network

MDPI

Article

Edge Data Center Organization and Optimization by Using Cage Graphs

Pedro Juan Roig [1,*], Salvador Alcaraz [1], Katja Gilly [1], Cristina Bernad [1] and Carlos Juiz [2]

[1] Computer Engineering Department, Miguel Hernández University, 03202 Elche, Spain
[2] Mathematics and Computer Science Department, University of the Balearic Islands, 07022 Palma de Mallorca, Spain
* Correspondence: proig@umh.es; Tel.: +34-966658388

Abstract: Data center organization and optimization are increasingly receiving attention due to the ever-growing deployments of edge and fog computing facilities. The main aim is to achieve a topology that processes the traffic flows as fast as possible and that does not only depend on AI-based computing resources, but also on the network interconnection among physical hosts. In this paper, graph theory is introduced, due to its features related to network connectivity and stability, which leads to more resilient and sustainable deployments, where cage graphs may have an advantage over the rest. In this context, the Petersen graph cage is studied as a convenient candidate for small data centers due to its small number of nodes and small network diameter, thus providing an interesting solution for edge and fog data centers.

Keywords: cage graph; data center design; graph integrity; graph theory; Petersen graph

Citation: Roig, P.J.; Alcaraz, S.; Gilly, K.; Bernad, C.; Juiz, C. Edge Data Center Organization and Optimization by Using Cage Graphs. *Network* **2023**, *3*, 93–114. https://doi.org/10.3390/network3010005

Academic Editors: Dawei Li, Minjun Xiao, Sadoon Azizi, Wei Chang, Ning Wang and Huan Zhou

Received: 22 November 2022
Revised: 11 January 2023
Accepted: 16 January 2023
Published: 18 January 2023

1. Introduction

Data center deployments are exponentially growing because of the rise in edge and fog domains powered by AI-based technologies [1]. This situation is bringing more attention to data center organization and optimization [2] in order to improve their performance in IoT environments [3]. In this context, machine learning techniques may help optimize performance by achieving increased efficiency in resource usage [4], leading to a decrease in energy consumption [5], which may lower the carbon footprint [6] in order to make the IoT environments more sustainable [7] and resilient [8].

Modern data centers are composed of physical hosts where remote computing resources are allocated within [9], usually being virtual machines or virtual containers [10]. In this sense, communications among virtual resources within the same physical machine are internal, which are much faster than east-to-west traffic, which takes place among diverse physical devices. The former takes place among internal means within a physical host, whereas the latter occurs through network connections among physical hosts [11].

Hence, in order to improve the performance of data centers, it is convenient to optimize the allocation strategy of virtual resources within physical nodes in order to gather interdependent resources and minimize the number of hops among nodes [12], which may lead to shorter migration times of the virtual resources between any pair of physical devices [13]. Such strategies to obtain efficient resource utilization are usually implemented through the use of generic algorithms, where multiple factors need to be taken into consideration, such as the current and predicted state of the virtual resources, the current and predicted application workloads, or the cost of migration to other hosts [14], as well as the data center network topology.

In this paper, a data center layout is proposed in order to minimize the number of hops between any pair of physical hosts, thus reducing the migration cost. The topology proposed is based on graph theory and, more specifically, on cage graphs, where all nodes share the same number of neighbors and the shortest cycle within the design. Furthermore,

some instances of small cage graphs are presented, such as Petersen, Heawood, and Robertson graphs, which may be useful for small to medium data centers, as its network diameter is only two in the first or three in the second and third, wehreas the number of links is not as high as that of complete graphs.

Additionally, we developed an algorithm to forward traffic through the 10 nodes within a Petersen graph in an optimized manner, thus providing a method to quickly reach any other node by using integer divisions and modular arithmetic, as opposed to searching for a match throughout the traditional routing tables or mac-address lists. The algorithmic method needs fewer operations to find the proper link to send traffic toward its destination, even though its actual performance improvement rate may depend on how it is implemented in hardware. Additionally, another algorithm devoted to Heawood graph was designed to forward traffic among its 14 nodes where the key point is the distance between source and destination node, as well as the outline to carry out a further algorithm for the Robertson graph to move among its 19 nodes.

The rest of this paper is organized as follows: Section 2 shows the motivation of this study. Section 3 presents an overview of graph theory regarding integrity and cage graphs; Section 4 explains the advantages of a Petersen graph to boost performance in small data centers, which is followed by Section 5, constructing a Petersen graph for small data centers. Section 6 is devoted to Heawood graphs and Robertson graphs; then, Section 7 presents an experimental analysis comparing the aforementioned topologies with other commonly used in data centers. Finally, Section 8 draws some final conclusions.

2. Motivation

Data center organization and optimization are key aspects when it comes to boosting operation and maintenance (O&M), which is basically driven by three converging forces: powerful operations, technology, and economics [15]. The first point is focused on data availability for real-time decision making. The second point is centered on the flexibility to deal with fast-changing situations. The third point is about delivering the best performance while reducing carbon emissions.

Optimizing facilities is about increasing efficiency, which produces a variety of benefits such as decreasing power consumption, which also reduces downtime; about increasing the mechanical capacity of the facility, thus improving HVAC efficiency and redundancy, prolonging equipment life span, and saving money on new equipment or unplanned maintenance; and becoming more climate-friendly, as both power consumption and HVAC needs are reduced [16].

According to the literature, there are many variables involved in achieving efficiency in data centers, such as servers, storage, networking, infrastructure, environment monitoring, cabling, backup power, cooling, and software. For example, some commonly used techniques in data centers are virtualization, containerization, server consolidation, storage unification, and enhanced cooling systems [17]. In this sense, network systems and storage capacities should be maximized by updating protocols to achieve higher bandwidth rates and capacity. As such, monitor power usage is necessary to avoid unexpected fallouts, whereas high-quality cabling guarantees high-speed rates while lowering downtime.

Moreover, physical space can be reduced by consolidating servers, whereas virtualization improves the usage ratio for both storage systems and central servers, as it reduces the number of servers needed and facilitates the accommodation of dynamically changing requirements. Additionally, the use of software-defined networks (SDNs) allows moving server bandwidth away from processing units within the infrastructure, although at the expense of an additional load cost. Moreover, software optimizations help improve efficiency, which may be automatically performed by implementing AI tools. Furthermore, hyperscale networks may optimize computer-intensive tasks through software to enhance caching, data encryption, or intelligent tiering. Additionally, service-oriented architecture (SOA) may be applied to increase the number of connections between applications and systems.

Nonetheless, power consumption may be seen as the greatest concern related to data center optimization. For this optimization, the applied power efficiency strategies may be classified into IT equipment, power infrastructure, HVAC systems, and airflow management [18]. The first one includes server consolidation, efficient data storage systems, and built-in server power management. The second involves the use of smart power distribution units (PDUs) and uninterruptible power supplies (UPSs). The third is the utilization of water-side or air-side economizers, using rack or in-row cooling, adjusting humidity, and employing sensors and controls to match IT loads with the level of cooling applied. The fourth involves efficient cooling airflow management, using a hot aisle and cold aisle layout and employing enclosure strategies.

However, the focus on this study was networking, from the point of view of the networking interconnection among the diverse nodes being part of a data center in order to reduce the number of hops between any pair of nodes so as to optimize the time required from a given source to a particular destination. Therefore, the main contribution of this study is related to the use of cage graph topologies to interconnect nodes in an edge data cente. As such, an algorithm was designed for each of the smallest cage graphs chosen in order to easily move traffic between any given pair of nodes, as described in Sections 5 and 6.

Additionally, those algorithms may be the core of an alternative way of packet forwarding in small data centers, instead of the traditional use of lookup tables in nodes, working either at layer two or three, according to the OSI model. The feasibility of this innovative model compared with the traditional lookup table model depends on the hardware implementation of the necessary operations defined by each model. This is descxribed in Section 7.1. Additionally, an experimental analysis was undertaken among some commonly used topologies in small data centers and the proposed small cage graphs by comparing the average number of hops between hosts and the average number of links per nodes as a way to compare performance and ease of design. This is described in Section 7.2.

It is to be noted that IoT edge devices are not usually installed in data centers. However, in this case, we considered a situation with IoT mobility with plenty of moving IoT devices. In this particular scenario, IoT devices may be moving around the edge domain, and the computing resources associated with each IoT device are located in a specific node according to the location of that IoT device at a given time, even though resource migration may be performed when the position of the device significantly changes to allocate them onto a more convenient node [19].

In this context, we considered that edge domains are circumscribed to a given area; thus, a small to medium amount of servers is usually required to deal with the users within the area, which is why the size of data centers should be kept small [20]. Hence, if a limit of 10 nodes is established, then the only cage graph available (apart from those related to complete graphs and complete bipartite graphs) is the Petersen graph, which we therefore thoroughly studied. On the other hand, if a limit of 20 nodes is established, then there are other two cage graphs available: the Heawood graph, which contains 14 nodes, and the Robertson graph, which includes 19 nodes. Therefore, the proposed IoT scenario is related to moving IoT devices within an edge domain, where the network topology of the data center serving such a domain is represented by a cage graph interconnecting the nodes.

Furthermore, the innovations proposed in this paper cover different areas. The most important one is the design of algorithms to define the behavior of the smallest cage graphs. Here, an algorithm was developed for moving around the nodes of a Petersen graph, which is composed of 10 nodes, where the locations of source and destination nodes lead to four different case scenarios. Moreover, another algorithm was developed to move about the nodes of a Heawood graph being formed by 14 nodes. In this case, a uniform pattern was established for all nodes, as they have a remote link toward the same distance, even though either clockwise or counterclockwise depends on whether the node is even or odd, resulting in an alternative sequence. Additionally, a Robertson graph was studied, which was made by 19 nodes, where each node has two remote links: one of them clockwise and

the other one counterclockwise, although their distances are not uniform among neighbor nodes. Hence, this prevented the existence of a regular pattern, as an algorithm similar to the previous ones being valid for all nodes was not possible; thus, it was necessary to consider each node as a separate case.

