IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 5 March 2024, accepted 14 May 2024, date of publication 3 June 2024, date of current version 11 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3408672

RESEARCH ARTICLE

SM-FPLF: Link-State Prediction for Software-Defined DCN Power Optimization

MOHAMMED NSAIF[1,2], GERGELY KOVÁSZNAI[3], ALI MALIK[4], AND RUAIRÍ DE FRÉIN[4]

[1]Department of Information Technology, Faculty of Informatics, University of Debrecen, 4032 Debrecen, Hungary
[2]Department of Computer Science, University of Kufa, Kufa 540011, Iraq
[3]Department of Computational Science, Eszterházy Károly Catholic University, 3300 Eger, Hungary
[4]School of Electrical and Electronic Engineering, Technological University Dublin, Dublin, D07 EWV4 Ireland

Corresponding author: Ruairí de Fréin (ruairi.defrein@tudublin.ie)

This work was supported by the Science Foundation Ireland under Grant 13/RC/2077_P2 and Grant 15/SIRG/3459.

ABSTRACT Efficient monitoring systems that optimize resource allocation, reduce energy usage through machine learning and flow aggregation routing techniques, are needed due to the escalating power consumption of data center networks, which, as has been recently reported, account for up to eight percent of global energy consumption, posing environmental operational concerns. We propose a software-defined data-center monitoring algorithm that reduces power consumption by: 1) using a GPU implementation of a Stacked Long Short-Term Memory Recurrent Neural Network (RNN) model for link utilization prediction, thus reducing monitoring overhead; and 2) utilizing a flow aggregation routing algorithm with feedback from online, OpenFlow-powered monitoring and machine learning modules. This combined approach results in a new algorithm called SMart-Fill Prefer Path First (SM-FPLF). In the context of SM-FPLF, the objective of this paper is to compare the: 1) training and validation loss curves for various models; 2) to evaluate the prediction accuracy of learning approaches for a range of prediction horizons; 3) to assess the time-cost and accuracy for different models, with a specific focus on the GuSLSTM and GuGRU models; 4) to analyze OpenFlow traffic with and without using the preferred prediction algorithm, the GuSLSTM model, assessing the accumulated power consumption per OpenFlow channel in the data-centre when SM-FPLF is applied. Our findings indicate that the GuSLSTM outperforms rival algorithms in terms of link utilization prediction accuracy over varying input sequence lengths. This accuracy is achieved whilst satisfying the SDN domain-specific requirement of a small computation time in a real-time implementation. Embedding a GuSLSTM in the SM-FPLF algorithm offers a power saving of 372 watts per OpenFlow channel, which is achieved in part due to a 13.7% CPU usage reduction in controllers and switches. These findings provide a valuable perspective into the performance and suitability of RNNs for real-time implementation as part of SDN solutions. They also shed light on their practical implications and benefits of using link utilization prediction in SDN management and power consumption optimization solutions.

INDEX TERMS Data center networks, software-defined networks, power consumption, machine learning, OpenFlow, monitoring, prediction, overhead.

I. INTRODUCTION

The increasing power consumption of Data Center Networks (DCNs) is becoming a major concern for network operators. The demand for information services is causing a dramatic increase in the usage of DCNs around the globe. The energy usage of DCNs accounts for between 1 % and 1.5 % of worldwide power consumption according to [1], [2], and this fraction is expected to grow to 8% by 2020 [2], [3]. Koomey reports that the increase in power consumption in DCN

The associate editor coordinating the review of this manuscript and approving it for publication was Sangsoon Lim.

was 56 % between 2005 and 2010 in [2] and [4]. Projections suggest a continued upward trend in the future. Current estimates suggest that by 2020, the energy consumption of DCNs in the US exceeded 139 billion kWh, and that interconnection devices (switches and links) consumed from 10 % to 20 % of the energy [2], [5]. The power consumption of switches and links in DCNs depends on the design of the device. It also depends on the network traffic and workloads transmitted by the device. This paper addresses the following problem: how can the power consumption of DCNs be optimized using a real-time monitoring system that collects the network statistics periodically and that predicts future link utilization of DCN links.

Recent studies have focused on optimizing the power consumption of links between switches in fat-tree and B-cube DCN topologies. These studies explored two distinct approaches. The first approach involved identifying a subset of the topology to support the traffic, while the second approach considered traffic scheduling. On one hand, in the case of the fat-tree and B-cube DCN topologies, the amount of traffic consolidation was increased, allowing for transmission over a selected portion of the topology instead of the entire network [6]. This led to significant power savings, particularly during periods when the traffic demand was low [7]. On the other hand, several studies advocated for the implementation of scheduling strategies to manage traffic flows in queues with different priorities. Each flow occupied a link's full bandwidth until its completion time or until it was preemptively suspended by other flows with higher priority [8], [9], [10]. Although scheduling strategies demonstrated the potential to improve flow control and network efficiency, they did not contribute to enhancing power consumption in comparison with the traffic consolidation strategies.

Compared to legacy networking systems, such as SNMP (Simple Network Management Protocol), which is used to manage and monitor network devices performance, and the Border Gateway Protocol (BGP), which is used as a routing protocol, Software-Defined Networking (SDN) is becoming increasingly popular for network provision and management tasks. In SDN, the forwarding elements are managed by a central controller which serves as the network's brain, and is responsible for the network's activities. Due to its flexibility, programmability and adaptability, SDN has the potential to solve many problems associated with traditional networks [11]. In the context of building an efficient network monitoring system, legacy networking systems suffer from the following issues: (1) the collection of inappropriate statistics due to variations in the types of distributed forwarding elements and (2) the inability of the monitoring system to cope with dynamic network changes, such as topology and routing changes [12].

In contrast, SDN's global view of the network allows it to perform efficient monitoring to gather statistical data about network flows, traffic per port, flow table status and so on. SDN's controller can monitor the forwarding elements residing on its domain using either an active or a passive approach. In the active approach, polling messages are periodically issued by the controller to obtain flow statistics. In the passive approach, the controller receives flow statistics from the forwarding elements when the flow tables encounter flow completions. Both active and passive monitoring techniques suffer from deficiencies. A large network overhead is the primary disadvantage of active monitoring. It causes a significant increase in controller resource usage. A disadvantage of passive monitoring approaches is that they cannot provide instantaneous measurements about the state of flows, which are currently in operation because the statistics relating to these flows are only sent to the controller after a flow has been terminated. In terms of configuration, the out-of-band deployment of SDN controllers is the most common deployment approach. In this scenario, the controller is connected to the data plane switches via a single hop and hence control messages can be sent immediately [13].

We focus on optimizing the power consumption by considering the control channel between data plane elements and the SDN controller. We contribute a new algorithm called SMart Fill Prefer Path First (SM-FPLF) that optimizes power consumption by considering use of this control channel. This is significant because these links can contribute meaningfully to the overall energy consumption of DCNs especially when they are utilized in the monitoring process.

Developing intelligent network applications using artificial intelligence and Machine Learning (ML) is becoming more practical [14]. Recently, ML techniques have been applied as a part of new solutions for a wide range of networking issues, including but not limited to traffic classification, routing optimization, security and Quality of Service (QoS) prediction [15].

Passive monitoring is not an effective monitoring strategy when the goal is to minimize the network power consumption in real time. This is due to the delay that typically occurs between an event of interest and when the monitoring report about it is transmitted. Consequently, we consider active network monitoring approaches when tackling the problem of power consumption reduction. Motivated by the advance of ML and its application to various computer networking challenges, as well as the programmability of SDNs, we contribute an approach that uses ML algorithms to predict the network traffic state, and then use these predictions to allocate the data plane resources efficiently. In summary, we contribute (1) a new ML model that receives the current state of data plane statistical monitoring reports and predicts the pattern of the next epoch, which also (2) reduces the network overhead required to obtain these reports when a periodic polling strategy is used.

A review of related approaches is provided in Section II to provide context for the SM-FPLF algorithm. Sections III, IV and VI provide the problem description, the switch energy model used to develop the SM-FPLF

algorithm in this paper, and the SM-FPLF problem formulation respectively. The SM-FPLF framework and its constituent algorithms are described in Section VI. The experimental setup used in this paper is described in VII. Section VIII introduces two models, based on the Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) networks, which are used as baseline algorithms in this paper. It also describes how Bidirectional LSTMs and GRUs, and Convolutional Long-term Recurrent Networks (CLRN) are evaluated using Graphics Processing Units (GPUs). Section IX describes the performance characteristics of the training and prediction routines used for the learning algorithms. A particular focus is placed on the convergence rate of the loss function, the accuracy of future prediction stops, and the computation time. We provide context for these results by referring to the limitations of applying SM-FPLF for power optimization. Finally, we conclude by highlighting the paper's main findings and outlining potential future directions in Sections X, and XI.

II. RELATED WORK

We discuss the state-of-the-art in power optimization and traffic prediction strategies that aim to improve energy usage in DCNs.

A. POWER OPTIMIZATION

There are two main approaches to reducing the power consumption of DCN-based Routing-Aware techniques: 1) Flow Aggregation Techniques (FATs); and 2) Flow Scheduling Techniques (FSTs). We identify and introduce FATs and FSTs approaches that demonstrate promise in the remainder of this section.

1) FLOW AGGREGATION TECHNIQUES

FATs consolidate as much traffic as possible in a subset of the DCN topology and turn off the other forwarding devices, subject to QoS constraints. An early example of this type of approach is the ElasticTree algorithm, which was introduced in [16]. The authors showed how the ElasticTree shrank and extended the topology according to the traffic demand. When the traffic level was low, the ElasticTree was shown to save a significant amount of energy. The study in [16] did not take into account the correlation between flows, which could affect the method's efficiency. The CoRrelation-aware Power Optimization (CARPO) approach presented in [17] aimed to address the correlation between flows and to dynamically aggregate them in a small subset of the DCN. This approach was designed to improve the latency of the DCNs.

To test the effect of the over-subscription factor on power usage in DCNs, the study presented in [6] examined a large number of over-subscription factors and introduced three algorithms, namely First-Fit, Best-Fit, and Worst-Fit, to find the most energy-efficient path in the DCN. The experiments were conducted using different DCN workloads, and the results indicated that the algorithms were able to save more energy by increasing the over-subscription factor of the DCN topology.

One recent study in [18] proposed an adaptive algorithm based on link utility and switch power consumption. The authors used an Integer Linear Program (InLP) to formulate an optimization problem that consisted of a linear objective function which was subject to multiple constraints on bandwidth, dropped packets, etc. Results showed that solving the InLP had a high time cost. Consequently, the authors proposed the Fill Prefer Path First (FPLF) algorithm to find the shortest-efficient energy paths in DCNs as a solution to the power reduction problem in DCNs. The power saving results achieved were promising, however, using the FPLF incurred a high CPU utilization cost on the controller. This was due to additional overhead costs which arose due to periodic monitoring. Finding optimal solutions for the power consumption optimization problem was also explored in [19]. The InLP model used in this paper aimed to minimize the number of active links while satisfying the traffic demands of the flows. Similar to the approach in [18], the InLP model incorporated several constraints, such as flow routing, link capacity, flow conservation, and link utilization. The authors evaluated a number of InLP solvers to find the best approach for the power consumption optimization problem. They demonstrated that the Gurobi solver exhibited superior performance in terms of scalability and runtime compared to other InLP solvers.

2) FLOW SCHEDULING TECHNIQUES

FSTs are used to manage requested flows in a queue based on their priority and to send them sequentially. FSTs aim to use the full bandwidth along the selected path when sending a flow. This approach is commonly called "path monopolization".

A collision-free domain and energy-efficient method was presented in [10]. The study sorted flows based on their size, i.e. from the smallest to the largest, and divided them into two lists: 1) a sending list; and 2) a waiting list. If a small flow arrived in the DCN, it preempted the larger one, monopolized the path, and was moved to the sending list. The lists were checked periodically for updates.

The authors in [20] claimed that FATs were not suitable for DCNs that had a limited number of flows. This scenario can occur when applications generate flows with a high bit rate, i.e. bandwidth-hungry applications. To resolve this problem, the authors proposed an approach named Willow, which considered the number of switches used and the time duration required to manage power optimization in DCNs. The authors of [21] aimed to optimize the greedy selection method used by Willow [20]. They proposed an algorithm called BEERS to find flows with similar sizes at each interval of time, which they called a harmonic set, and to schedule them with high priority. The results showed that the BEERS algorithm could save more power compared to the [20]. However, searching for candidate members of the harmonic set at specific times increased the complexity of the algorithm.

To improve the quality of flow delivery a Dynamic Flow Scheduling (DFS) approach was proposed in [9]. DFS aimed to balance the workload on network switches to reduce power consumption. To achieve this goal, the authors introduced a waiting time threshold in the queue line, which meant that the flows were scheduled based on whether the threshold time satisfied a QoS requirement. This class of technique is suitable for real-time protocols.

The links between the switches in DCNs was the focus of research that has been reviewed so far. It is also important to consider the OpenFlow channel, especially for the OpenFlow1.3 specification, which is described in [22]. The OpenFlow channel uses Ethernet. The OpenFlow protocol is designed to work on top of existing Ethernet networks and uses the same Ethernet frame format and communication mechanisms as traditional Ethernet. Although the power consumption of the OpenFlow channel in SDNs is relatively low compared to other networking components, it is still important to consider the power requirements of the entire SDN system when designing and deploying it, especially in many cases of Software-Defined-Data Center Network (SD-DCN) design where multiple OpenFlow channels are used. In conclusion, this paper aims to minimize the use of the OpenFlow channel for efficient energy usage and to reduce the overhead of SDN controller functions.

B. TRAFFIC PREDICTION

Given that many monitoring models use a traffic matrix to build the utilization matrix, we now discuss methods that have been used to predict the traffic matrix.

We categorize methods for network traffic prediction into two categories: linear prediction methods and non-linear prediction methods. Successful linear prediction has traditionally been carried out using a member of the Autoregressive Moving Average (ARMA) family of methods which was developed by the financial time-series community and under the guise of Infinite Impulse Response (IIR) filtering by the Signal Processing community [23]. ARMA combines AutoRegression and Moving Average components to model stationary time-series data. The Autoregressive Integrated Moving Average (ARIMA) is used when the data is assumed to be non-stationary. AutoRegressive Conditional Heteroskedasticity (ARCH) is an appropriate assumption when the traffic exhibits volatility. ARCH captures this volatility by assuming that the variance of last error term is related to the square of previous innovations [24]. The Holt-Winters (HW) algorithm uses a combination of exponential smoothing and trend estimation, to make predictions [25], [26]. For video quality prediction, trend removal via an IIR filter combined with fitting periodic, decaying exponentials to the time series has yielded good results [23].

