





Please cite the Published Version

Okafor, Kennedy Chinedu , Okafor, Wisdom Onyema, Longe, Omowunmi Mary , Ayogu, Ikechukwu Ignatius, Anoh, Kelvin  and Adebisi, Bamidele  (2025) Scalable Container-Based Time Synchronization for Smart Grid Data Center Networks. *Technologies*, 13 (3). 105 ISSN 2227-7080

DOI: <https://doi.org/10.3390/technologies13030105>

Publisher: MDPI AG

Version: Published Version

Downloaded from: <https://e-space.mmu.ac.uk/639063/>

Usage rights:  [Creative Commons: Attribution 4.0](https://creativecommons.org/licenses/by/4.0/)

Additional Information: This is an open access article which first appeared in *Technologies*, published by MDPI

Data Access Statement: All data generated or analyzed in this study are included in this manuscript, except the programming codes, which can be released upon reasonable request to the corresponding author.

Enquiries:

If you have questions about this document, contact openresearch@mmu.ac.uk. Please include the URL of the record in e-space. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from <https://www.mmu.ac.uk/library/using-the-library/policies-and-guidelines>)

Article

Scalable Container-Based Time Synchronization for Smart Grid Data Center Networks

Kennedy Chinedu Okafor^{1,2,3,*} , Wisdom Onyema Okafor⁴, Omowunmi Mary Longe² ,
Ikechukwu Ignatius Ayogu⁵, Kelvin Anoh³  and Bamidele Adebisi¹ 

¹ Department of Engineering, Manchester Metropolitan University, Manchester M1 5GD, UK; b.adebisi@mmu.ac.uk

² Department of Electrical and Electronic Engineering Science, University of Johannesburg, Johannesburg 2006, South Africa; omowunmil@uj.ac.za

³ School of Engineering, University of Chichester, Bognor Regis PO21 1HR, UK; k.anoh@chi.ac.uk

⁴ School of Computer Science and Technology, University of Bedfordshire, Luton LU1 3JU, UK; wisdom.okafor.pg82650@unn.edu.ng

⁵ Department of Computer Science, Federal University of Technology, Owerri PMB 1526, Nigeria; ignatius.ayogu@futo.edu.ng

* Correspondence: kennedy.okafor@mmu.ac.uk

Abstract: The integration of edge-to-cloud infrastructures in smart grid (SG) data center networks requires scalable, efficient, and secure architecture. Traditional server-based SG data center architectures face high computational loads and delays. To address this problem, a lightweight data center network (DCN) with low-cost, and fast-converging optimization is required. This paper introduces a container-based time synchronization model (CTSM) within a spine-leaf virtual private cloud (SL-VPC), deployed via AWS CloudFormation stack as a practical use case. The CTSM optimizes resource utilization, security, and traffic management while reducing computational overhead. The model was benchmarked against five DCN topologies—DCell, Mesh, Skywalk, Dahu, and Ficonn—using Mininet simulations and a software-defined CloudFormation stack on an Amazon EC2 HPC testbed under realistic SG traffic patterns. The results show that CTSM achieved near-100% reliability, with the highest received energy data (29.87%), lowest packetization delay (13.11%), and highest traffic availability (70.85%). Stateless container engines improved resource allocation, reducing administrative overhead and enhancing grid stability. Software-defined Network (SDN)-driven adaptive routing and load balancing further optimized performance under dynamic demand conditions. These findings position CTSM-SL-VPC as a secure, scalable, and efficient solution for next-generation smart grid automation.

Keywords: cloud computing; data center network; smart grid; software-defined network; containerization



Academic Editor: Dongran Song

Received: 10 December 2024

Revised: 17 February 2025

Accepted: 21 February 2025

Published: 5 March 2025

Citation: Okafor, K.C.; Okafor, W.O.; Longe, O.M.; Ayogu, I.I.; Anoh, K.; Adebisi, B. Scalable Container-Based Time Synchronization for Smart Grid Data Center Networks. *Technologies* **2025**, *13*, 105. <https://doi.org/10.3390/technologies13030105>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In today's Cloud-First era, where smart grid digital innovation is accelerating at breakneck speed, data centers face unprecedented demands—from managing massive data traffic and supporting multi-tenant environments to delivering cutting-edge analytics services. Yet, when traditional architecture reaches its limits, even powerful systems like the legacy SGDA struggle with computational overhead and service delays [1,2]. Thus, a solution that not only addresses these limitations but also optimizes performance across the board is needed.

By integrating innovative models like container optimization and time synchronization, these legacy systems can be transformed into highly efficient, resilient infrastructures that are fit for the future. For example, the integration of smart grid (SG) technologies into energy infrastructure marks a major step forward in sustainable energy management. However, to unlock its full potential, key improvements in system design are required. Traditional approaches often struggle with issues like scalability, efficiency, and synchronization, especially in distributed edge environments [1]. To overcome these challenges, the goal is to transform energy distribution and management by incorporating time synchronization and container-based optimization into the distributed edge architecture of the SG Fog [2]. This innovative strategy promises enhanced security, better resource utilization, smoother operations, and seamless coordination across the grid [3].

To date, some specialized literature has proposed several strategies to address computational overhead and service delays in legacy data center networks. However, these strategies are introduced mostly at a discrete level, only including real-time service recovery mechanisms [4]; a congestion-aware (CA) control with buffer and egress port CA algorithms and fine-grained window size adjustment [5]; Viseu, a latency-aware blockchain framework for virtual internet services at the edge [6]; backward congestion notification for end-to-end congestion management in Ethernet-based unified switch fabric [7]; machine learning-based predictive congestion control for TCP to mitigate packet loss and improve the quality of the experience [8]; and switch-assistant loss recovery for RoCEv2 to replace priority flow control in large-scale data centers [9]. Thus, this paper advances SG edge-to-cloud integrations by enhancing the reliability, resilience, and efficiency of service provisioning in SL-VPC beyond the discrete level.

As SGs continue to evolve, the growing demands on data centers have become increasingly apparent, particularly concerning storage capacity, computing power, and bandwidth capabilities [10]. The surge in data volume underscores the urgent need for expanded storage solutions. However, data center hardware has often been developed without fully considering the requirements of distributed systems, especially those managing substantial east–west traffic, such as SG enterprise DCNs [1–3]. These networks necessitate extensive computational resources and highly structured designs that can be effectively scaled as the impact of SGs on service delivery in distributed architecture continues to evolve.

Handling increased traffic and workload through intelligent automation, complex mathematical, and computational workflows is rarely implemented in legacy networks [11]. However, in modern network architectures, these approaches are often superior, enabling dynamic optimization, predictive scaling, and cost-effective resource utilization. As a result, a predictive learning curve, PLC (also referred to as intelligent analytics), is introduced into the SL-VPC for performance optimization in SGDCNs. Our spine–leaf virtual private cloud (SL-VPC) architecture implements the spine–leaf topology within a multi-cloud environment. This topology comprises two distinct layers of switches: leaf switches and spine switches. Leaf switches connect directly to individual servers, while spine switches interconnect all leaf switches, ensuring a highly scalable, low-latency, and efficient network. The architecture optimizes east–west traffic flow, enhances fault tolerance, and supports high-bandwidth applications, making it ideal for SGCN (i.e., on-premises and cloud) environments.

In this paper, container stack automation is integrated into an SL-VPC within an AWS cloud infrastructure. This is adaptable to any cloud ecosystem, including Azure and Google Cloud Platform (GCP) through network-defined functions which house containers for Kubernetes. By incorporating stochastic gradient descent (SGD) controllers, the system enables lightweight scheduling and efficient resource allocation. The results highlight that CTSM SL-VPC outperforms traditional DCN architectures in received energy data

and service throughput, while SGDA IPv6 Gateway and SGDA-VPC access exhibit higher encryption–decryption overhead and varying media access and packetization delays. The introduction of containerization, a CTSM, and software-defined CloudFormation stack (SD-CFS) enhances computational efficiency, automation, scalability, security, and compliance, supporting multi-cloud and hybrid environments.

A multi-combinational approach integrating edge computing, SDN, containerization, AI-driven optimization, and hybrid cloud architecture effectively addresses computational overhead and service delays in SGDCNs. Edge and fog computing reduce latency by processing data closer to the energy paying residents, while SDN and NFV optimize network traffic, minimizing congestion. Containerization and microservices enhance scalability and modularity, preventing system-wide failures. AI-driven optimization and SGD-based scheduling improve real-time analytics and resource allocation. Time-synchronized SL architecture ensures low-latency, high-throughput data exchange, while hybrid and multi-cloud integrations improve workload distribution, reliability, and security. With CTSM and SDN-CFS, the proposed SGDCN demonstrates significant gains in efficiency and performance.

The summary contributions of this paper include the following:

1. Developing a simple optimization framework utilizing stochastic gradient descent (SGD) and the implicit function theorem (IFT) for dynamic load balancing and resource management in the SGDA network.
2. Proposing a multi-queuing model (MQM) for real-time load management via cloud-based resource allocation and auto-scaling.
3. Establishing a decentralized edge-to-fog framework for efficient advanced metering infrastructure (AMI) sensor communication and energy monitoring.
4. Implementing predictive workload balancing and multipath routing to reduce latency and optimize traffic flow in SGDA.
5. Developing a security group firewalling strategy using OpenFlow to enhance network security and load management, providing a robust backend for data protection.
6. Conducting performance evaluations comparing SGDA efficiency with traditional data center network topologies.

The rest of the paper is structured as follows. In Section 2, general studies on DCN topologies are discussed. Section 3 discusses the methodology applied to fix the gaps. Section 4 presents the performance evaluation of the DCN while Section 5 presents the conclusion of the research findings and recommendations.

2. Relevant Literature Review

2.1. Topological Models

Legacy server-centric and switch-centric topologies are the two classifications of data center networks based on where the intelligence of the DCN is located [12]. In server-centric DCN design, the intelligence is built on the server, while in switch-centric cases the intelligence is built on the switch. In this paper, we examined server-centric topologies, switch-centric topologies, and spine–leaf architecture for prospective smart grid DCN designs.

In [13], the Dahu DCN model was proposed as an improvement powered by commodity Ethernet switches supporting direct link networks. By dynamically distributing traffic equally across links, this model eradicates congestion points. When performing load balancing using local data, it forwards traffic over non-minimal routes. However, the decentralized load balancing heuristic is not scalable for SG environments. SG data center implementations would be primarily restricted to multi-rooted tree switch-centered topologies such as Dahu [13] and traditional data center architecture. The authors of [14]

developed a changeable multi-tiered topology with substantial oversubscription by altering the amount of forwarding and receiving links at each access tier and aggregation tier switch. The fat-tree architecture [15] was proposed to describe how to sustain the total aggregate bandwidth of clusters with tens of thousands of nodes using mostly cheap Ethernet switches. The price difference between high-end switches and low-end switches is significant, which inspired the creation of the fat-tree network. Unfortunately, this model is not scalable for the SG ecosystem due to the limitation of the port switch. Various efforts in the literature seem to isolate SG applications. The traditional data center architecture denotes a multi-tiered design and vendor orientation.

Intelligent switches are used in the switch-centric category to conduct smart packet routing in a data center. In this category, some switch-centric data center topologies include JellyFish [16], DOS [17], FRINGE [18], and Skywalk [19]. In [20], the flattened butterfly network topology was designed originally for on-chip connectivity networks. The authors presented a flattened butterfly network for high-radix networks as a cost-effective topology. This is useful in load-balanced traffic where, relative to the Clos network, its efficiency is optimum. The advantage over the Clos network is achieved by removing redundant hops when they are not required for load balancing. Again, the network integration of SG systems may encounter computational overhead; however, efforts to build resilient systems have continued to date. For instance, a Layered Scalable Data Center (LaScaDa) was introduced as a DC topology in [21]. This is used for the construction of scalable and cost-effective networking infrastructures for data centers. Interestingly, SG systems will rely on server-centric DCN topologies to ensure robust server integration. The main server-centric models in the literature include [22] DCell, BCube, FiConn, HCNs and BCNs, DPillar, MCube BCDC, General Hypercube, HSDC, and Stellar Transformation [23]. Delay in service processing is a common problem of server-centric DCN topologies. This is unacceptable for SGDAs.

A new trend of DCN evolution has recently begun with spine–leaf architecture [23]. The architectural design is an example of Clos architecture, in which every leaf switch is linked to each of the spine switches and is a full-mesh topology. Depending on the capabilities of the networking switches, Layer 2 or Layer 3 technologies can be integrated with the spine–leaf mesh. Each link must be routed in Layer 3 spine–leaves, which is typically achieved using open shortest path first (OSPF) or equal-cost multipath routing (ECMP) with border gateway protocol (BGP) dynamic routing. Layer 2 employs a loop-free Ethernet fabric like transparent interconnection of lots of links (TRILL) or shortest path bridging (SPB). The modern-day software-defined solutions (smart grid networks, banking enterprise architecture) require more expanded east–west traffic, so the spine–leaf architecture is realistically ideal [24]. Similarly, the DCNs in [25–28] offer scalable attributes for enterprise adoption, though with several limitations. Table 1 summarizes the recent efforts toward data center designs and application deployments.

Table 1. Summary of DCN design models and applications.

References	Design Strategy	Design Limitations	Application Domain
X-NEST [29]	Optical switching	High computational workload	A large-scale distributed machine learning system
Flatter networks [30]	Composable and optical switching	High computational workload	High-performance computing workloads
Geo-DCN [31]	Multiple data centers distributed without data replication	High computational workload	Data centers that are geographically dispersed

Table 1. Cont.

