that the control message to create and init that VM is called as ctr_3, as that action is considered as action 3. Furthermore, the process $'Init'$ is defined to carry out the creation and initialization of the VM. Eventually, this is the modelling of action *init VM*, where W_i is quoted in (6.14), Q is cited in (6.15), and H_j is done in (6.16).

$$W_i = \Big(r_{in}\Big(IoT(-)\Big) \cdot s_{W_iQ}\Big(ctr_3(-)\Big) \cdot r_{QW_i}\Big(j_m\Big) \cdot Bind\Big(IoT(j_m)\Big)\Big) \cdot W_i \tag{6.14}$$

$$Q = \Big(r_{W_iQ}\Big(ctr_3(-)\Big) \cdot Select(-) \cdot s_{QH_j}\Big(ctr_3(j_m)\Big) \cdot r_{H_jQ}\Big(ACK, j_m\Big) \cdot s_{QW_i}\Big(j_m\Big)\Big) \cdot Q \tag{6.15}$$

$$H_j = \Big(r_{QH_j}\Big(ctr_3(j_m)\Big) \cdot Init(j_m) \cdot s_{H_jQ}\Big(ACK, j_m\Big)\Big) \cdot H_j \tag{6.16}$$

After the application of the encapsulation operator, it is attained the sequence of events in a timely manner, triggered by the IoT device getting into the fog ecosystem and asking for an *init VM* action, as shown in (6.17).

$$\partial_H\Big(W_i \mid\mid Q \mid\mid H_j\Big) = \Big(r_{in}\Big(IoT(-)\Big) \cdot c_{W_iQ}\Big(ctr_3(-)\Big) \cdot Select(-) \cdot c_{QH_j}\Big(ctr_3(j_m)\Big) \cdot \\ Init(j_m) \cdot c_{H_jQ}\Big(ACK, j_m\Big) \cdot c_{QW_i}\Big(j_m\Big) \cdot Bind\Big(IoT(j_m)\Big)\Big) \cdot \partial_H\Big(W_i \mid\mid Q \mid\mid H_j\Big) \tag{6.17}$$

Finally, after applying the abstraction operator, the external behaviour of the model prevails, as seen in (6.18).

$$\tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big) = r_{in}\Big(IoT(-)\Big) \cdot \tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big) \tag{6.18}$$

6.2.4 Action *cloud out*

It takes place when a moving IoT device departs from the fog environment by means of getting disconnected out of a wireless relay W_i through physical channel *out*, not getting reconnected to any other one. In that case, the VM associated to that IoT device leaves the fog domain and gets into a cloud, therefore, the point is to migrate that VM in the fog environment, specifically into a host H_j, to a cloud one.

This process is quite similar to the one exposed for the migration from cloud to fog, presented in action 1 (*cloud in*), although the operations to be done are undertaken the other way around. It is to be said that the process of migrating a VM to the cloud is carried out in the first place, and when it is done, the IoT device moves out of the fog environment.

Putting the focus on the model, it is to be noted that the control message to prepare this type of migration is called ctr_4, because this action is considered as action 4. Additionally, the process $'Find'$ is going to be defined to perform the lookup of the cloud identifier where the VM will be forwarded on. Eventually, this is the modelling of action *cloud out*, where W_i is quoted in (6.19), Q is cited in (6.20), H_j is done in (6.21) and C is referred to in (6.22).

$$W_i = \Big(s_{W_iQ}\Big(ctr_4(j_m)\Big) \cdot r_{QW_i}\Big(k_m\Big) \cdot Unbind\Big(IoT(j_m)\Big) \cdot Bind\Big(IoT(k_m)\Big) \cdot s_{out}\Big(IoT(k_m)\Big)\Big) \cdot W_i \tag{6.19}$$

$$Q = \Big(r_{W_iQ}\Big(ctr_4(j_m)\Big) \cdot Find(j_m) \cdot s_{QH_j}\Big(ctr_4(j_m,k_m)\Big) \cdot r_{H_jQ}\Big(CloudOut(j_m,k_m)\Big) \cdot s_{QC}\Big(CloudOut(j_m,k_m)\Big) \cdot r_{CQ}\Big(ACK,k_m\Big) \cdot s_{QW_i}\Big(k_m\Big)\Big) \cdot Q \tag{6.20}$$

$$H_j = \Big(r_{QH_j}\Big(ctr_4(j_m,k_m)\Big) \cdot MoveOut(j_m,k_m) \cdot s_{H_jQ}\Big(CloudOut(j_m,k_m)\Big)\Big) \cdot H_j \tag{6.21}$$

$$C = \Big(r_{QC}\Big(CloudOut(j_m,k_m)\Big) \cdot MoveIn(j_m,k_m) \cdot s_{CQ}\Big(ACK,k_m\Big)\Big) \cdot C \tag{6.22}$$

After the application of the encapsulation operator, it is obtained the sequence of events in a timely manner, triggered by the IoT device getting out of the fog domain and asking for an *cloud out* action, as viewed in (6.23).

$$\partial_H\Big(W_i \mid\mid Q \mid\mid H_j \mid\mid C\Big) = \Big(c_{W_iQ}\Big(ctr_4(j_m)\Big) \cdot Find(j_m) \cdot c_{QH_j}\Big(ctr_4(j_m,k_m)\Big) \cdot MoveOut(j_m,k_m) \cdot c_{H_jQ}\Big(CloudOut(j_m,k_m)\Big) \cdot c_{QC}\Big(CloudOut(j_m,k_m)\Big) \cdot MoveIn(j_m,k_m) \cdot c_{CQ}\Big(ACK,k_m\Big) \cdot c_{QW_i}\Big(k_m\Big) \cdot Unbind\Big(IoT(j_m)\Big) \cdot Bind\Big(IoT(k_m)\Big) \cdot s_{out}\Big(IoT(k_m)\Big)\Big) \cdot \partial_H\Big(W_i \mid\mid Q \mid\mid H_j \mid\mid C\Big) \tag{6.23}$$

Finally, after applying the abstraction operator, the external behaviour of the model is shown in (6.24).

$$\tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big) = s_{out}\Big(IoT(k_m)\Big) \cdot \tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big) \tag{6.24}$$

6.2.5 Action *migration out*

It is to be remarked that both this case, namely *migration out* (action 5), and *migration in* (action 2) undertake a VM migration from a source host to a destination host within the fog domain, even though this case takes the point of view of the source wireless relay, where the IoT devices leave it, whilst the other one does it from the destination wireless relay, where the IoT devices joins it. Hence, the same processes exposed in action 2 apply herein.

Focusing on the model, it is to be said the control message to prepare this kind of migration is related to as ctr_5, because this action is considered as action 5. It is to be noted that, after an IoT device leaves the wireless relay, two options may happen, such that there may be a VM migration, or otherwise. The former implies that there is an available host closer to the new wireless relay where the IoT is connected to, whereas the latter implies that there is not, hence no VM migration is needed. Eventually, this is the modelling of action *migrate out*, where W_i is quoted in (6.25), Q is cited in (6.26), H_j is related to in (6.27), H'_j is referred to in (6.28) and T is done in (6.29).

$$W_i = \bigg(s_{W_iQ}\Big(ctr_5(j_m)\Big) \cdot \bigg(r_{QW_i}\Big(j'_m\Big) \cdot Unbind\Big(IoT(j_m)\Big) \cdot Bind\Big(IoT(j'_m)\Big) \cdot s_{out}\Big(IoT(j'_m)\Big) + r_{QW_i}\Big(ACK\Big) \cdot s_{out}\Big(IoT(j_m)\Big)\bigg)\bigg) \cdot W_i \tag{6.25}$$

$$Q = \bigg(r_{W_iQ}\Big(ctr_5(j_m)\Big) \cdot \bigg(s_{QH_j}\Big(ctr_5(j_m, j'_m)\Big) \cdot r_{H_{j'}Q}\Big(ACK, j'_m\Big) \cdot s_{QW_i}\Big(j'_m\Big) \lhd\ Select(j_m) \rhd s_{QW_i}\Big(ACK\Big)\bigg)\bigg) \cdot Q \tag{6.26}$$

$$H_j = \bigg(r_{QH_j}\Big(ctr_5(j_m, j'_m)\Big) \cdot MoveOut(j_m, j'_m) \cdot s_{H_jT}\Big(MigrateOut(j_m, j'_m)\Big)\bigg) \cdot H_j \tag{6.27}$$

$$H_{j'} = \bigg(r_{TH_{j'}}\Big(MigrateIn(j_m, j'_m)\Big) \cdot MoveIn(j_m, j'_m) \cdot s_{H_{j'}Q}\Big(ACK, j'_m\Big)\bigg) \cdot H_{j'} \tag{6.28}$$

$$T = \Big(r_{H_jT}\Big(\mathit{MigrateOut}(j_m, j'_m)\Big) \cdot \mathit{Networking}(j_m, j'_m) \cdot s_{TH_{j'}}\Big(\mathit{MigrateIn}(j_m, j'_m)\Big)\Big) \cdot T \tag{6.29}$$

After the application of the encapsulation operator, it is obtained the sequence of events in a timely manner, triggered by the moving IoT device being already into the fog ecosystem and asking for a *migrate out* action, as performed in (6.30). It is to be noted that two possible actions may arise, depending on the result of $'\mathit{Select}'$ process.

$$\begin{aligned} \partial_H\Big(W_i \mid\mid Q \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big) = \Bigg(c_{W_iQ}\Big(ctr_5(k_m)\Big) \cdot \Bigg(c_{QH_j}\Big(ctr_5(j_m, j'_m)\Big) \cdot \mathit{MoveOut}(j_m, j'_m) \cdot \\ c_{H_jT}\Big(\mathit{MigrateOut}(j_m, j'_m)\Big) \cdot \mathit{Networking}(j_m, j'_m) \cdot c_{TH_{j'}}\Big(\mathit{MigrateIn}(j_m, j'_m)\Big) \cdot \\ \mathit{MoveIn}(j_m, j'_m) \cdot c_{H_{j'}Q}\Big(ACK, j'_m\Big) \cdot c_{QW_i}\Big(j'_m\Big) \cdot \mathit{Unbind}\Big(\mathit{IoT}(j_m)\Big) \cdot \\ \mathit{Bind}\Big(\mathit{IoT}(j'_m)\Big) \cdot s_{out}\Big(\mathit{IoT}(j'_m)\Big) \triangleleft \mathit{Select}(j_m) \triangleright c_{QW_i}\Big(ACK\Big) \cdot s_{out}\Big(\mathit{IoT}(j_m)\Big)\Bigg)\Bigg) \cdot \\ \partial_H\Big(W_i \mid\mid Q \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big) \end{aligned} \tag{6.30}$$

Finally, after applying the abstraction operator, the external behaviour of the model is revealed in (6.31).

$$\begin{aligned} \tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big)\Big) = \\ \Big(s_{out}\Big(\mathit{IoT}(j_m)\Big) + s_{out}\Big(\mathit{IoT}(j'_m)\Big)\Big) \cdot \tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big)\Big) \end{aligned} \tag{6.31}$$

6.2.6 Action ***kill VM***

It takes place when a moving IoT device leaves the fog environment by means of wireless physical channel *out*, not getting into the cloud environment. This implies that the IoT device does not need its associated VM any longer, so it may be terminated. Hence, the point is to kill that VM located into a host H_j within the fog.

This process is pretty similar to the one exposed for the creation of a VM in the fog environment, called action 3, although the operations to be done work the other way around. Hence, the process of killing the VM is carried out in the first place, and when it is done, the IoT devices moves away from the fog domain.

Focusing on the model, it is to be noted the control message to prepare this sort of migration is referred to as ctr_6, because this action is considered as action 6.

Furthermore, the process *'Kill'* is defined to carry out the termination of the VM. Eventually, this is the modelling of action *kill VM*, where W_i is quoted in (6.32), Q is cited in (6.33) and H_j is done in (6.34).

$$W_i = \Big(s_{W_iQ}\Big(ctr_6(j_m)\Big) \cdot r_{QW_i}\Big(-\Big) \cdot Unbind\Big(IoT(j_m)\Big) \cdot s_{out}\Big(-\Big)\Big) \cdot W_i \quad (6.32)$$

$$Q = \Big(r_{W_iQ}\Big(ctr_6(j_m)\Big) \cdot Find(j_m) \cdot s_{QH_j}\Big(ctr_6(j_m)\Big) \cdot r_{H_jQ}\Big(ACK,-\Big) \cdot s_{QW_i}\Big(-\Big)\Big) \cdot Q \quad (6.33)$$

$$H_j = \Big(r_{QH_j}\Big(ctr_6(j_m)\Big) \cdot Kill(j_m) \cdot s_{H_jQ}\Big(ACK,-\Big)\Big) \cdot H_j \quad (6.34)$$

After the application of the encapsulation operator, it is obtained the sequence of events in a timely manner, triggered by the IoT device getting off the fog environment and asking for a *Kill VM* action, as viewed in (6.35).

$$\partial_H\Big(W_i \mid\mid Q \mid\mid H_j\Big) = \Big(c_{W_iQ}\Big(ctr_6(j_m)\Big) \cdot Find(j_m) \cdot c_{QH_j}\Big(ctr_6(j_m)\Big) \cdot Kill(j_m) \cdot c_{H_jQ}\Big(ACK,-\Big) \cdot c_{QW_i}\Big(-\Big) \cdot Unbind\Big(IoT(j_m)\Big) \cdot s_{out}\Big(IoT,-\Big)\Big) \cdot \partial_H\Big(W_i \mid\mid Q \mid\mid H_j\Big) \quad (6.35)$$

Finally, after having applied the abstraction operator, the external behaviour of the model prevails, as shown in (6.36).

$$\tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big) = s_{out}\Big(IoT(-)\Big) \cdot \tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big) \quad (6.36)$$

6.3 Verification of the models

At this point, it may be interesting to compare the results obtained for the external behaviour of all the six models proposed with the external behaviour of the real systems running the same actions. The real system is being expressed as X, which accounts for a guarded linear recursive variable. Hence, for each of the actions described, the external behaviour of X is going to be presented in ACP notation in order to compare it with that obtained for the corresponding model.

Action 1: cloud in

- External behaviour of the model: it has already been deduced in (6.6):

$$\tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big) = r_{in}\Big(IoT(k_m)\Big) \cdot \tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big)$$

- External behaviour of the real system: an IoT device with an associated VM in the cloud gets into the fog domain, as in (6.37).

$$X = r_{in}\Big(IoT(k_m)\Big) \cdot X \tag{6.37}$$

Action 2: migrate in

- External behaviour of the model: it has already been deduced in (6.13):

$$\tau_I\bigg(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big)\bigg) = r_{in}\Big(IoT(j_m)\Big) \cdot \tau_I\bigg(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big)\bigg)$$

- External behaviour of the real system: an IoT device with an associated VM already in the fog environment gets to another wireless relay, as in (6.38).

$$X = r_{in}\Big(IoT(j_m)\Big) \cdot X \tag{6.38}$$

Action 3: init VM

- External behaviour of the model: it has already been deduced in (6.18):

$$\tau_I\bigg(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\bigg) = r_{in}\Big(-\Big) \cdot \tau_I\bigg(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\bigg)$$

- External behaviour of the real system: an IoT device without any associated VM gets into the fog ecosystem, so a new VM is assigned, as in (6.39).

$$X = r_{in}\Big(IoT(-)\Big) \cdot X \tag{6.39}$$

Action 4: cloud out

- External behaviour of the model: it has already been deduced in (6.24):

$$\tau_I\bigg(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\bigg) = s_{out}\Big(IoT(k_m)\Big) \cdot \tau_I\bigg(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\bigg)$$

- External behaviour of the real system: an IoT device with an associated VM in the fog domain gets out to the cloud domain, as in (6.40).

$$X = s_{out}\Big(IoT(k_m)\Big) \cdot X \tag{6.40}$$

Action 5: migrate out

- External behaviour of the model: it has already been deduced in (6.31):

$$\tau_I\bigg(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big)\bigg) =$$
$$\bigg(s_{out}\Big(IoT(j_m)\Big) + s_{out}\Big(IoT(j'_m)\Big)\bigg) \cdot \tau_I\bigg(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid H_{j'} \mid\mid T\Big)\bigg)$$

- External behaviour of the real system: an IoT device with an associated VM already in the fog ecosystem moves to another wireless relay, so it may or may not migrate, as in (6.41).

$$X = \Big(s_{out}\Big(IoT(j_m)\Big) + s_{out}\Big(IoT(j'_m)\Big)\Big) \cdot X \qquad (6.41)$$

Action 6: kill VM

- External behaviour of the model: it has already been deduced in (6.36):
$\tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big) = s_{out}\Big(IoT(-)\Big) \cdot \tau_I\Big(\partial_H\Big(W_i \mid\mid Q \mid\mid C \mid\mid H_j \mid\mid T\Big)\Big)$

- External behaviour of the real system: An IoT device with an associated VM in the fog domain gets off and it does not go into cloud, as in (6.42).

$$X = s_{out}\Big(IoT(-)\Big) \cdot X \qquad (6.42)$$

In short, it is clear that each pair of recursive expressions are multiplied by the same factors, hence, each pair shows rooted branching bisimilar expressions, as they both share the same actions and they both have the same branching structure. This a sufficient condition to get a model verified, therefore, all of the 6 models proposed herein with ACP get duly verified.

6.4 Summary

In this chapter, a generic IoT/fog model has been introduced by considering some essential building blocks. There are a bunch of wireless relays where IoT devices get connected to the fog environment, being represented by a block W_i, a group of hosts where the VM associated to those devices are stored, being rendered by a block H_j, a blanket topology portraying the paths interconnecting all hosts within the fog domain, being depicted by block T, an orchestrator managing all actions within the fog ecosystem, being characterized by block Q, and a backup cloud service allowing external VM associated to incoming devices to get into the fog system and internal VM associated to outgoing devices getting out of it, being illustrated by block C.

The interface between such a generic system and the exterior world is a wireless block W_i, which permits either the external IoT devices to get associated to it or the internal IoT devices to get disassociated to it. Depending on either of both situations, three different actions may happen, where cloud in, migrate in and init VM go attached to the former, whereas cloud out, migration out and kill VM go tied to the latter.

Each of those six actions are being specified and further verified by means of ACP by first describing the actions being carried out by each of the blocks involved in a given action, then a specification is obtained by applying the appropriate operators, and in turn, a verification is performed by comparing the external behaviour of the model with such of the real system, where all of the six actions described above achieve the verification required.

Overall, the model proposes a generic framework for describing the main parts of a fog/cloud environment, just considering the main functions of each one from a high-level point of view, hence not getting into practical details. This approach permits to abstract away from the DC topology being deployed, thus making the model fit to any of the DC designs proposed in the last chapter.

Therefore, the model presented denotes the behaviour of the main building blocks in a generic manner, allowing to study how they work all together when representing the main actions being carried out in a fog/cloud environment.

Chapter 7

Use case of Data Centre models

After having presented a general algebraic model for a fog environment, composed by some necessary building blocks, and taking into consideration that a wide range of options have been previously defined for the DC topology, it is time to focus on a real world implementation for this model.

The scenario selected is related to a linear deployment of sequential wireless relays, thus offering a non-stop coverage area along a predefined path. It is to be said that, in spite of referring to a linear deployment, such a path may not necessarily be a straight line, as long as a trajectory between a given pair of points may be fully included into the coverage area, which may allow the implementation of this type of linear deployment in any type of roads, railways or pipelines, regardless those being rectilinear or otherwise.

7.1 Sequential relay - schematic diagram

To start with, Figure 7.1 exhibits the schematic diagram for the overall topology proposed. There is a range of sequential wireless relays, whose coverage areas overlap, hence allowing for handover between any pair of neighbours. Then, each wireless relay is connected to a different host, which keeps the VM associated to different users. It is to be reminded that those users may be as close as possible to their associated VM, although it may happen that they are not in the closest one.

Furthermore, those hosts may be interconnected by means of a given topology. Initially, any of the topologies studied in the previous section regarding DC modelling might be used. However, fat tree has been selected for being the one with more layers of switches, and thus, minimizing the scalability issues. Nonetheless, if any other topology was to be selected, such as leaf and spine or any other topology, the only thing to be done would be to substitute the fat tree model layers with those related to the desired design, this is, just applying the corresponding expressions to such a topology and keeping the rest of the model as it is.