On the other hand, nonlinear forecasting methods frequently utilize Neural Networks (NNs) as shown in [27]. Domain expertise is required to apply the more sophisticated linear models introduced above. The authors of [28] showed that nonlinear traffic prediction modes based on NNs performed outperformed the linear forecasting models, ARMA and HW, they investigated. One advantage of NNs is that many algorithms can now be applied with minimal domain expertise, resulting in good performance. However, the authors of [29] demonstrated that additional novel feature domain transforms are can be used to enhance the quality of prediction and classification of future video jitter measurements using an algorithm called, FEATjitter. The enhancement was achieved via significant tuning of the learning algorithm, which may not always be possible. Moreover, a disadvantage of NN-based approaches is that training these models is computationally costly. However, an appealing characteristic of the approach in [23] is that modelling fitting is inexpensive. For example fitting the exponential model parameters incurs the cost of inverting a 2×2 matrix.

Recurrent Neural Networks (RNNs) are a type of NN that has recently experienced a big boom in natural language processing tasks and time-series analysis due to its ability to capture long-term dependencies in the data. The study in [28] used LSTM architectures to predict the Origin-Destination (OD) traffic matrix, i.e., the flow sequence, separately. The results showed that LSTM were well-suited for traffic matrix predictions and outperformed a number of linear methods and FeedForward Neural Networks (FFNNs) by many orders of magnitude. The study discussed in [30] explored the performance of three different types of RNNs: LSTM, GRU, and Bidirectional LSTM (BiLSTM), on both real-world, (i.e., GEANT backbone network traffic) and artificial datasets (i.e., generated using a testbed). The findings were encouraging. Each model exhibited unique strengths and weaknesses depending on the type of data being analyzed.

The authors of [31] employed hybrid models which comprised of multiple layers, including Convolutional Neural Networks (CNNs) and LSTMs, to enhance the accuracy of the OD output. Results showed that the model achieved substantially higher accuracy than state-of-the-art models. A few recent studies, including the one described in [32], focused on predicting the entire traffic matrix, which offers benefits for various network functionalities.

In conclusion, it is important to consider multi-step prediction models to optimize power consumption. Consequently, an objective of this research is to develop a multi-step prediction model for entire links in the DCN. This model will be integrated with our contribution in [18] in this paper, to further enhance the power consumption and network overhead in SD-DCNs environment.

III. PROBLEM DESCRIPTION AND SOLUTION

This paper is part of an ongoing project to optimize the power consumption in DCNs. In a previous contribution [18], we introduced the FPLF algorithm, a dual-direction algorithm for reducing power consumption and keeping the QoS at an acceptable level. This algorithm consisted of three components, a (1) Link-Utility, (2) Link-Cost, and (3) Fill-Shortest Path component.

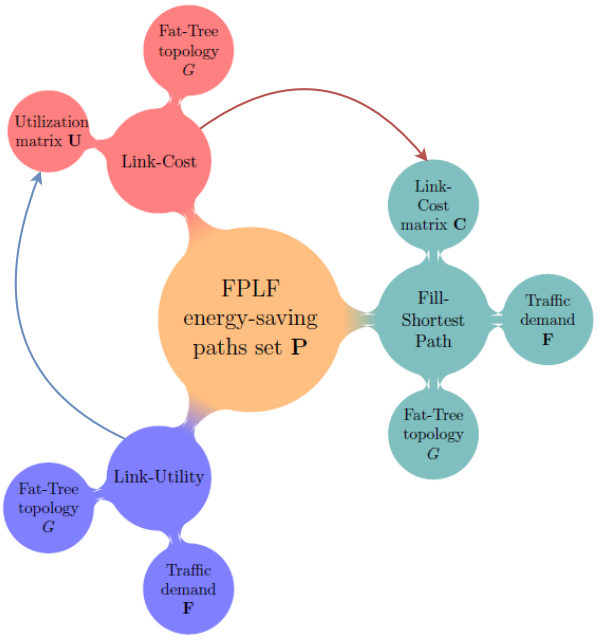


FIGURE 1. The FPLF algorithm (illustrated as the root node, in the centre) consists of three components: the link-utility, link-cost, and fill-shortest path components (which are illustrated as first-level leaf nodes). The parameters used by each of these components are illustrated as second-level leaf nodes. They include matrices representing the edge cost, **C**, and edge utilization **U**, as well as the traffic demand set, $\mathbb{F}$. These components are then combined to produce the final output, an energy saving path, $\mathbb{P}$..

Fig. 1 shows that a Link Utilization matrix, **U**, is produced by the Link Utility component. It is an input to the Link Cost component. To construct the Link Utilization matrix, statistics must be collected from the switches in the DCN. This is done periodically, with a period of 1 second. This monitoring process is costly. It consumes power and reduces the performance of the DCN. This reduction is explained by the fact that resources used for monitoring and then optimization cannot be used for other DCN-critical functions.

FPLF uses a consolidation strategy to optimize power usage in SDN-based DCNs by minimizing the count of ports that are in use. This optimization is performed subject to a ''maximum link utilization threshold'', to maintain link reliability, i.e., to prevent links from becoming overloaded and more susceptible to unplanned failures [33]. Additionally, it increases the number of active ports in proportion to the DCN workload to prevent performance degradation. However, FPLF only considers links between switches. It does not consider the switch-to-controller link. This approach is reasonable because the majority of links in a DCN are switch-to-switch, and thus, the payoff in terms of power saving is large if it considers these links.

Switches typically have one or more connections with the SDN controller. Reducing the usage of this connection may result in more efficient usage of SDN controller resources and reduce the power usage. Decreasing CPU usage is a potential saving from minimising the usage of this link. Given that messages (relating to flow modification, termination, etc.) from the switches invoke actions in the controller, the level of usage of the switch-to-controller link is a good proxy for CPU usage imposed on the controller due to changes in the network and minimising usage of this link has the potential to reduce controller workload.

Considering the potential for power saving from the strategies outlined above, in this paper, we apply Deep Learning (DL) approaches, specifically a RNN to predict the utilization matrix, **U**,for multiple steps of events in real-time. The proposed model is trained offline with different traffic patterns, which are characteristic of the DCN. The trained model is integrated with the rest of the system. This facilitates real-time testing by synchronizing it with the periodic monitoring function.

To quantify the performance of the integrated system we define the number of switch ports in the DCN to be M. The number of request and response ports state messages delivered over the i^{th} port is denoted, $R_{ps}(i)$. The total number of request and response messages in the DCN is the sum

$$\text{minimize} \sum_{i=1}^{M} R_{ps}(i). \tag{1}$$

The number of request and response messages in Equation (1) is subject to a threshold and the idle time of the link. The goal of this paper is to minimize the sum in Equation (1).

This paper makes the following contributions. (1) Dataset: We propose a method to generate the Utilization Matrix by utilizing realistic traffic distributions. To achieve this, we employ the D-ITG (Distributed Internet Traffic Generator) tool [34], to generate traffic between switches within the fat-tree topology while scheduling PortState OpenFlow messages. This approach facilitates the collection of training, validation, and testing data for our analysis. (2) Modeling: A DL model is produced to meet the requirements of the FPLF algorithm and to reduce the overhead while optimizing power usage for both controllers and switches in a DC-SDN architecture. (3) Real-time implementation: Using a series of simulation studies in the Mininet emulator, we evaluate our model and demonstrate that it achieves higher prediction performance and reduces overhead and energy in real-time.

IV. SWITCH ENERGY MODEL

The topology of a DCN can be either homogeneous or heterogeneous. In a homogeneous topology, all switches in the network are of the same type, which can provide advantages in terms of simplicity and ease of management, because all switches can be configured and managed in a similar way. On the other hand, in a heterogeneous topology, switches from multiple vendors or switches with different capabilities or configurations are used. This can provide greater flexibility and functionality, but it can also make the network more complex to manage and to troubleshoot. Therefore, DCNs are often designed to be homogeneous [35].

TABLE 1. List of notation.

Symbol	Description
$\mathbb{S}$	Set of nodes, where $\mathbb{S} = \{s_1, \ldots, s_N\}$.
$\mathbb{E}$	Set of the edges between switches, where $\mathbb{E} \subseteq \{e_{ij}, e_{ji} \mid s_i, s_j \in \mathbb{S}\}$.
$\mathbb{F}$	Set of flows, where a flow $f = (s_r, s_d, \lambda_f) \in \mathbb{F}$ is represented by source $s_r \in \mathbb{S}$, destination $s_d \in \mathbb{S}$, and bit rate $\lambda_f \in \mathbb{N}$.
$\mathbf{U}$	Utilization matrix where U_{ij} represents the utilization of the link e_{ij}.
$\mathbb{P}$	Set of energy-efficient shortest paths, where a path.
$\mathbf{U}(t)$	Utilization matrix at time-step t, where $U_{i,j}(t)$ represents the utilization of the link e_{ij} at time-step t.
$\hat{\mathbf{U}}(t)$	Prediction matrix at time-step t, where $\hat{U}_{i,j}(t)$ represents the predicted utility of the link e_{ij}.
T	Interval of prediction $t, t+1, t+2, \ldots, T$.
$\mathbf{C}$	Link-cost matrix where C_{ij} denotes the cost of the link e_{ij}.
ΔT_s	Amount of time required for the controller to discover the graph G.
ΔT_m	Time required to perform periodic polling.
ΔT_p	Time required to predict future link utilizations.
ΔT_l	Future state of each link.
ΔT_{req}	Controller-to-switch request time.
ΔT_{rep}	Switch-to-controller reply time.
ΔT_u	Time required to update the cost of each link.

The power consumption of DCN switches can be measured dynamically or statically. Dynamic measurement measures the power consumption of active links and the power depends on the speed of the link. Different devices have different power profiles. The power profile of a device characterizes the power it consumes as a function of the bit rate of the switch port under consideration. To provide benchmark values for the power involved, we cite the authors of [36] who state that the power consumed per port, when the switch port is operating at the speeds 10 Mbps, 100 Mbps, and 1 Gbps, is 63 mW, 260 mW, and 913 mW, respectively. The switch considered was the commercial Pronto switch, which has OpenFlow capabilities. In contrast, the NEC ProgrammableFlow Networking Suite *PF*5240 OpenFlow switch consumes significantly more power, consuming approximately 0.2 W/port, when it operates at 10 Mbps. In conclusion, the design of the switch plays a large role in its power consumption. In this case the Pronto switch consumes 31.5 % of the power of the NEC switch when operating at the same data rate. Given that the network manager may not have a choice in the equipment at their disposal, the network manager's focus should be on the manner in which the switch is used, to reduce power consumption. The present contribution looks to optimize switch usage to reduce power consumption.

The static measurement approach involves assessing the power consumed by the components that are responsible for keeping the system operational, maintaining connectivity, and performing background tasks. This includes considering the power consumed by components such as chassis, fans, and switching fabric. The components involved in both dynamic and static measurement are now listed and described.

We base our switch energy consumption on the work of [37] which presented measurements and derived a power consumption models for OpenFlow SDN devices. The model consisted of four parts. (1) the base power required to operate the switch component was denoted P_{base}; (2) the power consumption of the control traffic (*PacketIn*, *FlowMod*, and *PortState* messages), was denoted P_{cont}; (3) the power consumed by active links and the configuration of line speed (the underlay of links, which represents two ports between two switches), was denoted, P_{conf}; and (4) the power consumption of the processed OpenFlow traffic (power to process and forward packets, where a number of matches and actions are possible), was denoted P_{OvF}. The total power consumption of the switch was defined to be the sum

$$P_{\text{s}} = P_{\text{base}} + P_{\text{cont}} + P_{\text{conf}} + P_{\text{OvF}}. \tag{2}$$

We focus on the dynamic component of switch power consumption, specifically P_{cont} and P_{conf}. To minimize the P_{cont} component, the number of *PortState* messages used in the monitoring models of the FPLF algorithm must be reduced.

To minimize the P_{conf} component, it is necessary to ensure that the OpenFlow channel remains idle for the longest possible time without compromising the FPLF algorithm's performance. This affects the power usage of the SDN controller and the links from the OpenFlow switch to the controller. The number of active ports is denoted N_a. We assume that the power consumption of the port at full speed is P_{max}. When the port is operating at a lower speed the power consumed is determined by scaling P_{max} by a factor, $F_i(r)$, which is a function of the bit rate, r for the i-th port. We consider the set of configured speeds, $r = \{10, 100, 1000\}$ Mbps, for the i-th port.

The NEC ProgrammableFlow Networking Suite *PF*5240 OpenFlow switch consumes approximately 0.2 W/port when it operates at 10 Mbps. This power approximated by taking the product of the power consumption at full speed and the relative power consumption of the configured speed, $P_{max} \times F_i(r) = 0.3761 \times 0.5295 \approx 0.2$ W/port, as reported in [37].

The power consumed by active links and the configuration of line speed P_{conf}, is calculated by summing the scaling factor for the appropriate speed on each port, $F_i(r)$, and scaling it by the maximum speed power, P_{max},

$$P_{\text{conf}} = P_{max} \sum_{i=1}^{N_a} F_i(r). \tag{3}$$

We assume that switches can turn on or enter sleep mode based on local traffic states, using Wake-on-Arrival (WoA) to wake up the switch when needed for forwarding packets and the Sleep-on-Idle (SoI) technique, for switches to save power when idle, without considering the transition time in the evaluation [20].

V. PROBLEM FORMULATION

FPLF is a Traffic Engineering Algorithm (TEA) that establishes routing in DCNs so that only a subset of links and switches are used, and consequently, power consumption is optimized. Similar to other TEAs, it uses a traffic matrix to calculate the utilization matrix, $\mathbf{U}(t)$, at time t as part of the routing process. The ability to be able to predict future utilization matrices, for example the utilization at time t, is denoted as $\hat{\mathbf{U}}(t)$, and can be used to reduce power consumption and SDN controller overhead. In our approach, the FPLF link-cost component is supplied with the utilization matrix, $\mathbf{U}(t)$, every second. The workload time is divided into consecutive intervals. Each interval lasts one second. We assume that the utilization matrix remains constant during each one second interval.

We summarize the notation used to model the problem in Table 1 and we describe it in this section. We model the network topology using a directed weighted graph to capture the forward and backward direction of flows. The topology is described by the graph, $G = (\mathbb{S}, \mathbb{E})$. It is composed of a vertex-set, $\mathbb{S}$, and an edge-set, $\mathbb{E}$. Each switch in the network is a member of the vertex-set. It is denoted as, s_i, and it functions as an OpenFlow switch, facilitating the routing of information through the path determined by the SDN controller.

Given a topology which consists of a set of N switches. The utilization matrix $\mathbf{U}(t)$ has dimensions $N \times N$. The (i, j)-th entry of $\mathbf{U}$, which is denoted $U_{i,j}(t)$, represents the link utilization of the link connecting switch s_i to switch s_j. To capture the dynamics of the network, we increment the time stamp each time the utilization matrix is estimated. This process results in a 3D tensor of dimension, $N \times N \times T$. The entry $U_{i,j}(t)$ denotes the value of the utilization between switches s_i and s_j at time step t.

