International Journal of Cloud Applications and Computing
Volume 14 • Issue 1

A Distributed Cloud Architecture Based on General De Bruijn Overlay Network

Osama R. S. Ramadan, Al Azhar University, Egypt*
https://orcid.org/0009-0008-3326-3372

Mohamed Yasin I. Afifi, Al Azhar University, Egypt
https://orcid.org/0000-0003-0012-2326

Ahmed Yahya, Al Azhar University, Egypt

ABSTRACT

Distributed cloud systems enable the distribution of computing resources across various geographical locations. While offering benefits like accelerated content delivery, the scalability and coherence maintenance of these systems pose significant challenges. Recent studies reveal shortcomings in existing distributed system schemes to meet modern cloud application demands and maintain coherence among heterogeneous system elements. This paper proposes a service-oriented network architecture for distributed cloud computing networks. Using a De Bruijn network as a software-defined overlay network, the architecture ensures scalability and coherence. Through service-based addressing, requests are issued to designated service address bands, streamlining service discovery. The architecture's evaluation through extensive simulations showcases sustainable scalability and inherent load-balancing properties. The paper concludes with insights into future research directions, emphasizing the extension of the proposed architecture to emerging distributed cloud use cases and decentralized security.

KEYWORDS

Cloud Computing, Distributed Cloud, Distributed Load Balance, Overlay Networks

Distributed cloud computing systems are a type of cloud computing infrastructure that allows for the distribution of computing resources across multiple geographical locations. This architecture provides many benefits, including faster content delivery, greater visibility and manageability of hybrid cloud and multicloud architectures, and easier industry or regional regulatory compliance (Atieh, 2021). However, as the size and complexity of these systems grow, coordinating and maintaining consistency among the distributed resources become increasingly challenging. (Coady et al., 2015; Ageed & Zeebaree, 2024). In a recent cloud schemes survey Angel et al. (2021) found that major distributed system schemes in practice fail to scale up to modern cloud application demands and to maintain coherency between the heterogeneous elements of the system.

DOI: 10.4018/IJCAC.339892 *Corresponding Author

To address these challenges, we propose a new service-oriented network architecture for distributed cloud computing networks. The proposed architecture aims to provide a scalable, robust solution to face changing cloud conditions. The proposed architecture uses a de Bruijn (Chikhi et al., 2014) network as a software-defined overlay network built on top of a physical core network. The chosen overlay network is a software-defined network that can be set up on top of any weakly connected physical network and is guaranteed to self-stabilize (Feldmann & Scheideler, 2017). This set of features is crucial for ensuring network construction feasibility and adaptability to changes in the cloud environment.

The proposed architecture uses service-based addressing instead of node-based addressing; that is, the consumer nodes issue a service request to a service address band, unlike a regular cloud network architecture in which the service is requested from a known computing node. The network address space is divided into bands, and each service is allocated a specific band. This requires the consumer node to have knowledge only about the desired service band.

To evaluate the effectiveness of the proposed architecture, we created and tested a network model in multiple scenarios. The results of the simulations demonstrate the sustainable scalability of the proposed architecture while maintaining no central core of the network and the inherent load-balancing property of the proposed architecture.

Finally, the paper concludes with a discussion of future work directions, including the extension of the proposed architecture to new distributed cloud use cases, such as edge computing and internet of things (IoT) applications.

OVERLAY NETWORKS

An overlay network is a logical network built on top of a physical network that provides an abstract layer with the purpose of overlaying the existing physical network infrastructure. The goal of an overlay network is to allow more flexibility in how data is transmitted and processed, irrespective of the physical network's technical implementation. It uses the existing infrastructure to connect and allow communication between nodes, while adding an additional layer of abstraction to enable advanced routing and network management algorithms. Creating an overlay on top of an existing physical network enables new mechanisms, protocols, and services to be introduced that can enhance the overall performance and functionality of the system.

Overlay networks are commonly used in a variety of systems and applications, including peer-to-peer (P2P) file-sharing systems, content delivery networks (CDNs), virtual private networks (VPNs), and distributed cloud computing platforms. In each of these applications, the overlay network allows for application-optimized routing and network management algorithms, while abstracting out the details and limitations of the underlying physical network.

Overlay networks represent a crucial building block in the design and implementation of scalable, efficient distributed systems. Their ability to overcome the limitations of the underlying physical infrastructure and provide enhanced functionality makes them a fundamental component in modern networking architectures (Lua et al., 2005).

De Bruijn Graph

A de Bruijn graph, symbolized as $G = (V, E)$, is a form of directed graph commonly used in computer networks and bioinformatics. At its core, this graph provides a structured representation of sequential data, often DNA sequences in genomics or symbol sequences in network routing algorithms.

The set of nodes, denoted by V within a de Bruijn graph, encompasses all possible substrings of a predetermined length, typically referred to as k-mers. K-mers are contiguous sequences of symbols in which the length k determines the size of the substring. For instance, in a networking scenario, a 3-mer (k-mer of length 3) could represent a sequence of three consecutive bytes or characters within a packet payload.

Each node in the de Bruijn graph corresponds to a unique k-mer derived from the original sequence of symbols. The edges, denoted by E, represent transitions between adjacent k-mers. Specifically, an edge (ki, kj) exists if and only if the last k-1 symbols of ki match the first k-1 symbols of kj, indicating a sequential relationship between the two k-mers.

The essence of the de Bruijn graph lies in its ability to compactly capture the sequential relationships between k-mers. This graphical representation simplifies the analysis and manipulation of sequences, facilitating tasks such as genome assembly, sequence alignment, and network routing optimization algorithms.

De Bruijn Network

In computer networking, de Bruijn graphs are employed to model network topology, where nodes represent network devices, such as routers or processing nodes, and edges represent the connections or links between these devices. Each node is assigned a hash in the range of]0,1[that is unique to the node. The hash is represented by a finite number of digital bits that are treated as the symbols in the de Bruijn graph. An edge between two nodes (ki, kj) exists if and only if the last k-1 symbols of node i hash's bit representation match the first k-1 node j hash's bit representation.