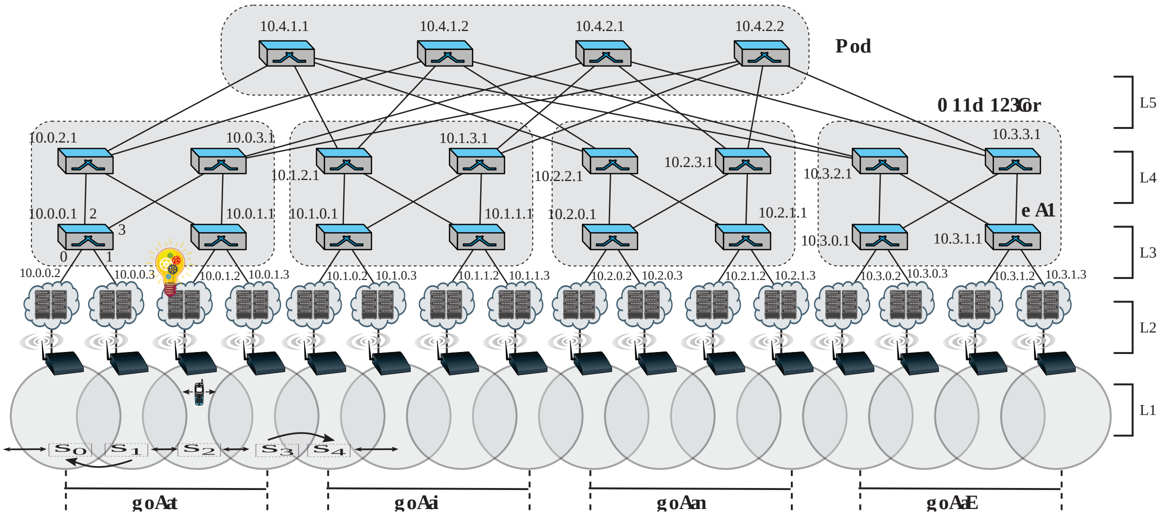


Figure 7.1: Schematic diagram of a fat tree sequential wireless relays.

This particular instance has been considered as a representative example of a DC model for fog environments because of its simplicity, in a way that each wireless relay is connected to a different host, thus allowing for an easy and straightforward scheme, whereas each IoT device may only move in a sequential fashion, hence making that the only wireless relays to get in and out of the system are those located at both ends, whilst the rest of them just execute roaming.

Other layouts are obviously available, leading to different sort of designs, although this particular setup permits a clearer understanding. Some of those variations might allow the connection of different wireless relays to the same host, the entry and exit operations at any wireless relay, the interconnection of wireless relays in a non linear fashion, permitting diverse kinds of branching designs, or the selection of a different switching topology, leading to other linking patterns among switches.

Regarding the functionality of the schematic diagram presented, the lower layer (L1) determines the area where an IoT device may get connected to the fog domain, that being formed by the overlapping of the coverage areas of a group of sequential wireless relays, allowing the handover among them. The next layer (L2) keeps the VM associated to the IoT within the fog ecosystem, meeting the requirement that both must be as close to each other as possible. With respect to the upper layers, they represent the switching topology interconnecting all hosts, considering that fat tree has been selected herein, but any other might do.

7.2 Topology framework for linear deployments

Regarding the working framework, it is important to remember the key concepts established in the previous sections, as this is a fog computing environment for moving IoT devices going around within a wireless domain, therefore, their associated computing assets ought to follow that movement as they get about. The working framework may be divided into layers, where the wireless layer exposes the area where physical IoT devices are moving about, the host layer keeps the associated VM to each moving IoT device connected into the system, and the fat tree layers are meant to permit the live VM migrations to happen between any couple of hosts.

To begin with, the *wireless layer* is formed by a string of sequential wireless relays, such that one is reached just after the previous one and just before the following one, thus making a well-ordered set as the set of natural numbers. This layout may be applied to an array of sequential wireless nodes, deployed in an ordered manner in any sort of sequential structure, not necessarily linear, such as a railway lines, roads or pipelines. Hence, IoT devices acting as end users might get into the topology only at any end and they may leave also at any end but not at any intermediate node.

Every wireless node has an associated fog computing asset, namely a VM, which is hosted in one of the servers being part of each of the facilities situated at the *host* *layer*. This way, when a new IoT user gets connected for first time into the system through one of the wireless nodes at any of the ends, a new VM is created in a host, and in turn, associated to that user.

Three options are to be considered herein, where the first one is to create the VM in the host directly connected to that wireless node, which is done only if there are available resources left in that given host, the second one is to create it in another host, and the third one is not to create it due to lack of computing resources available, and hence, the end user is ejected out of the system. With regards to the second option, after trying to allocate the VM in its directly connected host unsuccessfully, the other hosts belonging to the same pod are checked out for enough resources left, and if that is not possible, all hosts belonging to the other pods are checked out just in case.

Afterwards, if an existing user moves to a neighbouring wireless node, a handover takes place, and hence, its associated VM tries to move to the host directly connected to that new access point. This migration will be done if there are available resources in that new host, or if not, other hosts might be checked, or otherwise, migration will not take place. Eventually, after having performed a string of wireless handovers by an end user and its corresponding attempts of live VM migration, the end user will get off the system through one of its ends, which will cause its associated VM to be either terminated or migrated out to the cloud, and therefore, its computing resources will be freed from the fog environment to be used by other moving IoT devices.

Looking back to Figure 7.1, it may be seen that an IoT device is standing at W_2 and its associated VM is at H_2. The following move for that end user are either going to the left to W_1 or going to the right to W_3, which may try to trigger the process of live VM migration to get its computing assets as close as possible.

Regarding the identification of the devices within this framework, that is done on a layer basis, by taking the initial letter of the proper layer in capital letters, followed by a natural number indicating its order in that layer, starting from zero and moving from left to right. As per the layers, there are five of them, as it may be seen in Figure 7.1: Wireless (W), Host (H), Edge (E), Aggregation (A) and Core (C).

Additionally, the total amount of devices n per layer are: wireless ($^{k^3}/_{4}$), hosts ($^{k^3}/_{4}$), edge ($^{k^2}/_{2}$), aggregation ($^{k^2}/_{2}$), core ($^{k^2}/_{4}$), thus, the devices will be assigned a natural number going from 0 onwards up to $n - 1$, in order to take advantage of congruence relation in modular arithmetic for the expressions used in the model.

On the other hand, connections between two wireless layer nodes are called paths, which are named after its initial letter p in lower letters, followed by the number corresponding to the node being at the right end of the path. That makes p_0 for the path coming into the system from the left border, and $p_{k^3/4}$ for the path going out of the system to the right one. On the other hand, $p'_{k^3/4}$ stands for the path coming in from the right boundary, and p'_0 does it for the path going off to the left one.

This way, a generic IoT device standing at W_i may reach its neighbour W_{i+1} through path p_{i+1} going to the right, whereas it may go to its neighbour W_{i-1} through path p'_i going to the left. Likewise, W_{i-1} may get to neighbour W_i through p_i moving rightwards, whilst W_{i+1} may gain neighbour W_i through p'_{i+1} moving leftwards.

Furthermore, each wireless node i have an extra channel (0) to reach host i, whilst each of those have a channel (0) looking downwards to that wireless node and a channel (1) looking upwards to the switching topology. As per the connections in the switching topology, they are identified by the port numbers of the topology selected. Focusing on this particular case, Figure 5.5 depicts the model for all fat tree layers.

Putting the whole set of devices and connections together, two scenarios may be proposed by carrying out an algebraic formal description using ACP syntax and semantics. First of all, an ideal scenario is presented where physical paths always take the same direction, namely coming from left to right, and storage space is unlimited, hence a new VM may always be added. Then, a more realistic scenario is presented where physical paths may take both directions and storage space is limited, hence having restrictions for hosting a new VM in a particular host.

Therefore, each scenario may have its own particular expressions in order to describe the behaviour of just one moving device in both case scenarios, even though many devices may get concurrently into the system. Therefore, the overall algebraic expressions in both cases may be easily extended for n users, as in (7.1):

$$\Big|\Big|_{u=0}^{n} \left(||_{i=0}^{\frac{k^3}{4}-1} ||_{i'=0}^{\frac{k^3}{4}-1} ||_{j=0}^{\frac{k^2}{2}-1} ||_{j'=0}^{\frac{k^2}{2}-1} ||_{l=0}^{k-1} \left(W_i^u || H_{i'}^u || E_j^u || A_{j'}^u || C_l^u \right) \right) \cdot X \tag{7.1}$$

7.3 Scenario 1: one-way and no resource restrictions

This first scenario portrays physical devices moving only one way, namely from left to right. This is, IoT devices getting into the system from wireless node 0, then, going all the way through each wireless node i in a sequential manner, and finally, getting out of the system through wireless node $\frac{k^3}{4} - 1$. Furthermore, there are no storage constraints, meaning that the creation of a new VM in any host (even though host 0 will always do it) or the performance of a live VM migration is always possible.

Therefore, the sequence of events for this case scenario is pretty straightforward, considering that the term wireless node W_i relates to the physical coverage area shown as the bottom layer of the Figure 7.1, which is formed by an access point AP_i within such an area. Hence, it may be said that AP_i creates a coverage area W_i through the radiating power of its antennas. In summary, here they are the most relevant events.

1. Initially, a new IoT device gets into W_0:

    - First, it gets into the coverage area W_0 of wireless relay AP_0 through path p_0, either having already an associated VM up in the cloud, being described as IoT(C), or otherwise, having no associated VM at all, being denoted by IoT(-).
    - Then, it gets associated to access point AP_0, and in turn, it requests either a VM migration onto host H_0 (through function *MIGRATEin*), or otherwise, the creation of a brand new VM on host H_0 (through function *INIT*), where the former results in the VM migration from cloud C onto the fog environment (through function $FOG_{C \to 0}$), whilst the latter does it in the setup of a new VM in the fog domain (through function FOG_0).
    - Next, a confirmation (0), related to where the associated VM is located, is sent back to AP_0, and in turn, it is sent over to the IoT device, which gets bound to its VM location (through function *BIND*), resulting in $IoT(0)$.
    - Eventually, at some point, it will go to wireless relay W_1 through path p_1.

2. At a later stage, that IoT device moves into a neighbour wireless relay W_i from W_{i-1} through path p_i:

    - First, it gets disassociated from access point AP_{i-1} (through function $\neg AP_{i-1}$) and gets associated to AP_i (through function AP_i).
    - Then, it requests its VM to be moved from the host it was on to the host directly connected to its current wireless node (through function *MOVE*, which in this scenario always does it from host $i-1$ to i), with variables x and y holding generic destinations one way and the other, which in this case means that the VM gets moved from host H_{i-1} to host H_i, according to the pre-copy live VM migration strategy.
    - As stated before, the pre-copy technique is supposed to need just one iteration for simplicity purposes, thus resulting in the following sequence of events: first off, a copy of the source VM is requested by the destination host (through function $RQT(i, i-1)$), which is replied back (through function $COPY(i-1, i)$), following by the discard of the source VM (through function $ERASE(i, i-1)$), and finally, activating the new one upon confirmation (through function $DONE(i-1, i)$). It is to be noted that i implies the current host and $i-1$ refers to the previous host.
    - Furthermore, it is to be said that the IoT device gets unbound from the previous host (through function *UNBIND*) and rebound to the new host (through function *BIND*) at the end of the migration process.
    - Eventually, at a later time, it will move to wireless relay W_{i+1} through path p_{i+1}, repeating the same pattern.

3. Finally, that IoT device will get out of the system at some point after having reached wireless relay $W_{\frac{k3}{4}-1}$ through path $p_{\frac{k3}{4}-1}$:

    - After reassociating with $AP_{\frac{k3}{4}-1}$ and getting its VM migrated to host $H_{\frac{k3}{4}-1}$, with the corresponding bounding process, the IoT device will eventually get out of the system at some stage.

- At that point, this is the sequence of events: the IoT device needs to get unbound from that host (through function *UNBIND*), and then, its VM is requested to be either erased (through functions *KILL* and $\neg FOG_{\frac{k^3}{4}-1}$) or migrated up to the cloud (through functions *MIGRATEout* and $\neg FOG_{\frac{k^3}{4}-1 \to C}$).
- Once the associated VM has been terminated or it has been transferred to the cloud, where the former is denoted by (ACK), and the latter does it by (C), then the IoT gets disassociated from $\neg AP_{\frac{k^3}{4}-1}$ (through function $\neg AP_{i-1}$), and eventually, leaves the system through path $p_{\frac{k^3}{4}}$, accounting for $s_{p_{\frac{k^3}{4}}}(IoT(-))$ in the former and $s_{p_{\frac{k^3}{4}}}(IoT(C))$ in the latter.

Regarding the messages used for *live VM migration*, it is considered that the transfer is completed in the first iteration, for simplicity purposes. Hence, these are the four steps to be taken for it:

- *RQT*: requesting an existing VM to the source.
- *COPY*: sending a copy of that VM to destination.
- *ERASE*: requesting the erasement of that VM from the source.
- *DONE*: sending the confirmation to destination and in turn, upon receipt, activate that new VM.

These messages are exchanged between the host where the VM is currently located (being the *old host*) and the host where that VM is going to be moved (being the *new host*), taking the shortest path between them both through the fat tree switching infrastructure. Alternatively, there may well be redundant paths between both hosts, and in such a case, any of the available paths may be taken to get to the target, or otherwise, a load-balancing policy may be applied by taking more than one of those feasible paths.

As per the direction of the flow, the new host sends a request message asking for a clone of the VM to the old host, which in turn, replies back with a copy message containing a clone of that VM, and upon receipt, that clone is established into the new host. After that, an erase message is sent from the new host to the old one, asking for the termination of the VM located at the old host, an upon receipt, the VM is terminated from the old host, which in turn, sends back a done message to the new host, thus confirming the success of the VM migration.

In order to keep it simple, the four messages are expressed by the generic notation $MSG(x, y)$ in the layers related to the interconnection topology, where x and y are, respectively, the source and destination of the message being undertaken through the fat tree infrastructure, no matter which layer the communication is taking place. This generic notation may be seen as a common framework for all four types of messages, as the path to move between two hosts is independent of the type of message involved.

Additionally, in this first case scenario, i identifies both the wireless relay where the moving item is connected at a given moment and the host attached to that wireless node, where the VM associated to the moving IoT is always kept. Furthermore, a and b are the source and destination hosts for any given migration through the switching infrastructure, which identify both ends of the fat tree communication.

Taking into consideration the special characteristics in this case scenario, VM migration messages at the host layer needs to be undertaken between the current host i and its predecessor $(i - 1)$, or otherwise, with its successor $(i + 1)$, depending on whether i is the source or the destination of the message being under consideration.

Moreover, it is to be reminded that the condition for two hosts to be hanging on the same edge switch is $\left\lfloor \frac{a}{k/2} \right\rfloor = \left\lfloor \frac{b}{k/2} \right\rfloor$, whereas the condition for two hosts to be part of the same pod is $\left\lfloor \frac{a}{(k/2)^2} \right\rfloor = \left\lfloor \frac{b}{(k/2)^2} \right\rfloor$. Therefore, if two hosts belong to different pods, then $\left\lfloor \frac{a}{(k/2)^2} \right\rfloor \neq \left\lfloor \frac{b}{(k/2)^2} \right\rfloor$. Therefore, those three conditions may help identify the number of hops away between host a and host b.

Furthermore, it is to be noted that if the first condition is met in an edge switch, the message is sent straight to the destination host, whereas otherwise, it is sent through all aggregation switches within its own pod. Likewise, if the second condition is fulfilled in an aggregation switch, the message is sent over to the destination switch, whilst otherwise, it is sent through all core switches. Otherwise, if the third condition is met, the message is forwarded on to all cores towards the destination pod.

Here they are the algebraic expressions for all five entities described in scenario 1, such as W_i in (7.2), H_i in (7.3), E_j in (7.4), A_j in (7.5) and C_l in (7.6).

$$\begin{aligned}
W_i = \sum_{i=0}^{\frac{k^3}{4}-1} & \Bigg(\bigg(\Big(\Big(r_{p_0}(IoT(-)) \cdot AP_0 \cdot s_0(INIT) + r_{p_0}(IoT(C)) \cdot AP_0 \cdot s_0(MIGRATEin)\Big) \cdot \\
& r_0(0) \cdot BIND(IoT(0)) \cdot s_{p_1}(IoT(0)) \cdot W_1\Big) \\
& \quad \lhd\ i = 0 \rhd \\
& \quad \Big(r_{p_i}(IoT(i-1)) \cdot \neg AP_{i-1} \cdot AP_i \cdot s_0(MOVE(i-1,i)) \cdot \\
& \quad r_0(i) \cdot UNBIND(IoT(i-1)) \cdot BIND(IoT(i)) \cdot \\
& \quad \quad \bigg(\Big(UNBIND(IoT(i)) \cdot \Big(s_0(KILL) \cdot r_0(ACK) \cdot \neg AP_{\frac{k^3}{4}-1} \cdot s_{p_{\frac{k^3}{4}}}(IoT(-)) + \\
& \quad \quad s_0(MIGRATEout) \cdot r_0(C) \cdot \neg AP_{\frac{k^3}{4}-1} \cdot s_{p_{\frac{k^3}{4}}}(IoT(C))\Big) \\
& \quad \quad \quad \lhd\ i = \frac{k^3}{4} - 1 \rhd \\
& \quad \quad \quad \Big(s_{p_{i+1}}(IoT(i)) \cdot W_{i+1}\Big)\Big)\bigg)\bigg)\Bigg)
\end{aligned} \tag{7.2}$$

$$\begin{aligned}
H_i = \sum_{i=0}^{\frac{k^3}{4}-1} & \bigg(\Big(r_0(INIT) \cdot FOG_0 + r_0(MIGRATEin) \cdot FOG_{C\to 0}\Big) \cdot s_0(0) \\
& \lhd\ i = 0 \rhd \\
& \bigg(\bigg(\Big(r_0(KILL) \cdot \neg FOG_{\frac{k^3}{4}-1} + r_0(MIGRATEout) \cdot \neg FOG_{\frac{k^3}{4}-1\to C}\Big) \cdot s_0(ACK)\bigg) \\
& \quad \lhd\ i = \frac{k^3}{4} - 1 \rhd \\
& \quad \Big(r_0(MOVE(i-1,i)) \cdot s_1(RQT(i,i-1)) + r_1(RQT(i+1,i)) \cdot s_1(COPY(i,i+1)) \\
& \quad + r_1(COPY(i-1,i)) \cdot FOG_i \cdot s_1(ERASE(i,i-1)) + r_1(ERASE(i+1,i)) \cdot \neg FOG_i \cdot \\
& \quad s_1(DONE(i,i+1)) + r_1(DONE(i-1,i)) \cdot s_0(i)\Big)\bigg)\bigg) \cdot H_i
\end{aligned} \tag{7.3}$$

$$\begin{aligned}
E_j = \sum_{j=0}^{\frac{k^2}{2}-1} & \Bigg(\sum_{p=0}^{K-1} \bigg(r_p(MSG(x,y)) \cdot \\
& \Big(s_{b_{|k/2}}(MSG(x,y)) \lhd \left\lfloor \frac{a}{k/2} \right\rfloor = \left\lfloor \frac{b}{k/2} \right\rfloor \rhd \sum_{q=\frac{k}{2}}^{k-1} s_q(MSG(x,y))\Big)\bigg)\Bigg) \cdot E_j
\end{aligned} \tag{7.4}$$

$$A_j = \sum_{j=0}^{\frac{k^2}{2}-1} \left(\sum_{p=0}^{K-1} \left(r_p(MSG(x,y)) \cdot \right.\right.$$
$$\left.\left. \left(s_{\left\lfloor \frac{b}{k/2} \right\rfloor}(MSG(x,y)) \lhd \left\lfloor \frac{a}{(k/2)^2} \right\rfloor = \left\lfloor \frac{b}{(k/2)^2} \right\rfloor \rhd \sum_{q=\frac{k}{2}}^{k-1} s_q(MSG(x,y)) \right) \right) \right) \cdot A_j \tag{7.5}$$

$$C_l = \sum_{l=0}^{\frac{k^2}{4}-1} \left(\sum_{q=0}^{k-1} r_q(MSG(x,y)) \cdot \left(s_{\left\lfloor \frac{b}{(k/2)^2} \right\rfloor}(MSG(x,y)) \right) \right) \cdot C_l \tag{7.6}$$

7.4 Scenario 2: two-way and resource restrictions

This second scenario portrays physical devices going both ways, such as getting in or out of the system at any both ends, as well as moving around to the left or to the right at any given time. Therefore, IoT devices may get into the system or get off it at wireless nodes located at both ends, such as 0 and $\frac{k^3}{4} - 1$, and they may change the sense of direction at random. Furthermore, there are storage constraints, which bring it closer to real deployments, inducing that generation and migration of VMs may be undertaken by another host located either in the same pod or not, in case the host directly connected to a particular relevant wireless node has not enough available resources to do it, or otherwise, in the worst case scenario, VMs might be neither created nor migrated if no resources are available within the whole pod.

