Traffic Prediction-Based Routing in Software-Defined Networking Using Long Short-Term Memory and Harmony Search

Khadijeh Mirzaei Talarposhti[1], Sam Jabbehdari[2*], Amir Masoud Rahmani[3]

Abstract – Efficient traffic management in Software-Defined Networking (SDN) relies heavily on accurate traffic prediction to enable intelligent routing decisions. In this paper, we propose a novel routing framework that integrates deep learning-based traffic forecasting with an optimization-driven path selection mechanism. Specifically, Long Short-Term Memory (LSTM) networks are employed to predict future traffic patterns based on time-series data extracted from SDN switches. To enhance prediction accuracy, Fast Fourier Transform (FFT) is used to enrich the input features with dominant traffic harmonics. Furthermore, key forecasting parameters such as the number of lags, labels, and shifts are optimized using the Harmony Search (HS) algorithm. The predicted traffic volumes are then utilized to dynamically adjust routing paths, aiming to minimize latency, jitter, and packet loss while improving throughput. The proposed framework is implemented on a Mininet-based SDN testbed using a customized Ryu controller. Experimental results under both regular and chaotic traffic scenarios demonstrate that the LSTM-based forecasting significantly improves routing efficiency compared to traditional approaches, especially when enhanced by FFT preprocessing and HS optimization. This study highlights the potential of combining predictive intelligence and metaheuristic tuning to achieve adaptive and robust traffic-aware routing in SDN environments.

Keywords: Software Defined Networking, Long Short-Term Memory, Harmony Search, Intelligent Routing

1. Introduction

The increasing demand for high-speed and intelligent network services has driven the evolution of network architectures beyond the capabilities of traditional systems. Software-Defined Networking (SDN) has emerged as a powerful paradigm that decouples the control plane from the data plane, offering centralized management and programmability. These features enable the dynamic implementation of advanced services, such as real-time traffic engineering, adaptive routing, and predictive analytics.

Among these services, traffic-aware routing has garnered considerable attention due to its potential to enhance Quality of Service (QoS) by adjusting routing strategies to current or predicted network conditions. While conventional SDN routing mechanisms often rely on reactive metrics such as instantaneous link utilization or delay, recent research has demonstrated that incorporating traffic prediction can significantly enhance routing performance and stability, particularly in highly dynamic or congested environments.

In this study, we propose a novel routing framework for SDN that leverages traffic prediction using deep learning techniques. Specifically, Long Short-Term Memory (LSTM) networks are employed to forecast future traffic volumes based on time series collected from SDN switches. To improve the forecasting accuracy, we incorporate Fast Fourier Transform (FFT) to extract dominant periodic features from the traffic data, thereby enhancing the learning capability of the neural network.

Additionally, we address the challenge of parameter tuning in time series modeling by integrating the Harmony

1 Department of Computer Engineering, NT.C., Islamic Azad University, Tehran, Iran. Email: Mirzaei.iausari@gmail.com

2* Corresponding Author: Department of Computer Engineering, NT.C., Islamic Azad University, Tehran, Iran Email: Sam.jabbedari@iau.ac.ir

1 Department of Computer Engineering, SR.C., Islamic Azad University, Tehran, Iran. Email: rahmani74@yahoo.com

Received: 2025.09.23; Accepted: 2025.11.02

Search (HS) metaheuristic. This enables the optimization of key parameters, such as lag length, forecasting horizon, and data shift, resulting in more effective traffic prediction. The predicted traffic levels are then used to inform routing decisions in the SDN controller, enabling proactive and intelligent path selection.