References	Design Strategy	Design Limitations	Application Domain
Renewable Energy DCN [32]	Scheduling algorithm for reinforcement learning (RL)-based jobs	High computational workload	Big data analytics on a geo-distributed data center for DER Reinforcement learning Application
Aggregation DCN [33]	Combination of two aggregators—Two-Chain Topology	Routing complexity	Generic data center on round aggregation top-of-rack DCN
PEFS [34]	Scheduling strategy that is AI-driven, energy-aware, proactive, and fault-tolerant	Delay complexity	Cloud data centers (CDCs)
CALM [35]	Polynomial-time algorithm/collocation-aware survivable placement (CASP)	Resource usage complexity	Cloud data centers (CDCs)
DCell [36]	Hamiltonian-connected	High computational workload	Cloud data centers (CDCs)
Skywalk [37]	Low-latency interconnects	Smaller QoS metric coverage	Large-scale high-performance computing (HPC)
BML-BCube [38]	Distributed gradient synchronization algorithm	High computational workload	Large-scale distributed machine learning (DML)
Generalized Hypercube [39]	Exchanged generalized hypercube (EGH)	High computational workload	Cloud data centers (CDCs)
RRect [40]	Server-centric network linear diameter and scaled parallel paths	High computational workload	Enterprise data center
HSDC [41]	High scalability with hypercube network	High computational workload	Enterprise data center
WaveCube [42]	WaveCube optical multipathing and dynamic bandwidth provisioning	High computational workload	Optical data center networks
BCCC [43]	Diameter linearity	High computational workload	Enterprise server-centric data center
GBC [44]	Examines low-cost off-the-shelf switches and servers with any specified number of NIC interfaces.	High computational workload	Enterprise server-centric data center
LaScaDa [45]	Explores hierarchical row-based routing	High computational workload	Cloud data centers (CDCs)
cRetor [46]	Topology-aware routing scheme. A star algorithm for SDN controllers	High computational workload	Cloud data centers (CDCs)
Proposed DCN	Edge-to-cloud orchestration using the SGDA network	Autonomous/managed computational workload	Smart grid infrastructure/Smart Cities

2.2. Research Gaps in SG DCNs

Several research gaps persist in the design of DCNs for cloud-based services, particularly with handling high computational workloads. A crucial need exists for integrating software-defined networking (SDN) into these designs to enable quick routing updates. This integration would enhance the computational routing behavior, speed up convergence, reduce route overhead, and improve fault tolerance. While significant progress has been made in combining SDN with network function virtualization (NFV) as a multi-objective optimization problem, there remains a gap in deploying SDN effectively within spine-

leaf topologies [47]. Current studies focus on SDN applications such as optical transport networks [48], stateful data networking [49], OpenFlow-based SDN [50], and heuristic SDN [51] within lightweight cloud domains, but SDN's specific optimization in spine-leaf architecture remains underexplored. Furthermore, the use of stochastic gradient descent (SGD) computations for large-scale networks has shown promise in modern DCN designs. However, little effort has been made to apply SGD quality of service (QoS) metrics for optimizing SDN controller engine provisioning [52–58]. To bridge this gap, the introduction of container-based time synchronization models with overhead-controlled SGD presents a novel approach to achieving a lightweight DCN scenario. This paper introduced an overhead-controlled SGD within SD-SL topology as a potential solution to address the current gaps, particularly in optimizing the QoS for large-scale smart grid DCNs.

3. System Model

Figure 1 illustrates the IoT core infrastructure in SGDA for residential units depicting edge-to-cloud integrations. Due to energy workflow sensitivities, we applied mathematical optimization techniques used to handle dynamic load balancing, stability, and scalability. SGD and multivariable calculus tools, particularly the implicit function theorem (IFT), serve as the foundation for these optimizations. The goal of optimization is to minimize network overhead, balance traffic loads, and enhance resource utilization while considering various constraints such as bandwidth, processing capacity, and service delivery requirements. This ecosystem can be formalized with constraints to optimize performance across the SGDA, as shown in Figure 2.

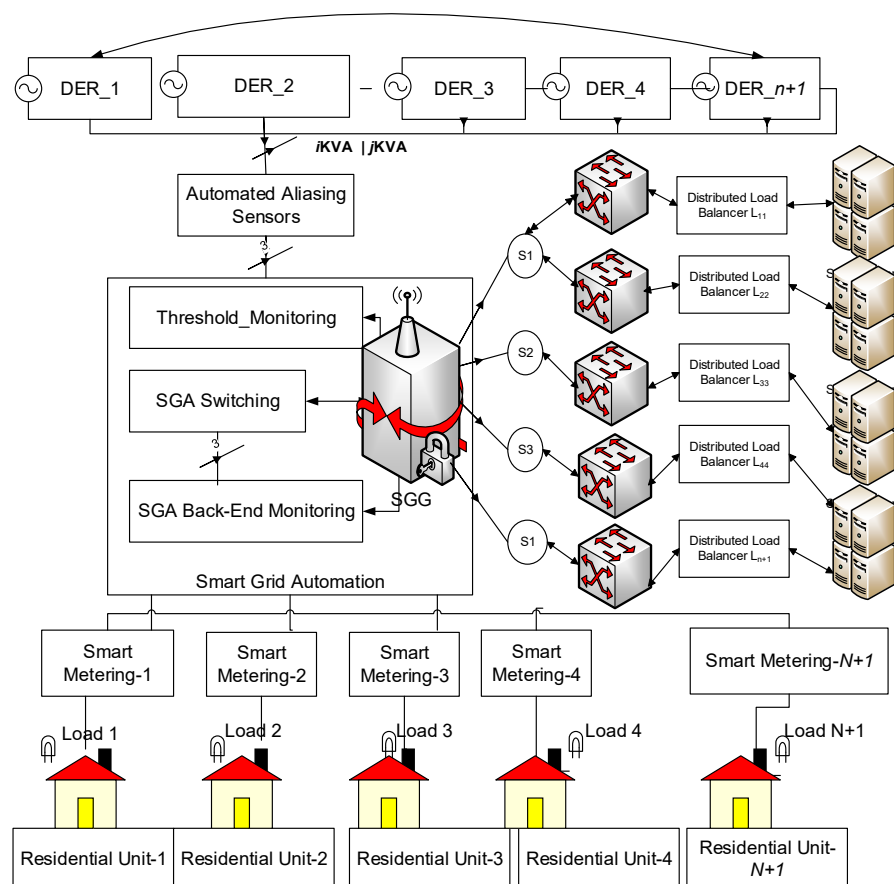


Figure 1. Residential units with layered SGDA with edge-to-cloud interfaces.

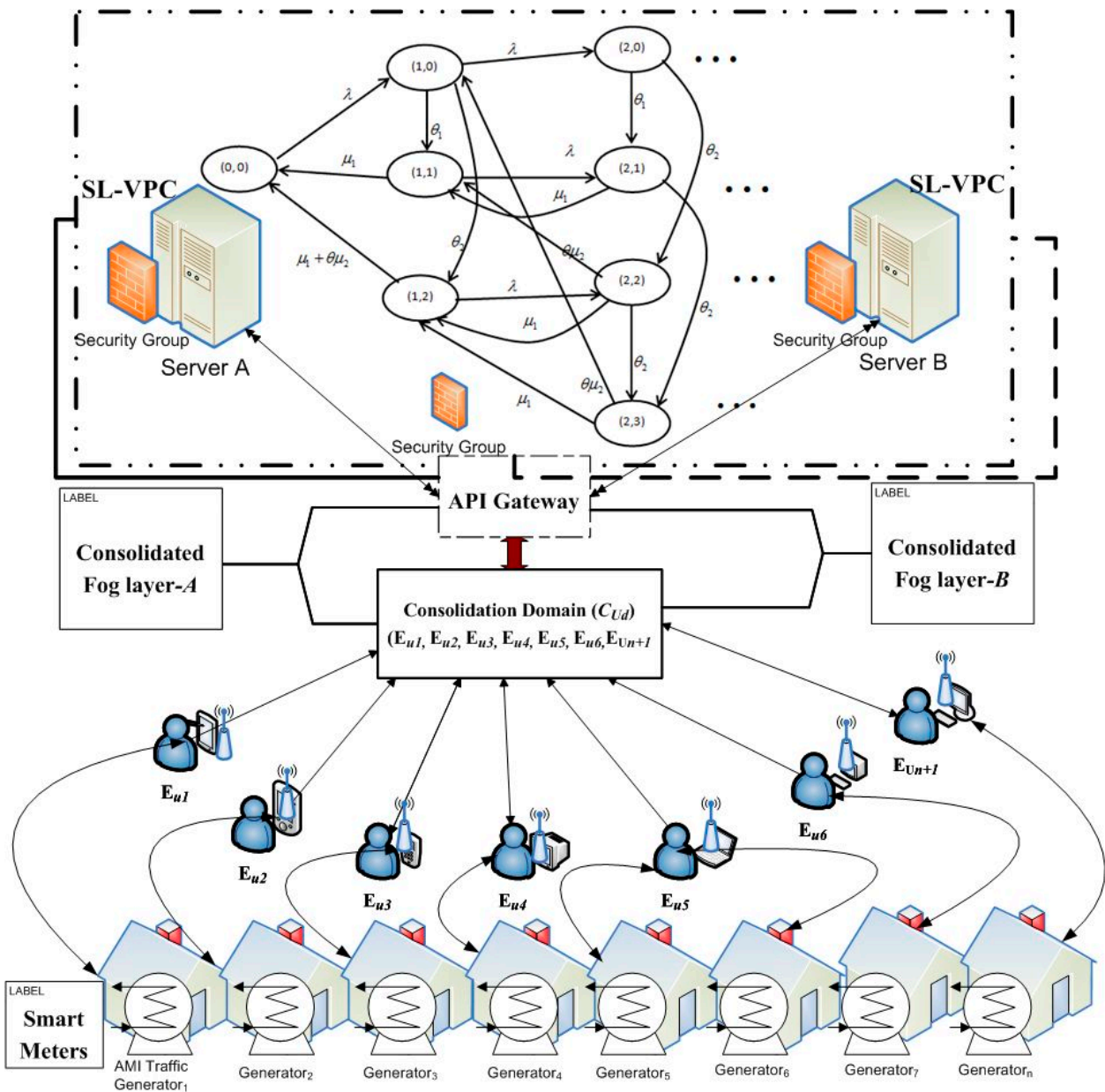


Figure 2. Smart grid edge-to-cloud integration using CTSM multi-queue system for heterogeneous fleet servers.

3.1. Analytical Framework-Scalable CTSM-SGDCN

A CTSM is introduced to a SGDCN to address real-time latency optimization and traffic workload across the SG infrastructure. SDN is integrated to enable intelligent automation of energy data traffic while SGD optimizes parameters dynamically.

3.1.1. Problem Definition and Optimization Formulation

This subsection describes the IoT smart grid infrastructure variables. Let us define an objective function $F(x, y)$, where $x = (x_1, x_2, \dots, x_n)$ represents the QoS network variables such as throughput, latency, or load, and $y = (y_1, y_2, \dots, y_m)$ are the constraints, including bandwidth limits, energy usage, and resource availability. The system is subject to constraints. (x, y) represent the capacity and operational limits of the infrastructure.

The optimization problem can be expressed as Equation (1):

$$\min_{x,y} F(x, y)$$

Subject to:

$$C_1(x, y) \leq b_1, C_2(x, y) \leq b_2, \dots C_m(x, y) \leq b_m \quad (1)$$

where $C_1, C_2, \dots C_m$ are constraint functions, and $b_1, b_2, \dots b_m$ are the upper bounds for these constraints (e.g., bandwidth, traffic capacity, or power consumption limits).

3.1.2. Implicit Function Theorem for Load Balancing

Using the IFT, we define the system of Layer 3 routers and their routing functions for the SL-VPC as Equation (2):

$$r L_i(x_1 \dots x_n, \dots y_1, \dots y_m) = 0, i = 1 \dots R \quad (2)$$

This formulation assumes the routers are part of a virtualized, differentiable system where redundancy and failover can occur in a seamless manner. The IFT allows us to express one variable as a function of others, simplifying the optimization problem, where

$L : \mathbb{R}^{n+m}$ represents a continuously differentiable virtual function across a product space such that $\mathbb{R}^n * \mathbb{R}^m$ forms a Cartesian product of virtual spaces. This model aims to optimize traffic load balancing while managing redundancy within the AMI of a SG network. The redundancy mechanism automatically disconnects nodes from the core internet gateway when multiple paths are available, ensuring efficient resource allocation (Equation (3)).

$$x, y, z = SGSB_{n+1}(x_1 \dots x_n, \dots y_1, \dots y_m). \quad (3)$$

where $SGSB_{n+1}$ denotes transparent failover clusters for load balancing.

Definition 1. *Implicit Transparent Failover Service (ITSF) and Load Balancing.*

The ITSF ensures the existence of differentiable fault-tolerant failover clusters $SGSB_1, SGSB_2, \dots SGSB_{n+1}$. This is required for load balancing across the SGDA network. The implicit function theorem provides the necessary differentiability conditions for achieving this balance in Equation (4). This guarantees uninterrupted energy service provisioning.

3.2. Mathematical Model of Traffic and Stability

The automation service metrics are defined as a tuple:

$$M = (Q, Sd, T, A)$$

where

$Q = \{Q_1, Q_2 \dots Q_n\}$ is a set of QoS metrics (e.g., throughput, traffic availability).

$Sd = (Sd_1, Sd_2, \dots Sd_{n+1})$ represents the SDN-OpenFlow components.

$T = (t_1, \dots t_2, \dots t_{n+1})$ is a set of temporal service invocations.

$A = (A_1, \dots A_2, \dots A_{n+1})$ represents the activities or transactions of end-users.

The CTSM leverages SL-VPN containers to optimize traffic handling and service provisioning through real-time stability monitoring. We now describe the main elements of the model in Equations (4) and (5).

3.2.1. CTSM Traffic Stability Model

i. QoS vector path stability:

To maintain real-time synchronization, the QoS stability equation ensures time-consistent data transmission (for high-density traffic), defined by the following equations.

$$Q_{vp}(t) = \left[\sum_{t=0}^{\infty} (Q_{vp_1}(t) + Q_{vp_2}(t) + \dots + Q_{vp_{n+1}}(t)) \right] \quad (4)$$

This guarantees predictable latency and resilient energy flow across the network.

ii. Stream Workload Path (SWP) Stability

The stream workload model ensures stable data synchronization across the SGDCN:

$$S_{swp}(t) = \left[\sum_{t=0}^{\infty} S_{swp_1}(t) + S_{swp_2}(t) \dots S_{swp_{n+1}}(t) \right] \quad (5)$$

By deploying containerized SDN functions, the network dynamically adjusts to peak loads, mitigating synchronization errors via auto-scaling fault tolerance.

3.2.2. CTSM Auto-Scaling and Traffic Optimization

As shown in Figure 2, we employed load balancing (i.e., Application Load Balancers (ALBs)/Network Load Balancers (NLBs)) to ensure scalability and fault tolerance. The ALB routes the HTTP/HTTPS traffic using WebSocket and content-based routing. The NLB handles TCP/UDP energy transactions, ensuring high throughput and low latency. We then used the auto-scaling group servers to dynamically adjust container instances based on CPU utilization and network request rates. We note a constraint optimization concern for traffic flow. Hence, to maintain synchronization under high-density traffic, the system enforces a constraint:

$$C_m(x, y) \leq b$$

where $C_m(x, y)$ represents network constraints such as bandwidth and latency.

Also, stochastic gradient descent (SGD) with constraints is considered. To dynamically optimize SGDA traffic, SGD updates network parameters using the loss function $J(\theta)$. This measures the deviation from optimal traffic balancing. The SGD update rule is given by Equation (6):

$$\theta^{(K+1)} = \theta^{(k)} - \mu \nabla J((\theta)^{(k)}) \quad (6)$$

where μ is the learning rate, and

$\nabla J J((\theta)^{(k)})$ is the gradient of the loss function with respect to the parameters θ .

This ensures real-time energy data synchronization, thereby improving load balancing efficiency, latency minimization, and resilient failover recovery.

3.2.3. CTSM Multi-Queue System

A multi-queuing model (MQM) for load management via cloud architecture is introduced. Once the edge AMIs generate events via the nodes to the cloud, the queuing system is activated. The CTSM for the SGDA network alongside its AMI is further analyzed to address real-time feedback. The microservice architectural design could benefit from classic segmentation, failure isolation, scalability, and active-active resilience and resource optimization under MQM. We assume our possible scenarios as load balancing across queues, microservices with dedicated queue states, task segmentation services, dynamic allocation via auto-scaling combined with cloud services (e.g., AWS SNS, GCP publish-subscribe, Kafka, etc.).