If the hash representation bits are base 2 binary bits, the graph can be constructed with the standard de Bruijn graph definition. Feldmann and Scheideler (2017) present a general de Bruijn graph that can use q-ary bits in the hash representation. Feldmann and Scheideler (2017) define and prove the general de Bruijn graph and network building protocols (buildList, general de Bruijn, and standard de Bruijn) for equivalence with the standard de Bruijn graph, routing correction, and self-stabilization. The set of edges in the general de Bruijn graph is a union of four edge sets:

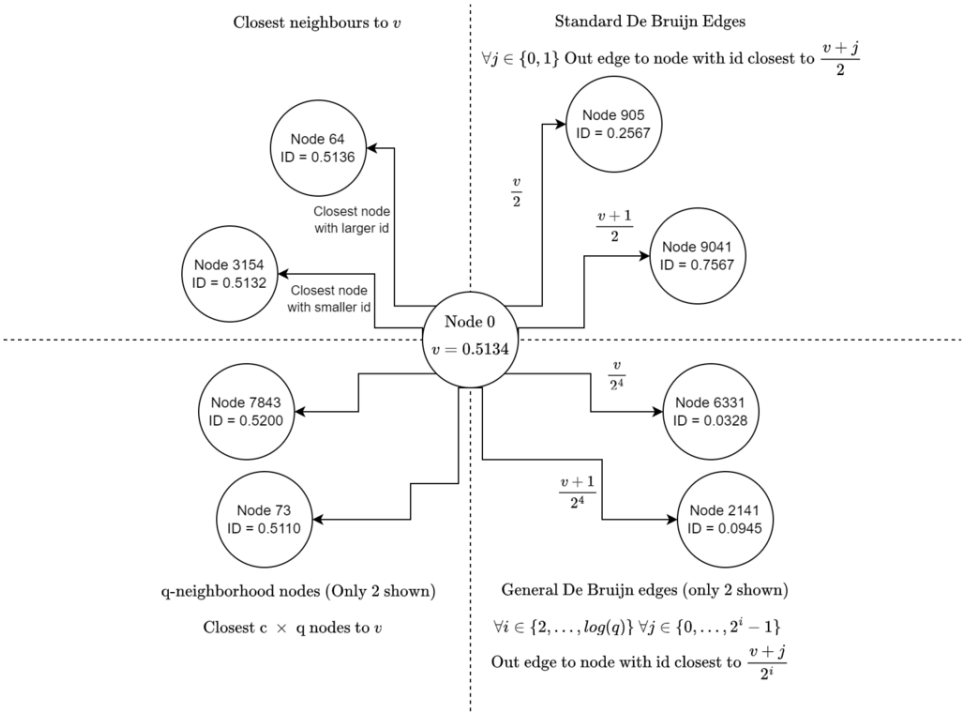
- List edges: These edge sets are outward edges from each node to the closest left and right nodes. They are included to facilitate the self-stabilization process.
- q-neighbor edges: These edge sets are outward edges from each node to the closest q nodes (q as of the base of the q-ary bits in the node hash representation). They are included to facilitate the self-stabilization process.
- Standard de Bruijn edges: These edge sets are outward edges from each node to two nodes where the last d-1 (d is the parameter of the graph) symbols of destination nodes match the first d-1 symbols of the source node. This represents a standard de Bruijn hop.
- General de Bruijn edges: These edge sets are outward edges from each node to nodes where the last d-1 symbols of destination nodes match the first d-1 symbols of the source node. This set is a projection of the standard de Bruijn hop on q dimensions, representing a general de Bruijn hop.

Figure 1 shows an example of a node's outgoing edges, categorized according to the four edge sets listed above. The proposed architecture in this paper is built on top of a general de Bruijn graph, referred to from here onward as de Bruijn graph. The properties of the de Bruijn graph make it particularly useful in constructing decentralized networks where each node doesn't have to store local information about all the other nodes in the system.

Here are some advantages to the results of these properties:

- Scalability: De Bruijn graphs exhibit excellent scalability properties, allowing them to accommodate a large number of nodes efficiently. As the number of nodes increases, the graph's structure remains manageable, enabling seamless expansion without compromising performance (Chikhi et al., 2014).
- Low diameter: The diameter of a de Bruijn graph, which represents the maximum distance between any two nodes in the network, is relatively low. This low diameter facilitates efficient routing and communication between nodes, reducing latency and enhancing overall system performance.

Figure 1. Outward Edges for a Node in a General De Bruijn Graph



- Efficient routing: De Bruijn graphs possess inherent properties that facilitate efficient routing algorithms. Because of the structured nature of the graph, routing decisions can be made deterministically based on the identifiers of the source and destination nodes. This deterministic routing reduces overhead and improves the reliability of message delivery in distributed environments.
- Fault tolerance: De Bruijn graphs are resilient to node failures and network partitions owing to their redundancy and distributed nature. In the event of node failures, the graph can dynamically adapt its routing paths to circumvent failed nodes, ensuring uninterrupted communication and maintaining system availability.
- Self-stabilization: De Bruijn graphs exhibit self-stabilizing properties; they can recover and stabilize from transient faults or inconsistencies without external intervention. This self-stabilization mechanism enhances the robustness and reliability of distributed cloud architectures, particularly in dynamic and unpredictable environments (Feldmann & Scheideler, 2017).
- Constant network overhead: The general de Bruijn graph from Feldmann and Scheideler (2017) has a constant number of messages sent by each node during every graph update cycle to maintain the self-stabilization of the graph.

LITERATURE REVIEW

Overlay networks have been extensively studied in the field of computer networks. Lua et al. (2005) conducted a comprehensive survey and comparison of various peer-to-peer overlay network schemes. They analyzed the characteristics, advantages, and limitations of different overlay network architectures and provided valuable insights into their performance and scalability.

Another notable paper by Ripeanu et al. (2002) discussed the mapping of Gnutella, an unstructured overlay network, and presented an analysis of its topology and performance. They examined the dynamics of the overlay network and highlighted the challenges of achieving efficient search and resource discovery in such decentralized systems.

The structured overlay network architecture has also been widely researched. Stoica et al. (2001) proposed Chord, a distributed lookup protocol for organizing nodes in a structured manner. Their work introduced consistent hashing and presented a decentralized algorithm for efficient node and resource location.

Faizian et al (2016) discussed the advantages of the de Bruijn graph in the context of distributed networks and showed that the de Bruijn graph could handle shortest path routing and a near-optimal performance in terms of diameter and load balancing. These properties make it an appealing choice for various distributed network applications. Faizian et al. (2016) also proposed a hop-limited all-path routing scheme (ALLPATH(H)) that complements the general de Bruijn graph's topology by providing effective routing capabilities, thus ensuring that traffic is efficiently distributed across the network.

Feldmann and Scheideler (2017) discussed self-stabilizing algorithms and explored the practical application of de Bruijn graphs in the domain of network topology. Their paper presented an approach to constructing a self-stabilizing version of a general de Bruijn graph that can be used as a basis for a computer network. In this context, the notion of self-stabilization was introduced as a key point of the paper. This property encompasses the ability of a system to recover from any initial state and subsequently converge toward correct operational behavior autonomously, without requiring any external intervention. Feldmann and Scheideler (2017) recognized the importance of this characteristic in the context of large-scale networks, where the occurrence of node failures or dynamic reconfigurations can severely disrupt normal network operation. By devising a self-stabilizing general de Bruijn graph construction algorithm, Feldmann and Scheideler (2017) tackled the challenge of preserving network connectivity and consistency, even in the face of disruptive faults and dynamic runtime events.