A second innovation is the consideration of these algorithms as an alternative method to carry out packet forwarding among the nodes of the data center. that the traditional way to perform this is by using routing and forwarding tables, where the destination layer-3 or layer-2 address is searched through a table in order to look up a match, which is achieved with the network portion of the former or with the whole address of the latter. For the former, the comparisons are made by using bitwise logical AND operations, the number of which depends on the prefix length For the latter, such comparisons involve all 48 bits of a MAC address. Hence, a possible innovation is undertaking packet forwarding through the arithmetic operations involved in the algorithms proposed, where the hardware implementation plays a key role in assessing this.

As a final innovation, an experimental analysis was carried out by calculating the average number of hops and the average number of links per node for some topologies employed in small data centers. In this way, the arithmetic values obtained for those topologies represent a performance measurement in the first case, as a smaller number of hops means a faster interconnection network, whereas those in the second case represent the ease of operation and maintenance, as a smaller number of links indicates a network topology that is easier to deal with and more easily forwards traffic.

A concrete example of IoT/Edge architecture related to this study is an edge domain where IoT devices may be moving around the coverage area of the domain. IoT devices need to have their computing resources as close as possible due to their constraints; hence, it would be convenient to deploy the computing nodes distributed throughout the edge domain and being linked together according to a cage graph design, supposing that the coverage area is somewhat circular, such that the whole domain is divided into influence areas dominated by a single node. In this way, when an IoT device becomes closer to the area of a new node, a migration of computing resources is carried out between the node having the resources and the new node being closer to the device. Cage graphs have the advantage of allowing movement from one node to another in a fixed upper bound of hops, such as two hops away for Petersen graphs or three hops away for Heawood or Robertson graphs. Therefore, using cage graphs as the network topology to interconnect the nodes in a distributed data center offers the certainty of undertaking the migration of computing resources in very few hops, while offering diverse redundant paths to in the case of any link failure.

Regarding some examples of a real-life implementation, the ideal situation is where users may be somewhat distributed within a wide area. In this context, smart agriculture may be a good scenario, as diverse crops can be disseminated throughout the field. Moreover, smart cities are another convenient scenario, as different points of service along the city can be interconnected. Additionally, IoT is a further scenario, as all devices within a smart factory may be linked together. In any of these cases, the interconnection among nodes may be achieved through one of the cage graph topologies proposed in order to arrive at the destination in two or three hops, such that a device moving through could obtain its computing resources from the closest location possible. For instance, in smart agriculture, it would possible for a fumigation robot to move along the field and migrate its computing resources to the closest node at any point.

3. Overview of Graph Theory

Graph theory is well known in the field of computer science [21]. Some background about integrity and cage graphs is provided in this section.

3.1. Integrity of Graphs

Communication networks need higher degrees of stability to perform properly, which means lower levels of vulnerability due to potential failure occurrences. In this sense, it may seem clear that network efficiency reduces when some nodes or edges are down. This leads to the definition of graph integrity ($I(G)$, also known as vertex integrity) [22] as a measure of the vulnerability of a graph G, as shown in (1), where $m(G-S)$ describes the order of the largest component of $G-S$, and $|S|$ is the order of the given subset S of $V(G)$. ore resilient graphs are obtained when integrity values are higher, meaning more nodes need to be deleted. Order refers to the number of nodes of a graph, whereas size refers to the number of its links.

$$I(G) = min\{|S| + \ m(G-S) : S \subseteq V(G)\} \tag{1}$$

Furthermore, any subset $S \subseteq V(G)$ may be called the Ith set of G if (2) applies.

$$I(G) = |S| + \ m(G-S) \tag{2}$$

Otherwise, edge integrity ($I'(G)$) may be defined in an analogous way by swapping $V(G)$ with $E(G)$ [23], as shown in (3).

$$I'(G) = min\{|X| + \ m(G-X) : X \subseteq E(G)\} \tag{3}$$

There is an intrinsic relationship between both values related to a given graph, such that vertex integrity $I(G)$ never exceeds edge integrity $I'(G)$. Hence, if G is a nontrivial connected graph of order n, then (4) applies [24].

$$2 \leq I(G) \leq I'(G) \leq n \tag{4}$$

Taking all this into consideration, integrity may be a more suitable parameter to measure network reliability, as it does not only account for node connectivity when some of those nodes are removed, but it also takes into account the remaining nodes that are functioning [25], which may give a more complete picture regarding network stability.

The values of integrity of some simple types of graphs are provided in Table 1 [26], whereas combinations of those result in diverse values of integrity [27].

Table 1. Integrity of some well-known kinds of graphs.

Graph Name	Symbol	Vertex Integrity	Edge Integrity
Complete graph	K_n	$I(K_n) = n$	$I'(K_n) = n$
Null graph	$\overline{K_n}$	$I(\overline{K_n}) = 1$	$I'(\overline{K_n}) = 1$
Star graph	$K_{1,n}$	$I(K_{1,n}) = 2$	$I'(K_{1,n}) = n+1$
Path graph	P_n	$I(P_n) = \lceil 2\sqrt{n+1} \rceil - 2$	$I'(P_n) = \lceil 2\sqrt{n} \rceil - 1$
Cycle graph	C_n	$I(C_n) = \lceil 2\sqrt{n} \rceil - 1$	$I'(C_n) = \lceil 2\sqrt{n} \rceil$
Complete bipartite graph	$K_{a,b}$	$I(K_{a,b}) = 1 + \min\{a, b\}$	$I(K_{a,b}) = m + n$
Wheel graph	W_n	$I(W_n) = \lceil 2\sqrt{n-1} \rceil$	$I(W_n) = \lceil 2\sqrt{n} \rceil + 1$
Spider graph	G_S	$I(G_S) = \lfloor (n+1)/2 \rfloor$	$I(G_S) = \lfloor (n+3)/2 \rfloor$
n-cube	Q_n	$I(Q_n) = 1 + 2^{n-1}$	$I(Q_n) = 2^n$

Additionally, from the definition proposed for vertex integrity, two extra parameters computationally useful may be defined [28]: $D_k(G)$ (5) and $E_l(G)$ (6), which better denote the stability of a network.

$$D_k(G) = min\{|S| : S \subset V(G), m(G-S) \leq k\}, \text{ where } k = \{1 \cdots |V(G)| - 1\} \tag{5}$$

$$E_l(G) = min\{m(G-S) : S \subset V(G), |S| = l\}, \text{ where } l = \{0 \cdots |V(G)| - 1\} \tag{6}$$

$D_1(G) = \alpha(G)$, meaning that the vertex cover of graph G, as well as $E_0(G) = m(G)$, represents the maximum order of any component of G additionally, $D_{|V(G)|}(G) = 0$ and $E_{|V(G)|}(G) = 0$.

The concepts of $D_k(G)$ and $E_l(G)$ lead to the definition of D integrity (7) and E integrity (8), respectively [29].

$$DI_k(G) = \sum_{k=1}^{|V(G)|-1} D_k(G) \tag{7}$$

$$EI_l(G) = \sum_{l=0}^{|V(G)|-1} E_l(G) \tag{8}$$

The values of these new kinds of integrity of some simple graphs are cited in Table 2, whereas combinations of these account for different values, whose calculation still represents an open problem in graph theory.

Table 2. D and E integrity of some well-known sorts of graphs.

Symbol	D Integrity	E Integrity
K_n	$DI_k(K_n) = {}^{(n^2-n)}/_2$	$EI_l(K_n) = {}^{(n^2+1)}/_2$
$\overline{K_n}$	$DI_k(\overline{K_n}) = 1$	$EI_l(\overline{K_n}) = 1$
$K_{1,n}$	$DI_k(K_{1,n}) = n$	$EI_l(K_{1,n}) = {}^{(n^2+n)}/_2$
G_S	$DI_k(G_S) = 4n - 4$, if $n \geq 3$	$EI_l(G_S) = 5n - 5$, if $n \geq 3$
$K_{a,b}$	$\begin{cases} DI_k(K_{a,b}) = \dfrac{3n^2 - n}{2}, \text{ if } n = m; \\ \dfrac{n^2 + 2mn - n}{2}, \text{if } n \neq m, n < m. \end{cases}$	$\begin{cases} EI_l(K_{a,b}) = \dfrac{3n^2 + 3n}{2}, \text{ if } n = m; \\ \dfrac{2nm + m^2 + m + 2n}{2}, \text{if } n \neq m, n < m. \end{cases}$

Furthermore, for any graph G, the lower bound of $DI_k(G)$ is $n - 1$, which is obtained if $G = K_{1,n-1}$, whereas the upper bound is ${}^{n\cdot(n-1)}/_2$, which is attained if $G = K_n$. Therefore, if $G_1 \cdots G_n$ are the components of a graph G, $DI_k(G_1 \cup \cdots \cup G_n) = DI_k(G_1) + \cdots + DI_k(G_n)$. On the other hand, for any $x \in V(K_p)$, $DI_k(K_p) = DI_k(K_n/x) + n - 1$, because $K_n/x = K_{n-1}$, leading to $DI_k(K_{n-1}) = {}^{(n-1)\cdot(n-2)}/_2$. Moreover, for any particular graph G, the lower bound of $EI_l(G)$ is $n + 1$, which is achieved when $G = K_2$, while the upper bound is ${}^{n^2+n}/_2$, which is attained if $G = K_n$.