- Dynamic allocation via auto-scaling

Let us use an auto-scaling use case as an example. Consider an M/M//2 MQM with myriads of complex heterogenous servers entering idle vacation mode whenever there is no workload. In the auto-scaling design, a fleet server S_f goes on vacation when the

others are active, S_{Aa} . Assume we have two heterogeneous servers, S_A and S_B , using the microservice-modified Poisson process parameter λ , and let μ_1 and μ_2 be the service rates of both auto-scaling servers, respectively. Here, $\mu_1 \neq \mu_2$. We assume that the durations of their fleet vacations are independent but distributed with an exponential stochastic variable, θ_1 and θ_2 . During vacation sessions, traffic arrival demands on S_A (a busy server with no fleet vacation) will cause S_B (a less busy server) to provision and service at lower rates.

In this case, the S_B will have a service rate of $\theta\mu_2$, $0 \leq \theta \leq 1$. Once the service is completed, S_B will continue with the expected service rate μ_2 until there is no more queue before it goes on its own fleet vacation. The workload arrival, W_a , from various endpoints follows first-come-first-served (FCFS) traffic discipline from the distribution load balancers.

- **Continuous Time Markov Chain**

From Figure 2, our fleet vacation M/M//2 MQM for S_A and S_B will be formulated with a CTMC [59] whose scenario states at the possible time T are designated by $(i, j)_{AB}$.

Here, $i \geq 0$ represents the number of transactional workloads (TWs) in the design. $j = 0, 1, 2, \dots, n$ represents fleet server status.

The fleet state $S_{i,j}$ ($0, 0$) denotes that both S_A and S_B are on vacation; the fleet state $S_{i,1}$ ($i, 1$) denotes $i \geq 0$, i.e., TWs are in the auto-scaling system.

Given that S_A is in a busy state, and S_B is non-vacation, $S_{i,1}$ ($i, 2$); $i \geq 0$, and the TWs are in the auto-scaling system.

Also, if S_A is in a busy state, and S_B is working during vacation, then $S_{i,1}$ ($i, 3$) $i \geq 0$, and the TWs are in the auto-scaling system. This implies that both S_A and S_B will be busy, working at optimal states.

We can then introduce a CTMC generator Q referred to as a quasi-birth-and-death (QBD) process [60]. The transitional carders $0, 1, \dots, n$ denote the possible epoch states, $\rho[0 = \{(0, 0)\}, 1 = \{(1, 0), (1, 1), (1, 2)\},$ and $i = \{(i, 0), (i, 1), (i, 2), (i, 3)\},$ if $i \geq 2$ [61]. The transition state diagram is also depicted in Figure 2, capturing all the layers of integration applied in our previous edge marketplace robots [62]. Therefore, by employing a dynamic auto-scaling strategy and CTMC, systems can efficiently manage and allocate heterogeneous server resources under varying workloads. This means that servers switch between active service and idle “vacation” modes, with adjustments in service rates (such as a reduced rate for provisioning tasks) governed by stochastic processes and modified Poisson parameters.

Our model provides a robust analytical framework to predict performance and optimize resource utilization when attached with edge nodes. The integration of a scalable container-based time synchronization mechanism (CTSM) with software-defined networking (SDN), implicit transparent failover service (ITSF), and stochastic gradient descent (SGD) in Figure 1 (i.e., SGDCN) ensures fault-tolerant time synchronization, optimizes workload distribution through auto-scaling and load balancing, and reduces latency while improving QoS under high-density energy transactions. By dynamically adjusting network parameters in response to fluctuating grid conditions, CTSM enhances the stability, efficiency, and real-time resilience of SGDCN, ensuring seamless and scalable operations within complex smart grid infrastructures. We shall now describe the functional components in Figure 2, linking their integration with our edge AMIs in Section 4.

4. SGDA Ecosystem

4.1. System Components

4.1.1. Edge Layer

We employed various computational techniques, including exponential, gamma, Bernoulli, and binomial distributions, to model key edge components of the SGDA system

in Figure 2. The exponential distribution was used to represent the life span of generation companies (GENCOs) and independent power plants (IPPs), and the gamma distribution captured the aggregated energy output. By simulating the energy generation over time with Minitab software version 21.1.0, we calculated the energy outputs and probabilities at different intervals, giving insights into energy production patterns. The results highlighted the role of exponential and gamma distributions in understanding the total megawatts produced by different GENCOs. For the distribution of energy to DISCOs, we applied Bernoulli and binomial distributions. The Bernoulli distribution was used to model the success or failure of individual smart meter readings, while the binomial distribution modeled the collective readings from multiple meters. These methods helped determine peak and non-peak energy usage, essential for load management.

4.1.2. Fog Layer

At the downstream zone (see Figure 1), the edge-to-fog layer in the SGDA operates as a distributed system, where AMI sensor networks leverage embedded computation for real-time energy monitoring and control. Advanced algorithms, including feedback loops, enable seamless communication with the consolidation layer, optimizing grid load balancing and system management. This layer integrates decentralized meter data management with the operation center and communication interfaces, enhancing control. A service-edge algorithm automates edge processes, enabling SG distribution automation to interconnect AMI sensors and field equipment for remote monitoring, control, and data aggregation. Dynamic interactions between energy users and the SG are supported through the CTSM dynamic dispatch mode, ensuring efficient data exchange across network elements. This is vital for secure, reliable SG communication. Additionally, the cloud-based, containerized infrastructure enables seamless east–west data stream migration from edge to cloud, achieving low-latency and high-throughput performance in SG networks.

4.1.3. Consolidation Domain (DC)

The CD integrates the CTSM within the SL-VPC to enhance computational efficiency and synchronize operations across containerized environments. As shown in Figure 1, this data-centric fabric management and automation layer optimizes performance by incorporating latency prediction within the SL-VPC. This domain plays a critical role in handling AMI data streams, where low latency and optimized traffic flow are essential. The SL-VPC backbone ensures uniform traffic distribution and minimizes latency using predictive workload balancing. Equal-cost multipath (ECMP) routing further enhances efficiency by distributing traffic evenly across multiple paths, mitigating bottlenecks and preventing loops. Its modular design scales dynamically, addressing port density challenges and ensuring seamless SG operations as traffic demands grow. The VPC and application services rely on container-based time synchronization microservices deployed over a stacked VPC. A container orchestrator encapsulates the cloud service stack, enabling structured deployment. SDN cloud functionality services (SDN-CFS) provision essential components, including the SG API gateway and OpenFlow controller interfaces, which manage network traffic and service functions. Load balancers and the OpenFlow console interface facilitate comprehensive flow table logging, covering operations like insertion, removal, updates, and data retrieval. The OpenFlow-enabled infrastructure functions as a firewall, static load balancer (SLB), convergence switch, and a hybrid volume bulk balancer (VBB) and volume billing container (VBC). Flow table configurations, managed by the OpenFlow controller, govern service deployment and ensure efficient network operations.

4.1.4. Mininet CloudFormation IaC

A container optimization strategy is designed to reduce computational overhead in legacy SGDA. An SDN-enabled container, built with infrastructure as code (IaC), manages the consolidation layer (Figure 1), optimizing resource allocation and workload management within SGDA. Using AWS CloudFormation, the Mininet framework enables CTSM validation, ensuring experimental accuracy. In the SL-VPC domain, IaC automates container deployment across multiple instances, creating a resilient environment for continuous integration and delivery. The AWS cloud architecture orchestrates these containers, ensuring scalability, reliability, and automated provisioning. Each container image functions as a lightweight, isolated service, bundling essential components like SG cloud code, runtime, libraries, and configurations. Once deployed, these images instantiate containers within the SDN-CFS engine, maintaining a consistent runtime environment. Additionally, the containerized infrastructure isolates key components such as gateways and load balancers, promoting network stability and uniformity across instances.

4.1.5. Security Group Firewalling

The backend security layer in the DCN design employs the SG OpenFlow firewall (Figure 2) to protect load control and management servers. This edge-to-cloud security framework leverages logical virtualization segmentation, creating tiered security functions within the OpenFlow table map services. These functions are virtualized and consolidated into a virtual security node, enhancing network flexibility and security. The firewall enforces access control policies, continuously monitoring legitimate and illegitimate connections in real time. Using containerized security instances, the system dynamically scales to accommodate increasing workloads. The OpenFlow firewall gateway manages firewall operations, service load balancing, and flow table security policies, ensuring controlled network access. For enhanced resilience, redundant security enforcement mechanisms integrate with ALBs and auto-scaling. ALBs distribute traffic across security-hardened instances, preventing overload attacks, while auto-scaling adjusts security resources dynamically to mitigate denial-of-service (DoS) risks. The CTSM embeds security at the edge AMI and SGDA, ensuring real-time threat detection and anomaly response.

4.2. SGDA AMI Hardware Architecture and System Components

The SGDA AMI (advanced metering infrastructure) edge layer integrates multiple components to facilitate real-time grid monitoring, control, and data aggregation. The key elements are as follows:

1. Transmission and Step-Down Transformers
 - Primary Setup: Three transmission transformers (models *A*, *B*, and *C*) are used. In our demonstration setup, each transformer steps down 240 V to 12 V.
 - Grid Voltage Adaptation: Although actual grids operate with power ratings ranging from 11 kVA to 133 kVA, our configuration employs a three-phase transformer to step down a primary voltage of 415 V to 240 V per phase before additional local step-downs.
2. Sensing and Data Acquisition
 - Sensor Deployment: Sensors continuously monitor power parameters—current and voltage—across both transmission and distribution networks.
 - Distribution-Level Monitoring: At the distribution level (DISCO), smart meters equipped with load-scheduling sensors provide enhanced monitoring of power consumption.

3. System Layers and Communication: The SG system operates through three integrated layers:
 - Metering Layer: Captures and transmits grid data from various points in the network.
 - Monitoring/Control Layer: Processes and analyses real-time grid parameters to support immediate operational decisions.
 - Cloud/IoT Interface: Utilizes wireless IoT modules and a dedicated VPC landing zone to support demand-side management (DSM) and provide centralized control.
4. Distribution Network Control
 - Relay Operations: The distribution network employs relays to route power efficiently to consumer blocks (see Figure 3).
 - Local Controller: A PIC18F4550 microcontroller manages DISCO operations. It interfaces with the following:
 - Current Sensing: An ACS712-30 sensor measures current usage.
 - Voltage Monitoring: A voltage divider circuit tracks voltage consumption, with data fed into the microcontroller's ADC to compute and display load demand.
 - RF Communication: An RF IoT module transmits real-time data to the cloud, enabling remote relay activation and precise load management.
5. Real-Time Feedback and Quality of Service (QoS)
 - The AMI edge layer facilitates dynamic feedback between generation companies (GENCOs) and residential consumers (see Figure 4).
 - After validating the accuracy of our smart grid neural network model, it was deployed in the design testbed. This implementation meets the QoS requirements by optimizing transmission capacity and minimizing time delays for critical services such as load management.
6. Advanced Data Aggregation and Algorithmic Processing

To ensure robust performance, key algorithms within the SG system include the following:

 - Traffic-Token Construction
 - Data Encryption
 - Sink Data Aggregation (SGDA)

For developing SGDA process node models, a conjugate batch gradient method was adopted instead of stochastic gradient descent (SGD). This method was chosen because it does not require access to the full training dataset from customer interface units (CIUs) and AMI node sets. The algorithm works as follows:

- **Initialization:** Start with an arbitrary CIU node vector, denoted as ϑ^0 .
- **Iterative Process:**
At each iteration i , a CIU row, $i(k)$ (where $i \in \{1, \dots, n\}$), is randomly selected from the layered cluster parent nodes.
- **Gradient Computation:**
The selected data streams are used to compute the gradient based on the local loss function, $\partial(x_i, y_i)$, as detailed in Algorithm 1.
- **Convergence:**
The procedure iteratively projects onto available hyper-node planes until convergence is achieved.

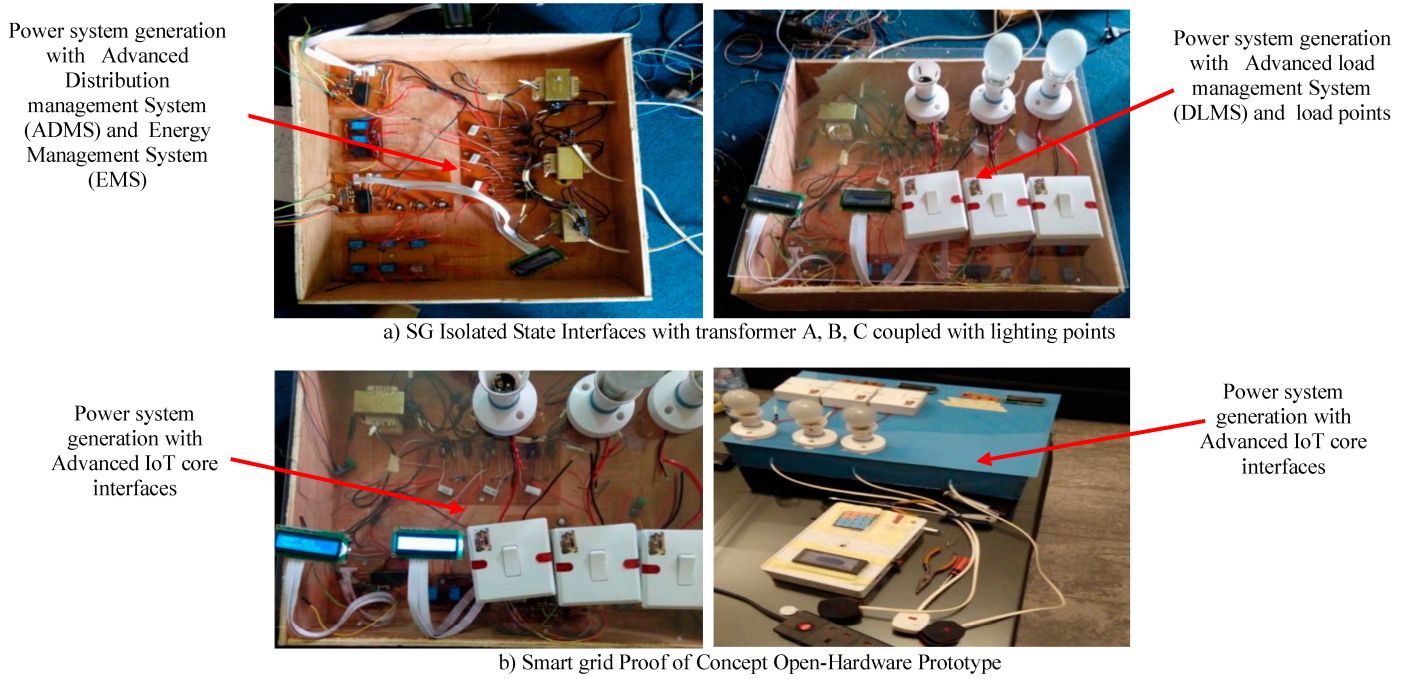


Figure 3. (a,b): Implementation of the load management AMI hardware in SGDA.