To validate our approach, we implement the proposed framework on a Mininet-emulated SDN environment with a customized Ryu controller. Our experiments encompass both regular and chaotic traffic patterns to assess the system's robustness. The results show that the proposed prediction-based routing approach significantly outperforms baseline methods in terms of latency, jitter, packet loss, and throughput. The main contributions of this paper are as follows:

- A predictive routing framework in SDN that leverages LSTM-based traffic forecasting to guide path selection.
- Integration of FFT-based feature enrichment to enhance the quality of traffic time series before model training.
- Use of Harmony Search optimization to tune key forecasting parameters and improve prediction performance.
- Design and implementation of a proactive routing module in the Ryu controller that adapts to forecasted traffic trends.
- Experimental validation under diverse traffic conditions (including chaotic scenarios) showing improved network metrics such as delay, jitter, loss, and throughput.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 describes the architecture and components of the proposed system. Section 4 outlines the implementation and experimental setup. Section 5 discusses results and evaluation. Finally, Section 6 concludes the paper and suggests directions for future research.

2. Related Works

The concept of software-defined networking offers unique characteristics that enable researchers to design powerful Traffic Engineering (TE) systems. A TE framework usually consists of a traffic measurement and a traffic management component [1]. A TE system that runs at the SDN controller utilizes the collected information to equip the network with certain desired functionalities by implementing them as network services [2]. Agarwal et al. [3] showed that the problem of computing the optimal paths in an SDN can be formulated as a multi-commodity flow problem. For this purpose, the controller can route traffic flows over multiple paths, resulting in an optimization problem that the controller solves in polynomial time.

Despite the extraordinary capabilities of the SDN controller in routing flows, applying the routing process to each flow will exceed the controller's computational capacities in real-world scenarios. Therefore, Hedera, a dynamic flow scheduling system [4], considers large flows, called elephant flows, and optimally reroutes them. Another TE system that detects elephant flows is called Mahout [5]. The difference is that Mahout monitors the host's socket buffer instead of polling network information to identify elephant flows. This reduces controller overhead. Cognitive Routing Engine (CRE) uses a cognitive routing algorithm module to find optimal paths while keeping low overhead in the controller traffic [6]. Another research that performs multipath routing instead of a single best route comprises the architecture in five components: detection of elephant flows, computing the shortest path, estimating delay and utilization, calculating the least-cost path, and rerouting traffic flows to the best path [7]. As shown above, the detection of elephant flows is a crucial component of many TE systems in the SDN research area. Hence, several research studies focus on effectively detecting elephant flows. For instance, Geremew and Ding used deep neural networks to detect elephant flows [8]. They apply a Convolutional Neural Network (CNN), Long-Short Term Memory (LSTM), and an autoencoder as the proposed method to prevent network congestion.

Recently, some research studies have relied on predicting the future state of the network to manage network traffic more effectively [2] To achieve this purpose, an efficient method for predicting network traffic is needed. The problem of predicting future traffic can be treated as a time series problem. Traditionally, mathematical techniques such as Autoregressive Moving Average (ARMA) and Autoregressive Integrated Moving Average (ARIMA) models were used in some applications, such as BitTorrent [9], file transfer [10], traffic prediction in 3G mobile networks [11], and public safety networks [12]. Since ARMA and ARIMA are designed to model linear systems, Anand et al. showed that Generalized Autoregressive Conditional Heteroskedasticity (GARCH) outperforms traditional regression systems on burst internet traffic with a change of variance over time [13].

Due to the limitations of mathematical models for time

series analysis, artificial intelligence-based methods are widely used in predicting network traffic. Multiple Layer Perceptron (MLP) [14, 15], dynamic Bilinear Recurrent Neural Networks (BLRNNs) [16], Adaptive Neuro-Fuzzy Inference System (ANFIS) [17], and Support Vector Machines (SVMs) that are improved by ant colony optimization [18] have been successfully tested on predicting network traffic. Another problem related to traffic management is traffic classification. Traffic classification categorizes the traffic, facilitating an understanding of the data flow [19]. It enables the proper utilization of network resources, provides Quality of Service (QoS), and prevents network attacks [20]. Deep learning algorithms are widely used in this area of research [19].

After these studies, we decided to develop an efficient predictive system that runs at the SDN controller. In the proposed method, the most popular deep learning algorithms are compared. It appears that the preprocessing section of such a system is a crucial part of the project, having a direct impact on the system's performance. The following sections will present the proposed method in detail.