Feldmann and Scheideler (2017) also outlined an algorithm that builds upon the foundation of the existing de Bruijn graph structure and seamlessly integrates self-stabilizing mechanisms to ensure fault recovery and system stabilization. Employing a decentralized control framework and using only locally available information, the algorithm achieved the fundamental design principles crucial for achieving self-stabilization within the graph. Thus, Feldmann and Scheideler (2017) proved the efficacy of their algorithm through formal proofs and analysis, demonstrating the correctness and convergence properties of their proposed algorithms.

The results presented by Feldmann and Scheideler (2017) showcased the self-stabilizing of the general de Bruijn graph constructed by their proposed algorithm against node failures and dynamic reconfigurations. The algorithm enabled the network to effortlessly rebound from arbitrary initial states, eventually converging toward a stable configuration.

OUR CONTRIBUTION

Our work introduces a novel approach by combining a service-oriented model with a de Bruijn graph-based overlay network architecture for distributed cloud computing systems. We depart from traditional node-based addressing to implement a service-based addressing mechanism, enhancing service discovery and allocation efficiency. By dividing the network address space into bands and assigning specific bands to each service, we simplify the service request process. Leveraging the de Bruijn network as a software-defined overlay ensures scalability and adaptability to dynamic cloud conditions. Through simulations, we demonstrate the architecture's sustainable scalability and inherent load-balancing properties. This innovative integration offers a promising solution to scalability and coherence challenges in distributed cloud environments.

PROPOSED ARCHITECTURE

Our proposed architecture for distributed cloud networks is a service-oriented de Bruijn graph-based network. Our aim in proposing this network is to optimize service discovery, network efficiency, and scalability. It consists of three interrelated components: service nodes, distributed cloud units, and service search and routing algorithms, each contributing to the overall function of the network. The functionality of de Bruijn nodes within our architecture is divided into service provision and overlay network management. The service nodes are responsible for service provisioning. Service provision is allocated to virtualized containers or machines running atop physical servers, called distributed cloud units, while overlay network management is centralized within an overlay network manager layer. This layer oversees the routing and graph construction for coexisting nodes. Hence, in our proposed architecture, a de Bruijn node consists of a service node and the shared overlay network management layer. Figure 2 shows the logical layers of a distributed cloud unit. New nodes seamlessly integrate into the network through a decentralized joining mechanism, leveraging the underlying weakly connected network infrastructure. To streamline service discovery and routing, we partition the de Bruijn hash space into bands, with each band associated with a specific service provided through the network. Service requests are directed to bands rather than specific service providers.

Service Nodes

In our proposed architecture, we introduce an approach where the network is represented in a general de Bruijn graph and each instance of a service provider is represented as a node in the graph. The service nodes can be either virtual machines coexisting with others on a physical server or directly stand-alone running on a physical server. We propose an approach where the de Bruijn hash space is divided into bands. Each band represents a subrange within the range]0:1[and is associated with a particular service provided through the network. Figures 3 and 4 show an example of a service band that is assigned a hash space with a range of [0.44:0.46]. Table 1 shows an example of nodes assigned

Figure 2. Logical Layers of a Distributed Cloud Unit

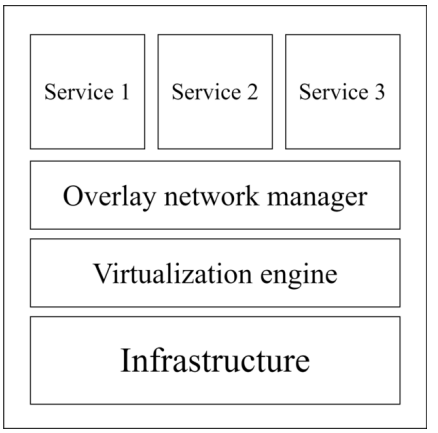


Figure 3. Network Hash Space

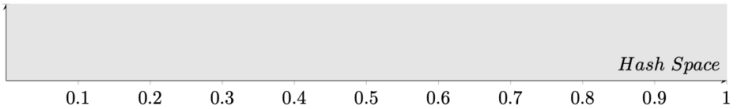


Figure 4. Subset of the Network Hash Space

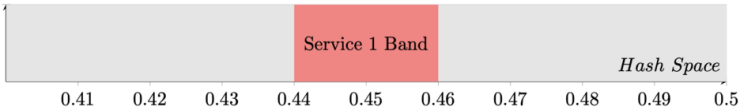


Table 1. Nodes Assigned to Service Band 1

Node	Service	Hash
X	1	0.4425123
Y	1	0.4525234
Z	1	0.4583961

to the Service 1 band from Figure 4. Each node is assigned a unique numerical identifier between 0 and 1 that is within a band allocated to the node's provided service. The client service request is issued to a band instead of a specific service provider, and the routing algorithm of the network implicitly tries to find an instance of the service provider within the band. Importantly, nodes belonging to the same band can be located in different geographical places.

To optimize service latency, the hashing function can be chosen to efficiently map nodes to their corresponding bands. The hashing function takes into account the geographic dispersion of nodes and aims to minimize packet travel time, thus reducing latency. Mapping nodes within the same band, which can be dispersed across different geographic locations, enables the distance traveled by packets to be minimized, resulting in improved service responsiveness.

Band Allocation and Management

In our proposed architecture, the allocation and management of bands are crucial for orchestrating service provisioning within the distributed cloud computing environment. Each service provider possesses a unique service ID, known exclusively to them, which serves as a distinct marker for the specific service offered within the network. Bands within the de Bruijn hash space are designated for specific services under the management of the network designer. This network designer assigns unique service IDs to providers, controlling the band allocation process and ensuring adherence to network policies.

The band allocation process relies on a one-way hashing function. This function takes the service ID as input and transforms it into a corresponding node ID within the designated band. The transformation is irreversible, ensuring that the original service ID cannot be deduced from the resulting node ID. To join the network, a node needs to authenticate first with the overlay network manager layer by providing the service ID. The overlay network manager uses the hash function to generate a unique node ID for the new node and starts introducing it to the system. The assumption of this mechanism is that the overlay network manager is a trusted software program. The discussion about ensuring the integrity of the overlay network manager is out of the scope of this work and should be investigated further in future work.

A critical aspect of band allocation is the unique mapping ensured by the hashing function. Each service ID is mapped to a distinct band within the hash space, preventing collisions and ensuring allocation integrity. This design mitigates the risk of unauthorized access and malicious activities.

The one-way nature of the hashing function enhances network security by making it computationally infeasible to reverse engineer the mapping from node ID back to the original service

ID. This security measure safeguards against impersonation and unauthorized access, fortifying the distributed cloud computing environment against potential threats.