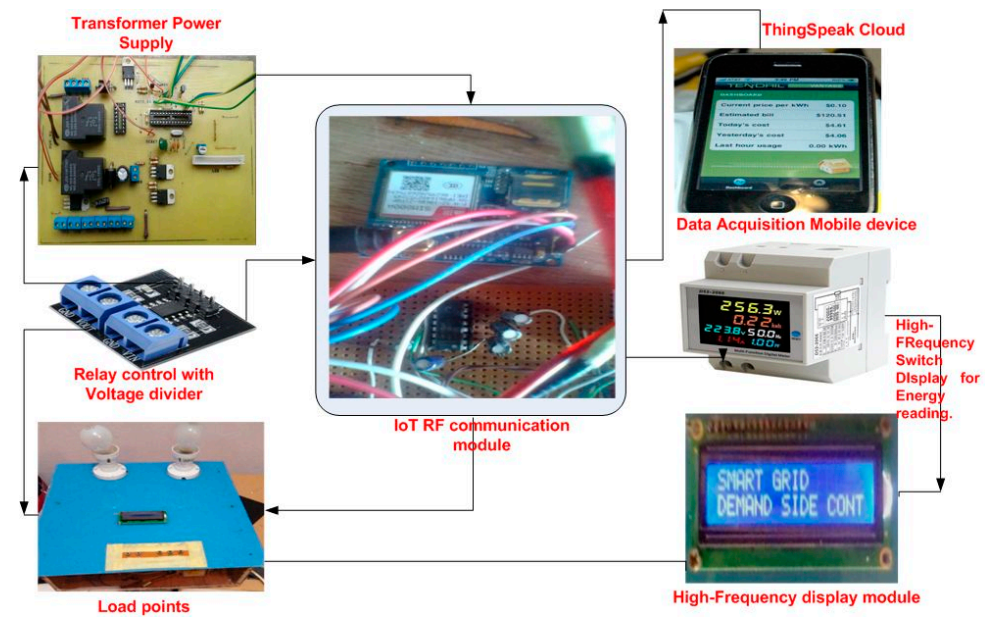


Figure 4. Proof-of-concept Advanced Metering Infrastructure (AMI) that employs full-duplex computational modeling of energy generation and distribution. This model utilizes exponential, gamma, Bernoulli, and binomial distributions to simulate GENCO lifespan, aggregated energy output, and smart meter reading accuracy for dynamic load management in the cloud. The SG system comprises key components such as smart load control switching modules, voltage and current sensors, and IoT RF communication modules, which monitor and manage electrical parameters while facilitating real-time data exchange. Enclosed edge aggregation boxes with both disabled and active load points organize and control distributed energy resources. Data acquisition mobile devices gather operational data, and high-frequency display modules provide energy readings and system status updates, enabling informed decision-making and effective grid management.

In practice, a continuous time Markov chain algorithm function is established. The algorithm outlines the customer interface unit (CIU) connection process at the edge layer. This takes the source ID, s , destination ID, d , and data size, m as inputs, and outputs a data

stream, n , arriving at the local aggregator, L_{ag} . Through iterative processing, Algorithm 1 sets the CIU and AMI destinations as vectors of different lengths while setting the cluster parent as AMI transmitter. To integrate stochastic gradient descent (SGD) variables into Algorithm 1, we can use them to optimize the decision-making process, particularly in determining the optimal data flow paths between CIU and AMI based on iterative learning. In this design, SGD is used to iteratively update the weight vector ω_i based on the gradient of the loss function $L(\omega_i)$. The learning rate η controls how much the weights are adjusted in each iteration, optimizing the connection between CIUs and AMI for better data flow efficiency.

Algorithm 1: Edge Connection with SGD Optimization

1: CIU Input: (s, d, m)
2: CIU Output: n
3: Initialize learning rate η for SGD Optimisation
4: Initialise weight vector ω_0 for AMI-CIU Connection
5: Determine (CIU +AMI Destination) as vectors of different lengths.
6: For (CIU = 0, CIU > 0, i++) **then**
7: Compute Gradient: $\nabla_{\omega_i} L(\omega_i)$ based on current load and connection
8: Update Weight: $\omega_i + 1 = \omega_i - \eta$.
9: Set Cluster Parent as (AMI Tx);: $\nabla_{\omega_i} L(\omega_i)$ (SGD Step)
10 Set Cluster parent as (AMI Tx) based on updated weights
11: End loop when convergence is achieved
12: End

4.3. Hardware Implementation

As described in Section 4.2, the edge AMI integration is now depicted in Figure 3, illustrating the load management within the SGDA architecture. The system integrates a PIC-based IoT microcontroller with a 10-bit ADC, converting sensor signals into digital data to calculate power consumption from current and voltage measurements. The PIC18F4550 controller manages GENCO operations, controlling power flow by energizing or de-energizing relays connected to the distribution bus. A display dashboard visualizes real-time power data, while a GSM module and RF IoT module establish secure cloud communication.

- i. System activation and control are accomplished with the following:
 - GENCO sources (A, B, C) under testing, displaying status on an LCD interface.
 - The RF IoT module configuration for real-time communication via activation start buttons which energizes relays, and routes power to the distribution buses.
 - The cloud-driven LCD displays available power and real-time load status.
- ii. Dynamic load management and smart control
 - Load variations trigger cloud-based automation as follows:
 - (a) As demand increases, the DISCO reports to the cloud control unit, which activates additional GENCOs to meet consumption needs (Figure 4).
 - (b) If demand exceeds capacity, the system implements smart load shedding, de-energizing relays to disconnect non-priority loads.
 - (c) When demand drops, the system takes GENCOs offline, preventing overproduction.
- iii. Neural network-driven self-healing optimizes power allocation, ensuring efficient demand–supply balancing while minimizing energy wastage. This seamless inte-

gration of SGDA hardware, IoT, and cloud infrastructure can improve real-time monitoring, control, and energy efficiency in next-generation smart grids.

4.4. Neural Edge Network Design

This section describes the design and implementation of the proposed SGDA leveraging a neural engine to predict energy consumption patterns, as shown in Figure 5. We assumed subscribers/customers at GENCOs/TCN/DISCOs. MATLAB R2022a was used to integrate preprocessing, feature selection, clustering, and a supervised learning algorithm for dynamic load management.

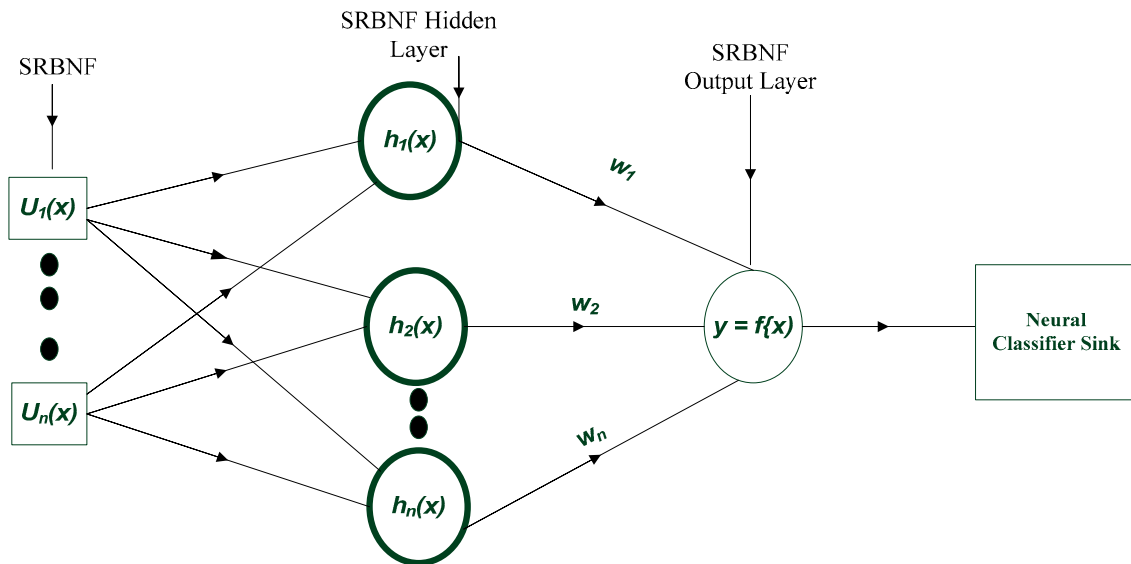


Figure 5. Computation of neural controller architecture for SG architecture.

4.4.1. Network Architecture

The core predictive model is based on a radial basis function network. In our implementation, the network consists of an input layer, a hidden layer, and an output layer.

In the hidden layer, the input vector is linearly mapped to a set of basis functions using weights w_i . The network's output is then defined as:

$$y = f(x) - W(x) = \sum_{i=1}^n w_i \varphi_i(x) \quad (7)$$

Specifically, each subscriber's data are represented as an input vector:

$$u = (u_1, u_2, u_3, \dots, u_n)^T \in \mathfrak{R}^n \quad (8)$$

where

$\varphi_i(x)$ = output of the i -th basis function

w_i = corresponding weight of node i

h = number of hidden nodes.

For classification, the network's output is constrained within the interval (0,1) and is further transformed by:

$$y = w(x) = \begin{cases} 1 & \text{if subscriber's load is at peak} \\ 0 & \text{if subscriber's load Non - peak} \end{cases} \quad (9)$$

4.4.2. Preprocessing and Learning Algorithm

Before training, the data undergo several preprocessing steps, such as the following:

- Dimensionality Reduction: To eliminate redundant features.
- Data Normalization: Scaling all input features to lie between 0 and 1, forming a consistent input-target matrix.
- Clustering: Pre-filtering the data into clusters helps in detecting anomalies such as faulty demand meters or abnormal customer behavior.

Following preprocessing, the learning algorithm employs the Stochastic Gradient Neural Process Controller (SG-NNPC), combined with an integrated forecasting engine to predict future energy demand. The dataset is randomly partitioned into training and testing sets, and model parameters are optimized to minimize the mean squared error (MSE). During training, the model iteratively computes hidden cluster centers—adjusting for inter-sample similarity—to refine its predictions. The overall performance is benchmarked using MSE, which for each hidden layer is calculated as:

$$\hat{\theta}_h = \frac{1}{s \times n_T} \sum_{r=1}^S \sum_{i=1}^{n_T} I(y_i^r \neq \hat{y}_i^r) \quad (10)$$

where

s = number of cross-validation runs;

n_T = the total number of test samples;

$I(\cdot)$ = Indicator function that returns 1 when the predicted value differs from the true value, and 0 otherwise;

h = number of hidden nodes.

4.4.3. Experimental Setup and Performance Evaluation

The SGDA system was evaluated using data samples collected from a DISCO in a simulated SG environment. In our experimental configuration (see Table 2), the neural network was set up with the following parameters:

Table 2. Parameters of SG neural network.

Generator	Discriminator Values
Input units	4
Output units	1
Activation	Levenberg_Marquardt
Hidden layers	10
Optimization goal	MSE (minimum square error)
Training epoch	56
Classifier output	25

Training was further validated using a 3-fold cross-validation process. As the number of AMI node samples increased, the model demonstrated improved learning, achieving the best validation performance at the 50th epoch with an MSE of 2.7359×10^{-11} . This low error margin indicates robust performance and a prediction accuracy of nearly 100%. Figure 5 illustrates the overall neural controller architecture, while Figure 6 presents the MSE plot for the SG edge neural network model.

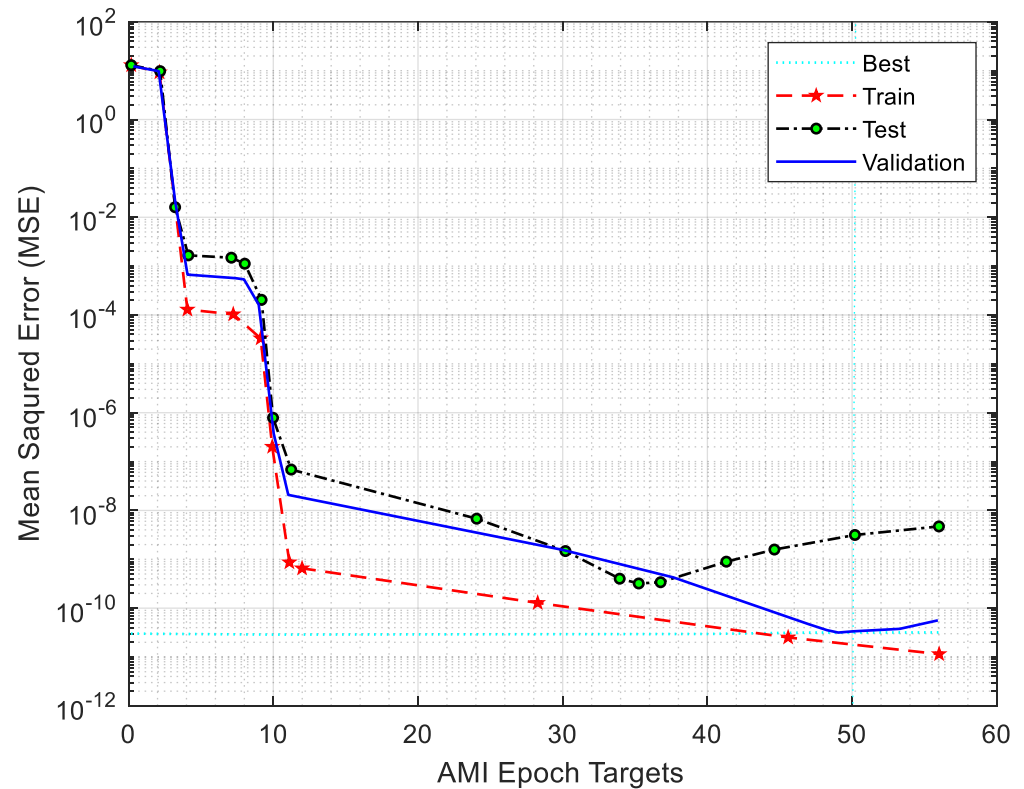


Figure 6. Mean square error plot for SG edge neural network model.

4.4.4. Data Aggregation via AMI Local Concentrator

In addition to prediction, the SGDA incorporates a data aggregation mechanism using an AMI local concentrator. Algorithm 2 outlines the procedural steps for the following:

- Executing batches of conjugate gradient updates concurrently across AMI nodes.
- Aggregating the data by averaging values computed for each iteration $\eta_{(k+1)}$.
- Synchronizing node operations to ensure efficient data flow and scalability.

The AMI local concentrator not only reduces computational overhead but also ensures that data streams—regardless of queue length—are effectively managed. This coordinated aggregation supports the dynamic load management required in smart grid environments.

4.5. Computational Complexity Analysis

The time complexity analysis of the CTSM involves mapping between the AMI user layer and the related service orchestrations for temporal data stream provisioning. The key components are as follows:

Let

N_u represent the port of the SDN user network feature, matching the number of CTSM services;

N_i represent the scope of service concurrent attributes, which matches the number of instantiating nodes;

E_u represent the scope of embedded security attributes;

E_i represent the complexity of SDN OpenFlow entries;

$$N_{max} = \text{Max}_j(N_u, N_i), \text{ and } E_{Max} = \text{Max}_k(E_u, E_i). \quad (11)$$

Using Equation (11), the time complexity of the CTSM-connected layered algorithms (Algorithms 1–7) for temporal complexity embedding is $O(N_u E_u + N_i E_i) = O(N_{Max}, E_{Max})$.