Taking the previous model as a base case scenario, there are some changes being introduced in order to show the characteristics of this new scenario:

- Path directions may be distinguished, hence, r_{p_i} means taking path i from left to right, whereas $r_{p'_i}$ means doing it the other way around, hence the model for W_i needs to be modified to reflect each path direction, taking into consideration that the moving item may change direction at any moment.
- VMs are located in any given host, hence, they may be considered to be in a generic location g, and the process of binding of an IoT device with its associated VM will be defined by $IoT(g)$. Furthermore, if a VM is not created due to the lack of available resources in any host, it is reflected by using NAK.

- VM operations, established from host to host through the switching structure, thus through the fat tree interconnections, may include a further argument at the beginning generically called A, standing for *Action*, indicating whether it is about *Initializing* (I), *Killing* (K) or *Moving* (M). Hence, generic messages have the structure $MSG(A, x, y)$, extending the one explained in scenario 1.

In order to deal with the searching for another host, either for initializing or moving actions, a general strategy may be established. To start with, the host directly connected to the wireless node where the moving item is associated at a given time is to be assessed, that being the ideal case. If such a host is not able to allow a new VM inside due to the lack of available resources, the hosts connected to the same edge switch will be checked out. Next, the hosts connected to the same pod may be scanned, and eventually, the hosts elsewhere within the topology may be looked up.

Regarding the different kinds of VM control flow, those are the appropriate messages taking part on each one, along with their meaning:

- *Initializing a VM* ($INIT_i()$):
  - First off, host i is checked out looking for available resources to incorporate a new VM by using $FREE_i$, and if that is the case, then that VM is created by means of FOG_i.
  - On the contrary, if that is not the case, query messages QRY(i,d) are sent to search for available resources in other hosts, looking for an appropriate host d, according to the order proposed above.
  - The answer for such queries are Response messages RSP(d,i), which may have two possible answers, such as $FREE_d$ or $FAIL$.
  - In case of the former, it implies that a distant host d meets the requirements, and in turn, an $ASSIGN$(i,d) message will be issued in order to create the appropriate VM in that remote host d, being responsed back with a $DONE$(d,i) message, meaning that the computing assets associated with the moving device located in wireless node i are stored in host d.

  - In case of the latter, a *NAK* message is sent over, meaning that there are no available resources for new VMs in the system, thus, the moving device is kicked off the system.
  - Eventually, if a VM has been created in any host within the fat tree structure, its generic host identifier g is forwarded on to the wireless node for the moving device to get bound to it, where g accounts for the host where the VM has been created, either i or d.

- *Migrating a VM* ($MOVE_i(g, g')$):

  - The process of searching for the closest host with available resources to store a VM is quite similar than the one described above, in a way that, first of all, the host i must be checked out in order to look for available resources to undertake the migration, which is done by means of $FREE_i$.
  - If this is not the case, it is carried out the exchange of QRY(i,d) and RSP(d,i) messages in order to look for the closest host d with enough available resources, which is shown by $FREE_d$, where d accounts for the host identifier being queried upon.
  - That way, if such a response is attained at any point, then a comparison needs to be run in order to check whether the available host d is closer from host i than the host currently keeping the VM, identified as host g.
  - This action will be regarded by comparing the hop count between host i and host g (current hop distance, expressed as $HOP\#_{i-g}$) with the hop count between host i and host d (new hop distance, exposed as $HOP\#_{i-d}$), and only in case the latter is lower, then VM migration will take place between host g and host d.
  - On the one hand, if VM migration needs to be performed, its related message exchange is similar to that explained for the previous scenario, although a new parameter is included to specify host i in order to kick off the migration process and to get the acknowledge receipt at the end of it.

  - Furthermore, the host identifier of the new host holding the VM is forwarded on to the wireless node for the moving device to get bound to it, which is sent as a generic host g', which includes the cases where host i or host d is the new VM destination. It is to be noted that variable g' in $MOVE_i(g, g')$ remains empty until this point, where g' gets assigned.
  - Therefore, these messages are analogous to those explained in the previous case scenario, such as $RQT_i(g', g)$, $COPY_i(g, g')$, $ERASE_i(g', g)$ and $DONE_i(g, g')$, even though it may be considered that an additional message to the local host i must be sent stating the new value of g', whose value may be i or d, and such a message is $RELAY_i(g')$, which is used to confirm back the operation in a successful manner.
  - On the other hand, if VM migration is not carried out, then a message with the host identifier g is forwarded on to the wireless node, meaning that VM migration is not necessary, thus VM stays on the same host as it is allocated at that point.

- *Killing a VM* ($KILL_i(g)$):
  - That process just terminates a VM standing on generic host g, which accounts for the case where a moving device which is about to leave the system and it does not need its VM any longer.
  - Such an action is performed through a couple of messages, where the first one is $DELETE_i(g)$ and the second one is $DONE_i()$, followed by an ACK message sent back to the wireless node i, causing the moving device to get disassociated from the AP where the moving device is connected to, which in turn, takes such a device out of the system.

- *Migrate a VM in* ($MIGRATE\_IN_i(C, g)$):
  - The concept is analogous to $INIT_i()$, as both require a host with enough available resources to allocate a VM into the system. However, in this case, C represents a location in a cloud of the VM associated with the moving

device entering into the system, whilst g does it for the host where that VM may be moved in.

- *Migrate a VM out* ($MIGRATE\_OUT_i(g, C)$):

  – The concept is analogous to $KILL_i(g)$, as both require a VM to be taken out of the system. However, in this case, g represents the host where the VM associated with the moving device leaving the system may be allocated, whereas C does it for a location in a cloud where that VM may be moved out.

On a side note, it is to be noted that the general layout proposed for both scenarios implies a fog implementation being deployed along a given trajectory, that being rectilinear or otherwise. This fact suggests that the only gateways between the system and the outside world are both ends of the design, meaning that the wireless relays being located right at both ends are the only points where a moving IoT device may get in or get off the system, whereas the rest of wireless relays just have a path to their direct sequential neighbours along the trajectory. Hence, that original layout might be slightly modified in order to obtain other specific features.

In that sense, there might be possible to permit moving IoT devices to join or leave the fog domain at any wireless relay, thus making the model more suitable for its deployment in a neighbourhood or in open field. For that matter, it might only be necessary to extend the capabilities of the wireless relays located at both ends to all wireless relays regardless their location, whilst the rest of the layer may not be affected in any way. Additionally, there might also be possible to allow more interconnections among wireless relays, thus making possible the establishment of more directly connected neighbours. This may represent a different interconnection outline at the wireless relay layer, even though the rest of layers may not be affected.

Anyway, here they are the algebraic expressions for the five entities described building up scenario 2, such as W_i in (7.7), H_i in (7.8), E_j in (7.9), A_j in (7.10) and C_l in (7.11).

$$
\begin{aligned}
W_i = \sum_{i=0}^{\frac{k^3}{4}-1} & \Bigg(\Bigg(\Big(r_{p_0}(IoT(-)) \cdot AP_0 \cdot s_0(INIT_i) + r_{p_0}(IoT(C)) \cdot AP_0 \cdot \\
& s_0(\mathit{MIGRATE\_IN}_i(C,g))\Big) \cdot r_0(g) \cdot BIND(IoT(g)) \cdot \Big(s_{p_1}(IoT(g)) \cdot W_1 \\
& + UNBIND(IoT(g)) \cdot \Big(s_0(KILL_i(g)) \cdot r_0(ACK) \cdot \neg AP_0 \cdot s_{p'_0}(IoT(-)) + \\
& s_0(\mathit{MIGRATE\_OUT}_i(g,C)) \cdot r_0(C) \cdot \neg AP_0 \cdot s_{p'_0}(IoT(C))\Big)\Big) \\
& + r_0(NAK) \cdot \neg AP_0 \cdot s_{p'_0}(IoT(-))\Big) \\
& \triangleleft\ i = 0 \triangleright \\
& \quad \Bigg(\Bigg(\Big(r_{p'_{\frac{k^3}{4}-1}}(IoT(-)) \cdot AP_{\frac{k^3}{4}-1} \cdot s_0(INIT_i) + r_{p'_{\frac{k^3}{4}-1}}(IoT(C)) \cdot AP_{\frac{k^3}{4}-1} \cdot \\
& \quad s_0(\mathit{MIGRATE\_IN}_i(C,g))\Big) \cdot r_0(g) \cdot BIND(IoT(g)) \cdot \Big(s_{p'_{\frac{k^3}{4}-1}}(IoT(g)) \cdot W_{\frac{k^3}{4}-2} \\
& \quad + UNBIND(IoT(g)) \cdot \Big(s_0(KILL_i(g)) \cdot r_0(ACK) \cdot \neg AP_{\frac{k^3}{4}-1} \cdot s_{p_{\frac{k^3}{4}}}(IoT(-)) + \\
& \quad s_0(\mathit{MIGRATE\_OUT}(g,C)) \cdot r_0(C) \cdot \neg AP_{\frac{k^3}{4}-1} \cdot s_{p_{\frac{k^3}{4}}}(IoT(C))\Big)\Big) \\
& \quad + r_0(NAK) \cdot \neg AP_{\frac{k^3}{4}-1} \cdot s_{p_{\frac{k^3}{4}}}(IoT(-))\Big) \\
& \quad \triangleleft\ i = \frac{k^3}{4} - 1 \triangleright \\
& \qquad \Bigg(\Bigg(r_{p_i}(IoT(g)) \cdot \neg AP_{i-1} \cdot AP_i \cdot s_0(MOVE_i(g,g')) \cdot \\
& \qquad \Big(r_i(g') \cdot UNBIND(IoT(g)) \cdot BIND(IoT(g')) + r_0(g)\Big) \cdot \\
& \qquad\quad \Bigg(\Bigg(UNBIND(IoT(g)) \cdot \Big(s_0(KILL_i(g)) \cdot r_0(ACK) \cdot \neg AP_{\frac{k^3}{4}-1} \cdot s_{p_{\frac{k^3}{4}}}(IoT(-)) + \\
& \qquad\quad s_0(\mathit{MIGRATE\_OUT}_i(g,C)) \cdot r_0(C) \cdot \neg AP_{\frac{k^3}{4}-1} \cdot s_{p_{\frac{k^3}{4}}}(IoT(C))\Big) \\
& \qquad\quad + s_{p'_{\frac{k^3}{4}-1}}(IoT(g)) \cdot W_{\frac{k^3}{4}-2}\Bigg) \\
& \qquad\quad \triangleleft\ i = \frac{k^3}{4} - 1 \triangleright \\
& \qquad\quad \Big(s_{p_{i+1}}(IoT(g)) \cdot W_{i+1} + s_{p'_i}(IoT(g)) \cdot W_{i-1}\Big)\Bigg)\Bigg) \\
& \qquad\qquad + \Big(r_{p'_i+1}(IoT(g)) \cdot \neg AP_{i+1} \cdot AP_i \cdot s_0(MOVE_i(g,g')) \cdot \\
& \qquad\quad \Big(r_i(g') \cdot UNBIND(IoT(g)) \cdot BIND(IoT(g')) + r_0(g)\Big) \cdot \\
& \qquad\qquad \Bigg(\Bigg(UNBIND(IoT(g)) \cdot \Big(s_0(KILL_i(g)) \cdot r_0(ACK) \cdot \neg AP_0 \cdot s_{p'_0}(IoT(-)) + \\
& \qquad\qquad s_0(\mathit{MIGRATE\_OUT}_i(g,C)) \cdot r_0(C) \cdot \neg AP_0 \cdot s_{p'_0}(IoT(C))\Big) \\
& \qquad\qquad + s_{p_1}(IoT(g)) \cdot W_1\Bigg) \\
& \qquad\qquad \triangleleft\ i = 0 \triangleright \\
& \qquad\qquad (s_{p'_i}(IoT(g)) \cdot W_{i-1} + s_{p_{i+1}}(IoT(g)) \cdot W_{i+1}))\Bigg)\Bigg)\Bigg)\Bigg) \qquad (7.7)
\end{aligned}
$$

$$H_i = \sum_{i=0}^{\frac{k^3}{4}-1} \Bigg(\Bigg(r_0(\mathit{INIT}_i())\cdot$$

$$\Big(\mathit{FOG}_i \cdot s_0(i)$$

$$\lhd\ \mathit{FREE}_i \rhd$$

$$\Big(\sum_{\substack{d=j\cdot k/2\\ d\neq i}}^{((j+1)\cdot(k/2)-1)} s_1(\mathit{QRY}(I,i,d))\cdot\Big(s_1(\mathit{ASSIGN}(I,i,d)) + r_1(\mathit{DONE}(I,d,i))\cdot s_0(d)$$

$$\lhd\ r_1(\mathit{RSP}(I,d,i)) = \mathit{FREE}_d \rhd$$

$$\Big(\sum_{\substack{d=j\cdot(k/2)^2\\ d\neq i}}^{((j+1)\cdot(k/2)^2-1)} s_1(\mathit{QRY}(I,i,d))\cdot\Big(s_1(\mathit{ASSIGN}(I,i,d)) + r_1(\mathit{DONE}(I,d,i))\cdot s_0(d)$$

$$\lhd\ r_1(\mathit{RSP}(I,d,i)) = \mathit{FREE}_d \rhd$$

$$\Big(\sum_{\substack{d=0\\ d\neq i}}^{((k^3/4)-1)} s_1(\mathit{QRY}(I,i,d))\cdot\Big(s_1(\mathit{ASSIGN}(I,i,d)) + r_1(\mathit{DONE}(I,d,i))\cdot s_0(d)$$

$$\lhd\ r_1(\mathit{RSP}(I,d,i)) = \mathit{FREE}_d \rhd$$

$$s_0(\mathit{NAK})\Big)\Big)\Big)\Big)\Big)\Big)\Big)$$

$$+\, r_1(\mathit{ASSIGN}(I,d,i))\cdot \mathit{FOG}_i \cdot s_1(\mathit{DONE}(I,i,d))\Bigg)$$

$$+\Bigg(r_0(\mathit{KILL}_i(g))\cdot$$

$$\Big(\neg \mathit{FOG}_i \cdot s_0(\mathit{ACK})$$

$$\lhd\ i = g \rhd$$

$$s_1(\mathit{DELETE}(K,i,g)) + r_1(\mathit{DONE}(K,g,i))\cdot s_0(\mathit{ACK})\Big)$$

$$+\, r_1(\mathit{DELETE}(K,g,i))\cdot \neg \mathit{FOG}_i \cdot s_1(\mathit{DONE}(K,i,g))\Bigg)$$

$$
\begin{aligned}
&+ \Bigg(r_0(MOVE_i(g, g')) \\
&\Bigg(s_0(i) \\
&\lhd \; FOG_i \rhd \\
&\Bigg(s_1(RQT_i(M, i, g)) + r_1(COPY_i(M, g, i)) \cdot FOG_i \cdot s_1(ERASE_i(M, i, g)) \\
&\quad + r_1(DONE_i(M, g, i)) \cdot s_0(i) \\
&\qquad \lhd \; FREE_i \rhd \\
&\qquad \Bigg(\sum_{\substack{d=j\cdot k/2 \\ d\neq i}}^{((j+1)\cdot(k/2)-1)} s_1(QRY(M, i, d)) \cdot \Bigg(s_1(RQT_i(M, d, g)) + r_1(COPY_i(M, g, d)) \\
&\qquad\qquad \cdot FOG_d \cdot s_1(ERASE_i(M, d, g)) + r_1(DONE_i(M, g, d)) \cdot s_1(RELAY_i(M, d, i)) \\
&\qquad \lhd \; r_1(RSP(M, d, i)) = FREE_d \quad \&\& \quad HOP\#_{i,d} < HOP\#_{i,g} \rhd \\
&\qquad\quad \Bigg(\sum_{\substack{d=j\cdot(k/2)^2 \\ d\neq i}}^{((j+1)\cdot(k/2)^2-1)} s_1(QRY(M, i, d)) \cdot \Bigg(s_1(RQT_i(M, d, g)) + r_1(COPY_i(M, g, d)) \\
&\qquad\qquad \cdot FOG_d \cdot s_1(ERASE_i(M, d, g)) + r_1(DONE_i(M, g, d)) \cdot s_1(RELAY_i(M, d, i)) \\
&\qquad \lhd \; r_1(RSP(M, d, i)) = FREE_d \quad \&\& \quad HOP\#_{i,d} < HOP\#_{i,g} \rhd \\
&\qquad s_0(g))\Bigg)\Bigg)\Bigg)\Bigg)\Bigg)\Bigg) \\
&\quad + r_1(RQT_i(M, g, i)) \cdot s_1(COPY_i(M, i, g)) \\
&\quad + r_1(ERASE_i(M, g, i)) \cdot \neg FOG_i \cdot s_1(DONE_i(M, i, g)) \\
&\quad + r_1(RQT_i(M, d, g)) \cdot s_1(COPY_i(M, g, d)) \\
&\quad + r_1(ERASE_i(M, d, g)) \cdot \neg FOG_g \cdot s_1(DONE_i(M, g, d)) \\
&\quad + r_1(RELAY_i(M, d, i)) \cdot s_0(d) \Bigg)\Bigg) \cdot H_i
\end{aligned}
\tag{7.8}
$$

$$
\begin{aligned}
E_j = \sum_{j=0}^{\frac{k^2}{2}-1} \Bigg(\sum_{p=0}^{K-1} \Bigg(& r_p(MSG(A, x, y)) \cdot \\
& \Bigg(s_{b_{|k/2}}(MSG(A, x, y)) \; \lhd \; \left\lfloor \frac{a}{k/2} \right\rfloor = \left\lfloor \frac{b}{k/2} \right\rfloor \; \rhd \sum_{q=\frac{k}{2}}^{k-1} s_q(MSG(A, x, y)) \Bigg)\Bigg)\Bigg) \cdot E_j
\end{aligned}
\tag{7.9}
$$

$$A_j = \sum_{j=0}^{\frac{k^2}{2}-1} \left(\sum_{p=0}^{K-1} \left(r_p(MSG(A,x,y)) \cdot \right.\right.$$
$$\left.\left. \left(s_{\left\lfloor \frac{b}{k/2} \right\rfloor}(MSG(A,x,y)) \lhd \left\lfloor \frac{a}{(k/2)^2} \right\rfloor = \left\lfloor \frac{b}{(k/2)^2} \right\rfloor \rhd \sum_{q=\frac{k}{2}}^{k-1} s_q(MSG(A,x,y)) \right) \right) \right) \cdot A_j \tag{7.10}$$

$$C_l = \sum_{l=0}^{\frac{k^2}{4}-1} \left(\sum_{q=0}^{k-1} r_q(MSG(A,x,y)) \cdot \left(s_{\left\lfloor \frac{b}{(k/2)^2} \right\rfloor}(MSG(A,x,y)) \right) \right) \cdot C_l \tag{7.11}$$

7.5 Model verification

Regarding verification of the aforesaid models, it may be performed by different means. First of all, it is to be considered that the ordered sequence of wireless relays proposed, which are numbered from 0 all the way to $\frac{k^3}{4}-1$, is clearly order isomorphic to the initial segment of natural numbers up to $\frac{k^3}{4}-1$, ordered by $<$, as they both show an established total order, meaning that antisymmetry, transitivity and totality properties apply. With that in mind, verification might be carried out using proof by mathematical induction or proof by contradiction.