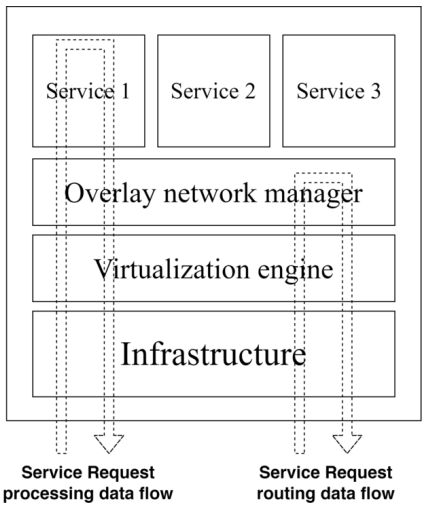
Distributed Cloud Units

In our proposed architecture, the functionality of the node is split into service-providing and overlay network management. The service-providing is allocated to virtualized containers or machines running on top of a physical server. Any number of nodes can be allocated to a single physical machine, with no constraints on the band range of the running services. The overlay network management of all the coexisting nodes of the server is grouped into an overlay network manager layer to reduce the overhead of network building and routing, as shown in Figure 2. This layer is responsible for handling all the routing and graph construction for the coexisting nodes. This layer keeps the local variables of each node and routes the arriving packages according to Algorithm 1 using the data of the recipient node. Figure 5 shows two scenarios: (a) an arriving request for a band for which the distributed cloud unit hosts a service node and (b) a request for a band for which the distributed cloud unit reroutes to the next node. The manager can further reduce the routing latency by checking if a coexisting node matches the service request band and routing it to that service-provider instance directly.

Another advantage of splitting the de Bruijn node functionality into two logical layers is enhanced data privacy. The data included in a user request for a certain band will never be available at any node other than a node within the requested band. The service request will be rerouted through the overlay network manager layer, which is assumed to be a trusted software program in the context of this proposal.

To establish the network, we begin with a weakly connected network, such as the internet or the 5G core network. This network provides a physical route to transmit packets from any node to all other nodes within the network. The de Bruijn network starts with an anchor node, which serves as the initial node to join the system and initiate the overlay network. Subsequent nodes can join the overlay network by issuing a join request to the anchor node; this request implies that a joining node needs to be able to address and communicate with at least an established node in the de Bruijn network within the framework of the core network. This process allows these nodes to integrate seamlessly into the network. It's worth highlighting that the anchor node does not function as a central hub; its significance lies solely in being the first node to join the network. Additionally, candidate nodes can join the network by introducing themselves to any existing node. Through this mechanism, a new

Figure 5. Data Flow in the Distributed Unit



node is guaranteed to integrate into the network at the correct position in the node list, as proposed by Feldmann and Scheideler (2017).

The process of initializing the network can go as follows:

1. The overlay network is initialized with a single node, which will act as the bootstrap node and form the initial basis of the de Bruijn network.
2. When a new node wants to join the network, it needs to know the physical address of the bootstrap node. The physical address of the bootstrap node can either be static and known or can be dynamically obtained through a peer-to-peer service discovery method—for example, a DNS search.
3. The new node establishes a connection to the bootstrap node. Once connected, the new node exchanges information on the overlay network level with the bootstrap node by calling the linearize function on the bootstrap node(Feldmann & Scheideler, 2017).
4. The new node constructs its own local list of the de Bruijn network neighbors and logical overlay connections by performing the buildList, general de Bruijn, and standard de Bruijn protocols (Feldmann & Scheideler, 2017).
5. The bootstrap node can also provide the new node with a set of initial connections or references to existing nodes in the network, helping the new node bootstrap its connectivity within the de Bruijn network.

The buildList, general de Bruijn, and standard de Bruijn protocols are invoked periodically to update network information. Each node calls the three protocols to update its connections list, essentially self-stabilizing the graph. This property of the general de Bruijn graph is very useful for our proposed architecture, allowing the network to optimally handle service requests in a distributed and dynamic cloud environment.

Service Search and Routing Algorithm

In our proposed architecture, the routing algorithm relies on certain segments of local information available to each node, including the following:

- d: Network diameter. An empirically tunable parameter that is essential for building the de Bruijn network. It represents the maximum number of allowed de Bruijn hops before the packet-routing timeouts.
- v.b: The band at which the node provides its service. It allows the node to identify its specific service domain within the network.
- v.q: The set of q neighbors to the node is another crucial piece of information. This set is constructed using the buildList, general de Bruijn, and standard de Bruijn protocols proposed by Feldmann and Scheideler (2017). It enables the node to maintain awareness of its neighboring nodes in the network. The maximum number of nodes in this set is 3q-2 (Feldmann & Scheideler, 2017).
- v.logq: To estimate the total number of nodes in the network, each node maintains v.logq. This value serves as an approximation of the network's size and aids in making informed routing decisions.

To facilitate the routing algorithm, we employed a multi-hop approach based on the de Bruijn graph. The algorithm takes into account the aforementioned local information and employs routing techniques to ensure service search and delivery.

When a node receives a service request, it begins the routing process by inspecting its v.b parameter. If the requested service falls within the same band, the node can directly process and respond to the request because it is within its service domain. However, if the requested service

lies outside the node's band, the routing algorithm is invoked. First, the node evaluates the v.logq parameter to estimate the size of the network. This estimation aids in selecting the most efficient routing path to reach the desired service.

Next, the node uses its v.q parameter to identify potential neighbor nodes that may provide the requested service or are closer to the requested band in the de Bruijn list of the network. By leveraging the buildList, general de Bruijn, and standard de Bruijn protocols, the node can determine a set of suitable candidate nodes that are likely to have the desired service. Once the target band is reached, the routing algorithm guides the service request to the nearest node within the band that offers the requested service. This ensures that the service request is efficiently directed to the appropriate provider, minimizing latency and maximizing service availability.

By incorporating these service search and routing algorithms, our architecture optimizes the efficiency and effectiveness of service discovery within the network. The combination of local information, multi-hop routing, and intelligent path selection enables nodes to efficiently locate and access services.

Algorithm 1. Service Request Routing

```
1.   Procedure Servicerequesthandler(b, r, RemHops)
2.   if v.band = b then
3.   Routing success;
4.   else if RemHops > 0 AND log(q) <= r then
5.   u <- determineNextDeBruijnHop();
6.   u -> ServiceRequestHandler(b, r - log(q), RemHops-1);
7.   else
8.   Routing failure;
9.   end if
10.  end Procedure
```

Algorithm 1 outlines the service request routing procedure. It takes three input parameters: b (the target band), r (a threshold value), and remHops (the remaining number of hops).

The algorithm begins by checking if the current node's band matches the target band. If they match, it signifies a successful routing, and the algorithm proceeds accordingly.

If the bands don't match and there are remaining hops (remHops > 0), the algorithm checks if the two's logarithm of q is less than or equal to the threshold value r. If this condition is satisfied, the algorithm proceeds to determine the next de Bruijn hop (u) and recursively invokes the ServiceRequestHandler procedure with updated parameters. This recursive step allows the algorithm to progress toward the desired band.