Utilization matrix prediction is the process of finding a prediction of $\mathbf{U}(t)$, which is denoted as $\hat{\mathbf{U}}(t)$. This prediction is obtained using a set of historical values. The SM-FPLF algorithm uses the developed prediction, $\hat{\mathbf{U}}(t)$, as an input to the process that calculates the routing strategy for future time intervals. The challenge in developing accurate predictions lies in being able to accurately model the inherent relationships among the utilization data set.

VI. PROPOSED FRAMEWORK AND ALGORITHMS

The architecture of the proposed solution is illustrated in Fig. 2. The framework consists of three main layers: (i) the infrastructure layer that represents the network topology, which is a fat-tree in this paper, (ii) the control layer that represents the network central management and information center, and (iii) the application layer. The novel aspects of this framework are the *SM-FPLF* and the *Monitoring and Prediction Model* modules, which are located in the application layer.

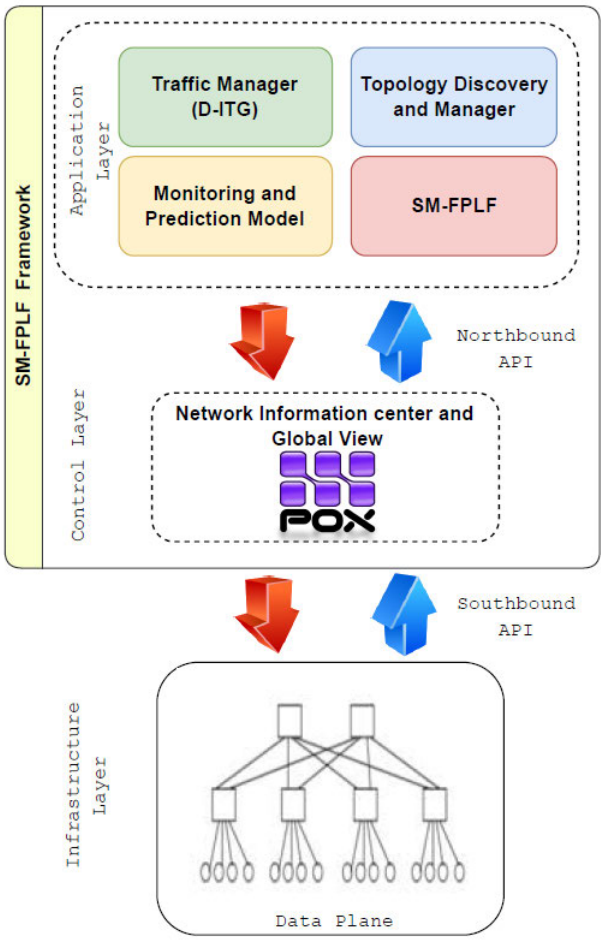


FIGURE 2. The architecture of the proposed framework and its main components. The contributions of this paper include the SM-FPLF and the Monitoring and prediction components.

A. CONTROL LAYER

The SDN controller implements management policies in response to network activity. Use of the POX controller is motivated by the fact that it is well-suited to the task of developing solution prototypes quickly [38]. The standard OpenFlow protocol [39] was used as a southbound interface to define the communication between the POX controller and the infrastructure's elements. POX's APIs were utilized on the northbound interface to develop the application layer modules.

B. APPLICATION LAYER

The application layer consists of four modules, which are now described.

1) TRAFFIC MANAGER (D-ITG)

In order to generate traffic that is similar to the traffic delivered on DCNs, we used the Distributed Internet Traffic

TABLE 2. Characteristics of the traffic used to generate training data: Weibull and pareto distributions were chosen to generate DCN-style diurnal traffic.

Characteristics	Type of Distribution
OFF periods	Weibull
ON periods	Pareto
Arrival times of packets	Weibull

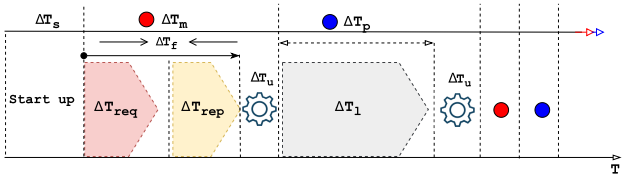


FIGURE 3. A breakdown of the time costs associated with the different subtasks in SM-FPLF. Two of the most expensive tasks are online link monitoring, which has a time denoted as ΔT_m, and prediction, denoted as ΔT_p.

Generator (D-ITG) [34]. D-ITG is a tool that can be used for producing a realistic packet-based network traffic by accurately emulating the workload of real world traffic [40]. The study in [41] showed that the lognormal, Weibull and Pareto distributions were suitable choices for generating DCN-style diurnal traffic. D-ITG is flexible enough to generate traffic patterns according to these distributions at both macroscopic and microscopic scales. Compared to other traffic generation tools, D-ITG is considered to be more reliable [42]. A range of types of traffic can be generated by D-ITG, for example, TCP, UDP, DNS and VoIP traffic.

Bursty ON-OFF traffic patterns are prevalent in DCNs according to [41]. Bursty ON-OFF traffic was generated when preparing training data, using D-ITG. The use of D-ITG allows for the customization of the characteristics of the generated bursty traffic. The duration of both ON and OFF periods was set using the distributions commonly observed in DCNs, i.e., the Pareto or Weibull distribution, in order to simulate real-world DCN traffic patterns. Table 2 summarizes the traffic generation strategy.

2) TOPOLOGY DISCOVERY AND PARSER

This module is responsible for detecting the current operational elements of the infrastructure layer, i.e. switches and links. Topology discovery is essential for the network controller as it allows it to build a global view of the network. We used the standard POX discovery module. To represent the network information as a graph for easy management and manipulation, the NetworkX tool [43] was used.

3) MONITORING AND PREDICTION MODEL

Flow aggregation based techniques for minimizing the network power consumption require a fine-grained monitoring system that reflects the actual state of the traffic being transmitted over the network. By knowing the current state of each link, a precise cost (or weight) can be determined. However, collecting the statistics required to acquire this state information with high accuracy typically involves a high frequency of messaging, which may increase the controller overhead. To address this issue, a new online prediction based system is developed in this paper to reduce this overhead.

Algorithm 1 Monitoring and Predicting Utilization

```
Input : Network topology G(𝕊, 𝔼)
while T do
    Utilization = [:[]]
    x = 0
    while mode = ΔT_m ∧ k ⩽ threshold do
        x ← x + 1
        foreach s_i ∈ 𝕊 do
            s_i ← polling statistics                ▷ ΔT_f
            Calculate: A
            foreach e_ij ∈ 𝔼 do
                Calculate: U_ij
                Utilization[e_ij:] ← [U_ij(x)]
                Update: C_ij                        ▷ ΔT_u
            end
        end
    end
    while mode = ΔT_p ∧ k ⩽ threshold do
        foreach e_ij ∈ 𝔼 do
            foreach x ∈ Utilization[e_ij] do
                U_ij(x) ← predicted value           ▷ ΔT_l
                Update: C_ij                        ▷ ΔT_u
            end
        end
    end
end
```

Fig. 3 illustrates how our proposed monitoring system operates. The illustrated approach is a hybrid monitoring system which collects statistics from a periodic polling function and develops predictions using an online approach from another process. Fig. 3 focuses on the different time periods involved in monitoring. The time required to discover the underlay elements and to construct the network graph, G, by the controller is denoted ΔT_s. The time required for periodic polling, which is used to collect the real-time statistics, is denoted as ΔT_m. It accounts for the time taken to run a number of processes. The controller-to-switch request time is, ΔT_{req}, and the switch-to-controller reply time is ΔT_{rep}. The time required to update the cost of each link according to the actual or predicted state is ΔT_u. The monitoring time is denoted ΔT_m. It is the sum of the times ΔT_{req}, ΔT_{rep}, and ΔT_u.

The time taken to predict the future state of links is denoted as ΔT_p. It accounts for the time taken to run two sub-processes. The time taken to forecast the future state of each link is ΔT_l. The time required to update the cost of each link according to the actual or predicted state is denoted as ΔT_u. The quality of the prediction delivered during the time, ΔT_p, is evaluated in this paper using the Root Mean Squared Error (RMSE). This entire monitoring approach process is re-run every T seconds.

The online link state monitoring function is developed as a module, which is incorporated in the implementation framework. Pseudocode for this module is presented in the listings in Algorithm 1. The main objective is to calculate link utilization and to update the cost accordingly. Utilization is represented as a list of lists in which each operational link in the network preserves the utilization value for x samples. At time $x+1$, a link e_{ij} with the utilization sample time indices $1, 2, \ldots, x$ is expressed as, $[e_{ij} : [U_{ij}(1), U_{ij}(2), \ldots, U_{ij}(x)]]$.

The controller starts by collecting the flow statistics from each switch, s_i, belonging to the network. The aggregated traffic matrix, $\mathbf{A}$, is calculated. This step represents the online monitoring period in lines 6-8. The collected data is subsequently analyzed and the utilization, U_{ij}, of each link, e_{ij}, is calculated in lines 9-10. In practice, the utilization value, U_{ij}, is the ratio of flow rates traversing the link, e_{ij}, with the bandwidth. The utilization value of each link is recorded in line 11. These values are used at the prediction stage.

Every time a new value of the utilization is calculated for a link, the cost is updated (in line 12). Once the periodic polling process has been run k times, then the monitoring process is switched off, and the prediction process is activated. In the prediction phase, which is described in lines 16-23, the future state of utilization of each link is predicted. The presented implementation is able to predict x future values. This equals the length of the utilization list of each link given in line 18-19. Every time when a new prediction value of utilization is calculated for a link, the cost is updated (in line 20).

4) SM-FPLF

The FPLF algorithm, introduced in [18], uses Dijkstra's algorithm [44] after a link-cost calculation step to determine the Energy-Efficient Shortest Path (ESP), $\mathbb{P}$. This selection process aims to identify the least-cost path. It optimizes energy consumption by favoring underutilized links and enhances forward energy optimization by setting the state of as many switches and ports as possible to be off (or in a sleep state). In this algorithm, the initial weight, w_{ij}, is 1 for all links in the graph. This weight configuration enables the selection of the shortest path between the source node, s_r, and the destination node, s_d, as the initial condition value. Once the first path is determined, the algorithm utilizes the active link cost, denoted as C_{ij}. This cost is a function of the link utilization. It is denoted by $p(U_{i,j}(t))$. It adjusts the link cost based on its utilization over time. It is parameterized by an upper bound parameter, M, which is a positive number that prevents $p(U_{i,j}(t))$ returning a negative value. Guidance on the selection of M and a threshold parameter, T_o, is given in [18]. In this paper, the threshold T_o is set to 90 %. The link utilization cost is computed using,

$$p(U_{i,j}(t)) = \begin{cases} M - (T_o - U_{i,j}(t)), & 0 < U_{i,j}(t) < T_o, \\ \infty, & U_{i,j}(t) \geqslant T_o, \\ M, & U_{i,j}(t) = 0. \end{cases} \tag{4}$$

Algorithm 2 Routing With Dijkstra's Algorithm

Input : Network topology $G(\mathbb{S}, \mathbb{E})$
1 $\forall(e_{ij} \in \mathbb{E})$: $w_{e_{ij}} = 1$ ▷ `initial weight`
2 $\forall(e_{ij} \in \mathbb{E})$: $C_{ij} = U_{ij}$ ▷ `the output of first two components of FPLF`
3 **foreach** *`new request`* **do**
4 Run Dijkstra's algorithm to find the shortest path with minimum link costs (C_{ij})
5 **end**

The strategy taken by the FPLF algorithm is to force all switches to use the specified link as long as it is underutilized. However, when the link utilization exceeds T_o, the FPLF algorithm redirects flows to the alternative ESP path. The changes in link weight are subject to traffic aggregation to maximize $U_{i,j}(t)$ for each link. The link utilization matrix, $U_{i,j}(t)$, at time t, is computed by summing the traffic during the t-th time period, e.g. $T_{ij}(t)$, where the traffic is the sum of different applications' flows, and dividing this sum by the bandwidth, B_{ij}, of the link, e_{ij},

$$U_{i,j}(t) = \frac{\sum_{e_{ij} \in \mathbb{E}} T_{ij}(t)}{B_{ij}}, \forall e_{ij}. \tag{5}$$

The path cost of a path from the source node, s_i, to the destination node, s_m, which consists of the sequence of links, $e_{ij}, e_{j,k}, \ldots e_{n,m}$, is the sum of the costs of each of the links

$$C_{path} = C_{ij} + C_{j,k} + \ldots + C_{n,m}. \tag{6}$$

In the new algorithm, SM-FPLF, the costs of links are dynamically adjusted according to Algorithm 1. The least cost path for new requests are determined by Algorithm 2 depending on the network state at the time of the request.

VII. EXPERIMENTAL SETUP

In this section, we describe the testbed, the data generation process and the data collection process which is used to train the ML models.

A. NETWORK EMULATION

A fat-tree topology, constructed with 4 cores, 16 hosts and 20 switches was modelled to represent the data plane during the evaluation process. The Mininet emulator [45] was used to emulate the proposed framework in the validation process. Mininet supports real and synthetic network topologies, running real kernel and application code on a single machine. Each host of the fat-tree topology was connected to an OpenvSwitch (OvS). Given that the proposed system is concerned with the out-of-band configuration, each OvS was connected to the controller by a secure OpenFlow channel.

B. TRAFFIC MATRIX GENERATION

Established methods were used to generate traffic matrices and produced nine sequence matrices with varying loads, including peak and valley traffic loads, by applying the

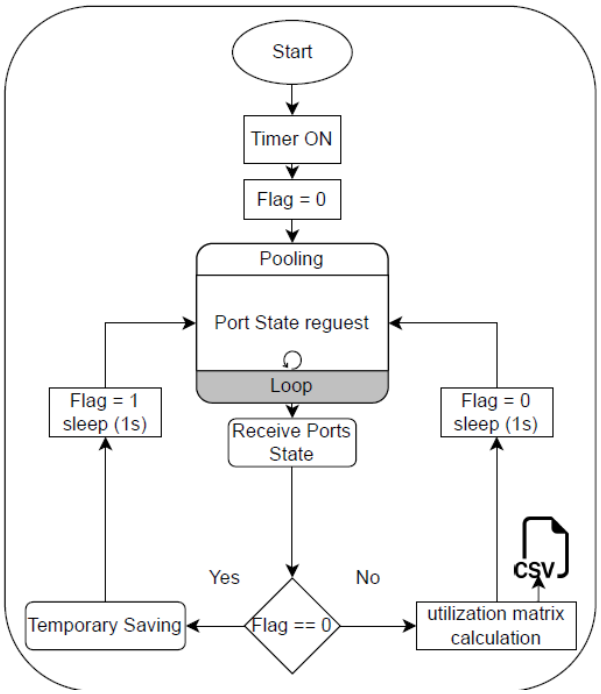


FIGURE 4. Utilization matrix generation.

Ranking Metrics Heuristic method proposed by Nucci in [46]. To create these matrices, we utilized the Fast Network Simulation Software (FNSS) [47], which is a Python-based toolchain simulator. Each sequence was simulated individually for approximately two hours using D-ITG, with the traffic characteristics set according to the approach outlined in [41]. We obtained 7250 observations for each OD pair using this simulation process.