3.2. Outline of Cage Graphs

An (r, g)-cage graph is an undirected connected simple graph being r regular, which has the smallest possible number of nodes for its girth. Hence, all nodes have the same number of incident edges, given by r, whereas the length of its shortest cycle is given by g. Thus, every node has r neighbors and girth g.

There are some special cages, such as those compiled in the following list:

- $(2, g)$ is the cycle graph C_g with g nodes;
- $(r, 2)$ is the multigraph of r edges between just two nodes;
- $(r, 3)$ is the complete graph K_{r+1} with $r + 1$ nodes;
- $(r, 4)$ is the complete bipartite graph $K_{r,r}$ with $2r$ nodes.

The number of nodes necessary to build an (r, g)-cage graph bears a lower bound, called the Moore bound, and an upper bound, named the Sauer bound [30]. The former is presented in (9), whereas the latter is shown in (10).

$$\text{cage}\,(r,g) \geq \text{Moore}\,(r,g) = \begin{cases} 1 + r\sum_{i=0}^{\frac{g-3}{2}}(r-1)^i \text{ if } g \text{ is odd} \\ 1 + r\sum_{i=0}^{\frac{g-4}{2}}(r-1)^i + (r-1)^{\frac{g-2}{2}} \text{ if } g \text{ is even} \end{cases} \tag{9}$$

$$\text{cage}(r,g) \leq \text{Sauer}\,(r,g) = \begin{cases} 2(r-2)^{g-2} \text{ if } g \text{ is odd} \\ 4(r-1)^{g-3} \text{ if } g \text{ is even} \end{cases} \tag{10}$$

There are some well-known Moore graphs [31], such as those where $g = 5$ and $r = \{3, 7, 57\}$, or those where $g = 6$ and $r = \{3, 4\}$, although most (r, g) cages contain a number of nodes between Moore (r, g) and Sauer (r, g).

The most well-known cage graphs are those where $r = 3$, resulting in $(3, g)$-cage graphs, which are presented in Table 3, where g values range from 5 to 12 [32]. Furthermore, the known values of g for (r, g)-cage graphs with $r = \{4, 5, 6, 7\}$ are also listed in Table 4 [33].

Table 3. $(3, g)$-cage graph classification for $g = \{5 \cdots 12\}$.

Symbol	Name	Order	Instances
$(3,5)$	Petersen graph	10	1
$(3,6)$	Heawood graph	14	1
$(3,7)$	McGee graph	24	1
$(3,8)$	Levi graph	30	1
$(3,9)$	First one: Biggs and Hoare graph Remainder: Brinkmann, McKay, and Saager	58	18
$(3,10)$	Balaban 10-cage graph Harries graph Harries–Wong graph	70	3
$(3,11)$	Balaban 11-cage graph	112	1
$(3,12)$	Tutte graph	126	1

Table 4. Known (r, g)-cage graph classification for $r > 3$.

Symbol	Name	Order	Instances
$(4,5)$	Robertson graph	19	1
$(4,6)$	Wong graph	26	1
$(4,7)$	Exoo graph	67	1
$(4,8)$	$(4,8)$-cage graph	80	1
$(4,12)$	$(4,12)$-cage graph	728	1
$(5,5)$	Robertson-Wegner graph Foster graph Meringer graph Wong graph	30	4
$(5,6)$	$(5,6)$-cage graph	42	1
$(5,8)$	$(5,8)$-cage graph	170	1
$(5,12)$	$(5,12)$-cage graph	2730	1
$(6,5)$	$(6,5)$-cage graph	40	1
$(6,6)$	$(6,6)$-cage graph	62	1
$(6,8)$	$(6,8)$-cage graph	312	1
$(6,12)$	$(6,12)$-cage graph	7812	1
$(7,5)$	Hoffman–Singleton graph	50	1
$(7,6)$	$(7,6)$-cage graph	90	1

3.3. Integrity of Cage Graphs

The aforementioned concept of integrity may also be applied to cage graphs in order to show how easy it is to divide the graph or the network it represents into various pieces by deleting the minimum amount of nodes.

Regarding cage graphs, there are some nonisomorphic instances for some given (r, g), such as 18 of them for $(3, 9)$, 3 of them for $(3, 10)$, and 4 of them for $(5, 5)$, where all the first numbers ones the same integrity (24), while all the second numbers one the same integrity of 28. However, one instance, the third one, has an integrity of 19, whereas the other three have an integrity of 18 [34].

The integrity of small cage graphs, such as $(3, g)$, where $g \leq 10$, and the rest of the small cage graphs with order up to 60 are given in Table 5 [35], where values for the vertex integrity are exhibited. The integrity for graphs with at least 30 nodes may be achieved by direct searching through all available subsets $S \in V(G)$, although the exhaustive procedure becomes harder as the order grows, leading to the search for theoretical bounds of integrity as opposed to direct searching through subsets.

Table 5. Integrity of small cage graphs.

Order	Symbol	Nonisomorphic Instances	Integrity
10	$(3, 5)$	1	6
14	$(3, 6)$	1	8
19	$(4, 5)$	1	11
24	$(3, 7)$	1	12
26	$(4, 6)$	1	14
30	$(3, 8)$	1	14
30	$(5, 5)$	4	18 or 19
40	$(6, 5)$	1	25
42	$(5, 6)$	1	22
50	$(7, 5)$	1	30
58	$(3, 9)$	18	24
70	$(3, 10)$	3	28

4. Advantages of a Petersen Graph to Boost Performance in Small Data Centers

Using a cage graph as a topology for a data center may bring several advantages, such as the aforementioned integrity values, making the underlying network more resilient and sustainable, as well as having steady values of regularity and girth, thus making the length of the available paths between any particular pair of nodes more predictable. However, the most interesting feature is the short values of network diameter, which is the maximum distance between any two nodes.

As edge computing admits a data center layout much simpler than their counterparts in the cloud, and even in the fog, because of the lower number of users involved within edge domains, the most convenient layout for edge environments using cage graphs seems to be the Petersen one. The Peterson is the smallest cage graph, allowing all users may be dealt with within just 10 nodes. Complete graphs and complete bipartite graphs are omitted so as to avoid full mesh-like topologies and to try and simplify the data center designs as much as possible.

Hence, as stated above, the order of a $(3, 5)$-cage graph is 10, which is the number of nodes involved in the topology, and the integrity is 6. Moreover, every node is three-regular, as each of those vertices have three links toward their counterparts, whereas the girth is five, such that the shortest cycle between any couple of vertices is five. The diameter is just two, resulting in only two hops away between a given source node and a particular destination node.

In this way, if an edge data center is composed of 10 nodes being interconnected as a Petersen graph, such a design may take advantage of an integrity value of six, which

makes for a really resilient layout, as diverse links may be down while not affecting the overall performance must, and a diameter value of two, which brings any node destination within only a two-hop upper bound.

Focusing on edge data centers, the aforesaid nodes may be considered as physical nodes, each of those keeping virtual nodes within. The latter may be migrated toward another physical node in order to stay as close as possible to their associated users in a moving IoT environment, which may be achieved in one or two hops, thus resulting in a network fast to be traversed and reducing the migration times, which also implies a higher energy saving rate.

The cage graphs are not planar graphs by definition, as the latter may be represented in a bidimensional layout without any pair of crossing links, which is not possible when dealing with the former. Hence, a Petersen graph may be drawn in a couple of alternative ways, such as in Figure 1, where a combination of a pentagon and a pentagonal star is exhibited, or in Figure 2, where a hierarchical layout is depicted, thus relieving the diameter of two for any pair of nodes, even though the first one is preferred for clarity purposes when assigning node identifiers so as to describe the forwarding strategy among nodes.

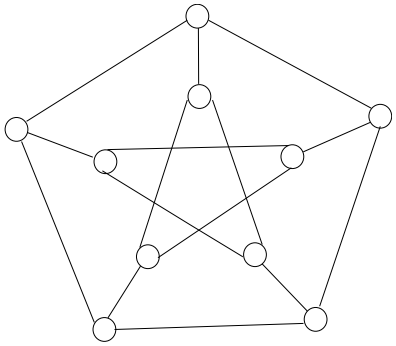

Figure 1. Petersen graph as a combination of a regular pentagon and a pentagonal star.

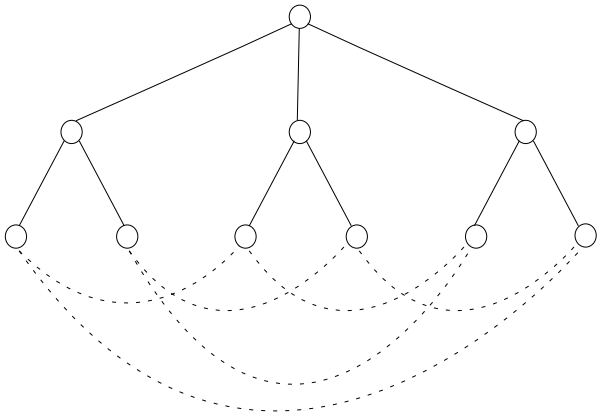

Figure 2. Petersen graph represented as a hierarchical layout.