The function determineNextDeBruijnHop() is discussed in detail in Feldmann and Scheideler (2017). In cases where the bands don't match and the above condition is not met, it signifies a routing failure. By following this routing algorithm, the system can handle service requests by navigating through the network based on the target band, threshold value, and remaining hops. This approach ensures efficient and reliable routing of service requests within the network.

The overhead of the proposed architecture in terms of routing is only the runtime execution of the ServiceRequestHandler function. The function is executed at every hop in the routing path, and the complexity of the function arises mainly from determineNextDeBruijnHop(), which searches a list of 3q-2 items. It has a big O notation of log (total number of network nodes).

Scale

In the landscape of cloud service providers leveraging distributed infrastructure across diverse geographical regions, including industry giants, such as Amazon, Google, and Azure, the proposed

network architecture holds versatile applicability. Although these providers maintain expansive backbone networks spanning multiple regions, our approach is designed to be able to seamlessly integrate within the confines of a singular data center.

The proposed network architecture offers dual functionality, serving as a comprehensive solution for building an entire distributed cloud network or as a specialized network within a designated data center. This flexibility is made possible through the integration of a dedicated translator or network entry manager. This component plays a pivotal role in facilitating communication by converting conventional network messages into a format understood by the internal de Bruijn network within the data center. By incorporating a translator or network entry manager, the proposed approach ensures compatibility with existing network infrastructures, making it a scalable solution for both overarching network implementations and localized, data center-centric applications. Figure 6 shows an example of a de Bruijn-based cloud data center connection with an example internet network.

SIMULATION RESULTS AND DISCUSSION

In this section, the results of three simulation scenarios are presented. The first simulation targets evaluating the proposed architecture's scalability, focusing mainly on how the proposed architecture manages the number of outgoing edges as the number of network nodes increase. The second simulation aims to show the network performance in terms of routing correctness. The third simulation shows the implicit load balancing of the routing algorithm.

Simulation Model

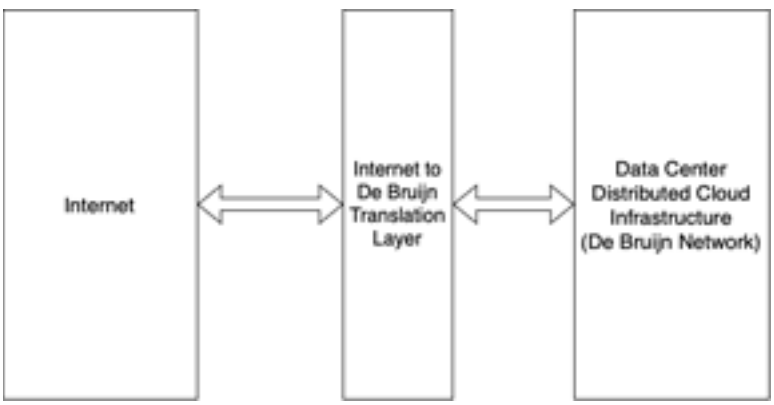
The simulation model provides a framework for evaluating the proposed service-oriented network architecture tailored for distributed cloud computing systems. It facilitates targeted assessments of scalability, routing correctness, and load balancing within the network, offering insights into its performance under various conditions.

Model Parameters

The simulation model includes the following parameters:

- Number of Nodes (numNodes): Defines the size of the network by specifying the total number of nodes present in the distributed cloud computing system
- Alphabet Size (q): Represents the size of the alphabet used for node identification and addressing within the network

Figure 6. De Bruijn Network-Based Data Center



- Dimensions (d): Determines the dimensions of the network topology, influencing the structure and connectivity patterns among nodes

Network Construction

Constructing the network includes these processes:

- Random node initialization: The simulation model initiates by randomly generating unique identifiers for each node within the specified alphabet size q.
- Topology generation: Using the general de Bruijn graph definition, the model constructs the network topology based on the generated node identifiers. Connectivity among nodes is established to form a distributed cloud computing system.
- Adjacency matrix generation: An adjacency matrix is generated to represent the connectivity relationships between nodes. This matrix serves as a fundamental data structure for routing operations and network analysis.

Scalability Evaluation

The simulation model offers the following capabilities for scalability evaluation:

- The simulation model enables the evaluation of network scalability by allowing the user to rerun simulations with varying numbers of nodes (\(numNodes\)).
- By systematically adjusting the network size and observing the impact on performance metrics, such as throughput, latency, and resource utilization, researchers can assess how well the architecture scales with increasing network size.

Routing Correctness Assessment

The simulation model offers the following capabilities for routing correctness assessment:

- The simulation model incorporates routing algorithms, such as de Bruijn-based routing, to assess the correctness and efficiency of routing within the network.
- By simulating various routing scenarios and tracking the paths taken by data packets between source and destination nodes, the results can evaluate the accuracy and reliability of the routing mechanism.

Load Balancing Evaluation

The simulation model offers the following capabilities for load balancing evaluation:

- Load balancing capabilities within the network are evaluated using the simulation model by monitoring the distribution of traffic and resource use across nodes.
- Through simulations with realistic workload patterns and varying levels of network activity, simulation results can assess the effectiveness of load balancing algorithms in evenly distributing workload and preventing resource bottlenecks.

The simulation model provides a flexible and robust framework for evaluating the proposed service-oriented network architecture within distributed cloud computing environments. By focusing on scalability, routing correctness, and load balancing, the simulation model yields results that can help us gain insights into the architecture's strengths, limitations, and areas for enhancement,

ultimately guiding the development and optimization of distributed cloud networks based on the proposed architecture.

Network Scalability

To assess the scalability of the proposed network architecture, we conducted multiple iterations of simulations across a range of node counts, from 200 to 10,000. The objective was to analyze how the network's performance scales with increasing numbers of nodes while maintaining a balanced distribution of outgoing edges per node.

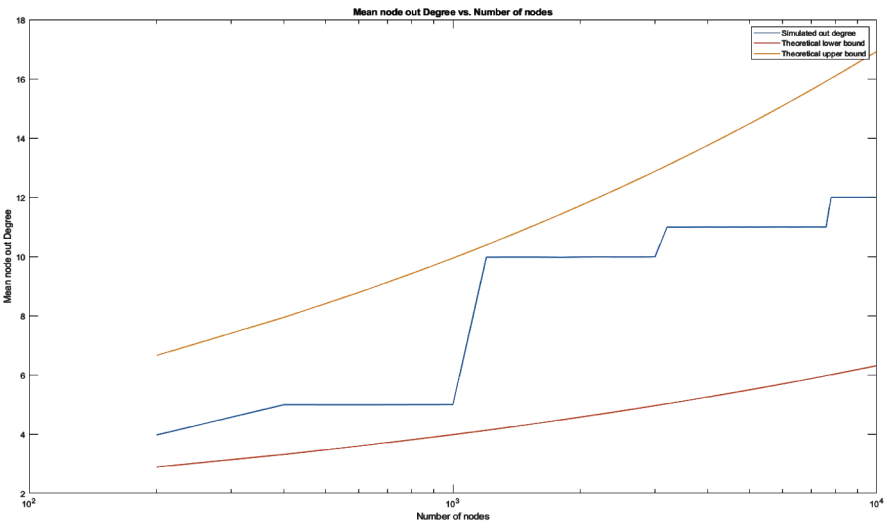
Simulation Setup

We conducted the following simulation tests for setup:

- Node count variation: Simulations were executed for node counts ranging from 200 to 10,000, allowing for a comprehensive evaluation of scalability across a broad spectrum of network sizes.
- Theoretical outgoing edges boundaries: The theoretical lower boundary of outgoing edges per node was calculated as the d-th root of the total number of nodes, while the upper boundary is 3* q - 2, where q represents the alphabet size used for node addressing (Feldmann & Scheideler, 2017).
- Simulation iterations: Each simulation iteration involved generating a network topology based on the specified node count and assessing the average outgoing degree of nodes within the network.

The simulation results, depicted in Figure 7, revealed that the actual average outgoing degree of nodes fell within the boundaries defined by the theoretical lower and upper limits across all tested node counts. This finding indicates that the proposed network architecture maintains a consistent level of connectivity and load distribution regardless of the network size. The architecture demonstrates its ability to adapt to varying network sizes while maintaining balanced connectivity. These findings reinforce the viability of the proposed architecture for distributed cloud computing systems and underscore its potential to support large-scale deployments with confidence in network performance and scalability.

Figure 7. Average Outgoing Degrees per Node



Network Routing Performance

We created a MATLAB model for the distributed cloud network. We wrote the model from scratch in MATLAB script language without the use of any specific toolbox. The network was fed with thousands of randomized service requests with a random entry point. The simulation model is divided into five parts:

- A weakly connected graph was constructed with a randomized ID for each node.
- The de Bruijn graph parameters were calculated for the constructed graph.
- A new directed graph was constructed from the weakly connected graph following the general de Bruijn connected edges rules.
- Randomized service requests were generated for random service bands and inserted into the network at a random starting node.
- The service request was propagated inside the network until it reached a service provider for the requested band or reached the hops threshold and terminated.

Figures 8 and 9 show the simulated performance of a network with a number of nodes equal to 256 and 1,024 nodes, respectively. The nodes are generated and assigned randomly to bands across the range]0,1[. The X axis represents different values for the diameter of the network. In subgraph (a), the Y axis represents the normalized success rate, calculated by dividing the total number of successfully routed service requests by the total number of generated requests. Subgraph (b) shows the relation between the network diameter and the average number of hops the service requested takes to reach a service-providing node. The average number of hops should not surpass the network diameter; otherwise, a large number of requests will not reach their destination. Subgraph (c) shows the relationship between the network diameter and the average outgoing degree of nodes. It can be

Figure 8. Simulation Results for a Network With 256 Nodes

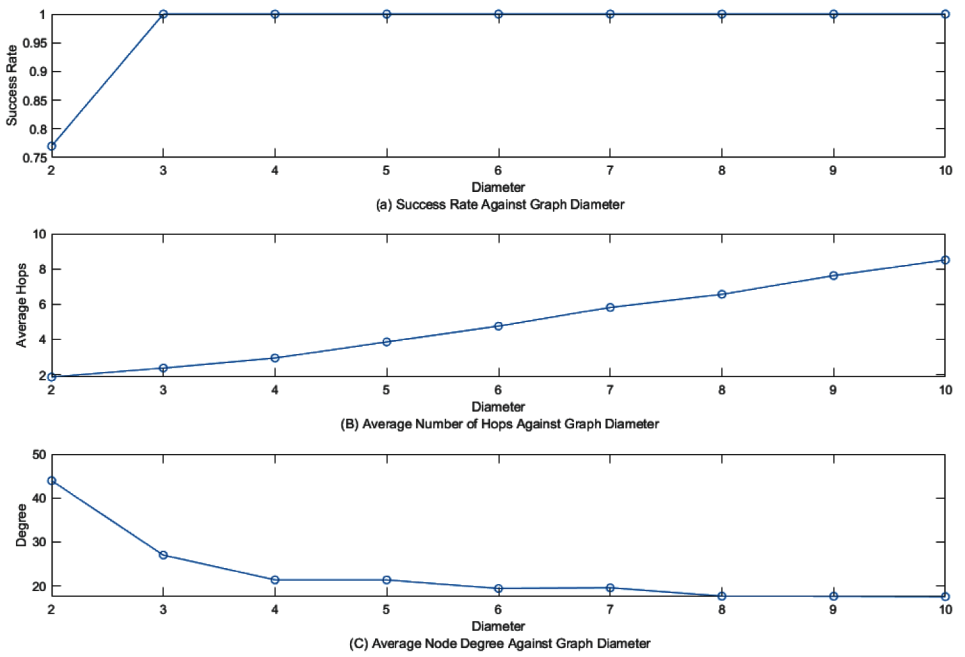
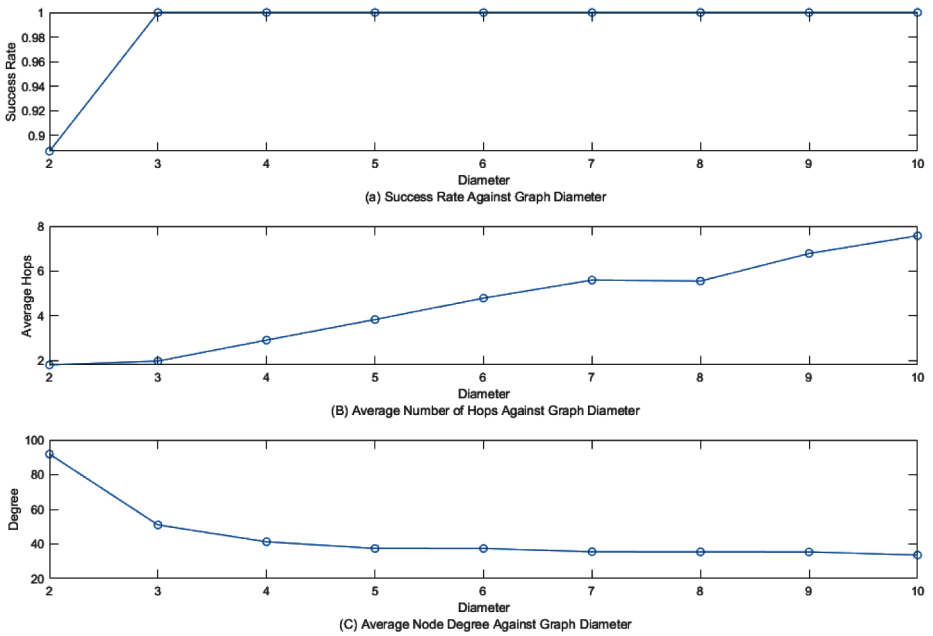


Figure 9. Simulation Results for a Network With 1,024 Nodes



observed that the average outgoing degree decreases rapidly with network diameter increase, indicating a trade-off between the number of established connections and the latency of request delivery.