All the counters in OvS, including Port Counters, Flow Counters, Group Counters, and others, typically represent cumulative values. They maintain a record of the total count or total amount of a specific metric since the counter was last reset or initialized. Consequently, the collection matrix at any given point in time is essentially an aggregated traffic matrix. In our case, the aggregated traffic matrix, $\mathbf{A}$, is generated during the monitoring phase, specifically within the time period ΔT_m.

Fig. 4 shows how this matrix is generated. At time t, a polling statistics message is issued by the controller. The aggregated traffic represented by $\mathbf{A}(t)$ is then temporarily stored until the subsequent time frame $t+1$, when a new traffic aggregated traffic matrix, $\mathbf{A}(t+1)$, is constructed. The controller then accurately calculates the traffic matrix for a specific time frame. It uses it to calculate the link utilization matrix, $\mathbf{U}(t)$, for each period and exports it to a CSV file. The link utilization matrix is normalized between 0 and 1.

C. LINK UTILIZATION TRANSFORMATION

The input sequence of data to the ML model consists of a sliding window of the previous ten samples of $\mathbf{U}(t)$. The output sequence predicts ten seconds of values for $\mathbf{U}(t)$. Since the SM-FPLF algorithm depends directly on the adjacency matrix of the topology graph, we only consider the adjacent values of $\mathbf{U}(t)$ when training the model.

The link utilization matrix is transformed into a vector of values for each time sample. This vector consists of the adjacent values of $\mathbf{U}(t)$, e.g. $U_{i,j}(t)$ at sample time t. Nine vectors were generated, one vector for each of the nine consecutive samples of the DCN workload that we generated and described in Section VII-B.

These vectors are used as inputs to a range of RNN models. We do not consider the remaining part of the link utilzation matrices at each time sample, $\mathbf{U}(t)$. This approach has two benefits which were determined from empirical evaluation of the system. Training the RNNs on the subset of the input data rather than the entire dataset, as described above, enhances the performance and accuracy of the DL model. The reason for this enhancement is that the model only considers the relevant adjacent values of $\mathbf{U}(t)$ when making predictions. It also helps to reduce the complexity of the model. This reduced complexity makes it easier to train and faster to execute the models. In summary, this strategy achieves better performance and accuracy while minimizing the overhead of DC-SDN networks.

VIII. RNNS: LINK UTILIZATION PREDICTION

We introduce the two baseline RNNs models for link utilization prediction, the GuGRU and GuLSTM models, which are GPU implementations of the GRU and LSTM, along with the challenges of using them for link utilization prediction. We then describe Bidirectional LSTM and GRU models, Stack LSTM (SLSTM) models and CLRNs. We compare and contrast these models qualitatively using a set of tables, e.g Table 3, 4, 5 and 6. We evaluate these Deep Neural Networks (DNNs) quantitatively using GPUs.

Training conventional RNNs poses challenges due to the problem of vanishing and exploding gradients. Vanishing gradients occur when the effect of input data at the very beginning of a long sequence is gradually lost. The result of this phenomenon is that the associated gradient values approach zero, which causes training progress to slow. Conversely, the exploding gradient phenomenon occurs when the gradients increase to a very high value, which causes large updates in the weights, and potential model instability. We use variants of RNNs, such as the LSTM [48] and GRU [49], which have been proposed as solutions to these problems, when the training data consists of sequences of link utilization data.

In terms of computation speed, GuDNN are DNNs which can be trained and used for inference on GPUs. GuDNNs can perform computations in parallel across multiple GPU cores, enabling faster training and inference times than traditional CPU-based NNs. The GuDNN library provides GPU-accelerated implementations of DL operations, including matrix multiplications and convolutions, which form a part of RNNs such as LSTMs and GRUs [50]. Given that

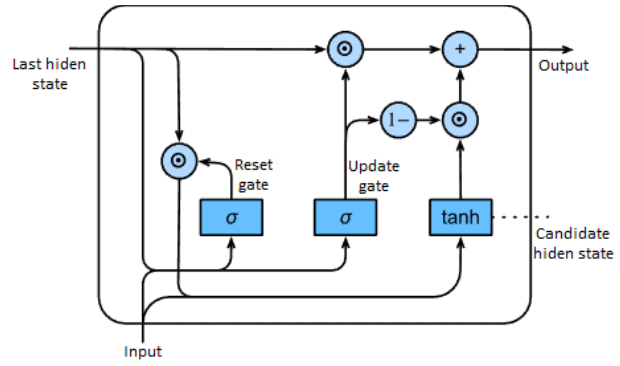


FIGURE 5. The structure of a GRU cell: The element-wise multiplication and sigmoid functions are denoted as ⊙ and σ, respectively. The GuGRU and GuLSTM models serve as baseline algorithms, setting a standard for link utilization prediction performance against which other algorithms are compared.

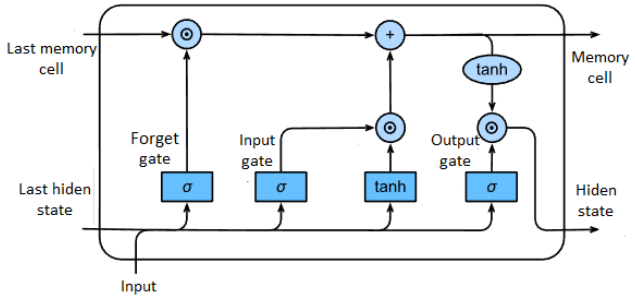


FIGURE 6. A three-gate architecture employed by the LSTM is illustrated for the purpose of comparison with the two-gate structure of the GRU cell shown in Fig. 5.

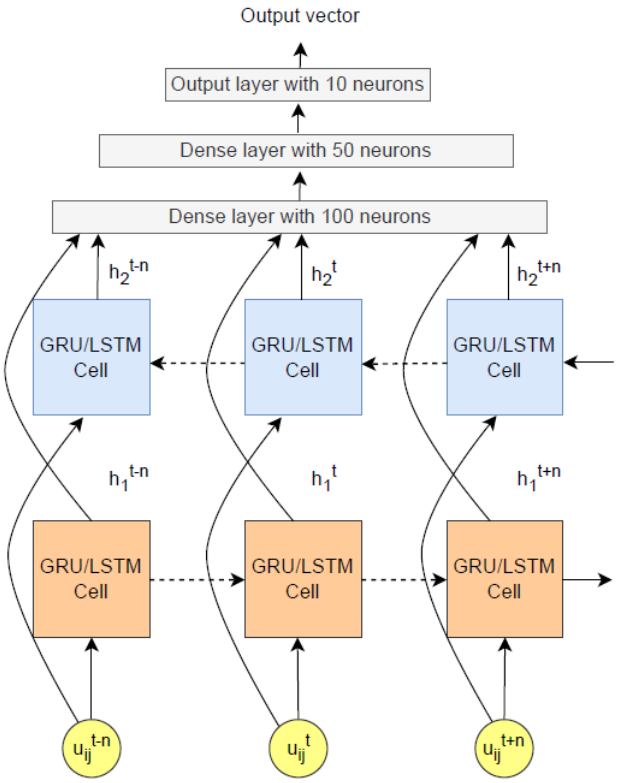


FIGURE 7. Bidirectional LSTM/GRU cell: this network processes the link utilization sequence in both directions, combining the forward and backward hidden states to create a final hidden state that incorporates information from both directions.

the goal of link state prediction is to train models suited to the different DCN patterns and traffic workloads that arise in dynamic networking situations, we run the proposed models using GuDNN. This allows us to speed up the training time and to handle large amounts of computation in parallel, which is a crucial requirement in our case.

A. BASELINE MODELS: GuGRU AND GuLSTM

GuGRUs and GuLSTMs are used when the flow of information must be controlled to avoid the vanishing and exploding gradient problems. The following notational conventions are used when describing their component gates. The sigmoid activation function is denoted, $\sigma(\cdot)$. Bias terms are denoted b_x and weight matrices, W_x and Q_x, where the subscript, x, identifies the gate the associated parameter pertains to. For example, the GRU shown in Fig. 5 and described in [49] consists of two gates, the update gate and the reset gate.

In the update gate, the bias term is b_r. The update gate uses two weight matrices, W_r and Q_r, to scale the current input and previous hidden state, which is denoted, $h(t-1)$. The update gate output, $r(t)$, is defined as

$$r(t) = \sigma(W_r U_{i,j}(t) + Q_r h(t-1) + b_r), \tag{7}$$

and it is used to determine how many of the past time steps should be passed to future processing steps.

The role of the reset gate is to decide how many past samples of the past information to forget at the current time step. It produces the output, $z(t)$, by computing

$$z(t) = \sigma(W_z U_{i,j}(t) + Q_z h(t-1) + b_z). \tag{8}$$

The weights, W_z and Q_z, and bias term, b_z, play a similar role to the corresponding terms in the update gate. The GuGRU generates candidate hidden states as follows

$$h'(t) = g(W U_{i,j}(t) + r(t) \odot Q h(t-1) + b).$$

Acceleration of GRU/LSTM layers is possible if certain criteria are met. For example, the LeakyReLU activation function, $g(\cdot)$, cannot be used. The implementation documentation in [51] states that the CUDNN LSTM/GRU acceleration currently works when the activation function is set to tanh$(\cdot)$. The final hidden state is defined as,

$$h(t) = z(t) \odot h(t-1) + (1 - z(t)) \odot h'(t).$$

The three gate architecture used by the LSTM model are illustrated in Fig. 6 for easy comparison with the two gate architecture used by the GRU in Fig. 5. The gates used by the LSTM are the input gate, $i(t)$, the forget gate, $f(t)$, and the output gate, $o(t)$. These gates work together in the LSTM to control the flow of information into, out of, and through the cell, respectively. The weights, W_i, W_f, W_0, Q_i, Q_f and

TABLE 3. CuDNNLSTM/CuDNNGRU model parameters overview.

Vector output technique			
Layer type	Output Shape	Number of the parameters	Note
CuLSTM/CuGRU	(None, 200)	162400	For CuGRU 121800
Dense	(None, 100)	20100	
Dense	(None, 50)	5050	
Dense	((None, 10)	510	
LSTM trainable params: 188, 060 GRU trainable params: 147, 460 Non-trainable params: 0			

TABLE 4. BiSLSTM/BiSGRU model parameters overview.

Vector output technique			
Layer type	Output Shape	Number of Parameters	Note
GuBiLSTM/GuBiGRU	(None, 10, 400)	324800	243600 for GuBiSGRU
BiLSTM/BiGRU	(None, 400)	963200	722400 for BiGRU
Dense	(None, 100)	40100	
Dense	(None, 50)	5050	
Dense	(None, 10)	510	
BiLSTM trainable params: 1, 333, 660 BiGRU trainable params: 1, 011, 660 Non-trainable params: 0			

Q_o, and the bias terms, b_i, b_f and b_o, play a similar role to the weights and bias terms in the gates introduced previously, and the gates are defined as follows,

$$i(t) = \sigma(W_i U_{i,j}(t) + Q_i h(t-1) + b_i), \quad (9)$$

$$f(t) = \sigma(W_f U_{i,j}(t) + Q_f h(t-1) + b_f), \quad (10)$$

$$o(t) = \sigma(W_o U_{i,j}(t) + Q_o h(t-1) + b_o). \quad (11)$$

Each LSTM cell receives a candidate cell state from the last LSTM cell,

$$\tilde{c}(t) = \tanh(W_c U_{i,j}(t) + Q_c h(t-1) + b_c), \quad (12)$$

and the memory cell state, $c(t)$, and the final hidden state, $h(t)$ by computing

$$c(t) = f(t) \odot c(t-1) + i(t) \odot \tilde{c}(t), \text{ and} \quad (13)$$

$$h(t) = o(t) \odot \tanh(c(t)). \quad (14)$$

The GuLSTM and GuGRU models were implemented as baselines to compare the results with other more advanced models. Table 3 shows the parameters for both the GuLSTM and GuGRU models. The models differ in terms of the number of parameters in the first layer. Both models use one LSTM/GRU layer with 200 cells, followed by two fully connected layers with 100 and 50 neurons, respectively. The output layer produces a vector of 10 values representing the next 10 seconds.

B. BIDIRECTIONAL, STACKED AND CONVOLUTIONAL

Fig. 7 shows the structure of the BILSTM and the BIGRU models [52]. During training, these networks process the link utilization sequence in both directions and concatenate the forward and backward hidden states to create a final hidden state that contains information from both directions. This approach can improve the model's performance and provide additional information to the network for better predictions. However, these models are more computationally expensive than their unidirectional counterparts, as they require twice the number of training parameters as shown in Table 4. Both models use one GUBiLSTM/GUBiGRU layer with 200 cells, followed by two fully connected layers with 100 and 50 neurons, respectively. The output layer produces a vector of 10 values representing the next 10 seconds of link utilization.

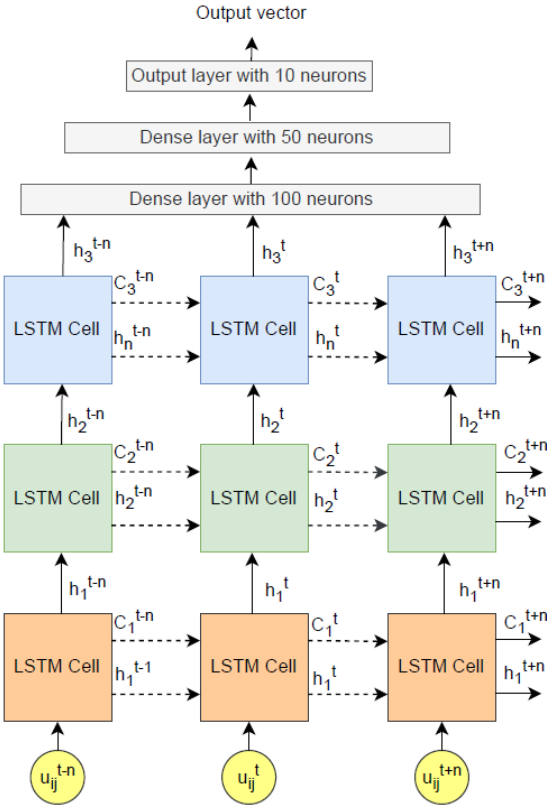


FIGURE 8. Visualization of the SLSTM model architecture.

Fig. 8 depicts the structure of the Stack LSTM model. Multiple LSTM layers are stacked on top of each other to create a deeper network. Each layer takes the output from the previous layer as an input. We evaluate this structure to determine if it can extract more complex patterns from the different sequences of DCN workloads. We also evaluate the associated computation time in real-time testing on an SDN. In these experiments, we stacked three LSTM layers followed by three fully connected layers. The parameters of the model are listed in Table 5.