5. Constructing a Petersen Graph for Small Data Centers

The first step to build up a model for a Petersen graph is to establish the node distribution. In order to perform this, two circles must be considered: the outer one, corresponding to the vertices of a pentagon, and the inner one, representing the vertices of a pentagonal star. Then, nodes from 0 to 4 are assigned to the former, while nodes from 5 to 9 are assigned to the latter. Furthermore, the first node within each circle is allocated to the upper position, thus marking the 12 hours in a clock layout, whereas the rest of nodes within a circle are sequentially assigned in a clockwise direction, as shown in Figure 3.

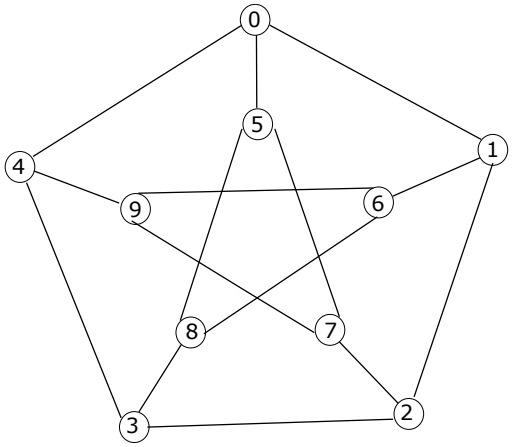

Figure 3. Petersen graph represented as a hierarchical layout.

Once the node identification scheme is established, the ports going from a particular node i to its neighboring nodes need to be identified as well, as depicted in Figure 4, where the predecessor node is the one linked through the counterclockwise link, namely port 0; the successor node is the one linked through the clockwise link, namely port 1; and the opposite node is the one located within the opposite circle through the intercircle link, namely port 2.

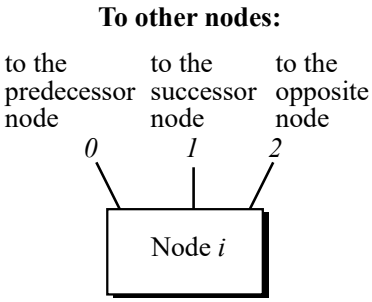

Figure 4. Petersen graph node with ports going to their neighboring nodes.

The outer circle belongs to a pentagon, such that the counterclockwise link from a particular node i connects to node $(i-1)_{|5} = (i+4)_{|5}$, whereas its clockwise link leads to node $(i+1)_{|5}$. Additionally, the inner circle belongs to a pentagonal star, such that the counterclockwise link from a given node i heads for node $(i-2)_{|5} = (i+3)_{|5}$, while its clockwise link heads for node $(i+2)_{|5}$. Furthermore, the opposite link from a particular node i always results in $(i+5)_{|5} = (i)_{|5}$, although it leads to node $i + 5 \cdot (-1)^{\lfloor i/5 \rfloor}$, being located in the opposite circle (outer or inner).

With this scheme, traffic forwarding among nodes is facilitated by applying arithmetic operations, such as integer divisions or modular arithmetic, compared with applying searching through forwarding tables, as is the case in traditional networks, such as routing tables or MAC-address tables, thus improving performance.

In order to forward traffic from a given source node i to a particular destination node j, consider the circles in which both are located, resulting in four different scenarios to be studied on an individual basis:

1. i and j are both in the outer circle, resulting in $\lfloor i/5 \rfloor = \lfloor j/5 \rfloor = 0$:
   - If $(j-i)_{|5} < 5/2$, then traffic is sent through port 1 to the successor node; otherwise, it is forwarded through port 0 to the predecessor node.
2. i and j are both in the inner circle, resulting in $\lfloor i/5 \rfloor = \lfloor j/5 \rfloor = 1$:
   - If $i_{|2} = j_{|2}$, then traffic is sent through port 1 to the successor node; otherwise, it is forwarded through port 0 to the predecessor node.

3. i is in the outer circle and j is in the inner one, where $\lfloor i/5 \rfloor = 0$ and $\lfloor j/5 \rfloor = 1$:
   - $\lfloor i/5 \rfloor < \lfloor j/5 \rfloor$; hence, if $(i+1)_{|5} = j_{|5}$, then traffic is sent through port 1 to the successor node; otherwise, if $(i+4)_{|5} = j_{|5}$, then it is sent through port 0 to the predecessor node, otherwise through port 2 to the opposite node within the inner circle.
4. i is in the inner circle and j is in the outer one, where $\lfloor i/5 \rfloor = 1$ and $\lfloor j/5 \rfloor = 0$:
   - $\lfloor i/5 \rfloor > \lfloor j/5 \rfloor$; hence, if $(i+2)_{|5} = j_{|5}$, then traffic is sent through port 1 to the successor node. Otherwise, if $(i+3)_{|5} = j_{|5}$, then it is sent through port 0 to the predecessor node; otherwise, it is sent through port 2 to the opposite node within the outer circle.

Therefore, each case scenario only needs one or two hops to reach any destination, which proves the improvement in performance compared with the traditional routing and forwarding tables, where searching through the entries of the table must be undertaken in order to find the proper match. This involves more computational resources in order to apply logical operations so as to achieve the result.

In summary, Algorithm 1 exhibits this process of forward traffic through arithmetic means, where all four different case scenarios are taken into account.

Algorithm 1: Moving from node i to node j in a Petersen graph.

```
if ⌊i/5⌋ = ⌊j/5⌋ = 0 then
  if (j − i)_|5 < 5/2 then
    send data through switch i, port 1
  else
    send data through switch i, port 0
  end if
else if ⌊i/5⌋ = ⌊j/5⌋ = 1 then
  if i_|2 = j_|2 then
    send data through switch i, port 1
  else
    send data through switch i, port 0
  end if
else if ⌊i/5⌋ < ⌊j/5⌋ then
  if (i + 1)_|5 = j_|5 then
    send data through switch i, port 1
  else if (i + 4)_|5 = j_|5 then
    send data through switch i, port 0
  else
    send data through switch i, port 2
  end if
else
  if (i + 2)_|5 = j_|5 then
    send data through switch i, port 1
  else if (i + 3)_|5 = j_|5 then
    send data through switch i, port 0
  else
    send data through switch i, port 2
  end if
end if
```

Moreover, the flow chart established by that algorithm may be expressed by means of a process algebra called Algebra of Communicating Processes (ACP) [36], which allows specifying and verifying concurrent distributed models in a formal algebraic manner. In this context, every node is seen as an object i, where two atomic actions are considered:

sending a generic message d through a port p, namely $s_{i\xrightarrow{p}}(d)$; and receiving such a message through port p, namely, $r_{i\xrightarrow{p}}(d)$.

Additionally, some operators need to be applied to those objects [37], such as a sequential one, denoted by $\cdot$; an alternate one, described by $+$; a concurrent one, cited by $\|$; or a conditional one, quoted by the expression $(true \lhd condition \rhd false)$, where true and false stand for the actions being performed depending on the boolean condition. If no action needs to be performed in a particular case, then symbol $\emptyset$ is represented therein.

Additionally, the encapsulation operator ∂_H turns internal atomic actions into communications, thus showing the sequence of events within the model. The abstraction operator τ_I masks all internal communications, hence revealing the external behavior of the model, which in turn may be compared with that of the real system [38]. At that point, if both external behaviors show the same string of actions and the same branching structure, then the model and the real system are considered to be rooted branching bisimilar, which is a sufficient condition to verify a model [39].

Expression (11) depicts the model of a given node i going from 0 to 9, where it may either be the source node of a transaction, which is represented by the neutral element of the product, being 1; otherwise, it may be an intermediate node of a transaction, which is denoted as $r_{x\xrightarrow{p}}(d)$, where x identifies the node where the message is coming in, in order to express that some traffic flow is received at one of its available ports p. Then, the process to forward it to the relevant destination starts. If node i were the destination node of a transaction (j), then no traffic forwarding toward other nodes would ever take place.

$$
\begin{aligned}
V_i \quad = \|_{p=0}^{2}\left(1 + r_{x\xrightarrow{p}}(d)\right) \cdot & \left(\left(s_{i\xrightarrow{1}}(d) \lhd \left((j-i)_{|5} < 5/2\right) \rhd s_{i\xrightarrow{0}}(d)\right) \lhd \left(\lfloor i/5 \rfloor = \lfloor j/5 \rfloor = 0\right) \rhd \emptyset\right) \cdot \\
& \left(\left(s_{i\xrightarrow{1}}(d) \lhd \left(i_{|2} = j_{|2}\right) \rhd s_{i\xrightarrow{0}}(d)\right) \lhd \left(\lfloor i/5 \rfloor = \lfloor j/5 \rfloor = 1\right) \rhd \emptyset\right) \cdot \\
& \left(\left(s_{i\xrightarrow{1}}(d) \lhd \left((i+1)_{|5} = j_{|5}\right) \rhd \left(s_{i\xrightarrow{0}}(d) \lhd \left((i+4)_{|5} = j_{|5}\right) \rhd s_{i\xrightarrow{2}}(d)\right)\right) \lhd \left(\lfloor i/5 \rfloor < \lfloor j/5 \rfloor\right) \rhd \emptyset\right) \cdot \\
& \left(\left(s_{i\xrightarrow{1}}(d) \lhd \left((i+2)_{|5} = j_{|5}\right) \rhd \left(s_{i\xrightarrow{0}}(d) \lhd \left((i+3)_{|5} = j_{|5}\right) \rhd s_{i\xrightarrow{2}}(d)\right)\right) \lhd \left(\lfloor i/5 \rfloor > \lfloor j/5 \rfloor\right) \rhd \emptyset\right) \cdot V_i
\end{aligned}
\tag{11}
$$