Let d denote the size of the SDN hidden databases/tables, ϑ denote the size of the QoS parameters, and the time complexity of CTSM become $O((d^2 + E_{Maxd}) * \vartheta)$.

The time complexity analysis (TCA) is given by

$$TCA = (\vartheta d^2 + \vartheta d E_{Max} + N_{Max} E_{Max}) \quad (12)$$

Given that the SDN identification of d and E_{Max} is proportional to N_{Max} , the TCA can be simplified to

$$O(\vartheta N_{Max}^2) \quad (13)$$

In the production environment, this analysis intrinsically justifies the importance of minimizing QoS metrics to reduce computational workload.

Algorithm 2: AMI Local Concentrator/Local Aggregator

```

1:   Input: local ID, destination ID, queue size, link Information
2:   Output: Infinite queue dispatched to the Global sink
   Procedure:
3:   Draw AM local Connection;
4:   AMI_Global Connection j
5:   Initialise  $i$  iterations T,  $\vartheta^k \leftarrow 0$ ;  $\vartheta^k$  (CIU and AMI)
6:   Map smart meter data of the individual nodes
7:   While all data(i) that hasn't been converged, do
8:   For all  $i$  (AMI)  $\in \{0, \dots, j + 1\}$  do read (i);
9:   For  $i := 0$  to  $N - 1$  do read ( $\varphi[i]$ );
10:  For  $i := 0$  to  $N - 1$  do read ( $\partial[i]$ );
11:  For  $i := 0$  to  $N - 1$  do read ( $n_{k+1}[i]$ );
12:  For  $j := 0$  to  $N - 1$  do read  $\vartheta r[i] = \varphi([i]); + \partial([i]); + \dots (n_{k+1}[i]);$ 
13:  For  $i := 0$  to  $N - 1$  do Write ( $j[i]$ );
       $\eta^{k+1}(\text{CIU and AMI}) = \frac{1}{N} \sum_{i=1}^N \eta^k$ 
14:  End

```

Algorithm 3: Container logical instantiation/Global Grid Concentrator

```

1: Input: grid workload sources
2: Output: autoscaling group instances
3: Create an Object instance:
   Instantiate an object from a map.
4: For node Iteration
   For each node  $i$  from I to  $N_{k+1}$ 
5: Call Open Firewall Function to add
    $F(I_0, J_1, K_1, N_{k+1}; \text{sg-link: sg-link})$ 
   sg-link
6: load container balancer
   Load balancer self-connection
7: Recycle loop ().
   execute recycle loop ()
8: Direct control
   directly control the container resources
9: End

```

Algorithm 4: SDN OpenFlow Firewall Services

1: Input: Call Schedule (), OpenFlow firewall services (), services bundling (), Source address, a destination address, 2 queue size, link, link Information

2: Output: OpenFlow Destination (Load balancer ports)

3: Parameters: OpenFlow_weight \leftarrow Empty, weight \leftarrow 0; weighted Moving \leftarrow 0; totalWeight \leftarrow 0;

4: OpenFlow_weight_Container_history_queue \leftarrow null;

5: $i \leftarrow 0$;

6: While < OpenFlow monitor Call Schedule do

7: queue \leftarrow (HistoryListSize—OpenFlow_monitor Call — i)

8: OpenFlow_weight \leftarrow fiboA1 + fiboB2;

9: OpenFlow weighted Moving \leftarrow weighted Moving+ (Container historyItem * weight);

10: Recycle loop;

11: end while

12: Calculate event filtering () & execute dynamic network balancing;

13: Legitimate initial Value \leftarrow (OpenFlow weight) * (pastInitialValue + trendPosteriorValue);

14: illegitimateTrendPosteriorValue \leftarrow (initialValue—pastInitialValue)+ (PosteriorValue)

15: If (Sensed event = 1), *then*,

16: Set services (),

17: Create another instance of the virtual node on OpenFlow;

18: Optimise flow table;

19: Do list control ();

End

Algorithms 3 and 4 were used to drop the impacts of data availability threats against the SGDA AMI.

Edge Streaming Algorithm

In Algorithm 5, once a smart meter joins a subnet group, binomial distribution is activated to optimize node placement. If a sink node is within range, the smart meter receives a TCP data stream message from the AMI base hub and assigns the sink node as the cluster parent. Nodes validate messages based on their level: higher-level nodes connect with different clusters, while lower-level nodes only interact within their cluster and with the parent node. Edge computing is utilized to enhance real-time processing, enabling each node to locally manage and detect associated AMI nodes. Cluster parent nodes use this edge-enabled infrastructure to create secure network data across all anchor nodes in the AMI subnet clusters.

Algorithm 7 begins by receiving an AMI network message and its type, then sends the output to the sink AMI local concentrator. For each iteration from 1 to K_n+1 , it processes encrypted data by gathering a random neighbor, waiting for a response from the AMI cluster node, and computing a new encryption key. It then sets the direction and curve level, applies homomorphic encryption, and packs the encrypted data. After waiting and preparing to send the data to the local aggregator, it checks if the construction is complete and, if so, sends the final data to the local concentrator.

Algorithm 5: CTSM Secured Network Construction

```

1:  Input: AMI Network design message
2:  Output: output in the Sink placed in the Cluster Servers
Procedure:
3:  Set CTSM VPC Encryption = 1
4:  If (a node is a source node) then
5:      Exit Call (AMI base hub)
6:      Perform Flooding (initial Level, base stationID)
7:      Wait for the TCP HELLO message to arrive
8:  If (a Smart Meter sensor node sends a message to the AMI node) then
9:      Set the message's parentID, recHopCnt, and recLevel
10:     Increment Net Information curEntry
11:     If (current hop count > recHopCnt + 1) then
12:         Set current hop count = recHopCnt + 1
13:     If (current hop count > recHopCnt + 1) then
14:         Break
15:     If (TOS LOCAL ADDRESS is not a leaf node) then
16:         Perform Inundation (current Level, current Node ID)
17:         If (message Type is Encrypted) then
18:             JOIN
19:             If (the maximum number of child nodes is not exceeded by the parent node) then
20:                 Set parent = parentID
21:             Else
22:                 Send message (RESET) to a node
23:             End If
24:         End If
25:     End If
26: End If
27: End If
28: End

```

Algorithm 6: CTSM Encrypted Data Construction

```

1:  Input: Network Message, Message type
2:  Output: output Sent to Sink AMI local concentrator
Procedure:
3:  For i = 1 to  $K_n+1$  do
4:      Obtain Data that Has Been Encrypted
5:      AggregateNode = gather neighbor(random)
6:      Wait for the AMI cluster Node to provide a response message
7:      newValue = ComputeKey(AMI cluster Node, KeySeed data, Received_KeySeed data)
8:      make Direction = directionValue(newValue)
9:      current CurveLevel = setCurveLevel
10:     Homomorphic Curve = encryptedData(direction, curveLevel, newUpdate)
11:     Packing(encryptedData)
12:     Wait
13:     Ready to send to the local aggregator
14:     If (the construction is complete) then
15:         Send to Local concentrator
16:     End If
17: End For
End

```

In Algorithm 7, TCP energy data flooding secures data aggregation from the AMI nodes to the SDN controller. The process begins with the AMI nodes. In smart grid mode, data from the CIUs are received by an AMI base node, which re-encrypts them along with its own data. The encrypted data are then sent from the source nodes to the global concentrator, enforcing encryption with specific key lengths during the transfer to the master AMI nodes. Before encrypted communication is sent to the sink node, the AMI nodes must be activated. To transmit upstream and downstream messages to the sink, both local and global concentrators must meet the encryption and decryption requirements. Upon receiving the event message, the sink node triggers the local concentrator function. Data from the edge nodes, collected by the local concentrator, are fully encrypted before being transmitted to the grid via the OpenFlow SL-VPC. Anomaly detection is confirmed using the TCP encryption filtering technique. Legitimate traffic arriving at the user AMI master nodes is decrypted while any illegitimate traffic is rejected.

Algorithm 7: CTSM Aggregated Data Security

1. **Receive Input:**
 - Get **AMI Network Message** and **Message Type** as input.
 2. **Send to Sink AMI:**
 - Send the **Message** and **Message Type** to the **Sink AMI Node** (AMI meters).
 3. **Iterate through Nodes:**
 - **For** each node $N(i)$, where $i = 1$ to K_n+1 (K_n is the number of AMI nodes):
 1. Apply the **Data Aggregation** function to the **Message** and **msgType**.
 2. **If** a metering node is set:
 - Send the **encrypted message** to the **Sink AMI Node**.
 3. **Else:**
 - **If** an AMI node receives an **update** message from the **receiver sensor node**:
 1. Connect to the **local concentrator (DMF)**.
 2. **If** an **event message** is encoded and received by the **local concentrator**:
 - Store the **encrypted data** (using **Homomorphic encryption**).
 4. **Decryption and Data Aggregation:**
 - Apply the **Decryption** function to the encrypted data:
decryptedData = Decrypt(encData)
 - Add the **decryptedData** to the **aggregatedData**:
aggregatedData += decryptedData
 5. **Global Concentrator Interaction:**
 - Retrieve data from the **Global Concentrator**.
 - Apply the homomorphic encryption curve:
HomomorphicCurve = newEncData(direction, curveLevel, aggregatedData)
 6. **Local Concentrator Completion:**
 - *If* all data is received from the **local concentrator child nodes**:
 1. Send an **encrypted message** to the **Grid OpenFlow Firewall ParentNode**.
 7. **Sink Node Actions:**
 - *If* the node is a **Sink Node**:
 1. Filter for **TCP encryption abnormalities**.
 2. Store the **encrypted data** from the message.
 8. **Final Decryption:**
 - Decrypt the **encData**:
decryptedData = Decrypt(encData)
 9. **Send to Master Station:**
 - Send the **decryptedData** to the **Customer Smart Meter Master Stations**.
 10. **End.**
-

4.6. Performance Evaluation

4.6.1. DCN Description

Given the edge-to-cloud integration, the performance of SG data aggregation (SGDA) was evaluated using a lightweight, container-based, time-synchronized spine-leaf data center deployed with AWS CloudFormation stack infrastructure. The assessment compared five traditional DCN topologies with the proposed SGDA, considering performance under stochastic gradient descent dependencies. Six DCN topologies—CTSM SL-VPC, DCell, Mesh, Skywalk, Dahu, and Ficonn—were subjected to performance benchmarking, as shown in Figure 7. To simulate real-world SGDCN traffic, Mininet was customized to replay *pcap* packet capture files, modeling SL-VPC traffic variations with SDN-CFS.

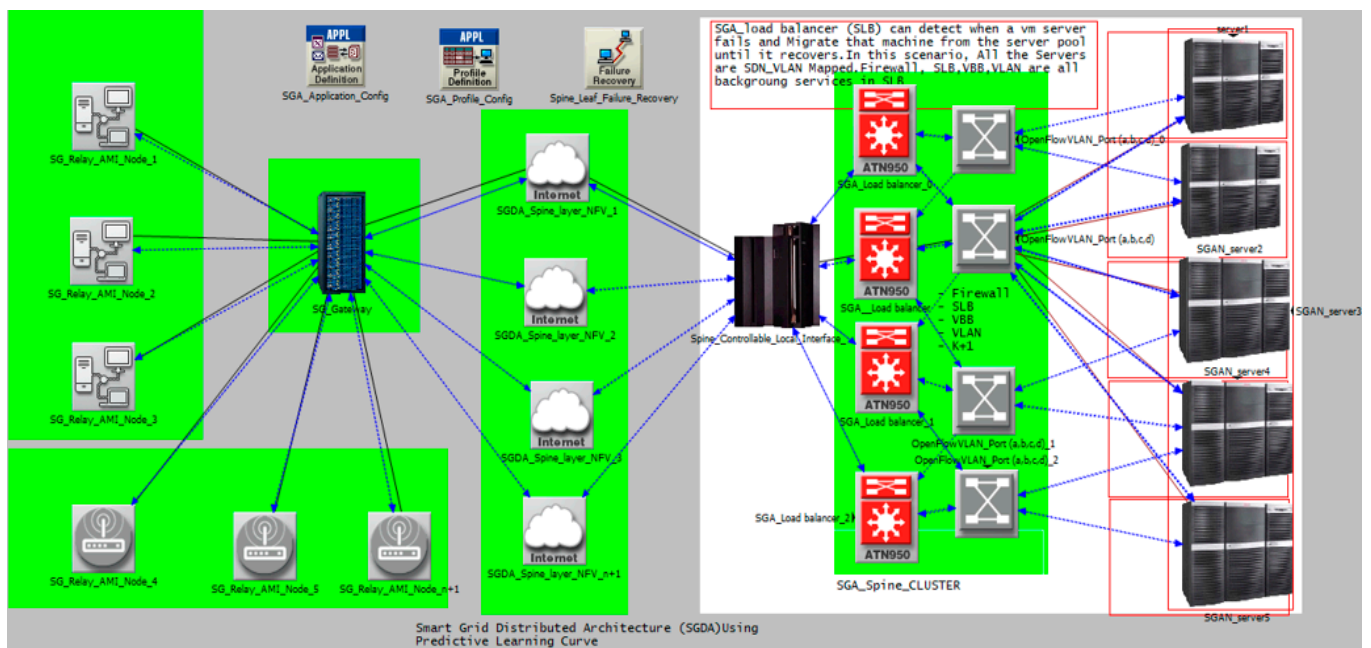


Figure 7. Simulated SGDA implementation. The edge-to-cloud AMI experiments were conducted on an EC2 HPC testbed featuring Intel Xeon Gold 6132 CPUs, NVIDIA GeForce GTX 1080Ti GPUs, and 192GB of RAM. We used Python 3.7.4 and PyTorch 1.1.0 to implement the CTSM modules on the EC2 HPC infrastructure. A Cisco Nexus 7700 core switch with 18 slots managed network connectivity, supporting up to 768×1 and 10 Gigabit Ethernet ports, 384×40 Gigabit Ethernet ports, and 192×100 Gigabit Ethernet ports, which efficiently handled the SG workloads and automation processes.

Performance analysis was conducted on Amazon EC2 HPC instances, running a kernel-level SDN switch and application code execution. For AMI sensor and field device integration, CISCO 7705 SAR-HC and SAR-We devices were used, enabling remote monitoring, control, and SG data aggregation. The SDN controller dynamically recalculated routes using reinforcement learning, optimizing data streams based on experiential patterns. Key SGDA performance indicators included service delays, throughput, energy data reception, cryptographic overhead, and service traffic availability.