CLRNs were proposed in [53] as a way to generate textual descriptions of videos and images. CLRNs leverage CNN layers for image feature extraction and LSTMs for sequence prediction. They have also recently been used for predicting

TABLE 5. SLSTM model parameters overview.

Vector output technique		
Layer type	Output Shape	Number of Note Parameters
CuDNNLSTM	(None, 10, 200)	162400
CuDNNLSTM	(None, 10, 200)	321600
CuDNNLSTM	(None, 100)	120800
Dense	(None, 100)	10100
Dense	(None, 50)	5050
Dense	(None, 10)	510
SLSTM trainable params: 620,460 Non-trainable params: 0		

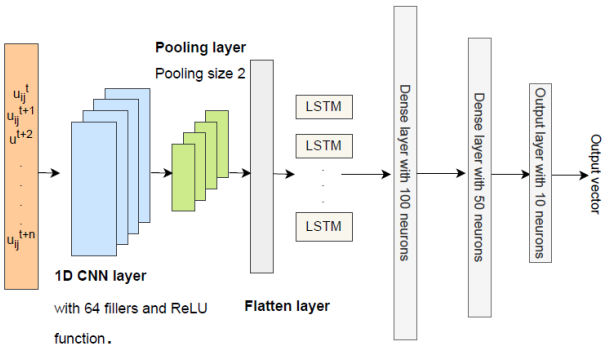


FIGURE 9. Visualization of the CLRN Model architecture.

traffic matrices [54]. To use CLRNs with link utilization matrices, we transformed the normalized link utilization matrix, **U**, into a time-series prediction problem for grayscale images, and used a CLRN model for sequence prediction in the following manner. Figure 9 illustrates the CLRN model used and Table 6 summarizes the characteristics of the model used. The CLRN models evaluated for link utilization prediction contained one CNN layer with one max pooling layer that interpreted sub-sequences of the input, which was provided as a sequence. When using the CLRN, we split the input sequences into two sub-sequences. For example. the sequence $U_{i,j}(1), U_{i,j}(2), \ldots U_{i,j}(T)$, was split into two sub-sequences of five samples, that the CNN model could process.

The TimeDistributed wrapper takes a layer as its argument and applies that layer to every time step in the input sequence. We applied it to the CNN layers, which allowed us to use the same CNN layers for each sub-sequence. This meant that we avoided using separate layers for each sub-sequence, which would have been computationally expensive. The CNN required several filters and the kernel size to be specified. The number of filters represents the number of interpretations of the input sequence. We set the number of filters to be 64. Increasing the number of filters did not always improve performance due to the small size of the input sub-sequences. The kernel size represented the number of time steps included in each interpretation operation of the input sequence. A kernel size of 1 or 2 was found to be appropriate when the ReLu function was used. Then we converted the output into a single long vector using a flattening layer to feed the GuLSTM layer. This architecture applied the convolutional and pooling layers as a local feature extractor that operated on the input sequence and identified local patterns and features in it. These local features were then passed to the LSTM layer, which was responsible for learning the long-term dependencies between these features and making predictions based on them.

IX. NUMERICAL EVALUATION

In this section we initially describe the computational environment used for evaluating link utilization prediction. We then evaluate the performance characteristics of the training and prediction routines used for each learning algorithm, focusing particularly on the convergence rate of the loss function, the accuracy of subsequent prediction steps and also the associated computation time. Having identified the most appropriate learning algorithm, we analyze real-time implementation considerations by integrating the selected ML model with the SM-FPLF algorithm and we analyze the level of activity on the OpenFlow channel to determine the level of power saving that is possible.

A. MODELS PERFORMANCE AND COMPUTATION TIME

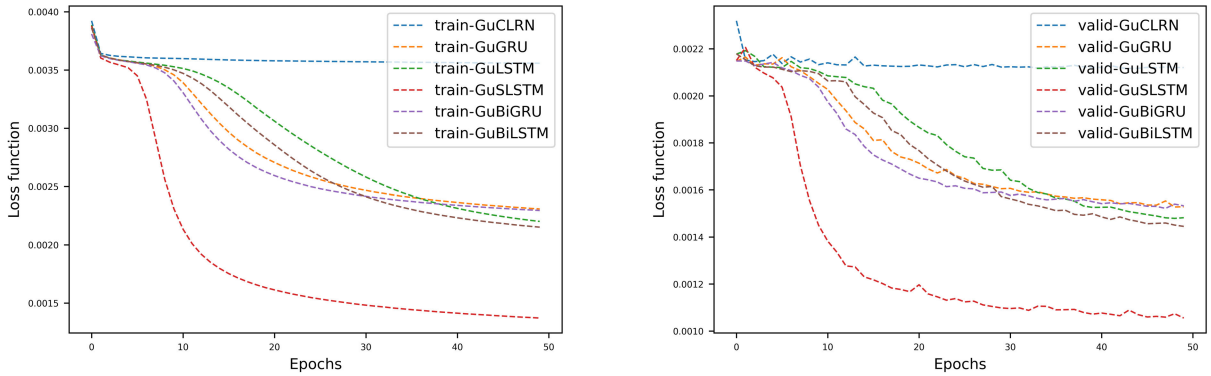
The models used in the training phase were implemented in Python 3.8 using GPU-TensorFlow 2.6. The Windows 11 operating system was used with 16 GB RAM, on a Core i7 processor with 12 logical processors. The NVIDIA GeForce GTX 1650 GPU was used with 4 GB RAM.

All models were trained on the first eight sequences of DCN workloads and tested on the last sequence. Models were trained using the ADAM optimizer and the Mean Absolute Error (MAE) loss function was used. A validation data size of 20% was used for all models. The number of epochs was set to 50 and the batch size was set to 256.

Fig. 10 shows the different learning curves of models during training. Models were compared using the MAE as the loss function, which was evaluated as a function of the number of epochs. The best learning curve, in terms of speed of convergence and minimization of the MAE, was observed for the GuSLSTM in Fig. 10 The worst convergence function was observed for the GuCLRN in Fig. 10. We believe that the poor performance for the GuCLRN may be explained by the small size of the input sub-sequence, which was 5×5. The CLRN model showed an improvement in the results when the size of the input sequence was 30. Evidence supporting this result is illustrated in Fig. 12. However, increasing the input sequence size is problematic, particularly, in light of the SDN domain-specific requirement we have regarding a small computation time, due to its long look-back sequence dependency. Performance improvements for other models was also observed when the input sequences were increased in length. Significantly, the performance of the GuBiLSTM and GuBiGRU models, which are illustrated in Fig. 10 did not outperform the unidirectional models, e.g. the GuLSTM and the GuGRU, despite having more training parameters.

TABLE 6. CLRN model parameters overview.

Vector output technique			
Layer taype	Output Shape	Number of Parameters	Notes
TimeDistributed(Conv1D)	(None, None, 2, 64)	2112	The flattening layer uses to transform 2D arrays resulting from pooling feature maps into a single linear vector.
TimeDistributed(MaxPooling1D)	(None, None, 1, 64)	0	
TimeDistributed(Flatten())	(None, None, 64)	0	
CuDNNLSTM	(None, 200)	212800	
Dense	(None, 100)	20100	
Dense	(None, 50)	5050	
Dense	(None, 10)	510	
Trainable params: 240,668 Non-trainable params: 0			



(a) Training Loss Progression Over Epochs: The GuSLSTM has the lowest cost.

(b) Validation Loss Trends During Model Training: The GuSLSTM has the lowest loss function value.

FIGURE 10. Comparison of model learning curves in terms of training loss progression and validation loss trends. All curves exhibit improvement with each epoch, which leads to a decrease in error values. The learning curves support the preference for the GuSLSTM model. Utilizing the mean absolute error criteria, the GuSLSTM's learning curve is characterized by faster convergence and effective objective minimization. Even though they have more training parameters, the GuBiLSTM and GuBiGRU models, do not outperform the unidirectional models, e.g. the GuLSTM and the GuGRU. This provides evidence that unidirectional models, which have smaller computational burdens, may be sufficient.

We conclude that using unidirectional models, which impose smaller computational burdens, may be sufficient.

To assess the performance of the RNN models in predicting link utilization, we conducted a comparison between the predicted and actual values. This comparison involved analyzing both the predicted and the ground-truth time series data for link utilization. These datasets were obtained from observations made on the Software-Defined Network (SDN).

To facilitate this evaluation, we serialized and stored all the trained algorithms using the Hierarchical Data Format (HDF5). This choice of storage format ensures efficient retention and easy retrieval of the models. In the subsequent testing phase, we invoked these serialized models within the same Python environment. This step exposed the models to previously unseen data, enabling us to test their performance on a new nine-sequence dataset (which we generated in VII-C). This dataset comprises approximately 378,000 observations. We summarize our findings as follows. All models had the ability to track the link utilization over time, but they produced errors when the link utilization changed suddenly, either by producing a sudden peak or a sudden trough. Fig. 11 plots the first true and predicted time step for each algorithm.

To quantify the error associated with each learning algorithm, for each time step, we computed the Root Mean Square Error (RMSE) between the ground-truth values and the predicted values,

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{U}_{i,j}(t) - U_{i,j}(t)\right)^2}. \tag{15}$$

In Fig. 11 the GuSLSTM model produced the most accurate predictions, even when the link utilization fluctuated quickly. In particular, it performed well when predicting sudden changes in link utilization values. We conclude that the GuSLSTM model was the best choice for the prediction component of the SM-FPLF algorithm. The SM-FPLF algorithm requires that a prediction algorithm used to perform link utilization prediction should be able to generate accurate prediction of what true future link utilization values will be. This allows the SM-FPLF algorithm to accurately change the cost of the links. The disadvantage of choosing a learning algorithm that yields inaccurate predictions is that this predictions might degrade the performance of SM-FPLF, particularly when the link utilization value fluctuates around the threshold.

Barplots in Fig. 13 summarize the RMSE results as the number of prediction steps into the future increases. The GuSLSTM model yielded the smallest error. As expected, the error increased gradually as the number of steps into

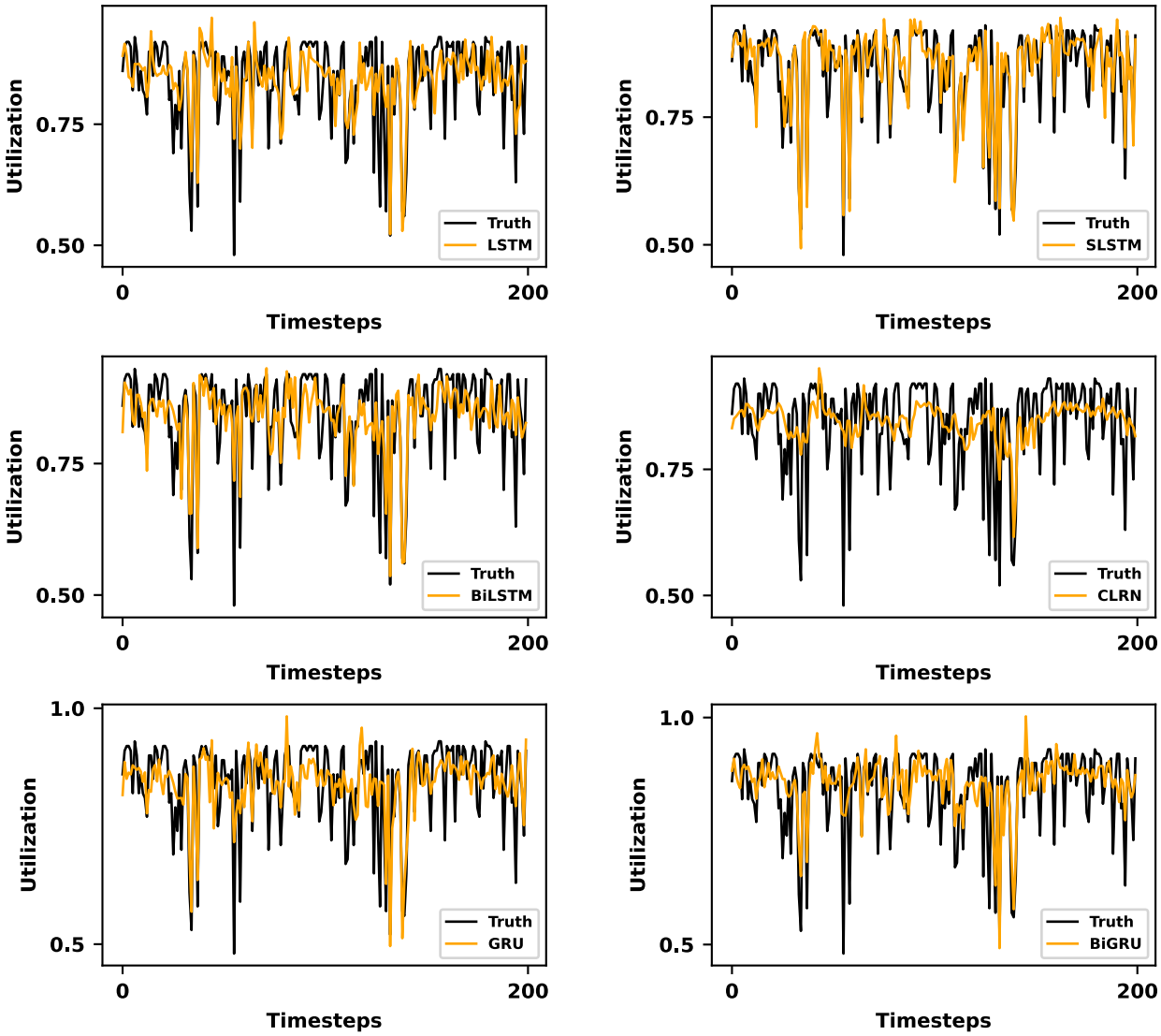


FIGURE 11. Prediction accuracy: The RMSE is plotted as a function of the time-step for each algorithm. Notably, the GuSLSTM generated superior prediction performance, characterized by its faster convergence and lower MAE during training. However, it is worth noting that all algorithms face challenges when accurately predicting abrupt changes in the link utilization time series.

the future for which we generated predictions increased. As a first impression, the current testing of the learning algorithms on unseen sequences, which originated from the same traffic characteristics and distribution but involve different DCN workloads, as detailed in Section VII-B, indicates their success in predicting the trend of link utilization. However, further evaluations of the nominated model (GuSLSTM) on real-time testing traffic, as presented and demonstrated in Section IX-E, are also provided.

Prediction error is a significant performance indicator, however, in a real-world setting the computation time is also determines if deploying an algorithm is feasible. We considered the prediction overhead of each prediction method to determine the best choice for a real-time implementation as a component of the SM-FPLF solution. Table 7 tabulates the training time, prediction time and prediction errors. The RMSE in the table represents the standard deviation of the residuals, revealing how tightly the observed data clusters around the predicted values, of the different prediction algorithms considered, for both the first and last prediction steps.

The GuCLRN, GuLSTM, and GuGRU models outperformed the other models in terms of training time and prediction time. These smaller training and prediction times are explained by the fact that they have fewer parameters than the other models. The GuSLSTM had a higher prediction time and training time; however, it outperformed the other models in terms of its prediction error. On the other hand, the GuBiGRU and GuBiLSTM models had the worst performance due the overhead imposed by their training and prediction time.