3. Prerequisites

3.1 SDN Architecture and OpenFlow Protocol

Software Defined Networking (SDN), because of its programmability and dynamic features, can improve network management [21]. In the SDN architecture, as depicted in Fig. 1, there are three layers: the data, control, and application layers. The data layer comprises switches and contains flow tables that are responsible for forwarding traffic. The centralized control layer is responsible for controlling and installing paths in flow tables. Lastly, the application layer encompasses network services and applications. The control layer communicates with the data and application layers through northbound and southbound interfaces, respectively [22].

One of the most popular SDN protocols is the OpenFlow protocol. In this protocol, all data packets with the same characteristics within the network are recognized as a flow [23]. Various parameters, including source and destination IP addresses, destination port numbers, and more, are used to match the characteristics of a flow. Matching and filtering of these parameters are performed in OpenFlow switches, where each data flow is represented as an entry in a table called the flow table.

As shown in Fig. 2, the flow table in the OpenFlow protocol replaces the traditional routing table. When a new packet enters a switch, it is first searched for in the flow table. If there is no match, it is then sent to the controller. After the controller routes the packet, it installs that route as a new flow entry in the switch's flow table

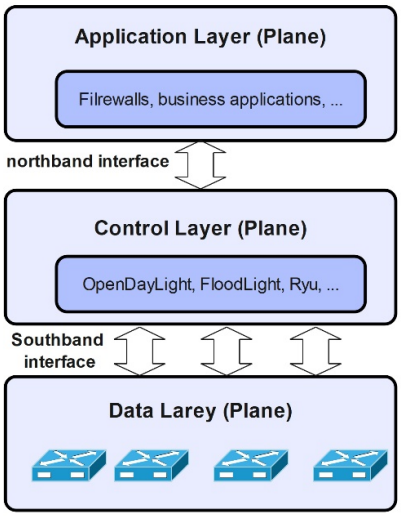
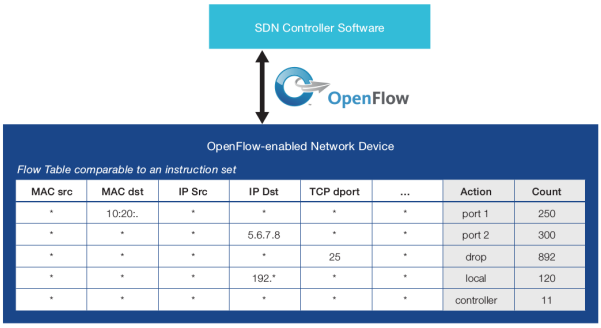


Fig. 1. The caption for a figure must follow the figure

. From then on, any packet from that flow reaching the switch will not require routing, and the route will be retrieved from the flow table. In OpenFlow, switches match and forward packets based on instructions found in the flow table. For example, as illustrated in Fig. 2, packets with a destination IP address of 5.6.7.8 are sent to port number 2. The last column in the flow table indicates the number of packets from this flow that have been sent along this route.



MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:..	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Fig. 2: An example of a flow table [22]

Numerous controllers provide the OpenFlow protocol, with the most popular ones summarized in Table 1. In this project, the Ryu controller was used to implement the prediction algorithm. Since this controller is written in Python, and Python has incredible capabilities for implementing artificial intelligence and machine learning algorithms due to its rich library of packages, we decided to

utilize this controller for this research.

Table 1. Comparison between the most popular OpenFlow controllers

Controller / *feature*	*Floodlight*	*Ryu*	*OpenDayLight*	*Pox*
Programming language	Java	Python	Java	Python
GUI	Web base/java	yes	Web base	Python+QT4
Documentation	good	medium	Very good	weak
modular	medium	medium	High	low
distributed	yes	no	Yes	no
environment	Linux, macOS, and Windows	Linux	Linux, macOS, and Windows	Linux, macOS and Windows
OpenFlow support	1.0, 1.3	1.0, 1.3, 1.4	1.0, 1.3, 1.4	1.0
Foundation	Big switch network	Nippo Telegraph Telephone And Corporation	Linux Foundation, CISCO, IBM, NEC, …	Nicira
Multinetwork support	yes	Yes	Yes	no
OpenStack support	no	yes	Yes	no

3.2 Long Short-Term Memory

The use of deep neural networks in today's problems continues to grow due to advancements in technology and the increased processing power of processors. LSTM is a popular deep neural network used in time series analysis. In this project, the performance of traffic prediction in an SDN-based network is examined.