Focusing in the first case scenario, attention must be paid to what happens to both moving devices and their associated VMs. In the former, proof by induction is obvious, as after a moving device has visited W_n, it may only go to W_{n+1}, hence, an external item may only get into the system through W_0 and it only eventually get off the system from $W_{\frac{k^3}{4}-1}$. Otherwise, proof by contradiction is also obvious, as no item getting into the system through W_0 may stay forever therein, as it may eventually reach the exit point $W_{\frac{k^3}{4}-1}$, because a moving device might eventually move, as it is not static.

Likewise, in the latter, proof by induction is clear, as the movement of a moving item implies the migration of its associated VMs to a host with the same identifier as the wireless relay where the moving item is connected to, because there are no constrictions in this scenario, thus live VM migrations are always feasible. This way, when a moving device first gets into the system at W_0, its associated VM gets created at H_0, whereas it gets terminated at $H_{\frac{k^3}{4}-1}$ when the moving item gets out of the system through $W_{\frac{k^3}{4}-1}$, whilst in the meantime, if the moving item moves from W_n to W_{n+1}, then its associated VM does it from H_n to H_{n+1}. Otherwise, proof by contradiction is also clear, as no associated VM may always be into the system, as it may be delete when its associated moving device leaves.

Getting down to the second case scenario, which might be seen as an extension of the first one, attention must be also paid to what occurs with both moving devices and their associated VMs. In the former, proof by induction is obvious as this case just extends the first one, because an item standing at W_n may later on visit either W_{n+1} or W_{n-1}, and this fact implies that paths to reach any of both ends and eventually leave the system may take random walks, making a variable number of steps to do so, but it finally happens. In that sense, proof by contradiction is also obvious as no item may be kept forever into the system, as any item may at some stage reach out a wireless node located in the border and eventually leave the system, although the number of steps to do so may be undetermined, but a lower bound may be found, which is just the first case related before.

On the other hand, in the latter, proof by induction is clear, as even though an associated VM may be held in a host with a different identifier than the wireless relay where the moving item is connected to, the system may try to allocate such a VM as close as possible to its associated item whenever it moves to another wireless relay, whilst entry and exit points for those moving devices are W_0 and $W_{\frac{k^3}{4}-1}$, respectively, whereas H_0 to $H_{\frac{k^3}{4}-1}$ are the preferred initial hosts for their associated VMs, and the next alternative solutions are another host hanging on the same end switch, following by another host located in the same pod, and finally another host located in another pod. Otherwise, proof by contradiction is also clear, as each associated VM is terminated when its moving device gets off the system, thus, no associated VM may be located forever into the system.

Putting the focus on ACP, an alternative verification technique may be employed by means of comparing the external behaviour of the model with that of the real system and studying whether they both are rooted branching bisimilar, that being a sufficient condition for verification to be considered.

It is to be noted that there are 5 layers involved in this model, where the upper layers correspond to the fat tree architecture, which have already been verified in (68), (69) and (70). Additionally, as commented before, any other type of switching topology might have been used in this model, such as the ones presented, modelled and verified in previous sections. Hence, the communication infrastructure may not make any difference when dealing with the verification of alternative switching models.

Therefore, it may seem clear that the switching topology is the base of this model, where the hosts are also included in it, as well as the wireless relays (along with its coverage areas) hanging on each of those hosts. This way, if the model is considered as a black box, then *scenario 1* only has a way in (through the left hand side) and a way out (through the right hand side) of every wireless node, whereas *scenario 2* has two ways in and other two ways out (through both the left hand side and the right hand side). Both cases as depicted in Figure 7.2.

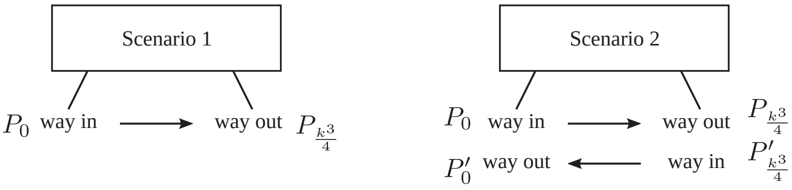


Figure 7.2: External behaviour of scenario 1 and scenario 2.

Additionally, it may seem clear that each layer just communicates with its directly connected layers, such as its lower and upper layers, hence, every other communication attempt may end up going deadlock. Furthermore, all communications exposed in the models are internal, except for the paths getting into and out of the system through the specific wireless relays located at both ends, namely, node 0 and node $\frac{k^3}{4} - 1$. Therefore, the encapsulation operator may convert all internal atomic actions into internal communications, and in turn, the abstraction operator may mask all internal atomic actions and communications, hence just revealing external atomic actions.

In this context, on the one hand, the first case scenario presents the following expressions for the communication among wireless relays in a concurrent manner, along with the encapsulation operator and the abstraction operator, and taking into account that moving devices may get into the system through path p_0 with either an associated VM located up in the cloud or without any VM whatsoever, whereas they may also get off the system in any of both ways, through path $p_{\frac{k^3}{4}-1}$, as in (7.12).

$$\tau_I\Big(\partial_H\Big(W_i \mid\mid H_{i'} \mid\mid E_j \mid\mid A_{j'} \mid\mid C_l\Big)\Big) = \Big(r_{p_0}\Big(IoT(-) + IoT(Cloud)\Big) + s_{p_{\frac{k^3}{4}-1}}\Big(IoT(-) + IoT(Cloud)\Big)\Big) \cdot \tau_I\Big(\partial_H\Big(W_i \mid\mid H_{i'} \mid\mid E_j \mid\mid A_{j'} \mid\mid C_l\Big)\Big) \qquad (7.12)$$

On the other hand, the second case scenario shows the following expressions for the communication among wireless nodes, adding up the encapsulation operator and the abstraction operator, considering the same conditions as before, along with the possibility of having bidirectional paths, as in (7.13).

$$\tau_I\Big(\partial_H\Big(W_i \mid\mid H_{i'} \mid\mid E_j \mid\mid A_{j'} \mid\mid C_l\Big)\Big) = \Big(r_{p_0}\Big(IoT(-) + IoT(Cloud)\Big) + r_{p'_{\frac{k^3}{4}-1}}\Big(IoT(-) + IoT(Cloud)\Big) + s_{p_{\frac{k^3}{4}-1}}\Big(IoT(-) + IoT(Cloud)\Big) + s_{p'_0}\Big(IoT(-) + IoT(Cloud)\Big)\Big) \cdot \tau_I\Big(\partial_H\Big(W_i \mid\mid H_{i'} \mid\mid E_j \mid\mid A_{j'} \mid\mid C_l\Big)\Big) \qquad (7.13)$$

Otherwise, the behaviour of the real system is described in a similar way, in both the first and the second case scenarios, as in (7.14) and (7.15), respectively.

$$X_1 = \Big(r_{p_0}\Big(IoT(-) + IoT(Cloud)\Big) + s_{p_{\frac{k^3}{4}-1}}\Big(IoT(-) + IoT(Cloud)\Big)\Big) \cdot X_1 \qquad (7.14)$$

$$X_2 = \Big(r_{p_0}\Big(IoT(-) + IoT(Cloud)\Big) + r_{p'_{\frac{k^3}{4}-1}}\Big(IoT(-) + IoT(Cloud)\Big) + s_{p_{\frac{k^3}{4}-1}}\Big(IoT(-) + IoT(Cloud)\Big) + s_{p'_0}\Big(IoT(-) + IoT(Cloud)\Big)\Big) \cdot X_2 \qquad (7.15)$$

Therefore, it may seem clear that the behaviour of both models and their corresponding real systems exhibit the same string of actions and the same branching structure, as seen in (7.16) for the first model and in (7.17) for the second one.

$$\tau_I\Big(\partial_H\Big(W_i \mid\mid H_{i'} \mid\mid E_j \mid\mid A_{j'} \mid\mid C_l\Big)\Big) \longleftrightarrow X_1 \qquad (7.16)$$

$$\tau_I\Big(\partial_H\Big(W_i \mid\mid H_{i'} \mid\mid E_j \mid\mid A_{j'} \mid\mid C_l\Big)\Big) \longleftrightarrow X_2 \qquad (7.17)$$

Hence, they are obviously rooted branching bisimilar, which is a sufficient condition for a model to get verified.

7.6 Summary

In this chapter, a particular instance of the model presented in chapter 6 has been put in place by supposing a linear path between any pair of successive wireless nodes, thus provoking them to be visited on a sequential manner. This scheme may not necessary imply the use of rectilinear trajectories to join any pair of neighbouring wireless nodes, as any other type of trajectory may be topologically equivalent, as long as there is no connection between any pair of non-neighbouring wireless nodes. Moreover, each wireless node is connected to a host, which is interconnected with the rest of hosts through a fat tree topology, even though any other may work out as well by just adapting the network interconnections within the design.

In this context, two different scenarios have been presented. On the one hand, the first scheme is an ideal one, where IoT devices may only move from left to right, which implies that each of those moving devices may only get into the system through the leftmost wireless node and may get out of the system through the rightmost one. Furthermore, there is no resource constraints in the hosts, hence no space limitation is set in the hosts, making possible to keep as many VM in as necessary, meaning that they will always be available to add an extra VM.

On the other hand, the second scenario is a more realistic one, where IoT devices may move both ways, this is, from left to right and the other way around, meaning that both wireless ends may be the entry and the exit point, as those devices may change direction at any time. Additionally, there are resource constrictions in the hosts when it comes to keeping VM in, thus making possible that some VM migrations might not be performed due to lack of availability in a given host.

Both schemes have been verified by different means, such as mathematical induction, proof by contradiction and by using ACP as described in the previous chapter, obtaining a successful verification in all of them.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis dissertation may be based on three different pillars, such as ACP, DC topologies and live VM migration. Regarding the first one, it is an abstract algebra which may be applied to describe processes without taking care of their real implementation details. Hence, it is a convenient solution for expressing the behaviour of non-deterministic systems, as only the relevant features prevail.

ACP defines the behaviour of an entity being part of a model by means of process terms, which are composed by atomic actions such as send or receive a message through a given channel towards another entity within the model, or otherwise, getting into the model or out of the model. In addition to it, a range of operators such as sequential, alternate, merge or conditional may be applied to those actions so as to describe how each entity works.

Once all entities belonging to a model are defined in algebraic terms, further operators are applied to unveil the external behaviour of the model proposed, which is in turn compared to the behaviour of the real system, and if they both run the same string of actions and have the same branching structure, it may be said that they are rooted branching bisimilar, which is a sufficient condition for a model to be verified.

In other words, bisimulation equivalence implies that a behavioural equivalence may be established between the process terms being part of an equational system describing the model and those related to the real system, as the equational logic of ACP is both sound and complete.

With regards to the second pillar, it is to be noted that fog environments do contain DCs with a small to medium size, hence huge designs and complex topologies may be discarded as a tradeoff may be necessary between efficiency and simplicity, as well as redundancy. Taking this into account, a range of generic designs have been presented, and in turn, proposed in order to fulfill those requirements, where special attention have been put in partial mesh topologies, as those are the most commonly used in production environments, although other designs have also been proposed.

It is to be reminded that ACP does not take time into consideration, which accounts for the models obtained to be timeless. This fact implies that performance in the DC topology models presented may not be measured in time units, as it usually happens, although an alternative view has been taken, such as measuring distance as a measure of time, provided that all links have the same characteristics. Hence, performance is measured herein as the minimum number of links between a given pair of devices.

As per the third pillar, it is focused on fog computing paradigm, which is an extension of cloud computing where the computing resources are located on the edge of the network, thus allowing for reduced latency and bandwidth use. This sort of cloud deployment is ideal for moving IoT devices, as those usually have limited computing resources and capabilities, and hence, an external server being close enough to such devices may be in charge of undertaking most of the processing whilst leaving them only with a few essential tasks. Hence, VM migration takes place within any couple of hosts connected by a DC topology, and that is modelled by using ACP.

Regarding the main hypotheses proposed in the Introduction section, it must be said that all of them have been satisfied. In that sense, chapter 5 proves that it is possible to model the VM migration path within a given DC topology being deployed in a fog environment by means of ACP, as it has actually been carried out and verified for all 15 DC topologies proposed. Morevoer, chapter 6 displays that it is possible to construct a generic model for a fog ecosystem in ACP, where the DC topology is abstracted away, as it has actually been represented and verified. Besides, chapter 7 exhibits that it is possible to build up a particular instance of such a general model in

ACP which is being focused on linear deployments, as it has actually been designed and verified for two different case scenarios.

With respect to the secondary hypotheses presented in the Introduction section, it must be noted that all of them have also been met for all topologies presented, according to the diverse subsections of chapter 5. In that sense, it is possible to model the VM migration path within a particular DC topology being implemented in a fog domain by using flow charts and pseudocode algorithms, whereas it is possible to obtain the list of devices and their appropriate ports for all redundant minimal paths for the models exposed in ACP, whilst it is possible to achieve the appropriate verification for the models expressed in ACP.

On the other hand, considering the goals proposed in the Introduction, they have all been met, as each of them has been developed in one of the chapters going from 2 to 7. In this sense, the review of the state of the art has been undertaken in chapter 2, where the three pillars cited above have been studied in detail. First off, a brief history of computing, cloud and fog has been exposed, leading to VM migration techniques. Next, an overview on FDT to obtain models has been carried out, going around the most commonly used, passing through process algebras and leading to ACP. Then, an outline of topologies for DC is fulfilled, where common network topologies are introduced and redundant topologies are exhibited.

Additionally, all those topologies have been associated with mathematical entities, hence the foundations of them have been presented in chapter 3 in order to deduce their main characteristics. In this sense, regular N-polytopes have been studied, exposing the main features of N-simplex, which may relate to full mesh topologies, followed by N-orthoplex, which may do to quasi full mesh designs, and going to N-hypercube, which may be proposed in its ordinary way, also known as plain, or otherwise, in its folded way, both being part of partial mesh designs.

Besides, a short introduction on trees has been exhibited, which may be seen as the base behaviour of the pair of tree-like designs presented herein, such as fat tree, which is a three-layer topology, or leaf and spine, which is a two-layer one. Both designs may be seen as partial mesh, where the latter offers better performance but it also suffers from scalability issues, whereas the former does not perform as good, but it is more scalable. It is also to be noted that those designs offer steady values of latency, which is ideal for networks dealing with audio and video streaming contents, making them the preferred solution in many real deployments.

Moreover, an overview of graph theory has also been presented, which may reveal the basics on directed and undirected graphs, where the de Bruijn ones have been seen as examples of the former, whilst the binary grid, a particular Hamming one and the simplest cage ones have been viewed as instances of the latter. Also, the foundations of toroidal structures have also been depicted in order to better understand how they work and the way they might be applied in the field of network designs.

Furthermore, some common network topologies for data centres are reviewed in chapter 4, paying special attention to partial mesh topologies, which are the ones typically used in production environments. In this context, tree-like designs are exposed along with a couple of instances, such as fat tree and leaf and spine, whereas graph-like designs are exhibited along with a pair of instances, such as N-hypercube and folded N-hypercube. Subsequently, all four designs are compared regarding the average distances among hosts and end switches, which after a careful analysis, it was found out that there is not a single topology being better in all situations.

Later on, other topologies are presented with a brief exposition of their main features. Those designs are full mesh, quasi full mesh, redundant hub and spoke, redundant ring, toroidal ring, de Bruijn graph, de Bruijn reverse graph, binary grid, Hamming graph, Petersen graph and Heawood graph. All of them, along with the instances of partial mesh cited above are the group of 15 topologies being put forward.

Afterwards, putting all together, each of the topologies proposed have been studied in chapter 5 according to the following order: a short preamble so as to indicate its core functions, a layout stating the nomenclature of its nodes, a flow chart in order to see how the algorithms may work, a pseudo code algorithm where the proper values are quoted, an algebraic definition where the behaviour of the model is cited by means of ACP syntax and semantics, a section about how to calculate the redundant paths, and eventually, a verification of the model proposed according to ACP in order to see if the model and the real system have both the same string of actions and the same branching structure, which is a sufficient condition to get the model verified.

It is to be noted that tree-like designs, such as fat tree and leaf & spine, and hub & spoke have switches with different roles, those being end switches or otherwise, and in those cases, each different category has been studied on its own by following the steps quoted above in a sequential order until the ACP model has been obtained. Afterwards, when ACP models for each type have been attained, then the redundant paths and the verification have been calculated for the whole structure. On the other hand, all graph-like designs are composed by switches with the same role, those being all end switches, and in such cases, they have been studied according to the steps cited above in a sequential manner, all the way to perform the verification.

The DC topologies presented and verified with ACP are fat tree, leaf and spine, N-hypercube, folded N-hypercube, full mesh, quasi full mesh, redundant hub & spoke, redundant ring, toroidal ring, de Bruijn graph, reverse de Bruijn graph, binary grid, Hamming graph, Petersen graph and Heawood graph. Moreover, the optimal VM migration paths have been modelled by means of ACP in all those cases.

After having exposed all those topologies, a generic model for a fog environment has been presented in chapter 6, based on a block diagram. It is to be noted that the block corresponding to the switching topology, also known as internetworking topology, has been abstracted away, such that it has been considered as a black box, which only has entry and exit points. That block is used to interconnect the hosts where the VMs being part of the fog domain are located, hence, it comprises the whole set of channels being employed to communicate one host to the others. This way, any of the topologies proposed above might be assimilated in the generic model, and taking into account that all of them have been already verified, it may be said that any of them accounts for a good switching solution for this scheme.

Apart from the internetworking topology block, other blocks have been proposed, such as a generic wireless relay block, where the moving IoT devices get into and out of the system, which triggers all actions available in the model. Besides, an orchestrator takes the decision as to which host is involved with the creation, migration or termination of a given VM associated with a particular moving IoT device. Furthermore, there is a generic host block, which represents a given host within the fog domain, considering that two of those host blocks are necessary to describe migration actions. Moreover, a cloud block is presented in order to denote VMs going up to the cloud from the fog, of the other way around.

This generic model accounts for six different actions, such as cloud in, migrate in, init, cloud out, migrate out and kill. The former three represent actions where the moving IoT device comes into a wireless relay, such that the first one implies that the IoT device has already a VM up in the cloud, whilst the second one denotes that it already has a VM in a previous host within that fog domain, and the third one states that an IoT device gets into the fog domain without any VM associated.

On the other hand, the latter three describe actions where the moving IoT device goes out of a wireless relay, such that the first one denotes that the IoT moves out to the cloud with its VM, whereas the second one implies that the IoT device keeps its VM within the fog domain, and the third one determines that an IoT device gets out of the fog domain and it does not need its associated VM any more.

It is to be said that all six actions have been expressed in ACP, revealing the external behaviour of all six models. Also, the real behaviour of those six actions have been expressed in ACP. Therefore, each modelled action has been compared to its corresponding real action, resulting that in all cases, those six pairs of actions have the same string of actions and the same branching structure, leading to the conclusion that they are both rooted branching bisimilar, which is a sufficient condition for the models to be verified.

Eventually, a specific instance of the aforesaid model has been proposed in chapter 7, where a linear deployment being composed of wireless relays has been designed, considering that those have been located sequentially, which permits to identify them in a linear manner. With that in mind, this implementation may be ideal for railways, highways or pipelines, as the adjective *linear* relates to a string of wireless relays traversed in a sequential way and not to a rectilinear implementation.

It is to be noted that each of those wireless nodes are connected to a different host, which in turn, are hanging on an interconnection topology. It has been decided to use fat tree, as the three layer design offers the possibility of having a good quantity of hosts with a steady bandwidth. However, any other type of topology might have been used and the model would have behaved in the same way.

Two case scenarios have been proposed, such that the first one is an ideal case where all hosts have unlimited resources to keep VMs in, meaning that no request may be rejected, and also the moving direction is only left to right, meaning that items may only enter into the system at the left end and they may only leave it at the right end. Otherwise, the second one is a more realistic case where hosts have limited resources to hold VMs in and moving IoT devices may change directions at any time, even though those items may only get into the system from any of both ends, and that is why the target for those deployments are railways or pipelines.