TABLE 7. Time-cost, trainable parameters, and RMSE measurement comparison. The GUSLSTM achieves the best first and last time-step RMSE measurement. The best scores are highlighted with bold font. The GuGRU has the lowest training and prediction time.

Metrics	GuLSTM	GuGRU	GuBiLSTM	GuBiGRU	GuSLSTM	CLRN
Number of trainable parameters	188,060	**147,460**	1,333,660	1,011,660	620,460	240,668
RMSE (First Time-Step)	0.04348	0.04491	0.04277	0.04477	**0.03521**	0.05626
RMSE (Last Time-Step)	0.05309	0.05407	0.05265	0.05412	**0.04497**	0.06450
Training time (sec)	2981	**2701**	5044	4393	7015	2449
Prediction time (sec)	17	**16**	29	30	26	17

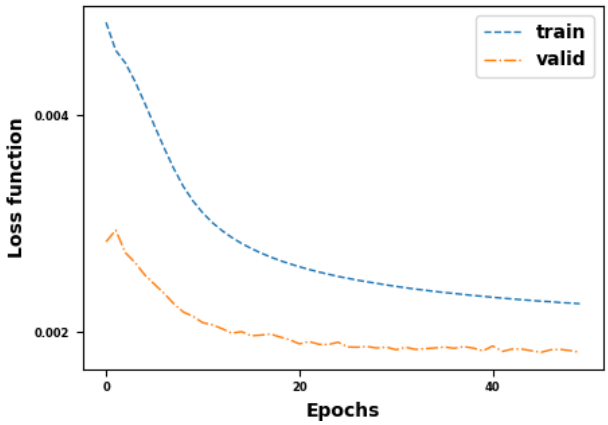


FIGURE 12. Effect of input sequence length on prediction accuracy: MAE error for 30-step input and 10-step prediction for the CLRN.

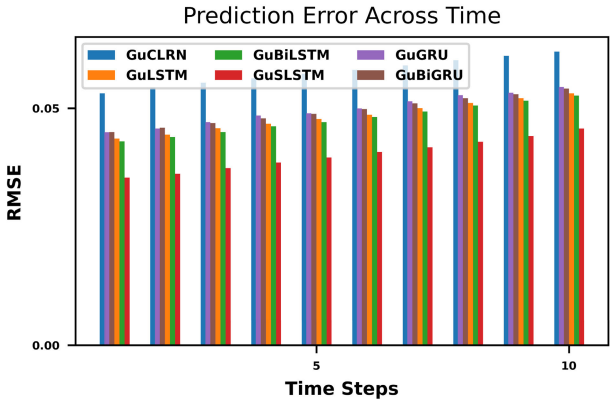


FIGURE 13. RMSE of the prediction generated by each algorithm compared to the ground-truth time series value. Predictions are compared for increasing numbers of time steps into the future. The GuSLSTM achieves the best performance. Predictions achieved by all values decrease as the prediction horizon increases.

In conclusion, this empirical evaluation demonstrated that the GuSLSTM was the most suitable RNN model for predicting link utilization in SD-DCNs networks. The model accurately predicted future values within an acceptable computation time. Evidence was obtained to support this result using a demanding scenario. The result in Table 7 shows the computation time was approximately 26 seconds and the first-step RMSE obtained was 0.03521. This makes GuSLSTM a viable option for real-time dynamic network management to optimize power usage using SM-FPLF.

B. MULTI-STEP SIZE AND RMSE VALUE

Two criteria that should be considered when the learning model is deployed in a real-time scenario are given as follows: (1) the number of predicted time-steps; and (2) the accuracy. Increasing the number of prediction time-steps causes the off time of the controller-OpenvFlow channel to be increased, which has the benefit of increasing power savings. We take the following approach with regard to the second consideration. To justify choosing a ten time-step prediction model, we adopt the strategy of increasing the number of time-steps and measuring the RMSE of each outcome. The results are compared with the results of the ten time-step model.

Increasing the number of prediction time-steps in multi-step time series problems may introduce more errors; a more in-depth treatment of this topic is given in [55]. We trained the GuSLSTM to predict twenty time-steps to assess the effect of doubling the number of time-steps on the RMSE value and compared the error with the error reported in Fig. 13 for ten time-steps.

The results in Fig. 14 indicate that the RMSE of the GuLSTM predictions increases as the time-step number increases in the twenty time-steps case. The RMSE increases even during the first ten time-steps, however, the RMSE is typically larger in the twenty time-step case than in the ten time-step case. This evidence suggests that the model may face challenges in learning and generalizing over the longer time-horizon. The time-cost of prediction for 378×10^3 observations (several thousands of link-states) increased to approximately 40 seconds for the twenty time-steps cases, compared to 26 seconds for the ten time-step scenario.

These results are consistent with those reported in the literature. According to [55], decreasing the number of prediction time-steps tends to reduce the error value. It is important to consider that the off time of the controller-OpenvFlow channel is a critical factor that we aim to maximize.

Following from the evaluation presented, we conclude that incorporating a ten time-step GuSLSTM prediction model in SM-FPLF, as demonstrated in Section IX-F, provides a good compromise between accuracy and time-cost. This claim is backed-up with empirical evidence. The accuracy of the model enabled SM-FPLF to achieve the same performance as the base-line algorithm, FPLF.

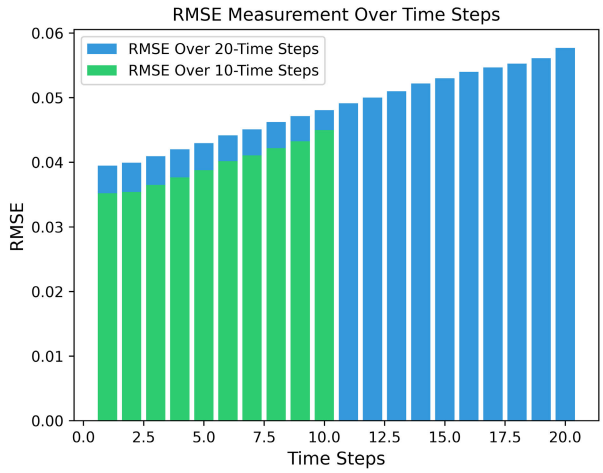


FIGURE 14. RMSE measurement for ten and twenty time-steps models as a function of time-step index. The RMSE increases as a function of the time-step index. The RMSE is larger for the twenty time-step model.

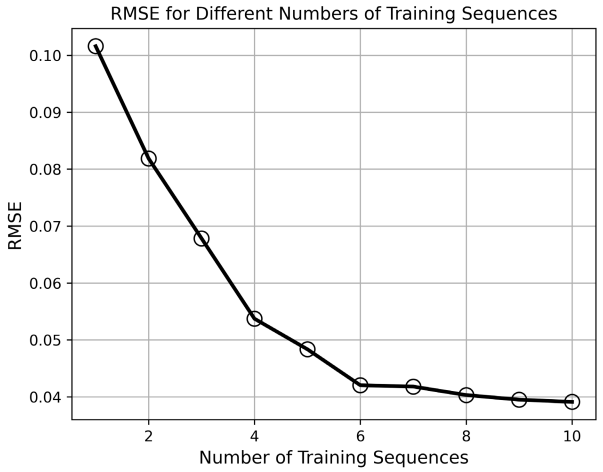


FIGURE 15. The average ten-step RMSE against the size of training sequences.

C. DATASET SIZE AND MODEL ACCURACY ANALYSIS

We adopt an empirical approach to determine the amount of training data required by SM-FPLF to be able to accurately track traffic patterns. A consequence of determining the appropriate training data set is that this allows SM-FPLF to perform and execute monitoring decisions correctly. We now describe how we determined that eight was an appropriate number of training sequences.

We trained the GuSLSTM for a set of different training sequences, $\{1, 2, 3, \ldots 10\}$ and computed the average RMSE obtained. Fig. 15 shows the average value of the RMSE obtained as a function of the number of training sequences.

The average RMSE decreases as a function of the number of sequences. The slope of this reduction is large for one up to six training sequences. The slope is greatly reduced when eight or more training sequences are used. We use the rate of change in RMSE as a function of the number of training sequences as a criteria for selecting eight as the appropriate value. The rate of decrease of the RMSE becomes small as we increase the number of sequences, which suggests that the increased computational effort, in terms of training and computing time, will not lead to significant increases in the algorithm's ability to track traffic trends.

D. WORKFLOW OF REAL-TIME IMPLEMENTATION

Before conducting real-time experiments, we introduce the workflow carried out by our real-time implementation of our hybrid model in Fig. 16. This figure emphasizes the interconnectedness between the polling scheme and the GuSLSTM model, along with the time required for the continuous processing of the statistics stream. It illustrates that the polling scheme and the learning model operate alternately, employing a time synchronization function to control the streaming direction of link state statistics.

Monitoring starts with a statistics collection process, which incurs a cost which is the sum of the execution times, $\Delta T_f = \Delta T_{req} + \Delta T_{rep}$. Consider the example where the execution time is $\Delta T_f = 1.808$ ms for twenty OpenFlow switches. This time corresponds to the difference between the timestamps of the first "SATAS-REQUEST" message and the last "SATAS-REPLY" message from twenty OpenFlow switches.

The calculation in Equation (5), incurs the time-cost of ΔT_u. This time-cost is in the order of nanoseconds and is considered to be negligible. We use the approximation, $\Delta T_m \approx 1.808$ ms, in the remainder of this discussion.

This operation is performed every second to faciltate the collection of the statistics necessary to perform predictions for ten time-steps. During this period, the polling scheme gathers the statistics that allow the SM-FPLF algorithm to calculate the energy saving paths set, $\mathbb{P}$. The polling scheme also provides the same values to the learning model. Once this step has been completed, polling is turned off. The values passed to the learning model are then used to predict the link states for future time-steps. The time-cost due to predicting the link states of the entire topology, ΔT_p, which consists of 48 links in a 4-ary fat-tree, is approximately 84.089 ms. This value was obtained empirically. Consequently, predictions of the next time-steps are available for the SM-FPLF algorithm, and the entire monitoring system pauses for the next period, T. This process ensures that the polling scheme and the GuSLSTM model interoperate seamlessly during real-time deployment, incurring an acceptable time-cost, and achieving an accurate prediction.

E. REAL-TIME IMPLEMENTATION

We consider the feasibility of integrating ML models with the SM-FPLF algorithm in this section. The goal is to minimize the level of communication activity on the OpenFlow channel, in the belief that minimizing traffic on this channel will lead to a reduction in power usage and SDN-controller overhead. In addition, we aim to achieve this reduction without causing a degradation in the performance of the

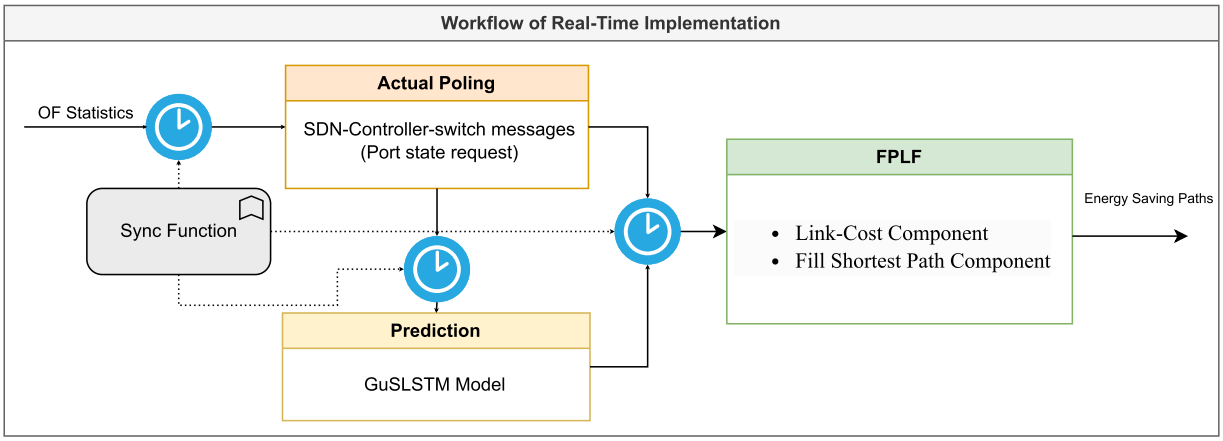


FIGURE 16. Workflow synchronization in the hybrid monitoring algorithm.

SM-FPLF algorithm components. We motivate some of the design decisions made in the implementation phase.

The following methodology was employed to evaluate the prediction accuracy of the learning approaches considered for SM-FPLF. We used the trade-off between accuracy and computation time-cost as a criterion to select the GuSLSTM to be the learning algorithm for SM-FPLF. This trade-off was performed using the model evaluation results presented in Fig. 11 and Table 7. The GuSLSTM accurately tracks the traffic, particularly when sudden changes occur. In addition, its moderate time-cost, in comparison with other learning models, qualify it for integration into SM-FPLF. We determined that implementing the GuSLSTM model would be the most suitable approach for predicting the next ten time-steps using recently recorded polling values.

To test the effectiveness of this approach, we reused the same test-bed that was described in Section VI. We integrated the GuSLSTM version of the SM-FPLF model into the SDN-controller for analysis. The purpose of this analysis was to determine the viability of our approach in terms of network performance, overhead reduction, and power usage. Before we could test the integrated model, we needed to synchronize the work of the polling scheme with the GuSLSTM. To achieve this, we implemented a scheduling algorithm and utilized multiple threads.