The simulation results show that given a service request for a band that exists in the network, a very confident success rate is attainable at a low graph diameter. The network routing success rate relies mainly on the diameter of the network, not the number of nodes; hence, the network can be scaled up exponentially in terms of nodes while only logarithmically growing in diameter.

As the diameter is increased, the average outgoing degree per node is rapidly decreasing, and the average number of hops per request is increased. As the average node outgoing degrees increases, the number of entries that must be searched through in every hop during routing increases. This pattern translates to a larger processing overhead in the overlay network routing management. On the other hand, the increase in number of hops during routing results in an overall higher latency time for service requests. A trade-off should be considered between the outgoing degrees and the number of hops by selecting a graph diameter value that results in a manageable number of connections per nod—thus manageable routing overhead—while still meeting the latency constraints of the cloud services.

Load Balancing

To evaluate the implicit load balancing of the routing algorithm, the network from the previous subsection is used with a modified scenario. The model is as follows:

- A weakly connected graph is constructed with a randomized ID for each node.
- The de Bruijn graph parameters are calculated for the constructed graph.
- A new directed graph is constructed from the weakly connected graph following the general de Bruijn connected edges rules.
- Randomized service requests are generated for certain service bands and inserted into the network at a random starting node.
- The service request is propagated inside the network until it reaches a service provider for the requested band or reaches the hops threshold and terminates.

To simulate a high-demand situation, the network is fed with 200,000 requests concentrated in the band of interest. At the end of the simulation, nodes within the band are checked for the number of requests that were processed by each. Figures 10 and 11 show the simulated load balance of a 256- and 32,768-node network for a band that has a range of [0.44:0.46], with six and 675 nodes having hashes within the band, respectively.

Figure 10. Simulation Results for Band Load Balance With Six Nodes

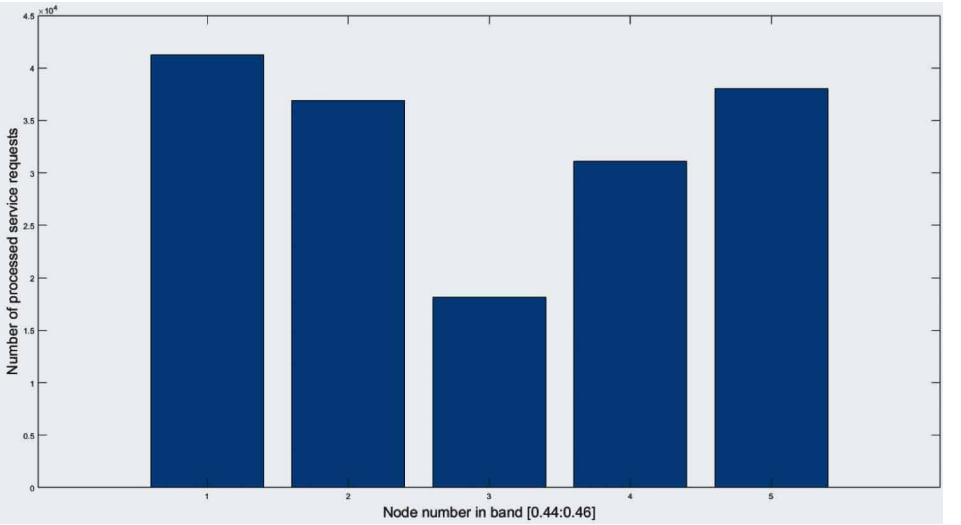


Figure 11. Simulation Results for Band Load Balance With 675 Nodes

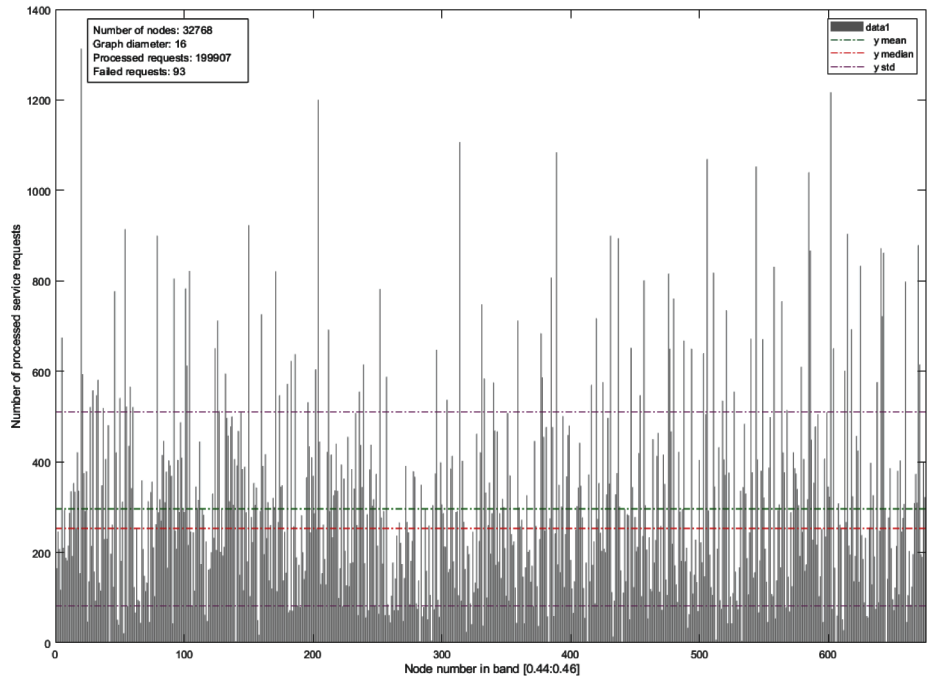


Figure 11 shows the mean and median of the tasks processed by each node. Note that the implicit load balancing arising from the de Bruijn routing is initially acceptable, but not comparable to state-of-the-art load-balancing techniques. To match state-of-the-art load-balancing algorithms, the hashing function of proposed architecture should be enhanced to better space out nodes so that the distribution of paths leading to nodes within the band is uniform. The simulation results show that the inherent load balancing of the proposed architecture holds as the network scales up in terms of the number of nodes.