In each of those cases, the model for each layer has been represented by means of ACP, considering the layers related to wireless relays and hosts, as well as the layers corresponding to the switching topology. Fat tree has been selected in this design, hence its corresponding three layers have been modelled, such as edge, aggregation and core, even though any other topology might actually have had its own layers. Eventually, both models have been verified, each one for its specific case scenario.

In summary, all three main hypotheses have been verified, where chapters 5, 6 and 7 have been dedicated to each of them, along with all three secondary hypotheses, which have been achieved on chapter 5 for all DC topologies presented. Additionally, all six objectives proposed have also been met, where each chapter from 2 to 7 has been devoted to cover one of those particular goals.

8.2 Future work

Regarding the future work and open research issues, the following points may be taken into account.

- The addition of time to the models of the different switching structures proposed, which may lead to achieve a more realistic approach, thus allowing to calculate performance measurements in time units.
- The consideration of specific parameters in the different links within the switching topologies, apart from just routing, such as the type of protocols implemented, the percentage of link usages or the rate of link delays.

- The application of some type of artificial intelligence to the switching topologies, by means of an orchestrator or otherwise, in order to get them adaptive to changing conditions so as to always get the fittest solution for packet forwarding.

- The study of other switching topologies being better fit for medium to large DCs, such as some variations of tree-like and graph-like designs, which achieve better performances at the cost of having more complex designs.

- The expansion of the general model presented for fog computing with the addition of blocking probabilities for IoT devices to get associated with wireless relays or for their associated VM to be migrated to hosts.

- The addition of time to the general model to make it more realistic, which also may allow to include queueing probabilities when accessing each building block.

- The extension of the general model so as to include some specific communication protocol for IoT devices, such as MQTT or CoAP, in order to better adapt it to real IoT message exchange flows.

- The implementation of the linear model proposed for fog computing with different switching topologies in order to compare the expressions obtained. A good exercise may be to do it with all 15 topologies presented herein.

- The variation in the linear model where an IoT device may join and leave the system at any wireless node, which basically may account for extending the capability of the outer nodes (create, migrate, terminate VMs) to every node. It is to be noted that, in such a case, all nodes may behave the same way, thus permitting an IoT device not to go through all wireless nodes.

- The study of different types of interconnections among wireless nodes within the linear model, thus obtaining a non linear design, which may permit redundant paths to get from a given source to a given destination, as well as avoiding single points of failure regarding wireless nodes.

All those items are out of the scope of the current research, although they might be addressed in further research in the future.

Chapter 9

Publications

The list of Papers generated by this thesis is presented below:

- Roig, P.J., Alcaraz, S., Gilly, K., *Formal Specification of Spanning Tree Protocol Using ACP*, in Elektronika Ir Elektrotechnika, 2017, Vol. 23(2), pp. 84-91. DOI: https://doi.org/10.5755/j01.eie.23.2.18005. Reference: [163].

- Roig, P.J., Alcaraz, S., Gilly, K., *Algebraic specification of ABP protocol using different time constraints*, in Proc. of the 21st International Conference ELECTRONICS, Palanga (Lithuania), 2017, pp. 1-6, IEEE Explore. DOI: https://doi.org/10.1109/ELECTRONICS.2017.7995232. Reference: [162].

- Roig, P.J., Alcaraz, S., Gilly, K., *Multicast Algebraic Formal Modelling using ACP - Study on PIM Dense Mode and Sparse Mode*, in Proc. of the 14th International Joint Conference on e-Business and Telecommunications - Volume 1: DCNET, Madrid (Spain), 2017, pp. 57-68, SciTePress. DOI: https://doi.org/10.5220/0006398800570068. Reference: [165].

- Roig, P.J., Alcaraz, S., Gilly, K., *FTP Algebraic Formal Modelling using ACP - Study on FTP Active Mode and Passive Mode*, in Proc. of the 7th Int. Conf. on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH, Madrid (Spain), 2017, pp. 362-373, SciTePress. DOI: https://doi.org/10.5220/0006465703620373. Reference: [164].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *Study on OSPF Algebraic Formal Modelling Using ACP*, in Elektronika Ir Elektrotechnika, 2018, Vol. 24(4), pp. 77-83. DOI: https://doi.org/10.5755/j01.eie.24.4.21484. Reference: [179].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *Study on Mobility and Migration in a Fog Computing Environment*, in Proc. of the 22nd International Conference ELECTRONICS, Palanga (Lithuania), 2018, 1-6, IEEE Explore. DOI: https://doi.org/10.1109/ELECTRONICS.2018.8443636. Reference: [178].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *OSPF Algebraic Formal Modelling using ACP - A Formal Description on OSPF Routing Protocol*, in Proc. of the 15th International Joint Conference on e-Business and Telecommunications - Volume 1: ICETE, Porto (Portugal), 2018, pp. 55-66, SciTePress. DOI: https://doi.org/10.5220/0006838702210232. Reference: [177].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *Algebraic Formal Modelling for EIGRP using ACP - Formal Description Modelling on EIGRP Routing Protocol*, in Proc. of 8th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH, Porto (Portugal), 2018, pp. 259-266, SciTePress.
DOI: https://doi.org/10.5220/0006838802590266. Reference: [176].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *Modelling VM Migration in a Fog Computing Environment*, in Elektronika Ir Elektrotechnika, 2019, Vol. 25(5), pp. 75-81. DOI: https://doi.org/10.5755/j01.eie.25.5.24360. Reference: [181].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *Algebraic Formal Modelling for HTTP Main Methods using ACP*, in Proc. of the 23rd International Conference ELECTRONICS, Palanga (Lithuania), 2019, 1-6, IEEE Explore. DOI: https://doi.org/10.1109/ELECTRONICS.2019.8765572. Reference: [180].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *Modelling a Leaf and Spine Topology for VM Migration in Fog Computing*, in Proc. of the 24th Int. Conference

ELECTRONICS, Palanga (Lithuania), 2020, pp. 1-6, IEEE Explore. DOI: https://doi.org/10.1109/IEEECONF49502.2020.9141611. Reference: [182].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., Aknin, N., *MQTT Algebraic Formal Modelling Using ACP*, in Proc. of the 24th International Conference ELECTRONICS, Palanga (Lithuania), 2020, pp. 1-5, IEEE Explore. DOI: https://doi.org/10.1109/IEEECONF49502.2020.9141589. Reference: [188].

- Alcaraz, S., Roig, P.J., Gilly, K., Filiposka, S., Aknin, N., *Formal Algebraic Description of a Fog/IoT Computing Environment*, in Proc. of the 24th International Conference ELECTRONICS, Palanga (Lithuania), 2020, pp. 1-7, IEEE Explore. DOI: https://doi.org/10.1109/IEEECONF49502.2020.9141602. Reference: [5].

- Roig, P.J., Alcaraz, S., Gilly, K., Filiposka, S., Aknin, N., *Formal algebraic modelling of a city-wide smart parking system*, in Proc. of the 2nd International Conference on Electrical, Communication, and Computer Engineering (ICECCE), Istanbul (Turkey), 2020, pp. 1-6, IEEE Explore.
DOI: https://doi.org/10.1109/ICECCE49384.2020.9179447. Reference: [174].

- Roig, P.J., Alcaraz, S., Gilly, K., Filiposka, S., Aknin, N., *Formal Algebraic Specification of an IoT/Fog Data Centre for Fat Tree or Leaf and Spine architectures*, in Proc. of the 2nd International Conference on Electrical, Communication, and Computer Engineering (ICECCE), Istanbul (Turkey), IEEE Explore, 2020, pp. 1-6, IEEE Explore.
DOI: https://doi.org/10.1109/10.1109/ICECCE49384.2020.9179445. Ref.: [175].

- Roig, P.J., Alcaraz, S., Gilly, *Arithmetic Study on Energy Saving for some common Data Centre Topologies*, in Proc. of the 8th European Conference on Renewable Energy Systems, Estambul (Turkey), 2020, pp. 744-751. ISBN: 978-605-86911-8-6. Reference: [166].

- Roig, P.J., Alcaraz, S., Gilly, K., Filiposka, S., *Fat Tree Algebraic Formal Modelling Applied to Fog Computing*, in ICT Innovations 2020, Machine Learning

and Applications, Communications in Computer and Information Science 2020, Vol. 1316, pp. 111-126, Springer, Cham (Switzerland).
DOI: https://doi.org/10.1007/978-3-030-62098-1_10. Reference: [190].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *Algebraic modelling of a generic Fog scenario for moving IoT devices*, in Lecture Notes on Networks and Systems 2021, Vol. 187, pp. 1-16, Springer, Singapore (Singapore).
DOI: https://doi.org/10.1007/978-981-33-6173-7_1. Reference: [183].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *Modelling a Plain N-Hypercube Topology for migration in Fog Computing*, in Lecture Notes on Electrical Engineering 2021, Vol. 736, pp. 595-608, Springer, Singapore (Singapore).
DOI: https://doi.org/10.1007/978-981-33-6987-0_47. Reference: [187].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *Modelling a Folded N-Hypercube Topology for migration in Fog Computing*, in Lecture Notes on Electrical Engineering 2021, Vol. 736, pp. 519-535, Springer, Singapore (Singapore).
DOI: https://doi.org/10.1007/978-981-33-6987-0_42. Reference: [186].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *Arithmetic Study about Energy Save in Switches for some Data Centre Topologies*, in Politeknik Dergisi 2021. DOI: https://doi.org/10.2339/politeknik.810896. Reference: [185].

- Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C., *Applying multidimensional geometry to basic fog computing designs*, in International Journal of Electrical and Computer Engineering Research, 2021, Vol. 1(1), pp. 1-8.
DOI: https://doi.org/10.53375/ijecer.2021.17. Reference: [184].

- Roig, P.J., Jornet, J., Albaladejo, A., Hernández, J.C., *Remote surveillance system in isolation for Covid-19*, in Proc. of the 3rd International Conference on Electrical, Communication, and Computer Engineering (ICECCE), Kuala Lumpur (Malaysia), IEEE Explore, 2021, pp. 1-5, IEEE Explore. DOI: https://doi.org/10.1109/ICECCE52056.2021.9514175. Reference: [189].

- Roig, P.J., Alcaraz, S., Gilly, K., Bernad, C., Filiposka, S., *De Bruijn-based and k-ary n-cube-based algebraic models in Fog environments*, in ICT Innovations 2021, Digital Transformation, Communications in Computer and Information Science 2022, Vol. 1521, pp. 126-141, Springer, Cham (Switzerland).
DOI: https://doi.org/10.1007/978-3-031-04206-5_10. Reference: [167].

- Roig, P.J., Alcaraz, S., Gilly, K., Bernad, C., Juiz, C., *Review on de Bruijn shapes in one, two and three dimensions*, in Journal of Physics: Conference Series, 2021, Vol. 2090, pp. 1-10.
DOI: https://doi.org/10.1088/1742-6596/2090/1/012047. Reference: [169].

- Roig, P.J., Alcaraz, S., Gilly, K., Bernad, C., Juiz, C., *Modeling of a Generic Edge Computing Application Design*, in Sensors, 2021, Vol. 21(21), Art. no. 7276, pp. 1-29. DOI: https://doi.org/10.3390/s21217276. Reference: [168].

- Roig, P.J., Alcaraz, S., Gilly, K., Bernad, C., Juiz, C., *Arithmetic Framework to Optimize Packet Forwarding among End Devices in Generic Edge Computing Environments*, in Sensors, 2022, Vol. 22(2), Art. no. 421, pp. 1-23. DOI: https://doi.org/10.3390/s22020421. Reference: [170].

- Roig, P.J., Alcaraz, S., Gilly, K., Bernad, C., Juiz, C., *Modeling an Edge Computing Arithmetic Framework for IoT Environments*, in Sensors, 2022, Vol. 22(3), Art. no. 1084, pp. 1-25. DOI: https://doi.org/10.3390/s22031084. Reference: [173].

- Roig, P.J., Alcaraz, S., Gilly, K., Bernad, C., Juiz, C., *Features for Riemannian N-manifold Classification*, in New Trends in Physical Science Research, 2022, Vol. 4, pp. 17-29. DOI: https://doi.org/10.9734/bpi/ntpsr/v4/2372B. Reference: [172].

- Roig, P.J., Alcaraz, S., Gilly, K., Bernad, C., Juiz, C., *De Bruijn Shapes: Theory and Instances*, in New Trends in Physical Science Research, 2022, Vol. 4, pp. 30-47. DOI: https://doi.org/10.9734/bpi/ntpsr/v4/2371B. Reference: [171].

Chapter 10

Appendix

10.1 Algorithm for a host

```
while forever do
    for $h \leftarrow 0$ to $M \times Z - 1$ do
        foreach $u$ in range(u) do
            if VM(u) $\subset h$ then
                if ready to send then
                    send $_{HOST\{h\},PORT\{0\}}$ (h,b,VM(u))
                end
            else
                if ready to receive then
                    receive $_{HOST\{h\},PORT\{0\}}$ (a,h,VM(u))
                end
            end
        end
    end
end
```

Algorithm 1: Host()

10.2 Algorithm for an edge switch in fat tree

```
while forever do
    for $i \leftarrow 0$ to $K^2/2 - 1$ do
        if receive $_{SWITCH\{i\},PORT\{p\}}$(d) then
            if $int(i/K/2) = int(b/K/2)$ then
                send $_{SWITCH\{i\},PORT\{b \bmod K/2\}}$(d)
            else
                for $q \leftarrow K/2$ to $K - 1$ do
                    send $_{SWITCH\{i\},PORT\{q\}}$(d)
                end
            end
        end
    end
end
```

Algorithm 2: Fat tree - edge()

10.3 Algorithm for an aggregation switch in fat tree

```
while forever do
    for $j \leftarrow 0$ to $K^2/2 - 1$ do
        if receive$_{SWITCH\{j\},PORT\{p\}}$(d) then
            if $int(j/(K/2)^2) = int(b/(K/2)^2)$ then
                send$_{SWITCH\{j\},PORT\{int(b/K/2)\ mod\ K/2\}}$(d)
            else
                for $q \leftarrow K/2$ to $K - 1$ do
                    send$_{SWITCH\{j\},PORT\{q\}}$(d)
                end
            end
        end
    end
end
```

Algorithm 3: Fat tree - aggregation()

10.4 Algorithm for a core switch in fat tree

```
while forever do
    for $l \leftarrow 0$ to $K^2/4 - 1$ do
        if receive$_{SWITCH\{l\},PORT\{p\}}$(d) then
            send$_{SWITCH\{l\},PORT\{int(b/(K/2)^2)\}}$(d)
        end
    end
end
```

Algorithm 4: Fat tree - core()

10.5 Algorithm for a leaf switch in leaf and spine

```
while forever do
    for $i \leftarrow 0$ to $t-1$ do
        if receive $_{SWITCH\{i\},PORT\{p\}}$ (d) then
            if $int(i/M) = int(b/M)$ then
                send $_{SWITCH\{i\},PORT\{b\ mod\ M\}}$ (d)
            else
                for $q \leftarrow M$ to $M+u-1$ do
                    send $_{SWITCH\{i\},PORT\{q\}}$ (d)
                end
            end
        end
    end
end
```

Algorithm 5: Leaf and spine - leaf()

10.6 Algorithm for a spine switch in leaf and spine

```
while forever do
    for $j \leftarrow 0$ to $u-1$ do
        if receive $_{SWITCH\{j\},PORT\{p\}}$ (d) then
            send $_{SWITCH\{j\},PORT\{int(b/M)\}}$ (d)
        end
    end
end
```

Algorithm 6: Leaf and spine - spine()

10.7 Algorithm for an N-hypercube switch

```
while forever do
    for $i \leftarrow 0$ to $2^N - 1$ do
        if receive $_{SWITCH\{i\},PORT\{p\}}$(d) then
            if $i = int(b/M)$ then
                send $_{SWITCH\{i\},PORT\{b \bmod M\}}$(d)
            else
                for $j \leftarrow 0$ to $N - 1$ do
                    if $((int(i/2^j))\ mod\ 2) \neq ((int(int(b/M)/2^j))\ mod\ 2)$ then
                        send $_{SWITCH\{i\},PORT\{M+j\}}$(d)
                    end
                end
            end
        end
    end
end
```

Algorithm 7: N-hypercube()

10.8 Algorithm for a folded N-hypercube switch

```
while forever do
    for $i \leftarrow 0$ to $2^N - 1$ do
        if $receive_{SWITCH\{i\},PORT\{p\}}(d)$ then
            if $i = int(^b/_M)$ then
                $send_{SWITCH\{i\},PORT\{b\ mod\ M\}}(d)$
            else
                $distance = 0$
                for $j \leftarrow 0$ to $N - 1$ do
                    if $((int(^i/_{2^j}))\ mod\ 2) \neq ((int(^{int(b/M)}/_{2^j}))\ mod\ 2)$ then
                        $distance = distance + 1$
                    end
                end
                if $distance \leq (N + 1 - distance)$ then
                    for $j \leftarrow 0$ to $N - 1$ do
                        if $((int(^i/_{2^j}))\ mod\ 2) \neq ((int(^{int(b/M)}/_{2^j}))\ mod\ 2)$ then
                            $send_{SWITCH\{i\},PORT\{M+j\}}(d)$
                        end
                    end
                end
                if $distance \geq (N + 1 - distance)$ then
                    for $j \leftarrow 0$ to $N - 1$ do
                        if $((int(^i/_{2^j}))\ mod\ 2) = ((int(^{int(b/M)}/_{2^j}))\ mod\ 2)$ then
                            $send_{SWITCH\{i\},PORT\{M+j\}}(d)$
                        end
                    end
                    $send_{SWITCH\{i\},PORT\{M+N\}}(d)$
                end
            end
        end
    end
end
```

Algorithm 8: Folded N-hypercube()

10.9 Algorithm for a full mesh switch $\{N$-simplex$\}$

```
while forever do
    for $i \leftarrow 0$ to $N$ do
        if receive $_{SWITCH\{i\},PORT\{p\}}$(d) then
            if $i = int(b/M)$ then
                send $_{SWITCH\{i\},PORT\{b \bmod M\}}$(d)
            else
                if $i = int(a/M)$ then
                    for $q \leftarrow 0$ to $N$ do
                        if $q \neq i$ then
                            send $_{SWITCH\{i\},PORT\{M+q\}}$(d)
                        end
                    end
                else
                    send $_{SWITCH\{i\},PORT\{M + int(b/M)\}}$(d)
                end
            end
        end
    end
end
```

Algorithm 9: Full mesh() $\{N$-simplex$\}$

10.10 Algorithm for a quasi full mesh switch $\{N$-orthoplex$\}$

```
while forever do
    for $i \leftarrow 0$ to $2N - 1$ do
        if $receive_{SWITCH\{i\},PORT\{p\}}(d)$ then
            if $i = int(^b/_M)$ then
                $send_{SWITCH\{i\},PORT\{b \ mod \ M\}}(d)$
            else
                if $i = int(^a/_M)$ then
                    for $q \leftarrow 0$ to $2N - 1$ do
                        if $q \neq i \ AND \ q \neq (i + N) \ mod \ 2N$ then
                            $send_{SWITCH\{i\},PORT\{M+q\}}(d)$
                        end
                    end
                else
                    $send_{SWITCH\{i\},PORT\{M + int(^b/_M)\}}(d)$
                end
            end
        end
    end
end
```

Algorithm 10: Quasi full mesh() $\{N$-orthoplex$\}$

10.11 Algorithm for a spoke switch in a redundant star

```
while forever do
    for $i \leftarrow 0$ to $n - 1$ do
        if $\mathtt{receive}_{SWITCH\{i\},PORT\{p\}}(d)$ then
            if $i = int(b/M)$ then
                $\mathtt{send}_{SWITCH\{i\},PORT\{b\ mod\ M\}}(d)$
            else
                for $q \leftarrow M$ to $M + 1$ do
                    $\mathtt{send}_{SWITCH\{i\},PORT\{q\}}(d)$
                end
            end
        end
    end
end
```

Algorithm 11: Redundant hub and spoke - spoke()

10.12 Algorithm for a hub switch in a redundant star

```
while forever do
    for $j \leftarrow 0$ to $1$ do
        if $\mathtt{receive}_{SWITCH\{j\},PORT\{p\}}(d)$ then
            $\mathtt{send}_{SWITCH\{j\},PORT\{int(b/M)\}}(d)$
        end
    end
end
```