In order to achieve the sequence of events occurring when all nodes taking part in the topology are running in a concurrent fashion, the aforesaid encapsulation operator may be applied so as to represent the channels where communications may happen. This leads to deadlocking the rest of the potential internal actions. In this sense, (12) shows the result of such an action, where it also may be seen that there is no longer an initial term, as all nodes are executed at once in a concurrent manner. In order to keep it simple, communications are identified by the sending end of each channel, thus obtaining an analogous expression to the one shown above, although this one involves all 10 nodes, whereas the previous one involves just 1 node.

$$
\begin{aligned}
\|_{i=0}^{9}\partial_H(V_i) \quad = \|_{i=0}^{9}\|_{p=0}^{2}\Bigg(\Bigg(c_{i\xrightarrow{1}}(d) \lhd \left((j-i)_{|5} < 5/2\right) \rhd c_{i\xrightarrow{0}}(d)\Bigg) \lhd \left(\lfloor i/5 \rfloor = \lfloor j/5 \rfloor = 0\right) \rhd \emptyset\Bigg) \cdot & \\
\left(\left(c_{i\xrightarrow{1}}(d) \lhd \left(i_{|2} = j_{|2}\right) \rhd c_{i\xrightarrow{0}}(d)\right) \lhd \left(\lfloor i/5 \rfloor = \lfloor j/5 \rfloor = 1\right) \rhd \emptyset\right) \cdot & \\
\left(\left(c_{i\xrightarrow{1}}(d) \lhd \left((i+1)_{|5} = j_{|5}\right) \rhd \left(c_{i\xrightarrow{0}}(d) \lhd \left((i+4)_{|5} = j_{|5}\right) \rhd c_{i\xrightarrow{2}}(d)\right)\right) \lhd \left(\lfloor i/5 \rfloor < \lfloor j/5 \rfloor\right) \rhd \emptyset\right) \cdot & \\
\left(\left(c_{i\xrightarrow{1}}(d) \lhd \left((i+2)_{|5} = j_{|5}\right) \rhd \left(c_{i\xrightarrow{0}}(d) \lhd \left((i+3)_{|5} = j_{|5}\right) \rhd c_{i\xrightarrow{2}}(d)\right)\right) \lhd \left(\lfloor i/5 \rfloor > \lfloor j/5 \rfloor\right) \rhd \emptyset\right) \cdot \partial_H(V_i) &
\end{aligned}
\tag{12}
$$

As stated above, formal algebraic verification implies comparing the external behavior of the proposed model with that of the real system. Focusing on the former, this is attained by applying the abstraction operator to the formal algebraic specification for the whole model, which is the last equation shown above. The outcome of such operation is denoted in (13), where all internal communications are masked, resulting in an empty set as no action from the aforementioned equation remains unmasked.

$$||_{i=0}^{9}\tau_I(\partial_H(V_i)) = \emptyset \tag{13}$$

With respect to the behavior of the real system X, this is a closed system, as all actions are bounded into it, while none of them are related to anything outside. Hence, it is clear that there is no external behavior regarding a closed system, as all actions take place internally. This may be mathematically described as setting such external behavior as an empty set, as denoted in (14).

$$X = \emptyset \tag{14}$$

Eventually, both external behaviors match, as they both result in $\emptyset$, meaning that they may be considered as rooted branching bisimilar, which is a sufficient condition to verify the model, as depicted in (15).

$$(\partial_H(V_i)) \longleftrightarrow X \tag{15}$$

6. Applying Other Cage Graphs for Small Data Centers

After having shown how to apply the Petersen graph to interconnect nodes in a data center with 10 nodes, we propose a couple of alternative cage graph designs: the Heawood graph and the Robertson graph. The motivation behind them is the same as the one described for Petersen graphs; hence, the focus in this section is on the algorithms and the formal algebraic descriptions.

6.1. Heawood Graph

Regarding the node distribution in a Heawood graph, also known as $(3, 6)$-cage graph, each node has just 3 links to its peers. Its overall shape is that of a tetradecagon, which is a polygon with 14 nodes and 14 edges; hence, each node has one link pointing out to its predecessor and another one to its successor. This context allows the sequential enumeration of all nodes from 0 to 13, such that if a node i is taken, then its predecessor link is to the node $(i-1)_{|14}$, whereas its successor link is to the node $(i+1)_{|14}$.

Additionally, the third link alternatingly points to the node located five positions clockwise or counterclockwise in a sequential manner, which is why it is referred to as a remote port. Therefore, the expression for the third link is $(i + 5 \cdot (-1)^i)_{|14}$, in a way that such a link in even nodes is forward, while in odd nodes, it is backward.

Figure 5 depicts the node distribution of a Heawood graph, where all the aforesaid considerations are shown. Moreover, Figure 6 exhibits the destination of the three ports described in each node.

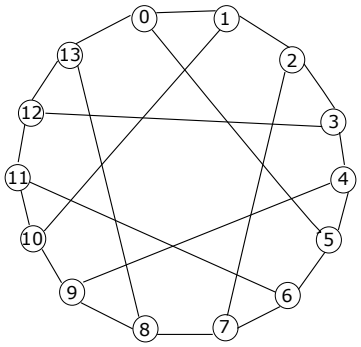

Figure 5. Node distribution of a Heawood graph.

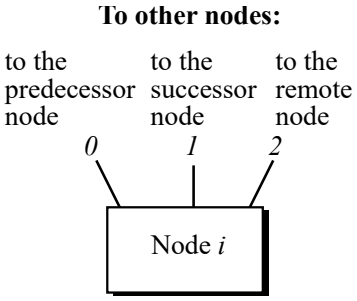

Figure 6. Heawood graph node with ports going to their neighboring nodes.

An important feature of the Heawood graph is that the maximum distance between any pair of nodes is just three hops away. To start, the distance from a particular source node i to a given destination node j is measured and further adjusted to remain within the interval from -6 to 7. Afterward, according to the adjusted distance and depending whether node i is even or odd, there are five scenarios to be studied on an individual basis, as shown in Algorithm 2.

Algorithm 2: Moving from node i to node j in a Heawood graph.

$dist = j - i$
if $dist > 7$ **then**
  $dist = -(14 - dist)$
else if $dist < -6$ **then**
  $dist = -(-14 - dist)$
end if
if $dist \in \{+1, +2, +3\}$ **then**
  send data through switch $i, port\ 1$
else if $dist \in \{-1, -2, -3\}$ **then**
  send data through switch $i, port\ 0$
else if $i\ mod\ 2 = 0$ **then**
  if $dist \in \{+4, +5, +6, +7\}$ **then**
    send data through switch $i, port\ 2$
  else if $dist \in \{-4, -5\}$ **then**
    send data through switch $i, port\ 1$
  else if $dist = -6$ **then**
    send data through switch $i, port\ 0$
  end if
else if $i\ mod\ 2 = 1$ **then**
  if $dist \in \{-4, -5, -6, +7\}$ **then**
    send data through switch $i, port\ 2$
  else if $dist \in \{+4, +5\}$ **then**
    send data through switch $i, port\ 0$
  else if $dist = +6$ **then**
    send data through switch $i, port\ 1$
  end if
end if

In order to obtain the formal algebraic model of a node in ACP, we must translate this algorithm into algebraic form by following the procedure shown for the ACP expression for the Petersen graph. In this way, (17) shows the formal algebraic model of a Heawood node V_i, where an auxiliary function (16) is previously defined to maintain the value of the adjusted distance between source node i and destination node j, whereas V_i uses that function in order to discriminate the five possible cases considered.

$$\begin{aligned} Distance \quad &= \Big(dist = j - i\Big) \cdot \\ &\Big(\Big(dist = -(14 - dist)\Big) \triangleleft\ dist > 7 \triangleright \Big(dist = -(-14 - dist) \triangleleft\ dist < -6 \triangleright \emptyset\Big)\Big) \end{aligned} \tag{16}$$

$$\begin{aligned} V_i \quad &= ||_{p=0}^{2}\Big(1 + r_{x \xrightarrow{p}}(d)\Big) \cdot \\ &\Big(s_{i \xrightarrow{1}}(d) \triangleleft\ Distance \in \{+1, +2, +3\} \triangleright \emptyset\Big) \cdot \\ &\Big(s_{i \xrightarrow{0}}(d) \triangleleft\ Distance \in \{-1, -2, -3\} \triangleright \emptyset\Big) \cdot \\ &\Big(\Big(s_{i \xrightarrow{2}}(d) \triangleleft\ Distance \in \{+4, +5, +6, +7\} \triangleright \Big(s_{i \xrightarrow{1}}(d) \triangleleft\ Distance \in \{-4, -5\} \triangleright s_{i \xrightarrow{0}}(d)\Big)\Big) \\ &\triangleleft\ i\ mod\ 2 = 0 \triangleright \emptyset\Big) \cdot \\ &\Big(\Big(s_{i \xrightarrow{2}}(d) \triangleleft\ Distance \in \{-4, -5, -6, +7\} \triangleright \Big(s_{i \xrightarrow{0}}(d) \triangleleft\ Distance \in \{+4, +5\} \triangleright s_{i \xrightarrow{1}}(d)\Big)\Big) \\ &\triangleleft\ i\ mod\ 2 = 1 \triangleright \emptyset\Big) \cdot V_i \end{aligned} \tag{17}$$

At this stage, the sequence of events happening when all nodes part of the topology are running in a concurrent manner is exhibited in (18).

$$\begin{aligned} ||_{i=0}^{13}\partial_H(V_i) \quad &= ||_{i=0}^{13}||_{p=0}^{2}\Big(c_{i \xrightarrow{1}}(d) \triangleleft\ Distance \in \{+1, +2, +3\} \triangleright \emptyset\Big) \cdot \\ &\Big(c_{i \xrightarrow{0}}(d) \triangleleft\ Distance \in \{-1, -2, -3\} \triangleright \emptyset\Big) \cdot \\ &\Big(\Big(s_{i \xrightarrow{2}}(d) \triangleleft\ Distance \in \{+4, +5, +6, +7\} \triangleright \Big(s_{i \xrightarrow{1}}(d) \triangleleft\ Distance \in \{-4, -5\} \triangleright s_{i \xrightarrow{0}}(d)\Big)\Big) \triangleleft\ i\ mod\ 2 = 0 \triangleright \emptyset\Big) \cdot \\ &\Big(\Big(s_{i \xrightarrow{2}}(d) \triangleleft\ Distance \in \{-4, -5, -6, +7\} \triangleright \Big(s_{i \xrightarrow{0}}(d) \triangleleft\ Distance \in \{+4, +5\} \triangleright s_{i \xrightarrow{1}}(d)\Big)\Big) \triangleleft\ i\ mod\ 2 = 1 \triangleright \emptyset\Big) \cdot \partial_H(V_i) \end{aligned} \tag{18}$$

At this point, the system modelled is closed; hence, its external behavior is equivalent to $\emptyset$, as shown in the Petersen graph. Therefore, the procedure described for this case applies herein to verify the model.