CONCLUSION

This paper introduces a network architecture specifically designed for distributed cloud computing networks. Its aim is to address the challenges arising from decentralization and scalability. The proposed architecture uses a de Bruijn graph-based overlay network alongside a service-banding approach, resulting in a highly scalable, service-oriented cloud network. By segregating the functionality of nodes into service-providing and overlay network management, achieved through virtualized containers or machines on a physical server, resource utilization is optimized. Table 2 shows a comparison between the proposed architecture and two prominent distributed cloud architectures. The architecture's performance is thoroughly examined through simulations that identify a strength in routing performance and inherit load balancing and show areas that may require further investigation. The scalability of the proposed architecture in terms of node connections is shown to have logarithmic growth in respect to the total number of nodes in the network. Future work may explore extending this architecture to accommodate

Table 2. Comparison Between Proposed Architecture and Recent Distributed Cloud Computing Architectures

Dimension	Blockchain-Based Cloud Architectures	Fog Computing	Proposed Architecture
Architecture	Decentralized, distributed ledger technology	Extends cloud computing to the edge of the network	Decentralized, service oriented
Scalability	Faces scalability challenges owing to consensus mechanisms	Can improve scalability by distributing computational tasks	Scales logarithmically with number of member nodes
Network model	Peer-to-peer (P2P) network model	Hierarchical network model with edge devices, fog nodes, and centralized cloud servers	Graph-based overlay network
Suitability	Suitable for applications requiring transparency, immutability, and trust	Suitable for applications demanding low latency, real-time data processing, and efficient bandwidth utilization	Enterprise cloud services, big data analytics, high-performance computing, mission-critical applications, content delivery networks, IoT
Strengths	Immutable ledger, decentralized consensus, enhanced security, transparency	Low latency, efficient bandwidth utilization, improved scalability	Decentralized stable network structure, improved scalability, service discovery, inherit load balancing.
Weaknesses	Scalability limitations, high computational overhead, potential latency	Management complexity, potential security risks at the edge, dependency on network connectivity	Overlay network overhead can become not neglectable with extremely high numbers of nodes; a trusted software program is needed to manage the node authentication.

Table 3. List of Abbreviations

Abbreviation	Definition
V	Set of all nodes in a de Bruijn graph
E	Set of all edges in a de Bruijn graph
K-mer	A sequence of symbols with length K
Q-ary	The base with which the node hash is represented as digital bits
q	The general de Bruijn network base. Used for hash Q-ary bits calculation, and for the routing algorithm.
d	Graph diameter
b	Band for which a requested service belongs to
v	Source node of a de Bruijn hop
u	Target node of a de Bruijn hop

emerging distributed cloud use cases, such as edge computing and cloud enterprise data warehouse, exploring security and privacy measures suitable for a decentralized distributed network, such as certificateless public auditing schemes and trusted software-based distributed node authentication schemes.

Table 3 includes a list of abbreviations.

REFERENCES

Ageed, Z. S., & Zeebaree, S. R. M. (2024). Distributed systems meet cloud computing: A review of convergence and integration. *International Journal of Intelligent Systems and Applications in Engineering*, *12*(11s), 469–490. https://ijisae.org/index.php/IJISAE/article/view/4468

Angel, N. A., Ravindran, D., Vincent, P. M. D. R., Srinivasan, K., & Hu, Y.-C. (2021). Recent advances in evolving computing paradigms: Cloud, edge, and fog technologies. *Sensors (Basel)*, *22*(1), 196. doi:10.3390/s22010196 PMID:35009740

Atieh, A. T. (2021). The next generation cloud technologies: A review on distributed cloud, fog and edge computing and their opportunities and challenges. *ResearchBerg Review of Science and Technology, 1*(1), 1–15. https://researchberg.com/index.php/rrst/article/view/18

Chikhi, A., & Rizk, G. (2013). Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms for Molecular Biology; AMB*, *8*(1), 22. doi:10.1186/1748-7188-8-22 PMID:24040893

Chikhi, R., Limasset, A., Jackman, S., Simpson, J. T., & Medvedev, P. (2014). On the representation of de Bruijn graphs. In R. Sharan (Ed.), Lecture Notes in Computer Science: Vol. 8394. *Research in Computational Molecular Biology. RECOMB 2014* (pp. 35–55). Springer Cham. doi:10.1007/978-3-319-05269-4_4

Coady, Y., Hohlfeld, O., Kempf, J., McGeer, R., & Schmid, S. (2015). Distributed cloud computing: Applications, status quo, and challenges. *Computer Communication Review*, *45*(2), 38–43. doi:10.1145/2766330.2766337

Cortes Gallardo Medina, E., Velazquez Espitia, V. M., Chipuli Silva, D., Fernandez Ruiz de las Cuevas, S., Palacios Hirata, M., Zhu Chen, A., González González, J. A., Bustamante-Bello, R., & Moreno-García, C. F. (2021). Object detection, distributed cloud computing and parallelization techniques for autonomous driving systems. *Applied Sciences (Basel, Switzerland)*, *11*(7), 2925. doi:10.3390/app11072925

Faizian, P., Mollah, M. A., Yuan, X., Pakin, S., & Lang, M. (2016). Random regular graph and generalized de Bruijn graph with k-shortest path routing. In *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (pp. 103–112). IEEE. doi:10.1109/IPDPS.2016.44

Faniyi, F., Bahsoon, R., Evans, A., & Kazman, R. (2011). Evaluating security properties of architectures in unpredictable environments: A case for cloud. In *Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture* (pp. 127–136). IEEE. doi:10.1109/WICSA.2011.25

Feldmann, M., & Scheideler, C. (2017). A self-stabilizing general de Bruijn graph. In P. Spirakis & P. Tsigas (Eds.), *Stabilization, safety, and security of distributed systems* (pp. 250–264). Springer International Publishing., doi:10.1007/978-3-319-69084-1_17

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., & Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, *7*(2), 72–93. doi:10.1109/COMST.2005.1610546

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *Computer Communication Review*, *31*(4), 149–160. doi:10.1145/964723.383071

Osama Ragab Shaban Ramadan obtained a B.Sc in Electronics and Communication Engineering in 2019, focusing on system architecture and design. He is currently engaged in research and development projects involving Lidar systems, distributed computing, and networking.

Mohamed Yasin I. Afifi is an Lecturer at the Electrical Engineering Department- Faculty of Engineering- Al-Azhar University, Cairo–Egypt. He received his B.Sc., M.Sc. and Ph.D. Degrees from Al-Azhar University in 2011, 2017, and 2021 respectively. His areas of interest include Cloud Computing, IoT, and Lightweight Cryptography. He can be contacted at email: mohamedyasin869@azhar.edu.eg

Ahmed Yahya [ORCID: 0000-0002-3271-058X] is currently working as a Professor in the Department of Electrical Engineering, A-Azhar University, Cairo, Egypt. He obtained his Ph.D. from Ain Shams University, Cairo, Egypt. He has published many research papers, which contributed significantly to the development of his field. He had been working as As Professor in various reputed Institution's in Abroad and Egypt. His research area focuses on Distributed Cloud Edge Computing, 5G Testbeds, LoRaWAN Networks, V2X Communications, Reconfigurable System-on-chip Design, Metamaterial-based Reconfigurable Antenna, and Intelligent Cognitive Radio Networks.