4.6.2. Evaluation Results

The CTSM SL-VPN architecture utilized the shortest path routing protocol with an SDN controller, as shown in Figure 8. Our results indicate that the SGDA collects energy data efficiently from the AMIs. The data transmission to the cloud data center network is consistent at both peak and off-peak load periods, supporting analytics tasks like load management, billing, and auditing. The observed data transmission process is highly reliable, nearing 100%, due to robust mechanisms such as full-duplex synchronization and

beacon collision avoidance among AMI nodes, local concentrators, and the cloud OpenFlow gateway. Load scheduling within the SGDA distributes load in varying percentages across various network architectures. The CTSM SL-VPN architecture achieved the highest reliability at 29.87%, followed by DCell at 19.48%, Mesh at 22.08%, Skywalk at 5.19%, Dahu at 15.58%, and Ficonn at 7.8%. This indicates that the CTSM SL-VPN architecture performs better than other DCN schemes in terms of reliable data stream reception and stability at the cloud, especially during load shifts from peak to off-peak periods. This improved reliability and stability are promising for effectively reducing utility bills and managing peak loads.

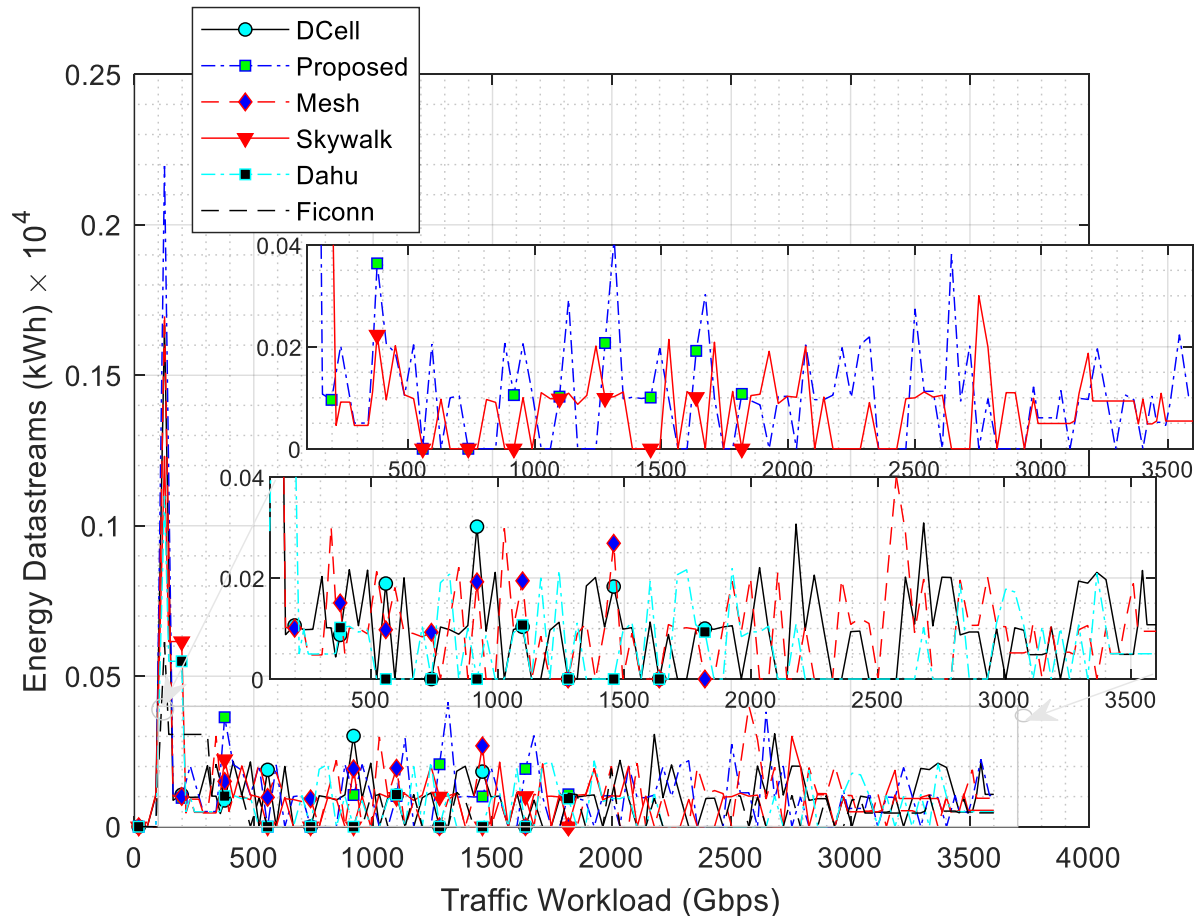


Figure 8. SGDA energy data received response.

Figure 9 shows the packetization service delays for energy data streams in the SGDA. This measures the total time needed to transmit data from the AMI to the cloud. These delays are crucial as they affect SG communication performance. Analysis using the platform statistics engine revealed the following packetization service delays for different SGDAs: CTSM SL-VPC had a delay of 13.11%, DCell 21.31%, Mesh 19.67%, Skywalk 18.03%, Dahu 16.39%, and Ficonn 11.49% during load scheduling on the SGDA. These results indicate that the CTSM SL-VPC architecture exhibits lower packetization service delays compared to other SGDA schemes, especially as energy demands transition from peak to off-peak periods. This performance improvement is promising for reducing utility bills and easing peak loads in the grid network.

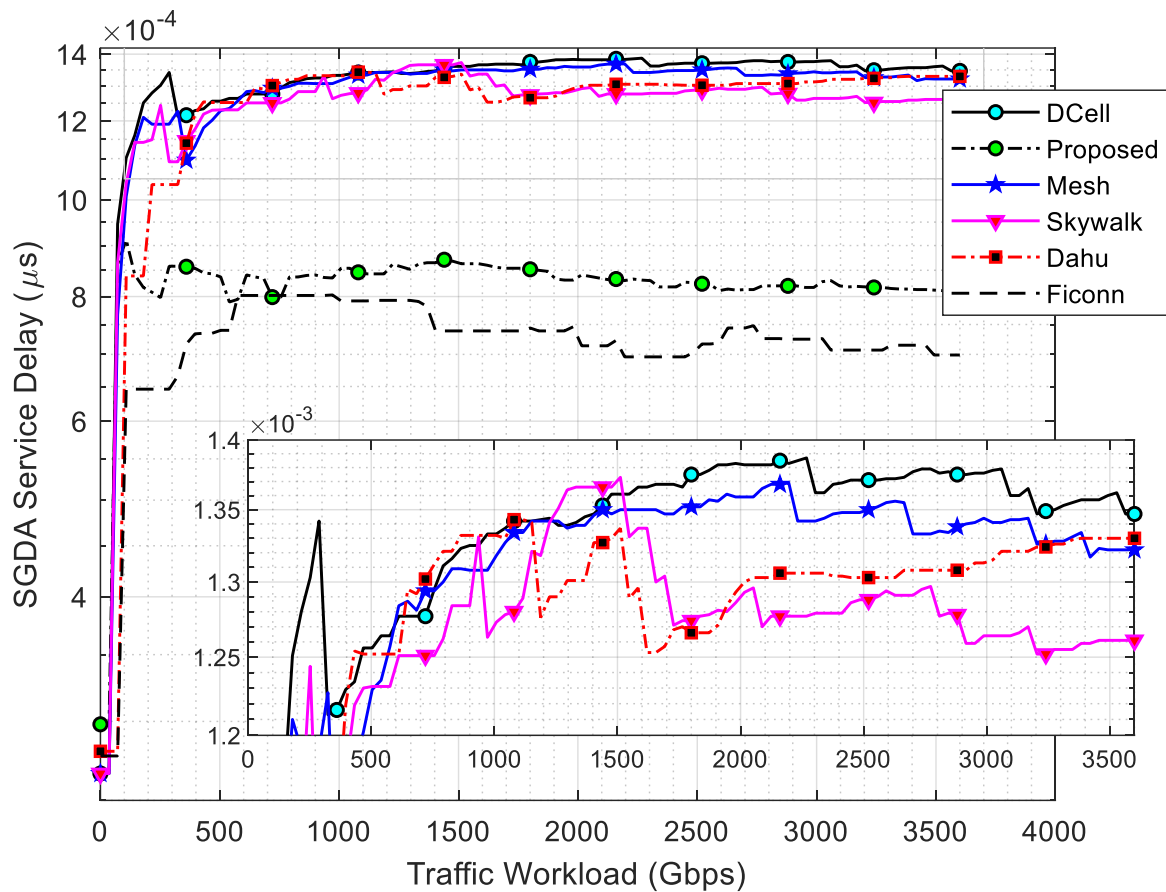


Figure 9. SGDA service delay response.

Figure 10 highlights the SGDA media access delays on DCN engine switching interfaces. It provides the time-frame between two successive resource allocations to likely related users during grid load management. With the daemon API, the Swarm orchestrator can update services using service parameters. During load scheduling on the SGDA, it was observed from the riverbed statistics engine that the CTSMs SL-VPN, DCell, Mesh, Skywalk, Dahu, and Ficonn gave 10.99%, 27.47%, 24.91%, 18.32%, 14.65%, and 3.66%, respectively. This implies that container isolation offers a reliable workload. Figure 8 indicates that due to the stateless container engine operated by the PLC, the proposed CTSM spine–leaf utilized optimum resources on the grid network as compared to other schemes as load demands in peak periods were moved to off-peak periods. This will make the objective of reducing administrative overhead feasible in the cloud.

Figure 11 shows the SGDA service throughput response under the PLC scheme. Regardless of SGDA DCN service limitations such as physical medium, assaults, computing power, and traffic protocols, the highest possible throughput is always chosen. When using SGDA in load scheduling mode, it was observed from the riverbed statistics engine that the CTSMs SL-VPN, DCell, Mesh, Skywalk, Dahu, and Ficonn offered 27.27%, 21.21%, 19.70%, 16.67%, 9.15%, and 3.33%, respectively. This means that when load demands at peak times were shifted to off-peak times, the proposed CTSM SL-VPN utilized optimum resources at scale while predictively managing the grid network when compared to other SGDA topologies.

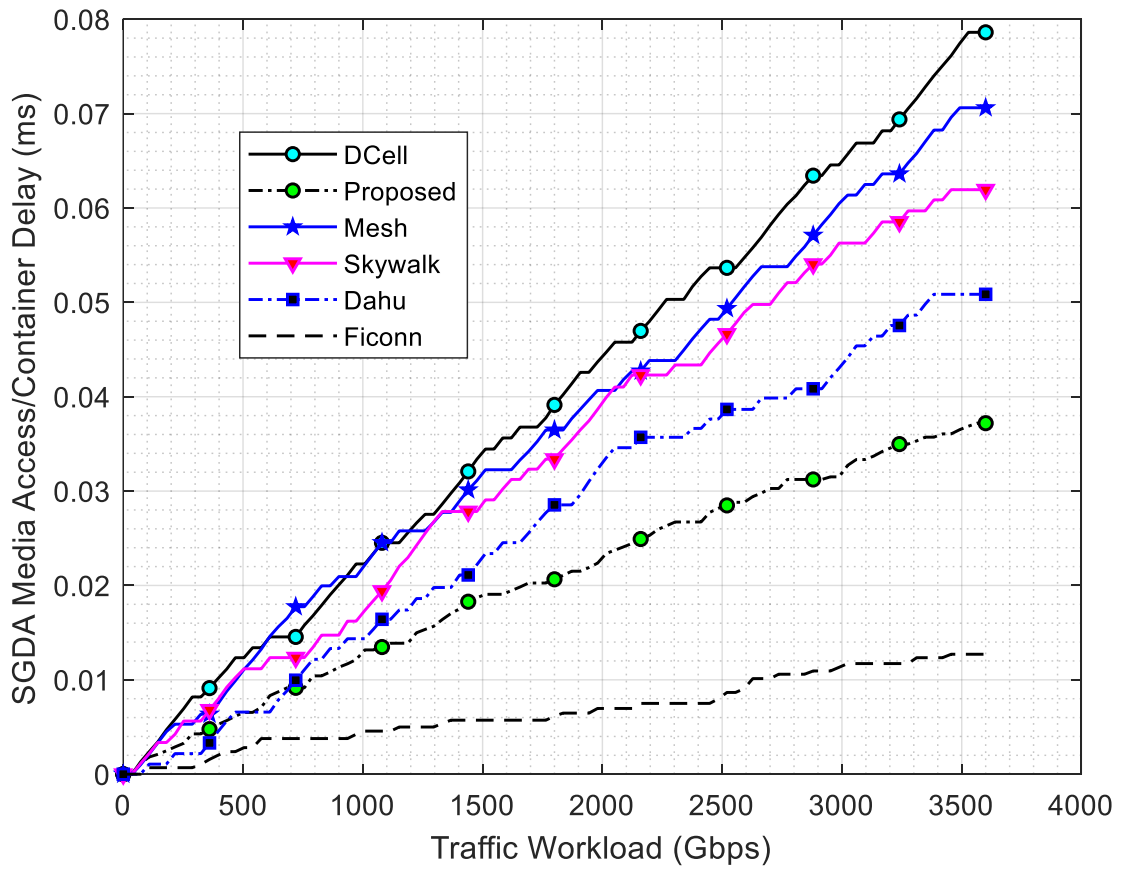


Figure 10. SGDA media access delay response.

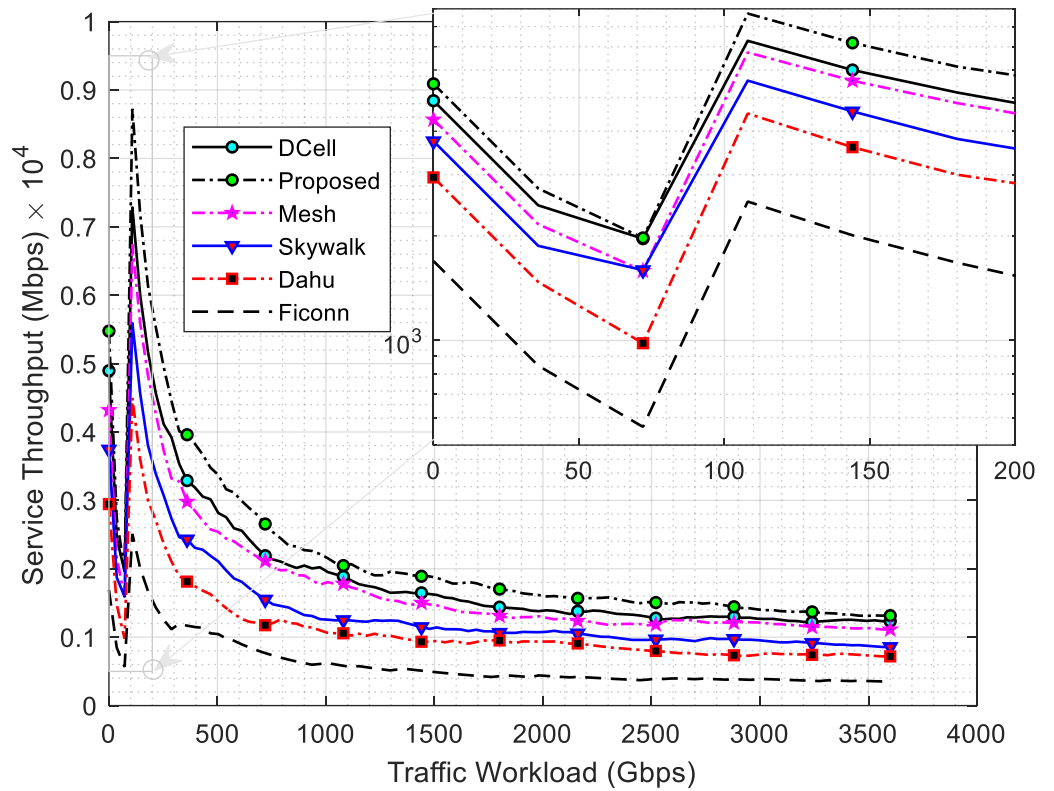


Figure 11. SGDA service throughput response.