Algorithm 12: Redundant hub and spoke - hub()

10.13 Algorithm for a redundant ring switch

```
while forever do
    for $i \leftarrow 0$ to $n - 1$ do
        if $receive_{SWITCH\{i\},PORT\{p\}}(d)$ then
            if $i = int({}^{b}/_{M})$ then
                $send_{SWITCH\{i\},PORT\{b \bmod M\}}(d)$
            else
                if $[{}^{-n}/_{2} \leq (int({}^{b}/_{M}) - i) < 0]$ then
                    for $q \leftarrow M$ to $M + 1$ do
                        $send_{SWITCH\{i\},PORT\{q\}}(d)$
                    end
                end
                if $(int({}^{b}/_{M}) - i) > {}^{n}/_{2}$ then
                    for $q \leftarrow M$ to $M + 1$ do
                        $send_{SWITCH\{i\},PORT\{q\}}(d)$
                    end
                end
                if $[0 < (int({}^{b}/_{M}) - i) \leq {}^{n}/_{2}]$ then
                    for $q \leftarrow M + 2$ to $M + 3$ do
                        $send_{SWITCH\{i\},PORT\{q\}}(d)$
                    end
                end
                if ${}^{-n}/_{2} > (int({}^{b}/_{M}) - i)$ then
                    for $q \leftarrow M + 2$ to $M + 3$ do
                        $send_{SWITCH\{i\},PORT\{q\}}(d)$
                    end
                end
            end
        end
    end
end
```

Algorithm 13: Redundant ring()

10.14 Algorithm for a toroidal ring switch

```
while forever do
    for $i \leftarrow 0$ to $n - 1$ do
        if receive $_{SWITCH\{i\},PORT\{p\}}$ (d) then
            if $i = int(^b/_M)$ then
                send $_{SWITCH\{i\},PORT\{b\ mod\ M\}}$ (d)
            else
                if $int(^{int(^b/_M)}/_n) \neq int(^i/_n)$ then
                    if $[^{-n}/_2 \leq (int(^b/_M)\ mod\ n - i\ mod\ n) < 0]$ then
                        send $_{SWITCH\{i\},PORT\{M+2\}}$ (d)
                    end
                    if $(int(^b/_M)\ mod\ n - i\ mod\ n) > {}^n/_2$ then
                        send $_{SWITCH\{i\},PORT\{M+2\}}$ (d)
                    end
                    if $[0 < (int(^b/_M)\ mod\ n - i\ mod\ n) \leq {}^n/_2]$ then
                        send $_{SWITCH\{i\},PORT\{M\}}$ (d)
                    end
                    if $^{-n}/_2 > (int(^b/_M)\ mod\ n - i\ mod\ n)$ then
                        send $_{SWITCH\{i\},PORT\{M\}}$ (d)
                    end
                end
                if $int(^b/_M)\ mod\ n \neq i\ mod\ n$ then
                    if $[^{-n}/_2 \leq (int(^{int(^b/_M)}/_n) - int(^i/_n)) < 0]$ then
                        send $_{SWITCH\{i\},PORT\{M+3\}}$ (d)
                    end
                    if $(int(^{int(^b/_M)}/_n) - int(^i/_n)) > {}^n/_2$ then
                        send $_{SWITCH\{i\},PORT\{M+3\}}$ (d)
                    end
                    if $[0 < (int(^{int(^b/_M)}/_n) - int(^i/_N)) \leq {}^n/_2]$ then
                        send $_{SWITCH\{i\},PORT\{M+1\}}$ (d)
                    end
                    if $^{-n}/_2 > (int(^{int(^b/_M)}/_n) - int(^i/_n))$ then
                        send $_{SWITCH\{i\},PORT\{M+1\}}$ (d)
                    end
                end
            end
        end
    end
end
```

Algorithm 14: Toroidal ring()

10.15 Algorithm for a de Bruijn graph switch

```
while forever do
    for $i \leftarrow 0$ to $k^n - 1$ do
        if receive $_{SWITCH\{i\},PORT\{p\}}$(d) then
            if $i = int(b/M)$ then
                send $_{SWITCH\{i\},PORT\{b \bmod M\}}$(d)
            else
                top ← -1
                flag ← 0
                while flag = 0 do
                    top ← top + 1
                    for $j \leftarrow 0$ to top do
                        if $int(i/k^{top-j}) \bmod k \neq int(int(b/M)/k^{n-1-j}) \bmod k$ then
                            flag ← 1
                        end
                    end
                end
                send $_{SWITCH\{i\},PORT\{M + int(int(b/M)/k^{n-1-top}) \bmod k\}}$(d)
            end
        end
    end
end
```

Algorithm 15: De Bruijn graph()

10.16 Algorithm for a de Bruijn reverse graph switch

```
while forever do
    for $i \leftarrow 0$ to $k^n - 1$ do
        if $\mathtt{receive}_{\mathtt{SWITCH\{i\},PORT\{p\}}}\mathtt{(d)}$ then
            if $i = int(^{b}/_{M})$ then
                $\mathtt{send}_{\mathtt{SWITCH\{i\},PORT\{b\ mod\ M\}}}\mathtt{(d)}$
            else
                $top \leftarrow -1$
                $flag \leftarrow 0$
                while $flag = 0$ do
                    $top \leftarrow top + 1$
                    for $j \leftarrow 0$ to $top$ do
                        if $int(^{i}/_{k^{n-1-j}})\ mod\ k \neq int(^{int(^{b}/_{M})}/_{k^{top-j}})\ mod\ k$ then
                            $flag \leftarrow 1$
                        end
                    end
                end
                $\mathtt{send}_{\mathtt{SWITCH\{i\},PORT\{M\ +\ int(int(b/M)/k^{top})\ mod\ k\}}}\mathtt{(d)}$
            end
        end
    end
end
```

Algorithm 16: De Bruijn reverse graph()

10.17 Algorithm for a binary grid switch

```
while $forever$ do
    for $i \leftarrow 0$ to $k^n - 1$ do
        if $\mathtt{receive}_{\mathtt{SWITCH}\{i\},\mathtt{PORT}\{p\}}(d)$ then
            if $i = int(b/M)$ then
                $\mathtt{send}_{\mathtt{SWITCH}\{i\},\mathtt{PORT}\{b\ mod\ M\}}(d)$
            else
                for $j \leftarrow 0$ to $n - 1$ do
                    if $int(i/k^j)\ mod\ k \neq int(int(b/M)/k^j)\ mod\ k$ then
                        $\mathtt{send}_{\mathtt{SWITCH}\{i\},\mathtt{PORT}\{M + j\}}(d)$
                    end
                end
            end
        end
    end
end
```

Algorithm 17: Binary grid()

10.18 Algorithm for a Hamming graph switch

```
while forever do
    for $i \leftarrow 0$ to $k^n - 1$ do
        if receive $_{SWITCH\{i\},PORT\{p\}}$(d) then
            if $i = int(b/M)$ then
                send $_{SWITCH\{i\},PORT\{b \bmod M\}}$(d)
            else
                $diff = 0$
                for $j \leftarrow 0$ to $n - 1$ do
                    if $int(i/k^j) \bmod k \neq int(int(b/M)/k^j) \bmod k$ then
                        $diff = diff + 1$
                    end
                end
                if $diff = n$ then
                    send $_{SWITCH\{i\},PORT\{M+n\}}$(d)
                else if $diff < n/2$ then
                    send $_{SWITCH\{i\},PORT\{M+n\}}$(d)
                else if $diff > n/2$ then
                    for $j \leftarrow 0$ to $n - 1$ do
                        if $int(i/k^j) \bmod k = int(int(b/M)/k^j) \bmod k$ then
                            send $_{SWITCH\{i\},PORT\{M+j\}}$(d)
                        end
                    end
                else /* If $diff = n/2$ */
                    for $j \leftarrow 0$ to $n$ do
                        send $_{SWITCH\{i\},PORT\{M+j\}}$(d)
                    end
                end
            end
        end
    end
end
```

Algorithm 18: Hamming graph() $\{H_k(n, n-1)\}$

10.19 Algorithm for a Petersen graph switch

```
while forever do
    for $i \leftarrow 0$ to 9 do
        if $receive_{SWITCH\{i\},PORT\{p\}}(d)$ then
            if $i = int(^b/_M)$ then
                $send_{SWITCH\{i\},PORT\{b\ mod\ M\}}(d)$
            else
                if $int(^i/_5) < int(^{int(^b/_M)}/_5)$ then
                    if $int(^b/_M)\ mod\ 5 \in \{i\ mod\ 5, (i+2)\ mod\ 5, (i+3)\ mod\ 5\}$
                     then
                        $send_{SWITCH\{i\},PORT\{M+2\}}(d)$
                    else if $int(^b/_M)\ mod\ 5 = (i+1)\ mod\ 5$ then
                        $send_{SWITCH\{i\},PORT\{M+1\}}(d)$
                    else /* If $int(^b/_M)\ mod\ 5 = (i+4)\ mod\ 5$                */
                        $send_{SWITCH\{i\},PORT\{M\}}(d)$
                    end
                else if $int(^i/_5) > int(^{int(^b/_M)}/_5)$ then
                    if $int(^b/_M)\ mod\ 5 \in \{i\ mod\ 5, (i+1)\ mod\ 5, (i+4)\ mod\ 5\}$
                     then
                        $send_{SWITCH\{i\},PORT\{M+2\}}(d)$
                    else if $int(^b/_M)\ mod\ 5 = (i+3)\ mod\ 5$ then
                        $send_{SWITCH\{i\},PORT\{M+1\}}(d)$
                    else /* If $int(^b/_M)\ mod\ 5 = (i+4)\ mod\ 5$                */
                        $send_{SWITCH\{i\},PORT\{M\}}(d)$
                    end
                else if $int(^i/_5) = int(^{int(^b/_M)}/_5) = 0$ then
                    if $(int(^b/_M) - i)\ mod\ 5 < {}^N/_2$ then
                        $send_{SWITCH\{i\},PORT\{M+1\}}(d)$
                    else
                        $send_{SWITCH\{i\},PORT\{M\}}(d)$
                    end
                else /* If$int(^i/_5) = int(^{int(^b/_M)}/_5) = 1$                  */
                    if $i\ mod\ 2 \neq int(^b/_M)\ mod\ 2$ then
                        $send_{SWITCH\{i\},PORT\{M+1\}}(d)$
                    else
                        $send_{SWITCH\{i\},PORT\{M\}}(d)$
                    end
                end
            end
        end
    end
end
```

Algorithm 19: Petersen graph()

10.20 Algorithm for a Heawood graph switch

```
while forever do
    for $i \leftarrow 0$ to 13 do
        if receive$_{SWITCH\{i\},PORT\{p\}}$(d) then
            if $i = int({}^{b}/_{M})$ then
                send$_{SWITCH\{i\},PORT\{b \bmod M\}}$(d)
            else
                $dist = int({}^{b}/_{M}) - i$
                if $dist > 7$ then
                    $dist = -(14 - dist)$
                else if $dist < -7$ then
                    $dist = -(-14 - dist)$
                end
                if $dist \in \{+1, +2, +3\}$ then
                    send$_{SWITCH\{i\},PORT\{M+1\}}$(d)
                else if $dist \in \{-1, -2, -3\}$ then
                    send$_{SWITCH\{i\},PORT\{M\}}$(d)
                else if $i \bmod 2 = 0$ then
                    if $dist \in \{+4, +5, +6, +7\}$ then
                        send$_{SWITCH\{i\},PORT\{M+2\}}$(d)
                    else if $dist \in \{-4, -5\}$ then
                        send$_{SWITCH\{i\},PORT\{M+1\}}$(d)
                    else /* If $dist \in \{-6\}$                    */
                        send$_{SWITCH\{i\},PORT\{M\}}$(d)
                    end
                else /* If $i \bmod 2 = 1$                          */
                    if $dist \in \{-4, -5, -6, +7\}$ then
                        send$_{SWITCH\{i\},PORT\{M+2\}}$(d)
                    else if $dist \in \{+4, +5\}$ then
                        send$_{SWITCH\{i\},PORT\{M\}}$(d)
                    else /* If $dist \in \{+6\}$                    */
                        send$_{SWITCH\{i\},PORT\{M+1\}}$(d)
                    end
                end
            end
        end
    end
end
```

Algorithm 20: Heawood graph()

Acronyms

5G 5th Generation of Cellular Networks

6G 6th Generation of Cellular Networks

ABP Alternating Bit Protocol

ACP Algebra of Communicating Processes

ADTs Abstract Data Types

AR Augmented Reality

BRP Bounded Retransmission Protocol

CAPEX capital expenditure

CCS Calculus of Communicating Systems

CoAP Constrained Application Protocol

COTS commercial-of-the-shelf

CPU Central Processing Unit

CSP Communicating Sequential Processes

DC data centres

EIGRP Enhanced Interior Gateway Routing Protocol

ETSI European Telecommunications Standards Institute

FDT Formal Description Techniques

FSM Finite State Machines

FTP File Transfer Protocol

HTTP Hypertext Transfer Protocol

IaaS Infrastructure as a Service

IEEE Institute of Electrical and Electronics Engineers

IEEE-1394 FireWire

IETF Internet Engineering Task Force

IGMP Internet Group Management Protocol

IoT Internet of Things

IP Internet Protocol

IPv4 Internet Protocol version 4

ITaaS IT as a Service

ITU-T International Telecommunication Union - Telecommunication Standardization Sector

LAN Local Area Network

MAC Media Access Control

MEC Multi-Access Edge Computing

MQTT Message Queue Telemetry Transport

MR Mixed Reality

OASIS Organization for the Advancement of Structured Information Standards

OPEX operational expenditure

OS Operating System

OSI Open Systems Interconnection

OSPF Open Shortest Path First

PaaS Platform as a Service

PIM Protocol Independent Multicast

POP3 Post Office Protocol revision 3

Promela Protocol/Process Meta Language

Pub/Sub Publisher/Subscriber

QoS Quality of Service

RAM Random Access Memory

RAN Radio Access Network

REST Representational State Transfer

RESTful based on REST architecture web services

RFC Request for Comments

RPC Remote Procedure Call

SaaS Software as a Service

SDL Specification and Description Language

SEES Smart Emergency Evacuation System

SMTP Simple Mail Transfer Protocol

Spin Simple Promela Interpreter

SSL Secure Sockets Layer

STP Spanning Tree Protocol

TCP/IP Transmission Control Protocol / Internet Protocol

TLS Transport Layer Security

UML Unified Modeling Language

VM Virtual Machine

VoIP Voice over IP

VR Virtual Reality

WAN Wide Area Network

WiFi-6 6th Generation of Wireless Fidelity

WiFi-7 7th Generation of Wireless Fidelity

WSN Wireless Sensor Networks

XOR Exclusive OR

Bibliography

[1] Jabril Abdelaziz, Mehdi Adda, and Hamid Mcheick. An Architectural Model for Fog Computing. *Journal of Ubiquitous Systems and Pervasive Networks*, 10:21–25, 03 2018.

[2] Bernhard K. Aichernig and Richard Schumi. How Fast is MQTT? - Statistical Model Checking and Testing of IoT Protocols. In *Quantitative Evaluation of Systems (QEST)*, pages 36–52, 2018.

[3] F. Al-Doghman, Z. Chaczko, A. R. Ajayan, and R. Klempous. A review on Fog Computing technology. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 001525–001530, 2016.

[4] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, volume 38, pages 63–74, 10 2008.

[5] S. Alcaraz, P. J. Roig, K. Gilly, S. Filiposka, and N. Aknin. Formal Algebraic Description of a Fog/IoT Computing Environment. In *24th International Conference Electronics 2020*, pages 1–7, 2020.

[6] Gerald Alexanderson. About the Cover: Euler and the Königsberg Bridges: A Historical View. *Bulletin (New Series) of the American Mathematical Society*, 43:567–573, 10 2006.

[7] M. Alizadeh and T. Edsall. On the Data Path Performance of Leaf-Spine Datacenter Fabrics. In *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, pages 71–74, 2013.

[8] Muhammad Anwar, Shangguang Wang, Muhammad Zia, Ahmer Jadoon, Umair Akram, and Syed Salman Raza Naqvi. Fog Computing: An Overview of Big IoT Data Analytics. *Wireless Communications and Mobile Computing*, 2018:1–22, 05 2018.

[9] Humaira Anwer, Farooque Azam, Muhammad Anwar, and Muhammad Rashid. *A Model-Driven Approach for Load-Balanced MQTT Protocol in Internet of*

Things (IoT), volume 993, chapter AISC (Complex, Intelligent, and Software Intensive Systems), pages 368–378. Springer, Cham, 06 2019.

[10] E. Archana, A. Rajeev, A. Kuruvila, and R. Narayankutti. A Formal Modeling Approach for QOS in MQTT Protocol. *Ad Hoc Networks*, 36:49–57, 2016.

[11] T. H. Ashrafi, M. A. Hossain, S. E. Arefin, K. D. J. Das, and A. Chakrabarty. Service Based FOG Computing Model for IoT. In *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*, pages 163–172, 2017.

[12] Ronald G. Askin and Girish Jampani Hanumantha. Queueing network models for analysis of nonstationary manufacturing systems. *International Journal of Production Research*, 56(1-2):22–42, 2018.

[13] Y. Atif, Jianguo Ding, and Manfred Jeusfeld. Internet of Things Approach to Cloud-based Smart Car Parking. *Procedia Computer Science*, 98:193–198, 12 2016.

[14] Hany F. Atlam, Robert J. Walters, and Gary B. Wills. Fog Computing and the Internet of Things: A Review. *Big Data and Cognitive Computing*, 2(10):1–18, 2018.

[15] Benjamin Aziz. A Formal Model and Analysis of an IoT Protocol. *Ad Hoc Networks*, 36(P1):49–57, January 2016.

[16] J. C. M. Baeten, T. Basten, and M. A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2009.

[17] J.C.M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131–146, May 2005.

[18] J.C.M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131–146, 2005.

[19] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards Predictable Datacenter Networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):242–253, August 2011.

[20] Mordechai Ben-Ari. *Propositional Logic: Deductive Systems*, pages 49–73. 01 2012.

[21] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.

[22] Jan Bergstra and Jan Willem Klop. *Algebraic Methods: Theory, Tools and Applications*, volume 15, chapter ACP: a universal axiom system for process specification, pages 445–463. 04 1987.

[23] Luca Bernardinello and Fiorella De Cindio. A Survey of Basic Net Models and Modular Net Classes. *Advances in Petri Nets (Lecture Notes in Computer Science)*, (609):304–351, 1992.

[24] Er. Showkat Bhat and Ishfaq Bashir. Edge Computing and Its Convergence With Blockchain in 5G and Beyond: Security, Challenges, and Opportunities. *IEEE Access*, 8:205340–205373, 11 2020.

[25] Arvind Kumar Bhatia and Gursharan Singh. A Review on Different Types of Live VM Migration Methods with Proposed Pre-Copy Approach. *International Journal of Engineering and Technology(UAE)*, 7:6–12, 10 2018.

[26] L. F. Bittencourt, M. M. Lopes, I. Petri, and O. F. Rana. Towards Virtual Machine Migration in Fog Computing. In *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 1–8, 2015.

[27] L.F. Bittencourt, E.R.M. Madeira, and N. Fonseca. *Resource Management and Scheduling*, chapter Cloud Services, Networking, and Management, pages 243–267. April 2015.

[28] Luiz Fernando Bittencourt, Roger Immich, Rizos Sakellariou, Nelson Fonseca, Edmundo Madeira, Marilia Curado, Leandro Villas, Luiz Silva, Craig Lee, and Omer Rana. The Internet of Things, Fog and Cloud Continuum: Integration and Challenges. *Internet of Things*, 3-4, 09 2018.

[29] Chiara Bodei, Pierpaolo Degano, Gian Ferrari, and Letterio Galletta. A Step Towards Checking Security in IoT. *Electronic Proceedings in Theoretical Computer Science*, 223:128–142, 08 2016.

[30] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. The Macmillan Press Ltd., 1976.

[31] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. Association for Computing Machinery.