LSTM is an improved version of recurrent neural networks (RNNs). RNNs were first developed by Williams and Zipser in the late 1980s, and they are mainly applied for modelling time series data. Various types of RNNs have been designed to overcome multiple challenges. However, RNNs have a weakness when it comes to gradients. Consequently, they struggle to capture non-stationary requirements over extended periods effectively. One solution to this problem, proposed by [24], involved adding a memory component to RNNs. Another improvement involved introducing skip connections, which allow information to be processed directly by the RNN without passing through the entire network. On the other hand, LSTM consists of elements, called cells, each of which has three gates: the input gate, output gate, and forget gate [25]. These gates allow changes to be made to the cell state vector, which is used to capture long-term dependencies. This controlled flow of information enables the network to remember multiple time dependencies. LSTM is used mainly for modelling long-term dependencies [26]. Fig. 3 depicts an LSTM cell.

One modification made to the LSTM cell, presented by [27], is known as the Gated Recurrent Unit (GRU). This improvement aims to predict long-term dependencies by optimally combining short-term information. The goal is to model adaptive dependencies at different time intervals.

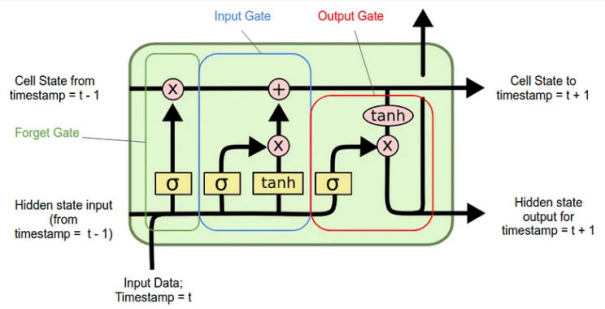


Fig. 3: an LSTM cell

Compared to LSTM, GRU has a simplified cell structure that operates based on a gating system, consisting of only a reset gate and an update gate. Its main difference from LSTM is that the cell state can be entirely revisited at each iteration and updated with short-term information through the reset gate. While LSTM provides a mechanism that limits the gradients of changes achievable at each iteration, GRU allows for information from the past to be more thoroughly ignored. LSTM, on the other hand, restricts the complete disregard of past information. Empirical research on cell architectures [28] has demonstrated that cells with gating mechanisms outperform classical RNN methods in terms of prediction results. A comprehensive study on different variations of neural networks has also demonstrated the superiority of LSTM over GRU [29]. Despite having fewer parameters, there is no significant computational advantage of GRU. Furthermore, the gating system in LSTM helps filter out irrelevant input information, resulting in higher accuracy in modeling time-variant behavior. Given the significant advantages of LSTM in time series prediction, it has been used in this article for traffic prediction.

3.3 Harmony Search Algorithm

Harmony Search is a metaheuristic algorithm used for optimization in various problem domains [30]. For example, in short-term load forecasting, a combination of Harmony search and fuzzy time series has been used [31]. Harmony Search was initially introduced by Geem in 2001 [32]. In this algorithm, each solution is referred to as a "harmony". Initially, a random population is generated. This initial population is ranked from best to worst in a structure called the Harmony Memory (HM) based on the fitness values. Two crucial parameters in this algorithm are the Harmony Memory Consideration Rate (HMCR) and Pitch Adjustment Rate (PAR), which significantly influence the

generation of new harmonies. Choosing HMCR and PAR may affect optimization performance. Some methods try to set these parameters automatically [33, 34]. The algorithm proceeds iteratively. In each iteration, a new harmony is created using the following algorithm:

First, a random number is generated, and if it is less than HMCR, a new harmony is created by randomly selecting components from the harmony memory. Otherwise, a completely random harmony is made without considering the memory. If the new harmony is chosen from the memory, a second random number is generated and compared to PAR. If it is less than PAR, the elements of the new harmony are adjusted within a small, defined range, just like pitching a musical instrument. After creating a new harmony, its fitness value is calculated and compared to the worst case in the harmony memory. If it is better, it replaces the worst case, and the harmony memory is reordered. These steps are repeated until termination conditions are met.

4. Proposed Method

Before delving into the details of the proposed method, it is necessary to examine the various sections of the project, as depicted in Fig. 4. This project comprises three main sections, which will be fully explained later. In the first section, a virtual network is created using the mininet package in Python, and the nodes begin to send traffic within the network. The second section is dedicated to the controller, whose program will be developed to monitor the traffic passing through each port and display it in a time series fashion. Finally, the third section, which is the most critical part of the project, receives the time series data and runs the prediction algorithm. We will provide a detailed description of these sections below.

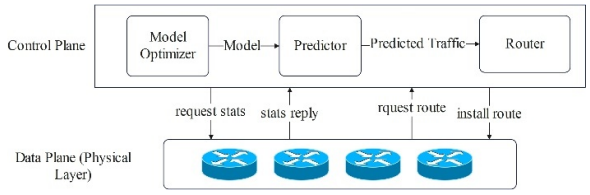


Fig. 4: Block diagram of the proposed method

4.1 Topology and Traffic Generation Scenarios

The mininet package, written in Python, provides capabilities for simulating SDN networks and is widely used by researchers in this field. In this section of the project, we utilized this package to design and simulate an SDN topology. A network with 11 switches, known as Abilene, is used in the experiments. Fig. 5 illustrates the topology and traffic-sending scenario, where each switch is connected to a host with the same index. For example, S1 is connected to H1, a host with index 1. A simple Python application runs threads on the hosts to simultaneously produce and send traffic through the network. The switches used are of the OpenVSwitch type and communicate with the controller using the OpenFlow 1.3 protocol.

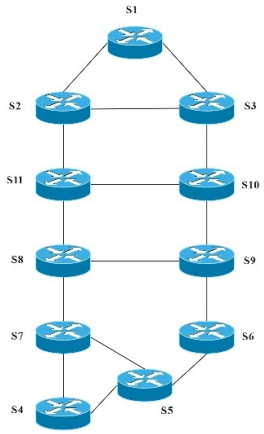


Fig. 5: Topology used in testing (Abilene)

For testing and further analysis, traffic is generated using two different methods. Two scenarios have been considered for this purpose. The first scenario is an idealized form where requests are made with a regular rhythm. In this scenario, hosts 1 and 2 both send traffic with sinusoidal functions, but with different frequencies. Eq. 1 represents the volume of traffic sent for this method. However, in the second scenario, the worst-case scenario is considered, and the volume of traffic is obtained from the logistic map function, which is a chaotic function. For this scenario, Eq. 2 has been used. An effort has been made to make the worst-case scenario as random as possible, which is why a chaotic function is used. Chaotic functions have a pseudo-random nature. Eq. 2 illustrates the relationship for the volume of traffic sent in this method. In the real world, traffic demands can be considered somewhere between these two scenarios.

$$\begin{cases} r = \dfrac{1+\sin\left(2\pi \dfrac{t}{Period}\right)}{2} \\ volume(t) = \text{int}\left[base\ volume \times r\right] \end{cases} \tag{1}$$

$$\begin{cases} x_0 = random \\ x_i = r \times x_{i-1} \times \left(1 - x_{i-1}\right) \\ volume(t) = \text{int}\left[base\ volume \times x_i\right] \end{cases} \tag{2}$$

where the $base\ volume$ is a constant representing the maximum traffic that can be sent from the links within the time interval t. For the sake of simulation concurrency, as mentioned earlier, the transmissions are carried out by two threads, each executing a ping command on the respective hosts at equal intervals and with a volume obtained from either Eq. 1 or Eq. 2.