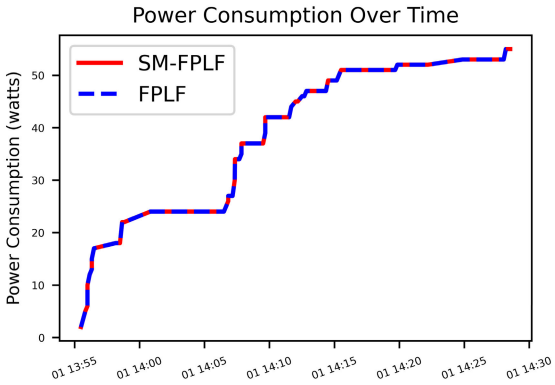
F. IMPLEMENTATION AND RESULTS

The experiments in [18] demonstrated that the FPLF algorithm could save power during periods of low traffic while maintaining reasonable QoS, such as by dropping packets. To test the performance of the algorithm with and without a prediction scheme and to determine whether the prediction model met the algorithm's requirements, we used a low-workload SDN network. We used the same traffic characteristics as described in Section VI-B1, but with new patterns of source and destination, which were generated using the Ranking Metrics and D-ITG approach. Multiple flows were gradually injected, and after approximately 35 minutes of the experiment, we recorded all port usage between the interconnecting switches. We implemented the experiment for both the actual scheme and the hybrid scheme. The results in Fig. 17a show that the algorithms consumed the same amount of power, during the simulation time, when the port speed was 10 Mbps, i.e., power consumption 0.2 watts. The results reaffirmed that the SM-FPLF model accurately predicted the utilization trend. Furthermore, SM-FPLF exhibited behavior similar to that of FPLF in optimizing power, with no discernible difference in the active topology subset. On the other hand, when we consider the OpenFlow channel, i.e., control plane, the results show a significant difference in power usage between the two algorithms in the time dimension. Fig. 17b shows the different accumulative power consumption in watts between the FPLF and SM-FPLF algorithms after 35 minutes. The OpenFlow channel power consumption using FPLF is denoted by C_{FPLF} and the OpenFlow channel power consumption using SM-FPLF is $C_{SM-FPLF}$. The power saving achieved using SM-FPLF is

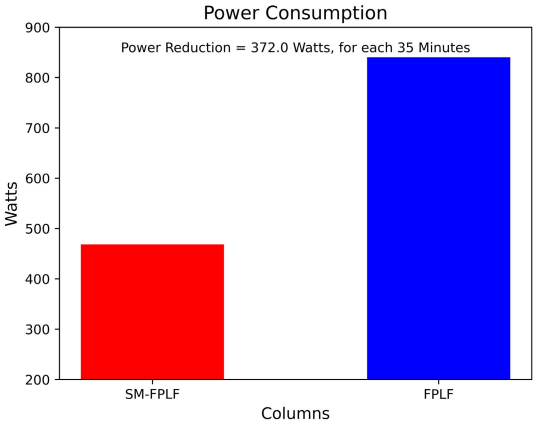
$$\Delta C = (C_{FPLF} - C_{SM-FPLF}) = 372 \text{ watts} \qquad (16)$$

per OpenFlow channel. We conclude that the GuSLSTM model maintains the performance of the SM-FPLF algorithm while reducing power consumption in the time dimension.

Wireshark was employed for the purpose of capturing and analyzing messages transmitted through the OpenFlow channel. This analysis considered the following packets: OFPT_PACKET_IN, OFPT_STATS_REPLY, OFPT_STATS REQUEST, OFPT_FLOW_MOD, OFPT_PACKET_OUT, and so on, for both scenarios. The results in Fig. 18 illustrate the difference in the number of transaction packets in the two scenarios. Specifically, Fig. 18a shows the actual pooling scheme scenario, where the number of packets reached up to 40 packets per second with frequent peaks of up to 80 packets per second. Whereas in Fig. 18b the prediction scheme scenario, the number of packets was approximately 40 with idle intervals when the ports were not in use. This presents an



(a) Power consumption of data plane (without counting the OpenFlow channel).



(b) Accumulated power consumption per OpenFlow channel in the DCN.

FIGURE 17. Power usage for SD-DCN data plane and control plane.

opportunity to save more power by incorporating it into the SM-FPLF algorithm.

The packets were then filtered to facilitate an analysis of the contribution of SM-FPLF, which is quantified here as a reduction in the port-state message overhead. As depicted in Fig. 18c, the difference between both scenarios is evident, where the number of packets remained approximately constant. This indicates that port-state messages make up the majority of transaction messages, in contrast to other types like OFPT_PACKET_IN. This result is reasonable given that the number of packets in our previous contribution [18] was reduced using one of the POX controller features. The optimization process in [18] involved sending the OFPT_FLOW_MOD message to all switches involved in the ESP instead of waiting for them to send OFPT_PACKET_IN messages to obtain the flow entry. This optimization process improved the installation of rules to the switches. The port state messages represents a significant component of the count of transaction messages, and the value remained at 40 packets/s. This result also serves to highlight the overhead associated with the polling mechanism used for such a power consumption optimization algorithms.

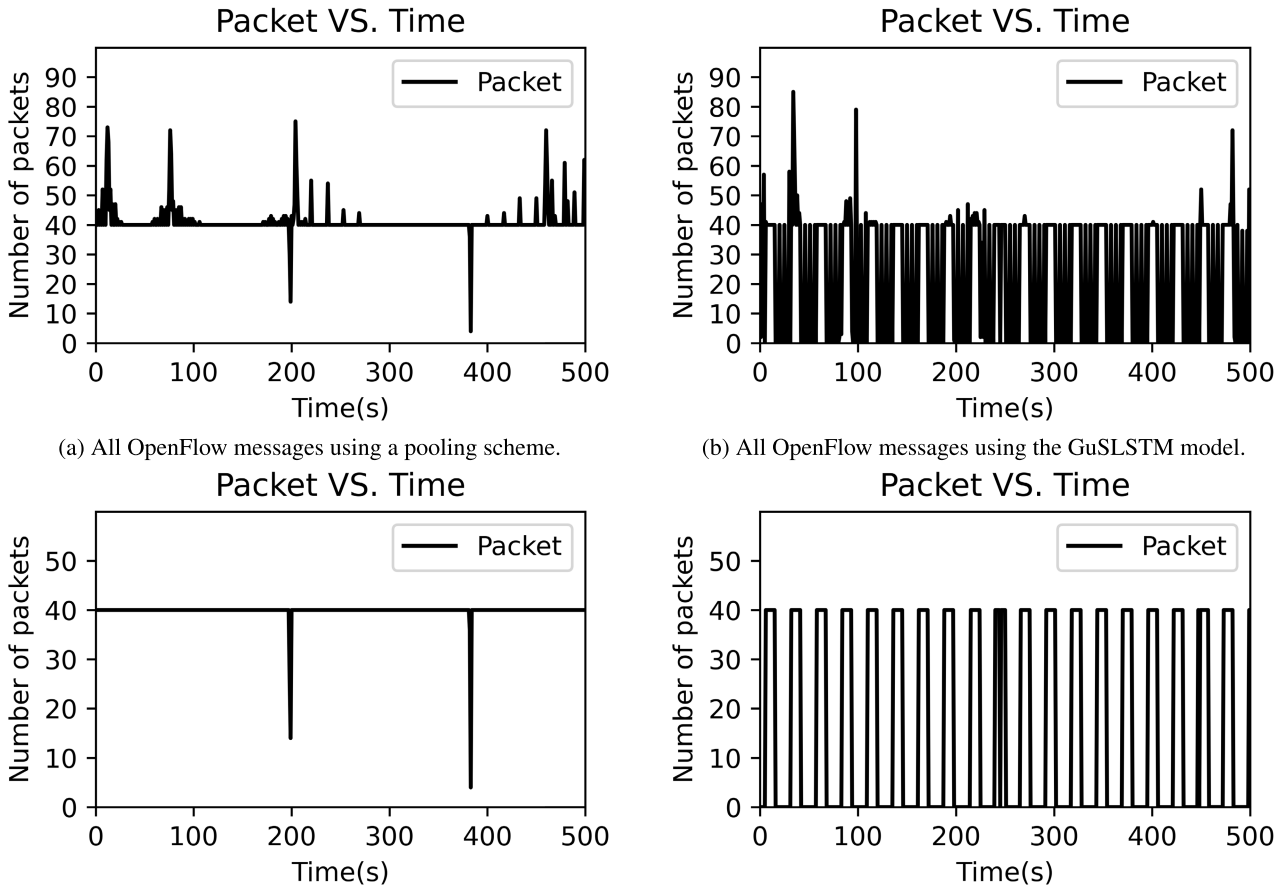
Conversely, with the SM-FPLP prediction scheme, the number of packets was approximately halved. This result is shown in Fig 18d. This halving occurs because it is sufficient to obtain ten actual utility values in the hybrid pooling scheme to predict ten feature values that are ready to be used by the SM-FPLF algorithm.

To measure CPU utilization with and without GuSLSTM, we employed an open-source Python package named psrecord [56]. This package provides a simple and effective way to record and analyze the activity of processes on a Linux system. We used psrecord to measure the CPU utilization of the ovs-switch process, which manages and controls any number of OpenvSwitch switches on the local machine. The OpenvSwitch datapath kernel module must be loaded for ovs-switch [57]. The tool tracked the CPU utilization during both scenarios. We report the results in Fig. 19. This enabled us to gain insights into the performance of the FPLF algorithm under both scenarios, e.g. with prediction and without, and to analyze the impact of GuSLSTM on the operation of the SM-FPLF. By comparing the CPU utilization graphs of the two scenarios, where Fig. 19b shows CPU usage with GuSLSTM and Fig. 19a shows CPU usage without the prediction scheme, we observed the performance improvements caused by GuSLSTM. Using SM-FPLF resulted in a 13.674% CPU usage reduction in the controller and the switches. Hence, the use of an ML model to manage the network topology and policies led to a lower processing requirement for the OpenFlow controller, resulting in a reduction in power consumption.

X. DISCUSSION AND LIMITATION

Despite the potential benefits of using hybrid actual/prediction monitoring models, for example FLPF and SM-FLPF, it is essential to take into account the upper and lower bounds for the number of time steps used in predictions. On one hand, this necessitates considering the algorithm's sensitivity and the potential for maximum/minimum errors. For example, in our case, the SM-FPLF algorithm adjusts the cost of a link based on its utilization value, as explained in Section VI-B4. The cost penalty in Equation (4) facilitates link selection in the next round of path installation or transitions links to an off state. Therefore, to ensure that the algorithm makes correct decisions, it should minimize errors in the vicinity of the utilization threshold. On the other hand, the time transition from idle or sleep mode to the ON state, which is out of coverage of the current study, is another crucial metric in this problem. For instance, we cannot take advantage of very short idle intervals (one or two seconds) to save power. This is why prediction techniques which successfully predict over a longer horizon should be considered. In short it is essential to achieve a trade-off between balancing the actual and prediction time-frame intervals to minimize errors while maintaining an acceptable prediction frame interval.

The scalability and traffic patterns of diverse data center topologies should be considered. This study uses the traffic patterns outlined in [41] as its benchmark. Applying SM-FPLF to other data center topologies with variations in load and traffic patterns could invoke the need for retraining



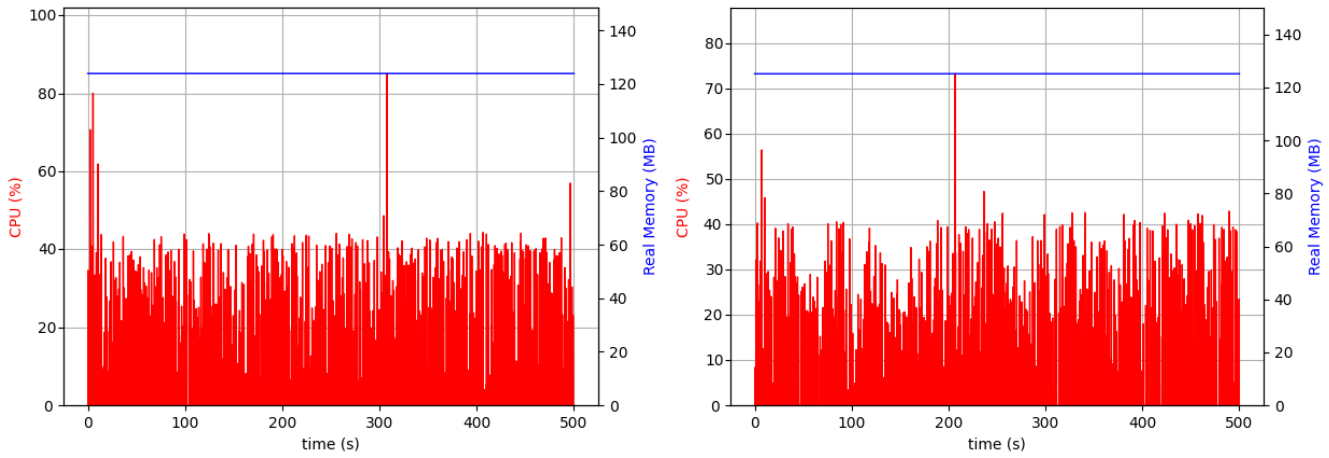
(a) All OpenFlow messages using a pooling scheme.

(b) All OpenFlow messages using the GuSLSTM model.

(c) Request/Response messages using a pooling scheme.

(d) Request/Response messages using the GuSLSTM model.

FIGURE 18. Analysis of network OpenFlow traffic with and without using the GuSLSTM model.



(a) CPU utilization without the GUSLSTM model.

(b) CPU Utilization with the GUSLSTM model

FIGURE 19. Comparison of the CPU usage when the GuSLSTM model is and is not used.

the model. This process is crucial for enabling the learning algorithm to adapt and recognize new patterns. Alternatively, Transfer Learning techniques could be employed to reduce training time. This approach would involve adding a few sequences that the model did not train on before. The authors of [58] reported that these techniques efficiently reduce the training time and make the learning phase easier.

On the other hand, in online prediction, time is crucial. As the number of links increases, the computation time also increases. For example, in Table 7, in the case of GuSLSTM, we reported a time-cost of 26 seconds for several thousand link-states. In such cases, computational resources in the data center should be increased, and the load should be distributed across multiple controllers to meet the system requirements for which the learning model was developed.

Generally, decisions regarding the application of RNNs in real-time SDN solutions are contingent upon the specific context and requirements. The model's capability to meet sensitivity and accuracy criteria for the given application is a crucial factor in this consideration.

XI. CONCLUSION

SD-DCN power consumption algorithms often rely on an active monitoring model to periodically poll DCN statistics, resulting in the following challenges: (1) a large SDN-controller overhead, and (2) a large energy consumption. The study presented in this paper proposes SM-FPLF as a practical solution to address these challenges. To minimize power consumption, our primary approach involves reducing the number of packets exchanged between the OvS and SDN Controller, with idle intervals when the ports are not in use. Additionally, the proposed framework reduces CPU usage by minimizing the use of the OvS-Controller link, thus reducing the controller's workload. In terms of theoretical implications, our study underscores the effectiveness of integrating learning algorithms in power optimization frameworks for DCNs. On a practical level, the proposed Smart SD-DCNs framework demonstrates good benefits in terms of power consumption reduction and monitoring efficiency.

We contribute a new algorithm called SM-FPLF that optimizes power consumption by considering use of this control channel. SM-FPLF complements the contribution in [18], which is called FPLF, and advances the state-of-the-art by incorporating learning algorithms into FPLF to further optimize performance. This contribution is significant because these links can affect the overall energy consumption of DCNs especially when they are utilized in the monitoring process. This contribution may be arranged into three categories: Dataset, Modelling and Real-time implementation. The dataset component of the contribution is centred on a new method for generating the Utilization Matrix by utilizing realistic traffic distributions. This contribution facilitates the collection of training, validation, and testing data for our analysis, but will also yield a dividend for the community, given the interest in the deployment of learning algorithms in this setting. The modeling contribution focuses on the design of a DL model that meets the requirement sof the FPLF algorithm so that overhead is reduced whilst power usage is optimized for both controllers and switches in a DC-SDN architecture. The impact of this contribution is strengthened by providing an evaluation of a real-time implementation of SM-FPLF.