[32] Alessio Botta, Walter Donato, Valerio Persico, and Antonio Pescapè. Integration of Cloud Computing and Internet of Things: A survey. *Future Generation Computer Systems*, 56, 10 2015.

[33] Stephen D. Brooks. On the relationship of CCS y CSP. *Lecture Notes in Computer Science (Automata, Languages and Programming)*, 154:83–96, 1983.

[34] James Caldwell. Structural Induction Principles for Functional Programmers. *Electronic Proceedings in Theoretical Computer Science*, 136, 12 2013.

[35] José M. Cámara, Miquel Moretó, Enrique Vallejo, Ramón Beivide, Carmen Martínez José Miguel-Alonso, and Javier Navaridas. Twisted Torus Topologies for Enhanced Interconnection Networks. *IEEE Transaction on Parallel and Distrbuted Systems*, 21(12), December 2010.

[36] Karel Casteels and Ted Tinker. De Bruijn Sequences of Higher Dimension. Master's thesis, University of California, 2018.

[37] Subrata Chakraborty, C. Fidge, Lin Ma, and Yong Sun. *M-ary Trees for Combinatorial Asset Management Decision Problems*, pages 117–127. 01 2014.

[38] Lucas Chaufournier, Prateek Sharma, Franck Le, Erich Nahum, Prashant Shenoy, and Don Towsley. Fast Transparent Virtual Machine Migration in Distributed Edge Clouds. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC1́7, New York, NY, USA, 2017. Association for Computing Machinery.

[39] Hossein Chegini, Ranesh Kumar Naha, Aniket Mahanti, and Parimala Thulasiraman. Process Automation in an IoT-Fog-Cloud Ecosystem: A Survey and Taxonomy. *IoT*, 2(1):92–118, 2021.

[40] Rayan Chikhi, Antoine Limasset, Shaun Jackman, Jared Simpson, and Paul Medvedev. On the Representation of De Bruijn Graphs. *Journal of computational biology : a journal of computational molecular cell biology*, 22, 01 2014.

[41] Y. Choi. Automated Validation of IoT Device Control Programs Through Domain-Specific Model Generation. *SEFM 2018: Software Engineering and Formal Methods*, 2018.

[42] C. C. Chou, Y. Chen, D. Milojicic, N. Reddy, and P. Gratz. Optimizing Post-Copy Live Migration with System-Level Checkpoint Using Fabric-Attached Memory. In *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, pages 16–24, 2019.

[43] S. Chouali, A. Boukerche, and A. Mostefaoui. Towards a Formal Analysis of MQTT Protocol in the Context of Communicating Vehicles. *MobiWac '17: Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access*, pages 129–136, 2017.

[44] Federico Ciccozzi, Ivano Malavolta, and Bran Selic. Execution of UML models: a systematic review of research and practice. *Software & Systems Modeling*, 18(3):2313–2360, June 2019.

[45] C. Clos. A study of non-blocking switching networks. *The Bell System Technical Journal*, 32(2):406–424, 1953.

[46] Harold Scott MacDonald Coxeter. *Regular polytopes*. Pitman Publishing, 1948.

[47] Oualid Demigha and Chamseddine Khalfi. Formal Analysis of Energy Consumption in IoT Systems. *Internet of Things, Big Data and Security*, 1:103–114, 2019.

[48] U. Deshpande, D. Chan, T. Guh, J. Edouard, K. Gopalan, and N. Bila. Agile Live Migration of Virtual Machines. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1061–1070, 2016.

[49] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[50] Huu Dinh, Alexander Dworkin, Christopher O'Neill, Scott Savage, Jimmy Leak, Mohammad Aazam, and Marc St-Hilaire. OmniBox: Efficient cloud storage by evaluating Dropbox and Box. In *Proceedings of the 24th International Conference on Telecommunications*, pages 1–6, 05 2017.

[51] Maithily Diwan and Meenakshi D'Souza. A Framework for Modeling and Verifying IoT Communication Protocols. In *International Symposium on Dependable Software Engineering 2017: Theories, Tools, and Applications*, pages 266–280, 10 2017.

[52] Jasenka Dizdarevic, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. *ACM Comput. Surv.*, 51(6), January 2019.

[53] Denis do Rosário, Matias Artur Klafke Schimuneck, João Camargo, Jéferson Campos Nobre, Cristiano Both, Juergen Rochol, and Mario Gerla. Service Migration from Cloud to Multi-tier Fog Nodes for Multimedia Dissemination with QoE Support. *Sensors (Basel, Switzerland)*, 18, January 2018.

[54] M.S. Doidge, P.A. Love, and J. Thornton. Monitoring distributed computing beyond the traditional time-series histogram. *EPJ Web Conf.*, 245:3036–3040, November 2020.

[55] Pranay Dutta and Prashant Dutta. Comparative Study of Cloud Services Offered by Amazon, Microsoft and Google. *International Journal of Trend in Scientific Research and Development*, Volume-3:981–985, 04 2019.

[56] A. El-Amawy and S. Latifi. Properties and performance of folded hypercubes. *Transactions on Parallel and Distributed Algorithms*, 2(1):31–42, 1991.

[57] Salim El Rouayheb, Costas Georghiades, Emina Soljanin, and Alex Sprintson. Bounds on Codes Based on Graph Theory. pages 1876 – 1879, 07 2007.

[58] ETSI GS MEC 003 V2.2.1. *Multi-access Edge Computing (MEC); Framework and Reference Architecture.* ETSI, December 2020.

[59] G. Exoo and R. Jajcay. Dynamic Cage Survey. *Electronic Journal of Combinatorics*, 1000, 2011.

[60] S. Rinaldi F. Disanto, A. Frosini. Square Involutions. *Journalof Integer Sequences*, 14(11.3.5):1–15, 2011.

[61] E. Fersman and B. Jonsson. Abstraction of Communication Channels in Promela: A Case Study. *Lecture Notes in Computer Science - SPIN Model Checking and Software Verification.*, 1885:187–204, 2000.

[62] S. Filiposka, A. Mishev, and C. Juiz. Community-based VM placement framework. *Journal of Supercomputing*, 71:4504–4528, October 2015.

[63] Sonja Filiposka, Anastas Mishev, and Katja Gilly. Community-based allocation and migration strategies for fog computing. pages 1–6, 04 2018.

[64] T. Fischer, C. Lesjak, D. Pirker, and C. Steger. RPC Based Framework for Partitioning IoT Security Software for Trusted Execution Environments. In *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0430–0435, 2019.

[65] Wan Fokkink. *Introduction to Process Algebra.* Springer, Berlin, Heidelberg, 2000.

[66] Wan Fokkink. Rooted Branching Bisimulation as a Congruence. *Journal of Computer and System Sciences*, 60:13–37, 2000.

[67] Wan Fokkink. Process Algebra: An Algebraic Theory of Concurrency. In *Algebraic Informatics. CAI 2009. Lecture Notes in Computer Science*, volume 5725, pages 47–77, 05 2009.

[68] Wan Fokkink. *Modelling Distributed Systems.* Springer, Berlin, Heidelberg, 2016.

[69] Behrouz A. Forouzan, Catherine Ann Coombs, and Sophia Chung Fegan. *Data Communications and Networking.* McGraw-Hill Higher Education, 2nd edition, 2000.

[70] Mattias Forsman, Andreas Glad, Lars Lundberg, and Dragos Ilie. Algorithms for Automated Live Migration of Virtual Machines. *J. Syst. Softw.*, 101(C):110–126, March 2015.

[71] Maciej Gazda and Wan Fokkink. Congruence from the Operator's Point of View: Compositionality Requirements on Process Semantics. volume 32, pages 15–25, 08 2010.

[72] V. Geetha and J. Bashkar. Programming smart environments using pi-calculus. *Procedia Computer Science*, pages 884–891, 2015.

[73] Marla Teresinha Barbosa Geller and Anderson Alvarenga de Moura Meneses. Modelling IoT Systems with UML: A Case Study for Monitoring and Predicting Power Consumption. *American Journal of Engineering and Applied Sciences*, 14(1):81–93, 2021.

[74] Malayam Parambath Gilesh, Subham Jain, S. D. Madhu Kumar, Lillykutty Jacob, and Umesh Bellur. Opportunistic live migration of virtual machines. *Wiley Online Library*, 2019.

[75] Katja Gilly, Sonja Filiposka, and Anastas Mishev. Supporting Location Transparent Services in a Mobile Edge Computing Environment. *Advances in Electrical and Computer Engineering*, 18:11–22, 11 2018.

[76] Anandi Giridharan. A Formal Model for Service Discovery Protocol (SDP) using SDL. In *Proceedings of the 9th International Conference on Machine Learning and Computing*, ICMLC 2017, pages 437–441, New York, NY, USA, 2017. Association for Computing Machinery.

[77] Alessandro Giua and Manuel Silva. Modeling, analysis and control of Discrete Event Systems: a Petri net perspective. *IFAC-PapersOnLine*, 50(1):1772–1783, 2017. 20th IFAC World Congress.

[78] Denis Gratias and Marianne Quiquandon. Discovery of quasicrystals: The early days. *Comptes Rendus Physique*, 20(7):803 – 816, 2019.

[79] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A Scalable and Flexible Data Center Network. *SIGCOMM Comput. Commun. Rev.*, 39(4):51–62, August 2009.

[80] Jan Friso Groote. Specification and verification of real time systems in ACP. *Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol Specification, Testing and Verification*, pages 261–274, 1990.

[81] Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. The MIT Press, 2014.

[82] Jan Friso Groote and Alban Ponse. The Syntax and Semantics of mCRL. *Algebra of Communicating Processes*, pages 26–62, 1995.

[83] Jan Friso Groote and Jaco van de Pol. A Bounded Retransmission Protocol for Large Data Packets. *AMAST '96: Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology*, pages 536–550, 1996.

[84] Giancarlo Guizzardi. *Ontological Foundations for Structural Conceptual Models*. PhD thesis, Universiteit Twente, October 2005.

[85] P. Habibi, M. Farhoudi, S. Kazemian, S. Khorsandi, and A. Leon-Garcia. Fog Computing: A Comprehensive Architectural Survey. *IEEE Access*, 8:69105–69133, 2020.

[86] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[87] Frank Harary, John P. Hayes, and Horng-Jyh Wu. A survey of the theory of hypercube graphs. *Computers & Mathematics with Applications*, 15(4):277–289, 1988.

[88] Isaiah H. Harney. *Colorings of Hamming-Distance Graphs*. PhD thesis, University of Kentucky, 2017.

[89] Michael Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *Operating Systems Review*, 43:14–26, 07 2009.

[90] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[91] K. Hofer-Schmitz and B. Stojanovic. Towards Formal Methods of IoT Application Layer Protocols. In *2019 12th CMI Conference on Cybersecurity and Privacy (CMI)*, pages 1–6, 2019.

[92] Gerard Holzmann, Elie Najm, and Ahmed Serhrouchni. SPIN model checking: An introduction. *STTT*, 2:321–327, 03 2000.

[93] Victoria Horan and Brett Stevens. Locating patterns in the de Bruijn torus. *Discrete Mathematics*, 339(4):1274 – 1282, 2016.

[94] M. Horváth and A. Iványi. Growing perfect cubes. *Discrete mathematics*, 308(19):4378–4388, 2008.

[95] M. Houimli, L. Kahloul, and S. Benaoun. Formal specification, verification and evaluation of the MQTT protocol in the Internet of Things. In *2017 International Conference on Mathematics and Information Technology (ICMIT)*, pages 214–221, 2017.

[96] Glenn Hurlbert and Garth Isaak. On the de Bruijn Torus problem. *Journal of Combinatorial Theory, Series A*, 64(1):50 – 62, 1993.

[97] Khaled Ibrahim, Steven Hofmeyr, Costin Iancu, and Eric Roman. Optimized pre-copy live migration for memory intensive applications. page 40, 11 2011.

[98] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. Goren, and C. Mahmoudi. Fog Computing Conceptual Model. *NIST Special Publication 500-325*, pages 1–14, 2018.

[99] Bergstra J.A. and Klop J.W. Verification of an alternating bit protocol by means of process algebra protocol. *Mathematical Methods of Specification and Synthesis of Software Systems 1985*, pages 9–23, 1986.

[100] J.A.Bergstra and J.W.Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[101] N. Jain, Bhatele. A., L. Howell, D. Bohme, I. Karlin, E. Leon, M. Mubarak, N. Wolfe, T. Gamblin, and M. Leiniger. Predicting the performance impact of different fat-tree configurations. *SC '17: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 50:1–13, November 2017.

[102] Robert Jamison and Gretchen Matthews. Distance k colorings of Hamming graphs. *Congressus Numerantium*, 183, 01 2006.

[103] Changyeon Jo, Youngsu Cho, and Bernhard Egger. A machine learning approach to live migration modeling. pages 351–364, 09 2017.

[104] Carlos Juiz. *Performance Modelling of Asinchronous Data Transfer Components in Soft Real-Time Systems*. PhD thesis, University of the Balearic Islands, 2001.

[105] J. B. Kapinya. Evolutionary Computing Solutions for the de Bruijn Torus Problem. Master's thesis, Vrije Universiteit Amsterdam, 2004.

[106] Joost-Pieter Katoen, Jaco van de Pol, Marielle Stoelinga, and Mark Timmer. A linear process-algebraic format with data for probabilistic automata. *Theoretical Computer Science*, 413(1):36–57, 2012.

[107] P. Kaur and A. Rani. Virtual machine migration in cloud computing. *InternationalJournal of Grid Distribution Computing 2015*, 8(5):337–342, 2015.

[108] David G. Kendall. Some problems in the theory of queues. *Journal of the Royal Statistical Society: Series B (Methodological)*, 13(2):151–173, 1951.

[109] S. Khan, S. Parkinson, and Y Qin. Fog computing security: a review of current applications and security solutions. *Journal of Cloud Computing*, 6(19):1–22, 2017.

[110] Evgeny Khorov, Ilya Levitsky, and Ian F. Akyildiz. Current Status and Directions of IEEE 802.11be, the Future Wi-Fi 7. *IEEE Access*, 8:88664–88688, May 2020.

[111] Alfred Kobsa. Generic User Modeling Systems. *User Modeling and User-Adapted Interaction*, 11:49–63, 2001.

[112] Lockefeer L., Williams D.M., and Fokkink W.J. Formal Specification and Verification of TCP Extended with the Window Scale Option. *Formal Methods for Industrial Critical Systems*, pages 63–77, 2014.

[113] R. Lai and A. Jirachiefpattana. *Communication Protocol Specification and Verification*, chapter Formal Description Techniques, pages 27–37. Springer, Boston, MA, 1998.

[114] R. Lanotte and M. Merro. A Semantic Theory of the Internet of Things. *Coordination Models and Language 2016.*, 2016.

[115] Tuan Le. A survey of live Virtual Machine migration techniques. *Computer Science Review*, 38:100304, 2020.

[116] Susan Leavitt. The Simple Complexity of Pascal's Triangle. 2011.

[117] H. G. Lee and N. Chang. Powering the IoT: Storage-less and converter-less energy harvesting. In *The 20th Asia and South Pacific Design Automation Conference*, pages 124–129, 2015.

[118] S. Lee, Y. Choe, and M. Lee. dT-Calculus: A Formal Method to Specify Distributed Mobile Real-Time IoT Systems. *Open Access*, August 2018.

[119] Jun Li, Xiaoman Shen, Lei Chen, Dumg Pham Van, Jiannan Ou, Lena Wosinska, and Jiajia Chen. Service Migration in Fog Computing Enabled Cellular Networks to Support Real-Time Vehicular Communications. *IEEE Access*, 7:13704–13714, 2019.

[120] Xiangwen Li, Vicky Mak, and Sanming Zhou. Optimal Radio Labellings of Complete M-Ary Trees. *Discrete Appl. Math.*, 158(5):507–515, March 2010.

[121] Zhihua Li, Xinrong Yu, Lei Yu, Shujie Guo, and Victor Chang. Energy-efficient and quality-aware VM consolidation method. *Future Generation Computer Systems*, 102:789 – 809, 2020.

[122] Kun Ma, Antoine Bagula, Clement Nyirenda, and Olasupo Ajayi. An IoT-Based Fog Computing Model. *Sensors MDPI*, 19(2783):1–17, 2019.

[123] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. Latency-Aware Application Module Management for Fog Computing Environments. *ACM Trans. Internet Technol.*, 19(1), November 2018.

[124] Naga Malleswari and G. Vadivu. Adaptive deduplication of virtual machine images using AKKA stream to accelerate live migration process in cloud environment. *Journal of Cloud Computing*, 8, 2019.

[125] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys Tutorials*, 19(4):2322–2358, 2017.

[126] S. Margariti, V. Dimakopoulos, and G. Tsoumanis. Modeling and Simulation Tools for Fog Computing - A Comprehensive Survey from a Cost Perspective. *Future Internet*, 12(5):89, 2020.

[127] S. Marir, F. Belala, and N. Hameurlain. A Formal Model for Interaction Specification and Analysis in IoT Applications. *MEDI 2018: Model and Data Engineering*, 2018.

[128] Mohammad Masdari and Hemn Khezri. Efficient VM migrations using forecasting techniques in cloud computing: a comprehensive review. *Cluster Computing*, 23:2629–2658, 2020.

[129] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, Berlin, Heidelberg, 1982.

[130] Daniel Minoli, Kazem Sohraby, and Benedict Occhiogrosso. IoT Considerations, Requirements, and Architectures for Smart Buildings-Energy Optimization and Next-Generation Building Management Systems. *IEEE Internet of Things Journal*, 4:269–283, 2017.

[131] Biswajeeban Mishra. *Performance Evaluation of MQTT Broker Servers*, pages 599–609. 07 2018.

[132] X. Molero, C. Juiz, and M. Rodeño. *Evaluación y Modelado del Rendimiento de los Sistemas Informáticos*. Pearson Prentince Hall, 2004.

[133] A. Munir, P. Kansakar, and S. U. Khan. IFCIoT: Integrated Fog Cloud IoT Architectural Paradigm for Future Internet of Things. *Distributed, Parallel, and Cluster Computing*, pages 1–9, 2017.

[134] M. I. Naas, J. Boukhobza, P. Raipin Parvedy, and L. Lemarchand. An Extension to iFogSim to Enable the Design of Data Placement Strategies. In *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–8, 2018.

[135] P. C. Nayak, D. Garg, A. k. Shakva, and P. Saini. A Research Paper of Existing Live VM Migration and a Hybrid VM Migration Approach in Cloud Computing. In *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 720–725, May 2018.

[136] Amnon Neeman. *Algebraic and Analytic Geometry*. Cambridge University Press, 2007.

[137] L. Ni, J. Zhang, C. Jiang, C. Yan, and K. Yu. Resource Allocation Strategy in Fog Computing Based on Priced Timed Petri Nets. *IEEE Internet of Things Journal*, 4(5):1216–1228, 2017.

[138] L. Ni, J. Zhang, and J. Yu. Priced Timed Petri Nets Based Resource Allocation Strategy for Fog Computing. In *2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI)*, pages 39–44, 2016.

[139] Rocco De Nicola. A gentle introduction to Process Algebras. 2013.

[140] Kennedy Okafor, Ifeyinwa Achumba, and Gordon Ononiwu. Leveraging Fog Computing For Scalable IoT Datacenter Using Spine-Leaf Network Topology. *Journal of Electrical and Computer Engineering*, 2017:1–11, 04 2017.

[141] Y.E. Oktian, E.N. Witanto, and S.G. Lee. A Conceptual Architecture in Decentralizing Computing, Storage, and Networking Aspect of IoT Infrastructure. *Industrial IoT as IT and OT Convergence: Challenges and Opportunities*, 2(2):205–221, 2021.

[142] Open Fog Consortium. *OpenFog Reference Architecture for Fog Computing*. Open Fog Consortium, 2017.

[143] Gerard O'Regan. *A Brief History of Computing*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[144] O. Osanaiye, S. Chen, Z. Yan, R. Lu, K. R. Choo, and M. Dlodlo. From Cloud to Fog Computing: A Review and a Conceptual Live VM Migration Framework. *IEEE Access*, 5:8284–8300, 2017.