Figure 12 highlights the SGDA availability from a fault-failure injection mode perspective. Since the system is autonomous, at full-scale subscription, three major SG security schemes were used to test for availability. In context, availability is the most important security objective in SGDA service provisioning. Therefore, establishing that availability in SGDA communications deals with various security types for network availability, a secured fault injection scheme is needed. Owing to the importance of providing seamless service from SGDA AMI power systems, protection against various attacks on the CTSM SL-VPN algorithm was explored. The CTSM scheme continuously monitors the state of the grid; for example, disruption, instability, etc. During load scheduling on the SG network, it was observed from the riverbed statistics engine that the proposed CTSM, SGDA IPv6 gateway, and CTSM SL-VPN access had 70.85%, 20.17%, and 8.98%, respectively. This implies that as load demands in the peak periods were shifted to the off-peak periods, the proposed CTSM provided optimum availability when compared to other SGDA schemes.

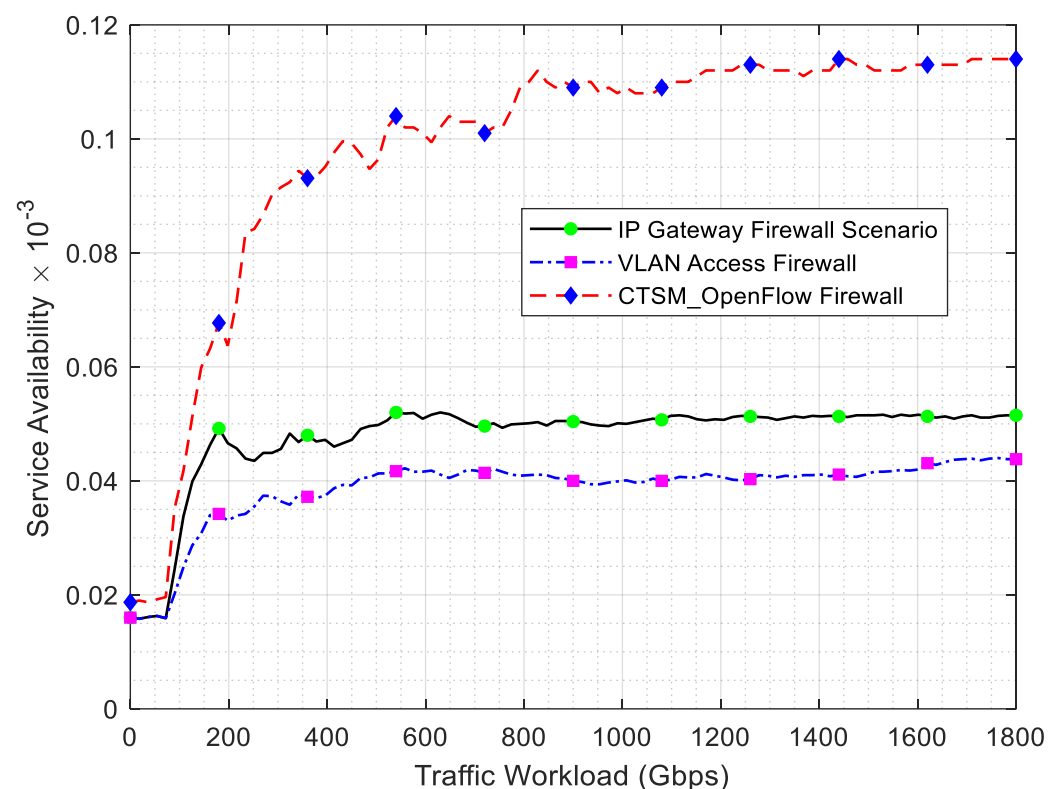


Figure 12. SGDA traffic availability response.

Figure 13 shows the SGDA encryption–decryption overhead subscription under the influence of these security schemes, viz. proposed CTSM, SGDA IPv6 gateway, and SGDA-VPC access. The proposed CTSM SL-VPN, SGDA IPv6 gateway, and SGDA-VPC access had 28.13%, 37.5%, and 34.37%, respectively, during load scheduling on the SGDA network, according to the riverbed statistics engine observation.

The implication is that as load demands in peak hours were transferred to off-peak periods, the projected CTSM SL-VPN offered the least overhead when compared to alternative schemes. This makes the goal of protecting the grid from attack vectors and payloads a lot more realistic.

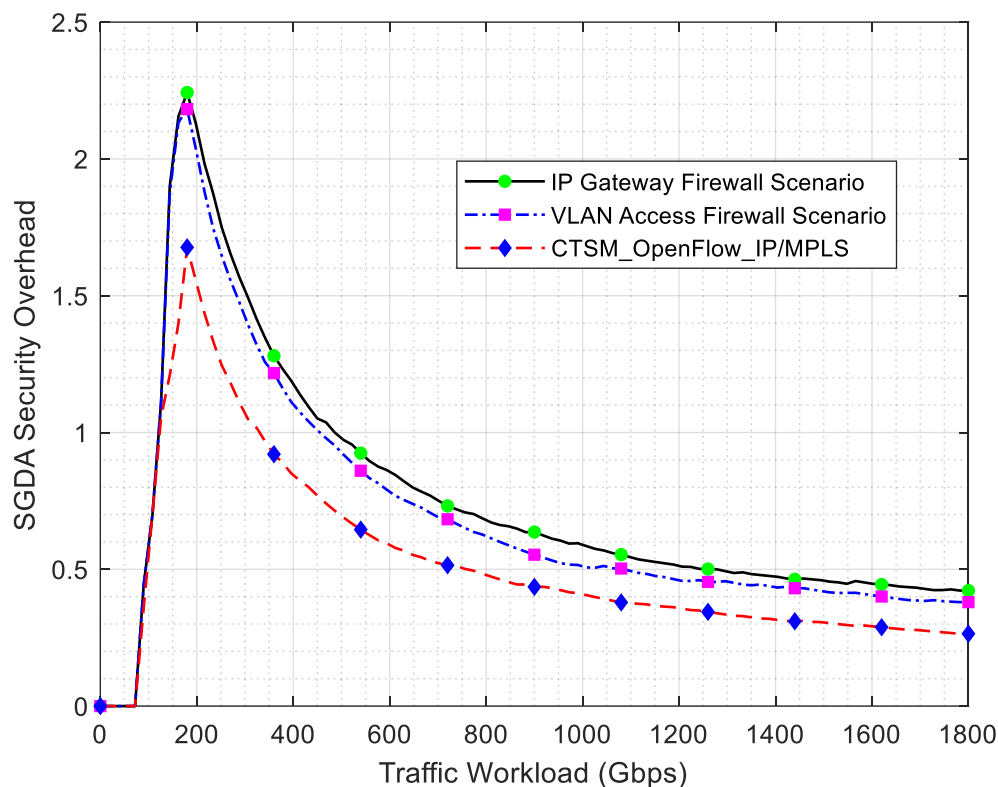


Figure 13. SGDA security overhead response.

Table 3 shows how load scheduling impacts SGDCN infrastructure. The sensitivity analysis interpretation is discussed below:

1. Received Energy Data: The CTSM SL-VPN architecture exhibits the highest percentage of received energy data (29.87%), indicating better energy consumption and efficiency compared to the other architectures. Sensitivity in context suggests that optimizing energy data reception is crucial for system performance, especially in high-demand SG systems.
2. Packetization Service Delays: The CTSM SL-VPN shows the lowest packetization delay (13.11%), making it the most sensitive to minimizing delays in packetization. Lower packetization delays result in a more responsive system, which is needed for AMI edge data processing.
3. Load Balancer Access Delays: The Ficonn architecture has the lowest load balancer access delay (3.66%), while the DCell architecture exhibits the highest (27.47%). This indicates that architectures with lower access delays can better handle dynamic traffic and high-volume requests, making them more sensitive to load balancing efficiency.
4. Service Throughput: CTSM SL-VPN leads with the highest service throughput (27.27%), which is significant for ensuring the efficient delivery of high-bandwidth services. Sensitivity in service throughput highlights the importance of network architectures with a high capacity to maintain performance under heavy loads.
5. Traffic Availability: CTSM SL-VPN also demonstrates high traffic availability (70.85%), which ensures stable network connections. Sensitivity in traffic availability means that architectures supporting greater availability are more resilient to disruptions, contributing to continuous edge-to-cloud service delivery.
6. Encryption–Decryption Overhead: SGDA IPv6 Gateway and SGDA-SL-VPC have the highest encryption–decryption overheads (37.50% and 34.37%), indicating these architectures are more sensitive to security-related processing costs. Minimizing

encryption overhead is crucial for optimizing performance, especially in our edge-to-cloud ecosystem (see Figure 2).

Table 3. Performance comparison of network architectures in SG systems.

Metrics	CTSM Spine-Leaf (%)	Dcell (%)	Mesh (%)	Skywalk (%)	Dahu (%)	Ficonn (%)	SGDA IPv6 Gateway (%)	SGDA-SL-VPC (%)
Received Energy Data kilowatt-hours (kWh)	29.87	19.48	22.08	5.19	15.58	7.8	17.07	11.69
Packetization Service Delays (μ s)	13.11	21.31	19.67	18.03	16.39	11.49	15.75	14.35
Load Balancer Access Delays (ms)	10.99	27.47	24.91	18.32	14.65	3.66	20.18	16.49
Service Throughput (Mbps)	27.27	21.21	19.70	16.67	9.15	3.33	14.92	11.24
Traffic Availability (Mbps)	70.85	37.19	30.45	25.81	18.34	10.91	20.17	8.98
Encryption–Decryption Overhead (Mbps)	28.13	33.44	30.93	26.19	22.91	12.78	37.50	34.37

The results show that the proposed CTSM SL-VPN architecture outperforms others in several selected key metrics. From the sensitivity analysis, our proposal is particularly effective in high-density SG environments due to its balance in energy efficiency and low latency. Conversely, architectures like Ficonn are more sensitive to load balancing. We argue that the proposed design shows higher sensitivity to encryption overhead, suggesting areas for optimization in future design.

5. Conclusions

This paper presented a container-based time synchronization spine–leaf data center network (DCN) for smart grid infrastructure, utilizing container stack technology. An efficient container time synchronization model (CTSM) for optimizing service provisioning in the SGDA was introduced. The design addresses key challenges in big data queuing management, and automation concerns. Furthermore, an adaptable stochastic gradient descent (SGD) computational model for the workload SL-VPC was created and experimented and found to be adaptable across multiple tiers. This paper has used an SDN-driven SGD approach to tackle issues inherent in both switch-centric and server-centric DCN designs for SGDCNs. Through a comparative study of five different topologies, the proposed approach was validated. The results demonstrate that the architecture proposed outperforms existing designs such as DCell, Mesh, Skywalk, Dahu, and Ficonn. Our results showcase the efficiency of the proposed design in the management of traffic workflows while maintaining satisfactory quality of service (QoS) metrics from edge to cloud.

Author Contributions: K.C.O.: writing—review and editing, conceptualization, formal analysis, validation, supervision, methodology, and funding acquisition. W.O.O.: writing—review and editing. O.M.L.: writing—review and editing, supervision. I.I.A.: writing—review and editing. K.A.: writing—review and editing, supervision, funding acquisition. B.A.: writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Federal Government of Nigeria through TETFUND (Tertiary Education Trust Fund), project award TETF/ES/UNIV/IMO STATE/TSAS/2021, issued in February 2022.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All data generated or analyzed in this study are included in this manuscript, except the programming codes, which can be released upon reasonable request to the corresponding author.

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

Abbreviations

SGDA	Smart Grid Distributed Architecture
DCN	Data Center Network
SDN	Software-Defined Network
SDN-CFS	Software-Defined CloudFormation Stack
CTSM	Container-based Time Synchronization Model
SL-VPC	Spine–Leaf Virtual Private Cloud
MQM	Multi-Queuing Model
LaScaDa	Layered Scalable Data Center
OSPF	Open Shortest Path First
ECMP	Equal-Cost Multipath Routing
BGP	Border Gateway Protocol
TRILL	Transparent Interconnection of Lots of Links
SPB	Shortest Path Bridging
IFT	Implicit Function Theorem
AMI	Advanced Metering Infrastructure
GENCOs	Generation Companies
ITSF	Implicit Transparent Failover Service
ALB	Application Load Balancers
NLB	Network Load Balancer
CTMC	Continuous Time Markov Chain
QBD	Quasi-Birth-and-Death Process
IPP	Independent Power Plants
DISCOs	Distribution Companies
ECMP	Equal-Cost Multipath
VBB	Volume Bulk Balancer
VBC	Volume Billing Container
AWS	Amazon Web Services
IaC	Infrastructure as Code
ALB	Application Load Balancers
DSM	Demand-Side Management
CIU	Customer Interface Unit
SRBNF	Subscriber Radial Basis Neural Network Function
SGRBNF	Stochastic Gradient Radial Basis Neural Function
SG-NNPC	Stochastic Gradient Neural Network Process Controller
AMI-SGDA	AMI Intelligent Stochastic Gradient Descent Algorithm

References

1. Zhao, L.; Li, Q.; Ding, G. An Intelligent Web-Based Energy Management System for Distributed Energy Resources Integration and Optimization. *J. Web Eng.* **2024**, *23*, 165–195. [[CrossRef](#)]
2. Liu, S.; Chen, L.; Lai, J. Integrated and Accountable Data Sharing for Smart Grids with Fog and Dual-Blockchain Assistance. *IEEE Trans. Ind. Inform.* **2024**, *20*, 4940–4952. [[CrossRef](#)]
3. Zhao, S.; Xu, S.; Han, S.; Ren, S.; Wang, Y.; Chen, Z.; Chen, X.; Lin, J.; Liu, W. PPMM-DA: Privacy-Preserving Multidimensional and Multisubset Data Aggregation with Differential Privacy for Fog-Based Smart Grids. *IEEE Internet Things J.* **2024**, *11*, 6096–6110. [[CrossRef](#)]