The practical advantages of using SM-FPLF are given as follows. In our evaluations using the RMSE metric, GUSLSTM outperformed other learning algorithms in terms of error reduction. Our real-time testing experimental results demonstrate that the proposed framework can reduce the network's power consumption by 372 watts per OpenFlow channel every 35 minutes of DCN operation, which represents a significant reduction in power consumption over time. Furthermore, it significantly reduces monitoring overhead by 13.674% when compared to the FPLF approach from the state-of-the-art, thus mitigating SDN controller throttling during peak times.

A common limitation of work which seeks to perform predictions for data center traffic are the issues of algorithm scalability and the time-varying traffic patterns that occur in diverse data center topologies. The use of benchmark datasets is often advocated as it ensures that the test scenarios considered are widely cited, which indicates that there is a broad consensus about their suitability. However this reliance on benchmark data sets has the drawback that the available benchmarks may not be sufficiently expressive to capture all types of traffic behaviour. We have discussed remedial actions such as retraining the model to ensure its effectiveness in recognizing new patterns. We also commented on the possibility of using transfer Learning techniques to expedite training time.

As a part of our future work, we will consider the performance benefits that might arise from deploying new, faster implementations of existing ML and DL techniques. An additional future research suggestions is to extend SM-FPLF so that it scales with increasing data center size and adapts to the traffic patterns of diverse data center topologies. One potential research approach would be to apply Transfer Learning techniques to reduce training time in the setting where data center topologies, loads and traffic patterns vary with time, in order to reduce retraining time for the model. Alternatively, this time-cost issue could be addressed via a multi-controller solution; the aim of this work would be to reduce the energy usage by considering the placement of multiple controllers. Finally, application layer components such as smart traffic classification and congestion traffic prediction could be used to improve real-time streaming protocols and lead to better network performance [59]. The ultimate goal is to develop a fully automated and intelligent SDN network that can adapt to changing demands, and optimize energy consumption while considering network performance in real time.

XII. ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to the Science Foundation Ireland for their generous support of this research. They would also like to thank the Department of Information Technology, University of Debrecen, and the School of Electrical and Electronic Engineering, TU Dublin, for providing them with the necessary resources and facilities to carry out this study.

REFERENCES

[1] Y. Zhang, K. Shan, X. Li, H. Li, and S. Wang, "Research and technologies for next-generation high-temperature data centers—State-of-the-arts and future perspectives," *Renew. Sustain. Energy Rev.*, vol. 171, Jan. 2023, Art. no. 112991.

[2] M. Nsaif, G. Kovásznai, A. Malik, and R. de Fréin, "Survey of routing techniques-based optimization of energy consumption in SD-DCN," *Infocommun. J.*, vol. 15, no. 1, pp. 35–42, 2023.

[3] X. Gao, A. Curtis, and B. Wong, "It's not easy being green," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 1, pp. 211–222, Aug. 2012.

[4] J. Koomey, "Growth in data center electricity use 2005 to 2010," *Completed Request New York Times*, vol. 9, p. 161, Aug. 2005.

[5] P. Sun, Z. Guo, S. Liu, J. Lan, J. Wang, and Y. Hu, "SmartFCT: Improving power-efficiency for data center networks with deep reinforcement learning," *Comput. Netw.*, vol. 179, Oct. 2020, Art. no. 107255.

[6] M. D. S. Conterato, T. C. Ferreto, F. Rossi, W. D. S. Marques, and P. S. S. de Souza, "Reducing energy consumption in SDN-based data center networks through flow consolidation strategies," in *Proc. 34th ACM/SIGAPP Symp. Appl. Comput.*, Apr. 2019, pp. 1384–1391.

[7] G. Kovásznai and M. Nsaif, "Integer programming based optimization of power consumption for data center networks," in *Proc. 13th Conf. PhD Students Comput. Sci.* Szeged, Hungary: Institute of Informatics, Univ. of Szeged, 2023, doi: 10.14232/actacyb.299115.

[8] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. IEEE 36th Annu. Found. Comput. Sci.*, Oct. 1995, pp. 374–382.

[9] J. Luo, S. Zhang, L. Yin, and Y. Guo, "Dynamic flow scheduling for power optimization of data center networks," in *Proc. 5th Int. Conf. Adv. Cloud Big Data (CBD)*, Aug. 2017, pp. 57–62.

[10] Y. Shang, D. Li, and M. Xu, "Greening data center networks with flow preemption and energy-aware routing," in *Proc. 19th IEEE Workshop Local Metrop. Area Netw. (LANMAN)*, Apr. 2013, pp. 1–6.

[11] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[12] P.-W. Tsai, C.-W. Tsai, C.-W. Hsu, and C.-S. Yang, "Network monitoring in software-defined networking: A review," *IEEE Syst. J.*, vol. 12, no. 4, pp. 3958–3969, Dec. 2018.

[13] C. Huang, J. Zhang, and T. Huang, "Updating data-center network with ultra-low latency data plane," *IEEE Access*, vol. 8, pp. 2134–2144, 2020.

[14] G. Xu, Y. Mu, and J. Liu, "Inclusion of artificial intelligence in communication networks and services," *ITU J. ICT Discov. Spec*, vol. 1, pp. 1–6, Oct. 2017.

[15] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 393–430, 1st Quart., 2019.

[16] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving energy in data center networks," in *Proc. NSDI*, vol. 10, 2010, pp. 249–264.

[17] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "CARPO: Correlation-aware power optimization in data center networks," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1125–1133.

[18] M. Nsaif, G. Kovásznai, A. Rácz, A. Malik, and R. de Fréin, "An adaptive routing framework for efficient power consumption in software-defined datacenter networks," *Electronics*, vol. 10, no. 23, p. 3027, Dec. 2021.

[19] G. Kovásznai and M. Nsaif, "Integer programming based optimization of power consumption for data center networks," in *Proc. 13th Conf. Ph.D Students Comput. Sci.*, 2022, pp. 76–80.

[20] D. Li, Y. Yu, W. He, K. Zheng, and B. He, "Willow: Saving data center network energy for network-limited flows," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 9, pp. 2610–2620, Sep. 2015.

[21] G. Xu, B. Dai, B. Huang, and J. Yang, "Bandwidth-aware energy efficient routing with SDN in data center networks," in *Proc. IEEE 17th Int. Conf. High Perform. Comput. Commun. 7th Int. Symp. Cyberspace Saf. Secur., IEEE 12th Int. Conf. Embedded Softw. Syst.*, Aug. 2015, pp. 766–771.

[22] The Open Networking Foundation, *OpenFlow Switch Specification Version 1.3.3*. Menlo Park, CA, USA: The Open Networking Foundation, 2013.

[23] R. de Fréin, O. Izima, and A. Malik, "Detecting network state in the presence of varying levels of congestion," in *Proc. IEEE 31st Int. Workshop Mach. Learn. Signal Process. (MLSP)*, Oct. 2021, pp. 1–6.

[24] R. de Frein, S. Rickard, and K. Drakakis, "Extracting garch effects from asset returns using robust NMF," in *Proc. IEEE 13th Digit. Signal Process. Workshop 5th IEEE Signal Process. Educ. Workshop*, Jan. 2009, pp. 200–205.

[25] J. Dai and J. Li, "VBR MPEG video traffic dynamic prediction based on the modeling and forecast of time series," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Aug. 2009, pp. 1752–1757.

[26] P. Cortez, M. Rio, M. Rocha, and P. Sousa, "Internet traffic forecasting using neural networks," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2006, pp. 2635–2642.

[27] V. B. Dharmadhikari and J. D. Gavade, "An NN approach for MPEG video traffic prediction," in *Proc. 2nd Int. Conf. Softw. Technol. Eng.*, vol. 1, Oct. 2010, pp. 1–57.

[28] A. Azzouni and G. Pujolle, "NeuTM: A neural network-based framework for traffic matrix prediction in SDN," in *Proc. IEEE/IFIP Netw. Operations Manag. Symp.*, Apr. 2018, pp. 1–5.

[29] T. Lisas and R. de Fréin, "Sequential learning for modeling video quality of delivery metrics," *IEEE Access*, vol. 11, pp. 107783–107797, 2023.

[30] D.-H. Le, H.-A. Tran, S. Souihi, and A. Mellouk, "An AI-based traffic matrix prediction solution for software-defined network," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2021, pp. 1–6.

[31] D. Aloraifan, I. Ahmad, and E. Alrashed, "Deep learning based network traffic matrix prediction," *Int. J. Intell. Netw.*, vol. 2, pp. 46–56, Jan. 2021.

[32] Z. Liu, Z. Wang, X. Yin, X. Shi, Y. Guo, and Y. Tian, "Traffic matrix prediction based on deep learning for dynamic traffic engineering," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2019, pp. 1–7.

[33] A. Malik and R. de Fréin, "A proactive-restoration technique for SDNs," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2020, pp. 1–6.

[34] A. Botta, W. de Donato, A. Dainotti, S. Avallone, and A. Pescapé, "D-ITG 2.8. 1 manual," *Comput. Interact. Commun. (COMICS) Group*, pp. 3–6, Oct. 2013.

[35] M. S. Yoon and A. E. Kamal, "Power minimization in fat-tree SDN datacenter operation," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–7.

[36] T. M. Nam, N. H. Thanh, N. Q. Thu, H. T. Hieu, and S. Covaci, "Energy-aware routing based on power profile of devices in data center networks using SDN," in *Proc. 12th Int. Conf. Electr. Eng./Electron., Comput., Telecommun. Inf. Technol. (ECTI-CON)*, Jun. 2015, pp. 1–6.

[37] F. Kaup, S. Melnikowitsch, and D. Hausheer, "Measuring and modeling the power consumption of OpenFlow switches," in *Proc. 10th Int. Conf. Netw. Service Manage. (CNSM) Workshop*, Nov. 2014, pp. 181–186.

[38] A. Malik, R. de Fréin, and B. Aziz, "Rapid restoration techniques for software-defined networks," *Appl. Sci.*, vol. 10, no. 10, p. 3411, May 2020.

[39] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[40] M. R. Nsaif, M. F. Abbood, and A. F. Mahdi, "Detection and prevention algorithm of ddos attack over the IoT networks," *TEM J.*, vol. 9, no. 3, p. 899, 2020.

[41] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 92–99, Jan. 2010.

[42] A. Botta, A. Dainotti, and A. Pescapé, "Do you trust your software-based traffic generator?" *IEEE Commun. Mag.*, vol. 48, no. 9, pp. 158–165, Sep. 2010.

[43] A. Hagberg, P. Swart, and D. S. Chult, "Exploring network structure, dynamics, and function using NetworkX," Los Alamos National Lab. (LANL), Los Alamos, NM, USA, Tech. Rep. LA-UR-08-05495; LA-UR-08-5495; TRN: US201006%%1254, 2008.

[44] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[45] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, Oct. 2010, pp. 1–6.

[46] A. Nucci, A. Sridharan, and N. Taft, "The problem of synthetically generating IP traffic matrices: Initial recommendations," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 3, pp. 19–32, Jul. 2005.

[47] L. Saino, C. Cocora, and G. Pavlou, "A toolchain for simplifying network simulation setup," in *Proc. 6th Int. Conf. Simul. Tools Techn.*, Brussels, Belgium, 2013, pp. 1–10.

[48] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space Odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.
[49] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," 2014, *arXiv:1406.1078*.
[50] *NVIDIA's Documentation on CuDNN*. Accessed: Jan. 30, 2023. [Online]. Available: https://docs.nvidia.com/deeplearning/cudnn/index.html
[51] *NVIDIA's Tf.Keras.Layers.LSTM*. Accessed: Feb. 1, 2023. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM
[52] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.
[53] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko, "Long-term recurrent convolutional networks for visual recognition and description," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 2625–2634.
[54] V. A. Le, P. Le Nguyen, and Y. Ji, "Deep convolutional LSTM network-based traffic matrix prediction with partial information," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, Apr. 2019, pp. 261–269.
[55] A. Venkatraman, M. Hebert, and J. Bagnell, "Improving multi-step prediction of learned time series models," in *Proc. AAAI Conf. Artif. Intell.*, 2015, vol. 29, no. 1, pp. 1–7.
[56] *Psrecord*. Accessed: Mar. 1, 2023. [Online]. Available: https://pypi.org/project/psrecord/
[57] *Ovs-vswitchd*. Accessed: Mar. 1, 2023. [Online]. Available: https://manpages.ubuntu.com/manpages/bionic/man8/ovs-vswitchd.8.html
[58] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proc. IEEE*, vol. 109, no. 1, pp. 43–76, Jan. 2021.
[59] M. Nsaif, G. Kovásznai, M. Abboosh, A. Malik, and R. D. Fréin, "ML-based online traffic classification for SDNs," in *Proc. IEEE 2nd Conf. Inf. Technol. Data Sci. (CITDS)*, May 2022, pp. 217–222.

MOHAMMED NSAIF received the M.S. degree in infocommunication technologies and communication systems from Kazan National Research Technical University, Russia. He is currently pursuing the Ph.D. degree with the Department of Information Technology, Faculty of Informatics, University of Debrecen. His research interests include software-defined networking, computer networks, wireless sensor networks, and machine learning.

GERGELY KOVÁSZNAI received the Ph.D. degree in formal methods and automated theorem proving from the University of Debrecen, Hungary, in 2007. He is currently an Associate Professor and the Head of the Department of Computational Science, Eszterházy Károly Catholic University, Eger, Hungary. Over the years, he was a Research Fellow with the Aristotle University of Thessaloniki, Greece; the Johannes Kepler University Linz, Austria; and the Vienna University of Technology, Austria. His research interests include formal methods, formal verification, operations research, and machine learning.

ALI MALIK received the Ph.D. degree in computing from the University of Portsmouth, U.K., in 2019. He was a Postdoctoral Researcher with the FOCAS Research Institute, Technological University Dublin, in areas related to data centers, monitoring, and software-defined networking. He is currently an Assistant Lecturer in computer engineering with the School of Electrical and Electronic Engineering, Technological University Dublin, Ireland. His current research interests include software-defined networks, vehicular networks, traffic engineering, machine learning, cybersecurity, microgrids, and power networks.

RUAIRÍ DE FRÉIN received the B.E. degree in electronic engineering and the Ph.D. degree in time-frequency analysis and matrix factorization from University College Dublin (UCD), Ireland, in 2004 and 2010, respectively. He is currently a CONNECT Funded Investigator and a Lecturer with the School of Electrical and Electronic Engineering, Technological University Dublin, Ireland. He held Marie Skłodowska-Curie fellowships at the KTH Royal Institute of Technology, Stockholm, and Amadeus SAS, Sophia Antipolis, France. Over the past few years, he has developed algorithms for predicting quality-of-delivery metrics for network management and monitoring strategies for small cell networks, and monitoring techniques for Internet Protocol TeleVision (IPTV). His research interests include machine learning, sparse signal processing, software-defined networks, vehicular networks, microgrids, and power networks.