6.2. Robertson Graph

A Robertson graph, also known as a $(4, 5)$-cage graph, is composed of 19 nodes in the shape of a nonadecagon, which is a polygon with 19 nodes and 19 edges. Each node has four links to its peers, making it possible to reach any of them within a three-hop distance. Figure 7 depicts the node distribution of a Robertson graph, where all those considerations are shown. Figure 8 exhibits the destination of the four ports described in each node.

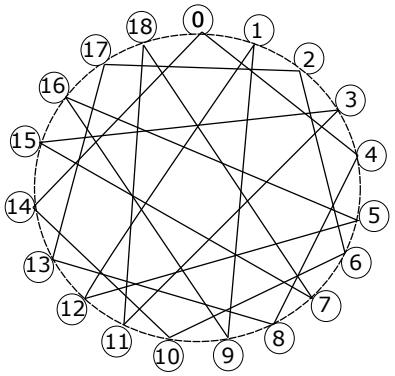

Figure 7. Node distribution of a Robertson graph.

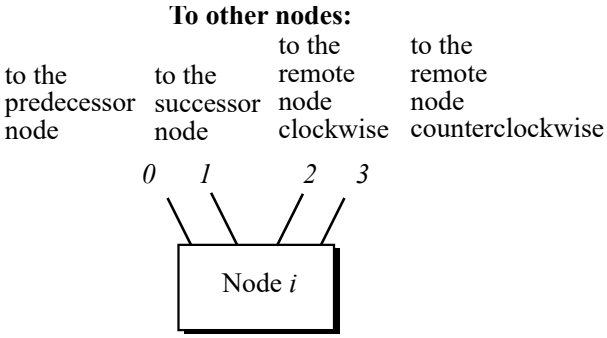

Figure 8. Robertson graph node with ports linking to their neighboring nodes.

Given a node i, one of the links must be with its predecessor, meaning node $(i-1)_{|19}$, and another one must be with its successor, meaning node $(i+1)_{|19}$. Regarding the remaining two links, one is with a remote node clockwise, whereas the other one moves to another remote node counterclockwise. Unfortunately, a regular pattern is not found to reach the destinations of both remote links no matter which node i is taken, although only some combinations are repeated in all cases, as shown in Table 6.

Table 6. Link destination for the nodes within a Robertson graph.

Node i	Predecessor Link	Successor Link	Remote Clockwise Link	Remote Counterclockwise Link
0	$18 \to (i-1)_{\|19}$	$1 \to (i+1)_{\|19}$	$4 \to (i+4)_{\|19}$	$14 \to (i-5)_{\|19}$
1	$0 \to (i-1)_{\|19}$	$2 \to (i+1)_{\|19}$	$9 \to (i+8)_{\|19}$	$12 \to (i-8)_{\|19}$
2	$1 \to (i-1)_{\|19}$	$3 \to (i+1)_{\|19}$	$6 \to (i+4)_{\|19}$	$17 \to (i-4)_{\|19}$
3	$2 \to (i-1)_{\|19}$	$4 \to (i+1)_{\|19}$	$11 \to (i+8)_{\|19}$	$15 \to (i-7)_{\|19}$
4	$3 \to (i-1)_{\|19}$	$5 \to (i+1)_{\|19}$	$8 \to (i+4)_{\|19}$	$0 \to (i-4)_{\|19}$
5	$4 \to (i-1)_{\|19}$	$6 \to (i+1)_{\|19}$	$12 \to (i+7)_{\|19}$	$16 \to (i-8)_{\|19}$
6	$5 \to (i-1)_{\|19}$	$7 \to (i+1)_{\|19}$	$10 \to (i+4)_{\|19}$	$2 \to (i-4)_{\|19}$
7	$6 \to (i-1)_{\|19}$	$8 \to (i+1)_{\|19}$	$15 \to (i+8)_{\|19}$	$18 \to (i-8)_{\|19}$
8	$7 \to (i-1)_{\|19}$	$9 \to (i+1)_{\|19}$	$13 \to (i+5)_{\|19}$	$4 \to (i-4)_{\|19}$
9	$8 \to (i-1)_{\|19}$	$10 \to (i+1)_{\|19}$	$16 \to (i+7)_{\|19}$	$1 \to (i-8)_{\|19}$
10	$9 \to (i-1)_{\|19}$	$11 \to (i+1)_{\|19}$	$14 \to (i+4)_{\|19}$	$6 \to (i-4)_{\|19}$
11	$10 \to (i-1)_{\|19}$	$12 \to (i+1)_{\|19}$	$18 \to (i+7)_{\|19}$	$3 \to (i-8)_{\|19}$
12	$11 \to (i-1)_{\|19}$	$13 \to (i+1)_{\|19}$	$1 \to (i+8)_{\|19}$	$5 \to (i-7)_{\|19}$
13	$12 \to (i-1)_{\|19}$	$14 \to (i+1)_{\|19}$	$17 \to (i+4)_{\|19}$	$8 \to (i-5)_{\|19}$
14	$13 \to (i-1)_{\|19}$	$15 \to (i+1)_{\|19}$	$0 \to (i+5)_{\|19}$	$10 \to (i-4)_{\|19}$
15	$14 \to (i-1)_{\|19}$	$16 \to (i+1)_{\|19}$	$3 \to (i+7)_{\|19}$	$7 \to (i-8)_{\|19}$
16	$15 \to (i-1)_{\|19}$	$17 \to (i+1)_{\|19}$	$5 \to (i+8)_{\|19}$	$9 \to (i-7)_{\|19}$
17	$16 \to (i-1)_{\|19}$	$18 \to (i+1)_{\|19}$	$2 \to (i+4)_{\|19}$	$13 \to (i-4)_{\|19}$
18	$17 \to (i-1)_{\|19}$	$0 \to (i+1)_{\|19}$	$7 \to (i+8)_{\|19}$	$11 \to (i-7)_{\|19}$

Nodes 0 and 13 share the same movements for clockwise and counterclockwise links, being +4 and -5, respectively. In this sense, nodes 1 and 7 share the movements +8 and -8, whereas nodes 8 and 14 share +5 and -4. Nodes 3, 12, 16, and 18 share movements +8 and -7; nodes 2, 4, 6, 10, and 17 share +4 and -4, which are the most repeated. However, the movements through the remote links of the neighbors of different nodes within a group do not match, meaning that each node needs to be separately assessed in order to design an algorithm for the Robertson graph. Within each node, the different movements to reach all the other nodes need to be shown, in a fashion similar to that in Algorithm 2.

As an example, if source node $i = 0$, then three hops on the successor link lead to nodes 1, 2, and 3, whereas three hops on the predecessor link lead to nodes 18, 17, and 16. Additionally, taking the remote clockwise link leads to node 4, which also leads to nodes 5 and 6 on its successor link. Moreover, taking the remote counterclockwise link

brings us to node 14, which also leads to 15 on its successor link, or to nodes 13 and 12 on its predecessor link. Furthermore, from neighbor node 1, it is possible to reach node 9 on its remote clockwise link, which also leads to nodes 8 and 10 or to node 12 on its counterclockwise link, which also leads to node 11. Finally from neighbor node 18, it is possible to reach node 7 through its clockwise link. Hence, all nodes are reached from node 0 within just three hops.

7. Experimental Analysis

This section is split into two subsections: the first is devoted to proposing an alternative manner of packet forwarding within data centers; the second is dedicated to an experimental analysis to compare the cage graph topologies proposed herein with some commonly used topologies in small data centers.

7.1. Alternative Method of Packet Forwarding

Network traffic is processed at either layer 2 or 3 of the OSI model, where the former is traditionally made by switches. The latter is typically achieved by routers. For the former, a switch composes its own MAC address table on the fly by inspecting the source MAC addresses of the frames coming in through any of their ports and by either adding a new register if such an address is not found in its own table or otherwise resetting the timer of that register within its own table if it does exist. For the latter, a router fills in its own routing table by reflecting either its directly connected routes or the routes statically configured by a network administrator; otherwise, the table is filled by receiving routing updates from other layer-3 devices being part of the same routing domain upon the same routing protocol.