[145] Samir Ouchani. *Ensuring the Functional Correctness of IoT through Formal Modeling and Verification*, pages 401–417. Model and Data Engineering (MEDI 2018), 01 2018.

[146] Edward J. Oughton, William Lehr, Konstantinos Katsaros, Ioannis Selinis, Dean Bubley, and Julius Kusuma. Revisiting Wireless Internet Connectivity: 5G vs Wi-Fi 6. *Telecommunications Policy*, 45(5):102127, 2021.

[147] U. Ozeer, G. Salaun, L. Letondeur, FG Ottogalli, and JM. Vincent. Verification of a Failure Management Protocol for Stateful IoT Applications. *FMICS 2020: Formal Methods for Industrial Critical Systems*, pages 272–287, 2020.

[148] D. Padua. *Encyclopedia of Parallel Computing*. Springer US, 2011.

[149] Joachim Parrow. Expressiveness of Process Algebras. *Electronic Notes in Theoretical Computer Science*, 209:173–186, 2008.

[150] C. A. Petri. Communication with automata. In *Technical Report RADC-TR-65-377 (New York University)*, 1966.

[151] Glenn Tesler Phillip E. C. Compeau, Pavel A. Pevzner. How to apply de Bruijn graphs to genome assembly. *Nature Biotechnology*, 29(11):987–991, 2011.

[152] L. Pierucci and P. J. Roig. UWB localization on indoor MIMO channels. In *2005 International Conference on Wireless Networks, Communications and Mobile Computing*, volume 2, pages 890–894 vol.2, 2005.

[153] C. S. R. Prabhu. Overview - Fog Computing and Internet-of-Things (IOT). *EAI Endorsed Transactions on Cloud Systems*, 3(10):1–23, 12 2017.

[154] A. Prinz, M. Scheidgen, and M.S. Tveit. A Model-Based Standard for SDL. *Lecture Notes in Computer Science - SDL 2007: Design for Dependable Systems*, 4745:1–18, 2007.

[155] L. Pudwell and R. Rockey. De Bruijn Arrays for L-Fillings. *Mathematics Magazine*, 87(1):57–60, 2014.

[156] C Puliafito, C. Vallati, E. Mingozzi, F. Merlino, G. andLongo, and A. Puliafito. Container Migration in the Fog: A Performance Evaluation. *Sensors 2019*, 19(7):1488, 2019.

[157] Han Qi, Muhammad Shiraz, Jie Yao Liu, Abdullah Gani, Zulkanain Rahman, and T. Altameem. Data center network architecture in cloud computing: review, taxonomy, and open research issues. *Journal of Zhejiang University SCIENCE C*, 15:776–793, 09 2014.

[158] Juan Quemada. Formal Description Techniques and Software Engineering: Some Reflections after 2 Decades of Research. In *Formal Techniques for Networked and Distributed Systems - FORTE 2004*, pages 33–42, 09 2004.

[159] A. Rahman, S. Roy, M. S. Kaiser, and M. S. Islam. A Lightweight Multi-tier S-MQTT Framework to Secure Communication between low-end IoT Nodes. In *2018 5th International Conference on Networking, Systems and Security (NSysS)*, pages 1–6, 2018.

[160] Bernhard Riemann. *Über die Hypothesen, welche der Geometrie zu Grunde liegen.* Springer, Spectrum, 1954.

[161] Alejandro Rodriguez, Lars Kristensen, and Adrian Rutle. On Modelling and Validation of the MQTT IoT Protocol for M2M Communication. In *International Workshop on Petri Nets and Software Engineering 2018*, 06 2018.

[162] P. J. Roig, S. Alcaraz, and K. Gilly. Algebraic specification of ABP protocol using different time constraints. In *21st International Conference Electronics 2017*, pages 1–6, 2017.

[163] P. J. Roig, S. Alcaraz, and K. Gilly. Formal Specification of Spanning Tree Protocol Using ACP. *Elektronika Ir Elektrotechnika*, 23(2):84–91, 2017.

[164] P. J. Roig, S. Alcaraz, and K. Gilly. FTP Algebraic Formal Modelling using ACP - Study on FTP Active Mode and Passive Mode. In *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH 2017*, pages 362–373. INSTICC, SciTePress, 2017.

[165] P. J. Roig, S. Alcaraz, and K. Gilly. Multicast Algebraic Formal Modelling using ACP - Study on PIM Dense Mode and Sparse Mode. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications - Volume 1: DCNET (ICETE 2017)*, pages 57–68. INSTICC, SciTePress, 2017.

[166] P. J. Roig, S. Alcaraz, and K. Gilly. Arithmetic Study on Energy Saving for some common Data Centre Topologies. In Erol Kurt, editor, *Proceedings of the 8th European Conference on Renewable Energy Systems*, pages 744–751, 2020.

[167] P. J. Roig, S. Alcaraz, K. Gilly, C. Bernad, and S. Filiposka. De Bruijn-based and k-ary n-cube-based algebraic models in fog environments. *Communications in Computer and Information Science (CCIS). Digital Transformation*, 1521:126–141, 2022.

[168] P. J. Roig, S. Alcaraz, K. Gilly, C. Bernad, and C. Juiz. Modeling of a Generic Edge Computing Application Design. *Sensors*, 21(21):1–29, November 2021.

[169] P. J. Roig, S. Alcaraz, K. Gilly, C. Bernad, and C. Juiz. Review on de Bruijn shapes in one, two and three dimensions. *Journal of Physics (IOP): Conference Series*, 2090(1):1–10, November 2021.

[170] P. J. Roig, S. Alcaraz, K. Gilly, C. Bernad, and C. Juiz. Arithmetic Framework to Optimize Packet Forwarding among End Devices in Generic Edge Computing environments. *Sensors*, 22(2):1–23, January 2022.

[171] P. J. Roig, S. Alcaraz, K. Gilly, C. Bernad, and C. Juiz. De Bruijn Shapes: Theory and Instances. *New Trends in Physical Science Research*, 4:30–47, May 2022.

[172] P. J. Roig, S. Alcaraz, K. Gilly, C. Bernad, and C. Juiz. Features for Riemannian N-manifold Classification. *New Trends in Physical Science Research*, 4:17–29, May 2022.

[173] P. J. Roig, S. Alcaraz, K. Gilly, C. Bernad, and C. Juiz. Modeling an Edge Computing Arithmetic Framework for IoT Environments. *Sensors*, 22(3):1–25, January 2022.

[174] P. J. Roig, S. Alcaraz, K. Gilly, S. Filiposka, and N. Aknin. Formal algebraic modelling of a city-wide smart parking system. In *2nd International Conference on Electrical, Communication, and Computer Engineering (ICECCE 2020)*, pages 1–6, 2020.

[175] P. J. Roig, S. Alcaraz, K. Gilly, S. Filiposka, and N. Aknin. Formal Algebraic Specification of an IoT/Fog Data Centre for Fat Tree or Leaf and Spine architectures. In *2nd International Conference on Electrical, Communication, and Computer Engineering (ICECCE 2020)*, pages 1–6, 2020.

[176] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. Algebraic Formal Modelling for EIGRP using ACP - Formal Description Modelling on EIGRP Routing Protocol. In *Proceedings of 8th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH 2018*, pages 259–266. INSTICC, SciTePress, 2018.

[177] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. OSPF Algebraic Formal Modelling using ACP - A Formal Description on OSPF Routing Protocol. In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications - Volume 1: ICETE 2018*, pages 55–66. INSTICC, SciTePress, 2018.

[178] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. Study on Mobility and Migration in a Fog Computing Environment. In *22nd International Conference Electronics 2018*, pages 1–6, 2018.

[179] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. Study on OSPF Algebraic Formal Modelling Using ACP. *Elektronika Ir Elektrotechnika*, 24(4):77–83, 2018.

[180] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. Algebraic Formal Modelling for HTTP Main Methods using ACP. In *23rd International Conference Electronics 2019*, pages 1–6, 2019.

[181] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. Modelling VM Migration in a Fog Computing Environment. *Elektronika Ir Elektrotechnika*, 25(5):75–81, 2019.

[182] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. Modelling a Leaf and Spine Topology for VM Migration in Fog Computing. In *24th International Conference Electronics 2020*, pages 1–6, 2020.

[183] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. Algebraic modelling of a generic Fog scenario for moving IoT devices. *Lecture Notes in Networks and Systems (LNNS)*, 187:1–16, 2021.

[184] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. Applying multidimensional geometry to basic fog computing designs. *International Journal of Electrical and Computer Engineering Research (IJECER)*, 1(1):1–8, 2021.

[185] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. Arithmetic Study about Energy Save in Switches for some Data Centre Topologies. *Journal of Polytechnic (Politeknik Dergisi)*, 2021.

[186] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. Modelling a Folded N-Hypercube Topology for migration in Fog Computing. *Lecture Notes in Electrical Engineering (LNEE)*, 736:519–535, 2021.

[187] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz. Modelling a Plain N-Hypercube Topology for migration in Fog Computing. *Lecture Notes in Electrical Engineering (LNEE)*, 736:595–608, 2021.

[188] P. J. Roig, S. Alcaraz, K. Gilly, C. Juiz, and N. Aknin. MQTT Algebraic Formal Modelling Using ACP. In *24th International Conference Electronics 2020*, pages 1–5, 2020.

[189] P. J. Roig, J. Jornet, A. Albaladejo, and J.C. Hernández. Remote surveillance system in isolation for Covid-19. *3rd International Conference on Electrical, Communication, and Computer Engineering (ICECCE 2021)*, pages 1–5, 2021.

[190] P.J. Roig, S. Alcaraz, K. Gilly, and S. Filiposka. Fat Tree Algebraic Formal Modelling Applied to Fog Computing. *Communications in Computer and Information Science (CCIS). Machine Learning and Applications*, 1316:111–126, 2020.

[191] R. Rojas-Cessa and Chuan-Bi Lin. Scalable two-stage Clos-network switch and module-first matching. In *2006 Workshop on High Performance Switching and Routing*, pages 1–6, 2006.

[192] Yonghui Ruan, Zhongsheng Cao, and Zongmin Cui. Pre-Filter-Copy: Efficient and Self-Adaptive Live Migration of Virtual Machines. *IEEE Systems Journal*, 10(4):1459–1469, 2016.

[193] Kai Sachs, Stefan Appel, Samuel Kounev, and Alejandro Buchmann. Benchmarking Publish/Subscribe-Based Messaging Systems. volume 6193, pages 203–214, 04 2010.

[194] A. S. Sadeq, R. Hassan, S. S. Al-rawi, A. M. Jubair, and A. H. M. Aman. A QoS Approach for Internet Of Things (IoT) Environment Using MQTT Protocol. In *2019 International Conference on Cybersecurity (ICoCSec)*, pages 59–63, 2019.

[195] Motoshi Saeki. *Handbook of Software Engineering and Knowledge Engineering*, chapter Formal Description Techniques, pages 239–261. 2005.

[196] G.L. Santos, M. Ferreira, L. Ferreira, J. Kelner, D. Sadok, E. Albuquerque, T. Lynn, and P.T. Endo. *Fog and Edge Computing: Principles and Paradigms*, chapter Integrating IoT + Fog + Cloud Infrastructures: System Modeling and Research Challenges, pages 51–78. John Wiley & Sons, Inc., 2019.

[197] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Resource Provisioning in Fog Computing: From Theory to Practice. *Sensors*, 19(10), 2019.

[198] Joe Sawada, Aaron Williams, and Dennis Wong. A surprisingly simple de Bruijn sequence construction. *Discrete Mathematics*, 339(1):127 – 131, 2016.

[199] Navrati Saxena, Abhishek Roy, Bharat J. R. Sahu, and HanSeok Kim. Efficient IoT Gateway over 5G Wireless: A New Design with Prototype and Implementation Results. *IEEE Communications Magazine*, 55:97–105, 2017.

[200] Ludwig Schläfli. *Gesammelte Mathematische Abhandlungen*, chapter Theorie der vielfachen Kontinuität, pages 167–387. Springer, Basel, 1950.

[201] D. Schüsselbauer, A. Schmid, and R. Wimmer. Dothraki: Tracking Tangibles Atop Tabletops Through De-Bruijn Tori. In *Proceedings of the 15th International Conference on Tangible, Embedded, and Embodied Interaction (TEI '21)*, volume 37, pages 1–10, February 2021.

[202] Amelie Schyn, David Navarre, Philippe Palanque, and Luciana Porcher. Formal description of a multimodal interaction technique in an immersive virtual reality application. In *Proceedings of the 15th Conference on l'Interaction Homme-Machine*, pages 150–157, November 2003.

[203] Abhishek Shakya, Deepak Garg, and Prakash Nayak. *Hybrid Live VM Migration: An Efficient Live VM Migration Approach in Cloud Computing*, pages 600–611. 12 2018.

[204] Carron Shankland and Mark Zwaag. The tree identify protocol of IEEE 1394 in mCRL. *Formal Aspects of Computing*, 10:509–531, 05 1998.

[205] Abhilasha Sharma and R. G. Sangeetha. Performance analysis of torus optical interconnect with data center traffic. *ETRI Journal*, April 2020.

[206] Abhilasha Sharma and R. G. Sangeetha. Terminal and broadcast reliability analysis of direct 2-D symmetric torus network. *The Journal of Supercomputing*, May 2020.

[207] E. Sherratt. SDL: Meeting the IoT Challenge. In *System Analysis and Modeling. Technology-Specific Aspects of Models. SAM 2016. Lecture Notes in Computer Science.*, volume 9959. Springer, Cham., 2016.

[208] F.A. Silva, I. Fé, and G. Gonçalves. Stochastic models for performance and cost analysis of a hybrid cloud and fog architecture. *The Journal of Supercomputing (2020)*, 2020.

[209] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. *SIGCOMM Comput. Commun. Rev.*, 45(4):183–197, August 2015.

[210] Ankit Singla. Fat-FREE Topologies. *HotNets '16: Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 64–70, November 2016.

[211] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362 – 391, 1983.

[212] D.M.Y. Sommerville. *An Introduction to the Geometry of N Dimensions.* Methuen & Co., London, 1929.

[213] Junsup Song and Moonkun Lee. *Process Algebra to Control Nondeterministic Behavior of Enterprise Smart IoT Systems with Probability*, pages 184–196. The Practice of Enterprise Modeling (PoEM 2019), 11 2019.

[214] Rabah Souam. The Schläfli formula for polyhedra and piecewise smooth hypersurfaces. *Differential Geometry and its Applications*, 20:31–45, 01 2004.

[215] A. Souri, A.M. Rahmani, and N.J. Navimipour. Formal verification approaches in the web service composition: A comprehensive analysis of the current challenges for future research. *International Journal of Communication Systems*, 31:3808, 2018.

[216] Alireza Souri and Monire Norouzi. A State-of-the-Art Survey on Formal Verification of the Internet of Things Applications. *Journal of Service Science Research*, 11:47–67, 06 2019.

[217] I. Stojmenovic. Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In *2014 Australasian Telecommunication Networks and Applications Conference (ATNAC)*, pages 117–122, 2014.

[218] I Stojmenovic and Wen S. The Fog Computing Paradigm: Scenarios and Security Issues. In M. Paprzycki M. Ganzha, L. Maciaszek, editor, *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, volume 2, pages 1–8, 2014.

[219] H. Su, S. Liu, B. Zheng, X. Zhou, and K. Zheng. A survey of trajectory distance measures and performance evaluation. *The VLDB Journal*, 29:3–32, 2020.

[220] I. Szentandrasi, M. Zacharias, J. Havel, A. Herout, M. Dubska, and R. Kajan. Uniform Marker Fields: Camera localization by orientable De Bruijn tori. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 319–320, 2012.

[221] Amir Taherkordi, Frank Eliassen, Michael Mcdonald, and Geir Horn. Context-Driven and Real-Time Provisioning of Data-Centric IoT Services in the Cloud. *ACM Transactions on Internet Technology*, 19(1(7)):1–24, 2019.

[222] Andrew Tanenbaum. *Computer Networks.* Prentice Hall Professional Technical Reference, 4th edition, 2002.

[223] F. Tao, L. Zhang, V.C. Venkatesh, Y. Luo, and Y. Cheng. Cloud manufacturing: a computing and service-oriented manufacturing model. In *Proceedings of*

the Institution of Mechanical Engineers Part B Journal of Engineering Manufacture, volume 225, pages 1969–1976, November 2011.

[224] Muhammad Ashar Tariq, Murad Khan, Muhammad Toaha Raza Khan, and Dongkyun Kim. Enhancements and Challenges in CoAP - A Survey. *Sensors*, 20(6391):1–29, 2020.

[225] A F F Vasquez and E Tamura. From SDL Modeling to WSN Simulation for IoT Solutions. *WEA 2018: Applied Computer Sciences in Engineering*, 2018.

[226] Anchal J. Vattakunnel, N. Suresh Kumar, and G Santhosh Kumar. Modelling and Verification of CoAP over Routing Layer Using SPIN Model Checker. *Procedia Computer Science*, 93:299 – 308, 2016. Proceedings of the 6th International Conference on Advances in Computing and Communications.

[227] S. Walklin. Leaf-Spine architecture for OTN switching. In *2017 International Conference on Computing, Networking and Communications (ICNC)*, pages 95–99, 2017.

[228] Jiacun Wang and William Tepfenhart. *Formal Methods in Computer Science*, chapter Petri Nets, pages 201–243. 06 2019.

[229] T. Wang, Z. Su, Y. Xia, and M. Hamdi. Rethinking the Data Center Networking: Architecture, Network Protocols, and Resource Sharing. *IEEE Access*, 2:1481–1496, 2014.

[230] Ting Wang, Zhiyang Su, Yu Xia, Bo Qin, and Mounir Hamdi. NovaCube: A low latency Torus-based network architecture for data centers. *IEEE Conference and Exhibition on Global Telecommunications (GLOBECOM)*, 2015.

[231] Xi Wang, Jianxi Fan, Cheng-Kuan Lin, Jingya Zhou, and Zhao Liu. BCDC: A High-Performance, Server-Centric Data Center Network. *Journal of Computer Science and Technology*, 33:400–416, 2018.

[232] M. Waqas, Y. Niu, M. Ahmed, Y. Li, D. Jin, and Z. Han. Mobility-Aware Fog Computing in Dynamic Environments: Understandings and Implementation. *IEEE Access*, 7:38867–38879, 2019.

[233] Richard West, Gary Wong, and Gerald Fry. Comparison of k-ary n-cube and de Bruijn Overlays in QoS-constrained Multicast Applications. In *Proceedings of the International Conference of Parallel and Distributed Processing Techniques and Applications (PDPTA) 2005*, pages 1336–1342, 01 2005.

[234] Chi Him Wong. *Novel universal cycle constructions for a variety of combinatorial objects*. PhD thesis, The University of Guelph (Canada), 2015.

[235] Ahmet Yazar, Huseyin Arslan, and Seda Dogan Tusha. 6G vision: An ultra-flexible perspective. *ITU Journal on Future and Evolving Technologies (ITU J-FET)*, 1(1), December 2020.

[236] A. Yousefpour, G. Ishigaki, and J. P. Jue. Fog Computing: Towards Minimizing Delay in the Internet of Things. In *2017 IEEE International Conference on Edge Computing (EDGE)*, pages 17–24, 2017.

[237] S. Zahra, M. Alam, Q. Javaid, A. Wahid, N. Javaid, S. U. R. Malik, and M. Khurram Khan. Fog Computing Over IoT: A Secure Deployment and Formal Verification. *IEEE Access*, 5:27132–27144, 2017.

[238] Ivona Zakarijaa, Frano Skopljanac-Macinab, and Bruno Blaskovic. Automated simulation and verification of process models discovered by process mining. *Automatika - Journal for Control, Measurement, Electronics, Computing and Communications*, 61(2):312–324, 2020.

[239] L. Zanzi, F. Cirillo, V. Sciancalepore, F. Giust, X. Costa-Pérez, S. Mangiante, and G. Klas. Evolving Multi-Access Edge Computing to Support Enhanced IoT Deployments. *IEEE Communications Standards Magazine*, 3(2):26–34, 2019.

[240] Qiang Zhu, Xu Jun-Ming, Xinmin Hou, and Min Xu. On reliability of the folded hypercubes. *Information Sciences*, 177(8):1782–1788, April 2007.