4. Zhang, Z.; Ma, J.; Mao, S.; Liang, P.; Liu, Q. A Low-Overhead and Real-Time Service Recovery Mechanism for Data Center Networks with SDN. In Proceedings of the 2024 International Conference on Networking and Network Applications (NaNA), Yinchuan, China, 9–12 August 2024; pp. 184–189.
5. Jie, X.; Han, J.; Chen, G.; Wang, H.; Hong, P.; Xue, K. CACC: A Congestion-Aware Control Mechanism to Reduce INT Overhead and PFC Pause Delay. *IEEE Trans. Netw. Serv. Manag.* **2024**, *21*, 6382–6397. [[CrossRef](#)]
6. Kathiravelu, P.; Zaiman, Z.; Gichoya, J.; Veiga, L.; Banerjee, I. Towards an internet-scale overlay network for latency-aware decentralized workflows at the edge. *Comput. Netw.* **2022**, *203*, 108654, ISSN 1389-1286. [[CrossRef](#)] [[PubMed](#)]
7. Jiang, W.; Ren, F.; Wu, Y.; Lin, C.; Stojmenovic, I. Analysis of Backward Congestion Notification with Delay for Enhanced Ethernet Networks. *IEEE Trans. Comput.* **2014**, *63*, 2674–2684. [[CrossRef](#)]
8. Welzl, M.; Islam, S.; von Stephanides, M. Real-Time TCP Packet Loss Prediction Using Machine Learning. *IEEE Access* **2024**, *12*, 159622–159634. [[CrossRef](#)]
9. Meng, Q.; Zhang, Y.; Zhang, S.; Wang, Z.; Zhang, T.; Luo, H.; Ren, F. Switch-Assistant Loss Recovery for RDMA Transport Control. *IEEE/ACM Trans. Netw.* **2024**, *32*, 2069–2084. [[CrossRef](#)]
10. Chouikhi, S.; Esseghir, M.; Meerghem-Boulahia, L. Energy Consumption Scheduling as a Fog Computing Service in Smart Grid. *IEEE Trans. Serv. Comput.* **2023**, *16*, 1144–1157. [[CrossRef](#)]
11. Bi, J.; Zhang, K.; Yuan, H.; Zhang, J. Energy-Efficient Computation Offloading for Static and Dynamic Applications in Hybrid Mobile Edge Cloud System. *IEEE Trans. Sustain. Comput.* **2023**, *8*, 232–244. [[CrossRef](#)]
12. Vahdat, A.; Al-Fares, M.; Farrington, N.; Mysore, R.N.; Porter, G.; Radhakrishnan, S. Scale-Out Networking in the Data Center. *IEEE Micro* **2010**, *30*, 29–41. [[CrossRef](#)]
13. Radhakrishnan, S.; Tewari, M.; Kapoor, R.; Porter, G.; Vahdat, A. Dahu: Commodity switches for direct connect data center networks. In Proceedings of the Architectures for Networking and Communications Systems, San Jose, CA, USA, 21–22 October 2013; pp. 59–70. [[CrossRef](#)]
14. Alqahtani, J.; Hamdaoui, B. Rethinking Fat-Tree Topology Design for Cloud Data Centers. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6. [[CrossRef](#)]
15. Misra, S.; Mondal, A.; Khajjayam, S. Dynamic Big-Data Broadcast in Fat-Tree Data Center Networks with Mobile IoT Devices. *IEEE Syst. J.* **2019**, *13*, 2898–2905. [[CrossRef](#)]
16. Jiang, W.; Qi, J.; Yu, J.X.; Huang, J.; Zhang, R. HyperX: A Scalable Hypergraph Framework. *IEEE Trans. Knowl. Data Eng.* **2019**, *31*, 909–922. [[CrossRef](#)]
17. Al-Fares, M.; Loukissas, A.; Vahdat, A. A Scalable, Commodity Data Center Network Architecture. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 63–74. [[CrossRef](#)]
18. Greenberg, A.; Hamilton, J.R.; Jain, N.; Kandula, S.; Kim, C.; Lahiri, P.; Maltz, D.A.; Patel, P.; Sengupta, S. VL2: A Scalable and Flexible Data Center Network. In Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, Barcelona, Spain, 16–21 August 2009.
19. Xia, Y.; Hamdi, M.; Chao, H.J. A Practical Large-Capacity Three-Stage Buffered Clos-Network Switch Architecture. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 317–328. [[CrossRef](#)]
20. Singla, A.; Hong, C.-Y.; Popa, L.; Godfrey, P.B. Jellyfish: Networking Data Centers Randomly. *arXiv* **2012**, arXiv:1110.1687v3.
21. Ye, X.; Mejia, P.; Yin, Y.; Proietti, R.; Yoo, S.J.B.; Akella, V. DOS—A scalable optical switch for datacenters. In Proceedings of the 2010 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), La Jolla, CA, USA, 25–26 October 2010; pp. 1–12.
22. Guo, Z.; Yang, Y. Collaborative Network Configuration in Hybrid Electrical/Optical Data Center Networks. In Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, 19–23 May 2014; pp. 852–861. [[CrossRef](#)]
23. Guo, D.; Xie, J.; Zhou, X.; Zhu, X.; Wei, W.; Luo, X. Exploiting Efficient and Scalable Shuffle Transfers in Future Data Center Networks. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 997–1009. [[CrossRef](#)]
24. Li, X.; Lung, C.-H.; Majumdar, S. Energy aware green spine switch management for Spine-Leaf datacenter networks. In Proceedings of the IEEE International Conference on Communications (ICC), London, UK, 8–12 June 2015; pp. 116–121. [[CrossRef](#)]
25. Wang, G.; Zhang, Y.; Yu, J.; Ma, M.; Hu, C.; Fan, J.; Zhang, L. HS-DCell: A Highly Scalable DCell-Based Server-Centric Topology for Data Center Networks. *IEEE/ACM Trans. Netw.* **2024**, *32*, 3808–3823. [[CrossRef](#)]
26. Udeze, C.C.; Okafor, K.C.; Okezie, C.C.; Okeke, I.O.; Ezekwe, C.G. Performance Analysis of R-DCN Architecture for Next Generation Web Application Integration. In Proceedings of the 2014 IEEE 6th International Conference on Adaptive Science & Technology (ICAST), Ota, Nigeria, 29–31 October 2014; pp. 1–12.
27. Lv, M.; Fan, J.; Fan, W.; Jia, X. A High-Performant and Server-Centric Based Data Center Network. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 592–605. [[CrossRef](#)]

28. Guo, C.; Lu, G.; Li, D.; Wu, H.; Zhang, X.; Shi, Y.; Tian, C.; Zhang, Y.; Lu, S. BCube: A high performance, server-centric network architecture for modular data centers. In Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, Barcelona, Spain, 16–21 August 2009; pp. 63–74.
29. Lu, Y.; Gu, H.; Yu, X.; Li, P. X-NEST: A Scalable, Flexible, and High-Performance Network Architecture for Distributed Machine Learning. *J. Light. Technol.* **2021**, *39*, 4247–4254. [[CrossRef](#)]
30. Taubenblatt, M.; Maniotis, P.; Tantawi, A. Optics enabled networks and architectures for data center cost and power efficiency. *J. Opt. Commun. Netw.* **2022**, *14*, A41–A49. [[CrossRef](#)]
31. Emara, T.Z.; Huang, J.Z. Distributed Data Strategies to Support Large-Scale Data Analysis Across Geo-Distributed Data Centers. *IEEE Access* **2020**, *8*, 178526–178538. [[CrossRef](#)]
32. Xu, C.; Wang, K.; Li, P.; Xia, R.; Guo, S.; Guo, M. Renewable Energy-Aware Big Data Analytics in Geo-Distributed Data Centers with Reinforcement Learning. *IEEE Trans. Netw. Sci. Eng.* **2020**, *7*, 205–215. [[CrossRef](#)]
33. Das, S.; Sahni, S. Two-Aggregator Topology Optimization Using Single Paths in Data Center Networks. *IEEE Trans. Cloud Comput.* **2021**, *9*, 807–820. [[CrossRef](#)]
34. Marahatta, A.; Xin, Q.; Chi, C.; Zhang, F.; Liu, Z. PEFS: AI-Driven Prediction Based Energy-Aware Fault-Tolerant Scheduling Scheme for Cloud Data Center. *IEEE Trans. Sustain. Comput.* **2020**, *6*, 655–666. [[CrossRef](#)]
35. Lo, H.-Y.; Liao, W. CALM: Survivable Virtual Data Center Allocation in Cloud Networks. *IEEE Trans. Serv. Comput.* **2021**, *14*, 47–57. [[CrossRef](#)]
36. Wang, X.; Erickson, A.; Fan, J.; Jia, X. Hamiltonian Properties of DCell Networks. *Comput. J.* **2015**, *58*, 2944–2955. [[CrossRef](#)]
37. Fujiwara, I.; Koibuchi, M.; Matsutani, H.; Casanova, H. Skywalk: A Topology for HPC Networks with Low-Delay Switches. In Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, 19–23 May 2014; pp. 263–272. [[CrossRef](#)]
38. Wang, S.; Li, D.; Cheng, Y.; Geng, J.; Wang, Y.; Wang, S.; Xia, S.; Wu, J. A Scalable, High-Performance, and Fault-Tolerant Network Architecture for Distributed Machine Learning. *IEEE/ACM Trans. Netw.* **2020**, *28*, 1752–1764. [[CrossRef](#)]
39. Wang, G.; Lin, C.-K.; Fan, J.; Cheng, B.; Jia, X. A Novel Low-Cost Interconnection Architecture Based on the Generalized Hypercube. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 647–662. [[CrossRef](#)]
40. Li, Z.; Yang, Y. RRect: A Novel Server-Centric Data Center Network with High Power Efficiency and Availability. *IEEE Trans. Cloud Comput.* **2020**, *8*, 914–927. [[CrossRef](#)]
41. Zhang, Z.; Deng, Y.; Min, G.; Xie, J.; Yang, L.T.; Zhou, Y. HSDC: A Highly Scalable Data Center Network Architecture for Greater Incremental Scalability. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 1105–1119. [[CrossRef](#)]
42. Chen, K.; Wen, X.; Ma, X.; Chen, Y.; Xia, Y.; Hu, C.; Dong, Q.; Liu, Y. Toward A Scalable, Fault-Tolerant, High-Performance Optical Data Center Architecture. *IEEE/ACM Trans. Netw.* **2017**, *25*, 2281–2294. [[CrossRef](#)]
43. Li, Z.; Guo, Z.; Yang, Y. BCCC: An Expandable Network for Data Centers. *IEEE/ACM Trans. Netw.* **2016**, *24*, 3740–3755. [[CrossRef](#)]
44. Li, Z.; Yang, Y. GBC3: A Versatile Cube-Based Server-Centric Network for Data Centers. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 2895–2910. [[CrossRef](#)]
45. Chkirkbene, Z.; Hadjidj, R.; Fofou, S.; Hamila, R. LaScaDa: A Novel Scalable Topology for Data Center Network. *IEEE/ACM Trans. Netw.* **2020**, *28*, 2051–2064. [[CrossRef](#)]
46. Jia, Z.; Sun, Y.; Liu, Q.; Dai, S.; Liu, C. cRetor: An SDN-Based Routing Scheme for Data Centers with Regular Topologies. *IEEE Access* **2020**, *8*, 116866–116880. [[CrossRef](#)]
47. Zhang, C.; Wang, X.; Dong, A.; Zhao, Y.; Huang, M.; Li, F. Dynamic network service deployment across multiple SDN domains. *Trans. Emerg. Telecommun. Technol.* **2020**, *31*, e3709. [[CrossRef](#)]
48. Montero, R.; Agraz, F.; Pagès, A.; Perelló, J.; Spadaro, S. SDN-based parallel link discovery in optical transport networks. *Trans. Emerg. Telecommun. Technol.* **2019**, *30*, e3512. [[CrossRef](#)]
49. Mahmood, A.; Casetti, C.; Chiasserini, C.F.; Giaccone, P.; Härrri, J. Efficient caching through stateful SDN in named data networking. *Trans. Emerg. Telecommun. Technol.* **2018**, *29*, e3271. [[CrossRef](#)]
50. He, Q.; Wang, X.; Huang, M. OpenFlow-based low-overhead and high-accuracy SDN measurement framework. *Trans. Emerg. Telecommun. Technol.* **2018**, *29*, e3263. [[CrossRef](#)]
51. Singh, A.K.; Maurya, S.; Kumar, N.; Srivastava, S. Heuristic approaches for the reliable SDN controller placement problem. *Trans. Emerg. Telecommun. Technol.* **2020**, *31*, e3761. [[CrossRef](#)]
52. Bonnabel, S. Stochastic Gradient Descent on Riemannian Manifolds. *IEEE Trans. Autom. Control.* **2013**, *58*, 2217–2229. [[CrossRef](#)]
53. Costilla-Enriquez, N.; Weng, Y.; Zhang, B. Combining Newton-Raphson and Stochastic Gradient Descent for Power Flow Analysis. *IEEE Trans. Power Syst.* **2021**, *36*, 514–517. [[CrossRef](#)]
54. Liu, Y.; Huangfu, W.; Zhang, H.; Long, K. An Efficient Stochastic Gradient Descent Algorithm to Maximize the Coverage of Cellular Networks. *IEEE Trans. Wirel. Commun.* **2019**, *18*, 3424–3436. [[CrossRef](#)]
55. Lei, Y.; Hu, T.; Li, G.; Tang, K. Stochastic Gradient Descent for Nonconvex Learning Without Bounded Gradient Assumptions. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *31*, 4394–4400. [[CrossRef](#)]

56. Pu, S.; Olshevsky, A.; Paschalidis, I.C. A Sharp Estimate on the Transient Time of Distributed Stochastic Gradient Descent. *IEEE Trans. Autom. Control* **2021**, *67*, 5900–5915. [[CrossRef](#)]
57. Peng, X.; Li, L.; Wang, F.-Y. Accelerating Minibatch Stochastic Gradient Descent Using Typicality Sampling. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *31*, 4649–4659. [[CrossRef](#)]
58. Luo, X.; Qin, W.; Dong, A.; Sedraoui, K.; Zhou, M. Efficient and High-Quality Recommendations via Momentum-Incorporated Parallel Stochastic Gradient Descent-Based Learning. *IEEE/CAA J. Autom. Sin.* **2021**, *8*, 402–411. [[CrossRef](#)]
59. Lin, L.; Cao, J.; Lam, J.; Rutkowski, L.; Dimirovski, G.M.; Zhu, S. A Bisimulation-Based Foundation for Scale Reductions of Continuous-Time Markov Chains. *IEEE Trans. Autom. Control* **2024**, *69*, 5743–5758. [[CrossRef](#)]
60. Perez, J.F.; Van Houdt, B. Exploiting Restricted Transitions in Quasi-Birth-and-Death Processes. In Proceedings of the IEEE 2009 Sixth International Conference on the Quantitative Evaluation of Systems, Budapest, Hungary, 13–16 September 2009; pp. 123–132. [[CrossRef](#)]
61. Maurya, V.N. Mathematical Modelling and Steady State Performance Analysis of a Markovian Queue with Heterogeneous Servers and Working Vacation. *Am. J. Theor. Appl. Stat.* **2015**, *4*, 1–10. Available online: <https://sciencepublishinggroup.com/article/10.11648/j.ajtas.s.2015040201.11> (accessed on 16 February 2025).
62. Okafor, K.C.; Anoh, K.; Chinebu, T.I.; Adebisi, B.; Chukwudebe, G.A. Mitigating COVID-19 Spread in Closed Populations Using Networked Robots and Internet of Things. *IEEE Internet Things J.* **2024**, *11*, 39424–39434. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.