When a switch receives a frame, it reads its destination MAC address and searches it through the registers within its own forwarding table. As MAC addresses contain 48 bits, all of them need to match. Comparing a bit requires a bitwise AND logical operation; hence, 48 AND gates are necessary to obtain a match. There are diverse methods to perform this, such that a matching operation halts when any of the bits do not match, thus avoiding the repeated comparison of all bits, although the number of such operations within a matching lookup depends on the number of entries within the MAC address table and how quick a matching entry is found.

When a router receives a packet, it reads its destination IP address and tries to accommodate it within any of the network addresses registered within the routing table. IPv4 addresses contain 32 bits, although the length of a network address depends on its subnet mask, with a most common value of 24 bits. As shown above, a bitwise AND logical operation is needed to compare a bit, meaning an AND gate for each of those comparisons, whose number depends on the amount of entries installed in the routing table and how fast a matching entry is found.

An alternative method of working with routing and forwarding tables is the use of arithmetic operations in order to define the next hop toward a destination. Focusing on the proposed algorithms for small cage graphs, integer divisions and modular arithmetic are used, as well as assessing determined small values for node identifiers and distance.

The aforementioned operations are usually implemented by a specific integrated circuit, which contains a number of logical gates. Additionally, the values to be operated are small; thus, the amount of bits is very low, thus reducing the overall number of necessary logical gates.

Regarding performance, the hardware implementation makes a difference, as traditional routing and forwarding tables implement the matching operation through AND logical gates, which is supposed to be faster. However, the alternative proposed method based on the designed algorithms may be faster when it comes to small node identifiers, such as the ones used in the proposed topologies. For instance, Ref. [40] exhibits a layout with low power consumption when a small number of bits is involved. Conversely, Ref. [41]

depicts a scheme to attain efficient lookup tables. Therefore, the hardware implementation and the design are key to establishing the most convenient method of packet forwarding.

7.2. Comparing Experimental Analysis with Other Common Topologies

In order to experimentally analyze the proposed cage graphs, some common statistic measurements were made, referring to both central tendency and dispersion, as well as the appropriate coefficient of variation. These statistics refer to the number of hops from a given node within a topology to any other one, along with the number of links per node in each topology. In this way, the former may be associated with to the performance of the system: the lower the number of hops to reach any other peer, the better. Otherwise, the latter may be assigned to ease of operation and maintenance of the topology: the lower the amount, the simpler the interconnections and packet forwarding.

Ten topologies were selected for this statistic analysis, where full mesh was the ideal one in theory, although it does not scale well in practice. Moreover, two instances of N hypercubes were taken, such as those where $N = \{3, 4\}$, along with their two folded N-hypercube counterparts. Additionally, two instances of k-ary n cube were chosen: 3-ary 2-cube and 4-ary 2-cube. Eventually, the three instances of small cage graphs studied herein were also considered: Petersen, Heawood, and Robertson graphs.

Those commonly used topologies provide a suitable representation of simple architectures for small data centers. To start, Table 7 presents the average number of hops to reach any other node and the average number of links per node. The average number of hops away, also known as the arithmetic mean values, was calculated by summing the multiplication of the different numbers of hops from one node to the others by the number of nodes at that distance, all divided by the number of nodes to reach overall, where the reference node is not taken into consideration.

The average numbers of links per node were easily obtained, as the topologies studied herein are all graph-like designs, where all nodes have the same number of links; hence, the average is just that number with no dispersion.

Table 7. Average number of hops and average number of links per node.

Topology	Average Number of Hops	Average Number of Links Per Node
Full mesh (8 nodes)	$(1{\cdot}7)/7 = 1$	7
3-hypercube (8 nodes)	$(1{\cdot}3{+}2{\cdot}3{+}3{\cdot}1)/7 = 1.71$	3
4-hypercube (16 nodes)	$(1{\cdot}4{+}2{\cdot}6{+}3{\cdot}4{+}4{\cdot}1)/15 = 2.13$	4
Folded 3-hypercube (8 nodes)	$(1{\cdot}4{+}2{\cdot}3)/7 = 1.43$	4
Folded 4-hypercube (16 nodes)	$(1{\cdot}5{+}2{\cdot}10)/15 = 1.67$	5
3-ary 2-cube (9 nodes)	$(1{\cdot}4{+}2{\cdot}4)/8 = 1.50$	4
4-ary 2-cube (16 nodes)	$(1{\cdot}4{+}2{\cdot}6{+}3{\cdot}5)/15 = 2.13$	4
Petersen graph (10 nodes)	$(1{\cdot}3{+}2{\cdot}6)/9 = 1.67$	3
Heawood graph (14 nodes)	$(1{\cdot}3{+}2{\cdot}6{+}3{\cdot}4)/13 = 2.07$	3
Robertson graph (19 nodes)	$(1{\cdot}4{+}2{\cdot}12{+}3{\cdot}2)/18 = 1.89$	4

Focusing on the average number of hops, other statistic measurements are listed in Table 8. The mode and median were calculated regarding central tendency measurements, whereas variance and standard deviation represent dispersion measurements.

We then calculated the coefficient of variation for all those topologies, which is the standard deviation divided by the arithmetic mean. It results in a dimensionless value between 0 and 1, where values above 0.30 are not considered to be homogeneous, meaning that the arithmetic mean is not representative of the data set [42].

Table 8. Statistics related to the average number of hops in each network topology.

Topology	Avg. Hops	Mode	Median	Variance	Std. Dev.	Coefficient of Variation
Full mesh	1	1	1	0	0	∞
3-hypercube	1.71	1.5	2	0.49	0.70	0.41
4-hypercube	2.13	2	2	0.78	0.88	0.41
Folded 3-hyp.	1.43	1	1	0.24	0.49	0.35
Folded 4-hyp.	1.67	2	2	0.22	0.47	0.28
3-ary 2-cube	1.50	1.50	1.50	0.25	0.50	0.33
4-ary 2-cube	2.13	2	2	0.60	0.77	0.36
Petersen gr.	1.67	2	2	0.22	0.47	0.28
Heawood gr.	2.07	2	2	0.53	0.73	0.35
Robertson gr.	1.89	2	2	0.32	0.57	0.30

Discussion

From the point of view of the average number of links per node, the most repeated value is four links per item. Thus, higher values result in more complex topologies from the point of view of operation and maintenance, even though performance improves, which is the case of folded four-hypercube compared with their counterparts of 16 nodes. The case of full mesh beats all of the rest performance-wise.

Some topologies present three links per node, such as the three-hypercube, along with the Petersen and Heawood graph designs. Those cage graphs have more nodes than a three-hypercube, which does not happen with the other topologies selected. Hence, using cage graphs provides simpler topologies from the point of view of operation and maintenance compared with other topologies with similar numbers of nodes, resulting in less links in the overall design.

Focusing on the average number of hops away, for the instances representing the same shape (N-hypercubes, folded N-hypercubes, k-ary N-cube, or cage graphs), this value is lower for the instance with fewer nodes. However, cage graphs provide better results for the same number of nodes and average links per node.

The coefficient of variation is lower for Petersen graph and folded four-hypercube, whose values are considered as acceptable as being under 0.30, while the values for the rest of topologies are slightly above this value. Considering all parameters, such as average number of hops, average number of links per node, and the coefficient of variation, the Petersen graph may be an interesting solution to balance performance and simplicity of operation and maintenance. The Heawood and Robertson graphs are good solutions if some more nodes are needed in a data center.

8. Conclusions

In this study, a data center organization and optimization technique was developed for edge or fog environments, whose main feature is having two or three hops between a small number of physical hosts involved, as a given number of virtual hosts may be allocated to a physical host.

This paper started with Introduction and Motivation sections, followed by an overview of graph theory, where we described the main types of graphs. After that, the concept of integrity of a graph was introduced as a measure of its reliability, which can be linked to the concept of network stability, where integrity values for some common type of graphs were provided.

Afterward, cage graphs were presented as designs with the particularity that all nodes have the same values of regularity and girth, where the number of nodes for each available instance may have both a lower bound and an upper bound. In this sense, all known three-regular cage graphs were introduced, whose girth values ranged from 5 to 12. The most well-known r-regular cage graphs with $r > 3$ were also described, while their integrity values for topologies whose order grows up to 60 were listed.

After that, the smallest cage graph (excluding complete graphs and complete bipartite graphs), namely the Petersen graph, also known as $(3,5)$-cage graph,was proposed as a topology for data centers with 10 nodes, leading to a diameter of 2, which creates a really fast interconnection network among hosts. Hence, a scheme for linking together those 10 nodes was described, resulting in four case scenarios depending on the location of the source and destination nodes, which were denoted in both algorithmic and algebraic methods.

Additionally, two other small cage graphs were studied: the Heawood graph, also known as the $(3,6)$-cage graph, and the Robertson graph, also known as the $(4,5)$-cage graph. The former is composed of 14 nodes, whereas the latter is composed of 19 nodes, although both have a diameter of 3. The former was described in an algorithmic and an algebraic manner, while only the algorithmic form of the latter was outlined. All three cage graphs proposed herein present a coefficient of variation among the lowest compared with those of other commonly used network topologies in data centers.

In summary, the forwarding scheme proposed for a Petersen graph topology improves the performance compared with that of the traditional routing tables and MAC-address tables, as forwarding is achieved by integer divisions and modular arithmetic, as opposed to searching through the aforesaid tables for a matching entry, which requires more computation resources.

Author Contributions: Conceptualization, P.J.R.; formal analysis, P.J.R.; supervision, P.J.R., S.A., K.G., C.B. and C.J.; validation, P.J.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACP	Algebra of Communicating Processes
AI	Artificial Intelligence
CNN	Convolutional Neural Networks
DL	Deep Learning
DS	Data Science
FDT	Formal Description Techniques
IoT	Internet of Things
LAN	Local Area Network
MEC	Multiaccess Edge Computing
ML	Machine Learning
WAN	Wide Area Network

References

1. Sarker, I.H. AI-Based Modeling: Techniques, Applications and Research Issues Towards Automation, Intelligent and Smart Systems. *SN Comput. Sci.* **2022**, *3*, 158. [CrossRef]
2. Saha, S.; Sarkar, J.; Dwivedi, A.; Dwivedi, N.; Narasimhamurthy, A.M.; Roy, R. A novel revenue optimization model to address the operation and maintenance cost of a data center. *J. Cloud Comput.* **2016**, *5*, 1. [CrossRef]
3. Prabhu, C.S.R. Overview - Fog Computing and Internet-of-Things (IOT). *EAI Endorsed Trans. Cloud Syst.* **2017**, *3*, 154378. [CrossRef]
4. Daid, R.; Kumar, Y.; Hu, Y.C.; Chen, W.L. An effective scheduling in data centres for efficient CPU usage and service level agreement fulfillment using machine learning. *Connect. Sci.* **2021**, *33*, 954–974. [CrossRef]
5. Yang, Z.; Du, J.; Lin, Y.; Du, Z.; Xia, L.; Zhao, Q.; Guan, X. Increasing the energy efficiency of a data center based on machine learning. *J. Ind. Ecol.* **2022**, *26*, 323–335. [CrossRef]
6. Grealey, J.; Lannelongue, L.; Saw, W.Y.; Marten, J.; Méric, G.; Ruiz-Carmona, S.; Inouye, M. The Carbon Footprint of Bioinformatics. *Mol. Biol. Evol.* **2022**, *39*, 1–15. [CrossRef] [PubMed]
7. Sovacool, B.; Monyei, C.G.; Upham, P. Making the internet globally sustainable: Technical and policy options for improved energy management, governance and community acceptance of Nordic datacenters. *Renew. Sustain. Energy Rev.* **2022**, *154*, 111793. [CrossRef]

8. Al Kez, D.; Foley, A.M.; Ahmed, F.W.; O'Malley, M.; Muyeen, S.M. Potential of data centers for fast frequency response services in synchronously isolated power systems. *Renew. Sustain. Energy Rev.* **2021**, *151*, 111547. [CrossRef]
9. Andrews, D.; Newton, E.J.; Adibi, N.; Chenadec, J.; Bienge, K. A Circular Economy for the Data Centre Industry: Using Design Methods to Address the Challenge of Whole System Sustainability in a Unique Industrial Sector. *Sustainability* **2021**, *13*, 6319. [CrossRef]
10. Hashemi, M.; Javaheri, D.; Sabbagh, P.; Arandian, B.; Abnoosian, K. A multi-objective method for virtual machines allocation in cloud data centres using an improved grey wolf optimization algorithm. *IET Commun.* **2021**, *15*, 2342–2353. [CrossRef]
11. Ullah, A.; Nawi, N.M.; Ouhame, S. Recent advancement in VM task allocation system for cloud computing: Review from 2015 to 2021. *Artif. Intell. Rev.* **2022**, *55*, 2529–2573. [CrossRef] [PubMed]
12. Mukherjee, D.; Ghosh, S.; Pal, S.; Akila, D.; Jhanjhi, N.Z.; Masud, M.; AlZain, M.A. Optimized Energy Efficient Strategy for Data Reduction Between Edge Devices in Cloud-IoT. *Comput. Mater. Contin.* **2022**, *72*, 125–140. [CrossRef]
13. Dhaya, R.; Ujwal, U.J.; Sharma, T.; Singh, P.; Kanthavel, R.; Selvan, S.; Krah, D. Energy-Efficient Resource Allocation and Migration in Private Cloud Data Centre. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 3174716.
14. Shi, F.; Lin, J. Virtual MAChine Resource Allocation Optimization in Cloud Computing Based on Multiobjective Genetic Algorithm. *Comput. Intell. Neurosci.* **2022**, *1*, 7873131. [CrossRef]
15. What Are Data Centers? How They Work and How They Are Changing in Size and Scope. Available online: https://www.networkworld.com/article/3599213/what-are-data-centers-how-they-work-and-how-they-are-changing-in-size-and-scope.html/ (accessed on 28 December 2022).
16. 8 Tips to Optimize Your Data Center's HVAC and Energy Use for 2022. Available online: https://galooli.com/blog/top-8-tips-to-optimize-your-data-centers-hvac-and-energy-use-for-2022/ (accessed on 28 December 2022).
17. Data Center Optimization Strategies. Available online: https://www.akcp.com/blog/data-center-optimization-strategies/ (accessed on 28 December 2022).
18. Best Practices for Data Center Network Optimization. Available online: https://www.techtarget.com/searchdatacenter/tip/Best-practices-for-data-center-network-optimization/ (accessed on 28 December 2022).
19. Román, R.; López, J.; Mambo, M. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. *Future Gener. Comput. Syst.* **2018**, *78*, 680–698. [CrossRef]
20. Mishra, S.; Puthal, D.; Sahoo, B.; Sharma, S.; Xue, Z.; Zomaya, A. Energy-Efficient Deployment of Edge Dataenters for Mobile Clouds in Sustainable IoT. *IEEE Access* **2018**, *6*, 56587–56597. [CrossRef]
21. Harary, F. *Graph Theory*; CRC Press: Boca Raton, FL, USA, 2019; pp. 1–288. ISBN 978-0-4294-9376-8.
22. Basavanagoud, B.; Policepatil, S.; Jakkannavar, P. Integrity of Graph Operations. *Trans. Comb.* **2021**, *10*, 171–185.
23. Dunkum, M.; Laphier, D. Edge integrity of nearest neighbor graphs and separator theorems. *Discret. Math.* **2019**, *342*, 2664–2679. [CrossRef]
24. Mahde, S.S.; Mathad, V., Cangul, I. Accessibility Integrity of Graphs. *Adv. Stud. Contemp. Math. (Kyungshang)* **2021**, *31*, 33–47.
25. Drange, P.G.; Dregi, M.S., van't Hof, P. On the Computational Complexity of Vertex Integrity and Component Order Connectivity. *Algorithmica* **2016**, *76*, 1181–1202. [CrossRef]
26. Saravannan, M.; Sujatha, R.; Sundareswaran, R.; Sahoo, S.; Pal, M. Concept of integrity and its value of fuzzy graphs. *J. Intell. Fuzzy Syst.* **2018**, *34*, 2429–2439. [CrossRef]
27. Basavanagoud, B.; Policepatil, S. Integrity of Wheel Related Graphs. *Punjab Univ. J. Math.* **2021**, *53*, 329–338. [CrossRef]
28. Mahde, S.S.; Mathad, V. Hub edge integrity of graphs. *Online J. Anal. Comb.* **2020**, *15*, 1–10.
29. Mahde, S.S.; Sibih, A.M.; Mathad, V. D Integrity and E Integrity Numbers in Graphs. *Appl. Math. Inf. Sci.* **2020**, *14*, 453–458.
30. Diudea, M.V.; Rosenfeld, V.R. The truncation of a cage graph. *J. Math. Chem.* **2017**, *55*, 1014–1020. [CrossRef]
31. Bretto, A.; Faisant, A.; Gillibert, L. New graphs related to (p,6) and (p,8)-cages. *Comput. Math. Appl.* **2011**, *62*, 2472–2479. [CrossRef]
32. Pisanski, T.; Servatius, B. Graphs. In *Configurations from a Graphical Viewpoint*; Birkhäuser Advanced Texts Basler Lehrbücher; Birkhäuser: Boston, MA, USA, 2013; pp. 15–53. ISBN: 978-0-8176-8363-1.
33. Araujo-Pardo, G.; Balbuena, C.; Héger, T. Finding small regular graphs of girth 6, 8 and 12 as subgraphs of cages. *Discret. Math.* **2010**, *310*, 1301–1306. [CrossRef]
34. Gao, S.; Knoll, F.; Manganiello, F.; Matthews, G. Codes for distributed storage from 3-regular graphs. *Discret. Appl. Math.* **2017**, *229*, 82–89. [CrossRef]
35. Atici, M.; Crawford, R.; Ernst, C. The integrity of small cage graphs. *Australas. J. Comb.* **2009**, *43*, 39–55.
36. Fokkink, W. *Modelling Distributed Systems*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2017; pp. 1–174. ISBN: 978-3-540-73938-8.
37. Groote, J.F.; Mousavi, M.R. *Modeling and Analysis of Communicating Systems*, 1st ed.; MIT Press: Cambridge, MA, USA, 2014; pp. 1–392. ISBN: 978-0-262-02771-7.
38. Bergstra, J.A.; Middleburg, C.A. Using Hoare Logic in a Process Algebra Setting. *Fundam. Informaticae* **2021**, *179*, 321–344. [CrossRef]
39. Fokkink, W. *Introduction to Process Algebra*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 1–151. ISBN: 978-3-662-04293-9.
40. Erra, R. Implementation of a Hardware Algorithm for Integer Division, Master's Thesis, Electrical Engineering, Faculty Graduate College, Oklahoma State University, Stillwater, OK, USA, 2019.

41. Ahn, Y., Lee, Y., Lee, G. Power and Time Efficient IP Lookup Table Design Using Partitioned TCAMs. *Circuits Syst.* **2013**, *4*, 299–303. [CrossRef]
42. Brown, C.E. Coefficient of Variation. In *Applied Multivariate Statistics in Geohydrology and Related Sciences*; Springer: Berlin/Heidelberg, Germany, 1998.