4.2 Traffic Data Retrieval and Time Series Generation

For this stage, the Ryu controller is used. This controller is written in Python and is highly customizable. The Ryu controller is customized to transform the port traffic into a suitable format for the prediction program. The operation of this part is as follows: the controller sends "request stats" packets to all switches at equal intervals. After receiving responses from the switches, it calculates the volume of traffic passing through each port by taking the difference and converting it into bits per second based on the interval. It then stores this information in a table for use by the prediction program. In other words, the traffic from each port is transformed into a time series by the controller and made available to the prediction program. Fig. 6 illustrates the traffic signals obtained over a specified period. The figure shows two time series obtained from regular (Fig. 7-a) and chaotic (Fig. 7-b) traffic generation functions.

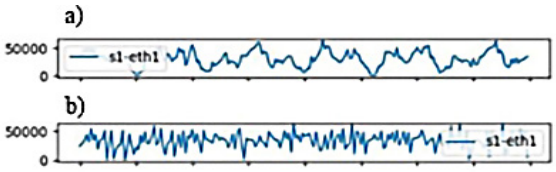


Fig. 6: Sample signals for regular (a) and chaotic (b) traffic of the port S1-eth1

4.3 Traffic Analysis and Prediction

In this section, one of the most crucial parts of the proposed method is explained. As mentioned earlier, the Ryu controller reads the status of each switch's ports at equal intervals, prepares them as a time series, and makes them available to the analyzer. Fig. 7 illustrates the architecture of the model optimizer. The traffic analyzer receives traffic data, performs initial preprocessing on it, and prepares the data for the next stage.

After the normalization, the analyzer adds new features to the time series using the Fast Fourier Transform (FFT). Before explaining the process of adding new features to the time series, let's first take a look at Fourier theory. According to Fourier's theory, any function can be represented as a sum of sinusoidal functions with different domains and frequencies, each with a coefficient of the primary frequency. Each of these sinusoidal functions is called a harmonic. To determine the coefficients of these functions, the Discrete Fourier Transform (DFT) or Fast Fourier Transform (FFT) can be used, which can be obtained from Eq. 3 [35].

$$\begin{cases} let\ x_0, x_1, \ldots, x_{N-1}\ \ be\ Complex\ Numbers \\ X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi kn}{N}} \qquad k = 0,1,\ldots,N-1 \end{cases} \tag{3}$$

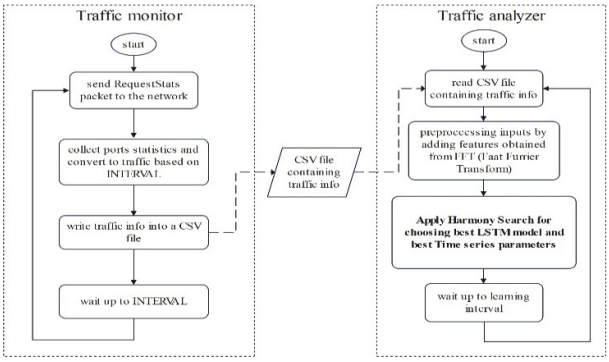


Fig. 7: Model optimizer part of the project

We propose a selection algorithm that chooses the most essential harmonics, obtained from FFT, so that their sum produces a shape similar to the input time series. The algorithm is as follows: After obtaining the Fourier coefficients, the harmonics are sorted based on the magnitude of their coefficients, considering that the larger the domain of a harmonic, the more critical it will be in shaping the final function. Then, a few of the most essential harmonics, which are the most crucial components of our tasks, are added as new features to the time series. These new features will improve the accuracy of our machine-learning model. Fig. 8 shows an example of three harmonics obtained from the traffic of port eth1 of switch s1, along with their sum. As seen in Fig. 8, the sum of these three harmonics closely resembles the original signal. This time series will be used for machine learning in the following steps.

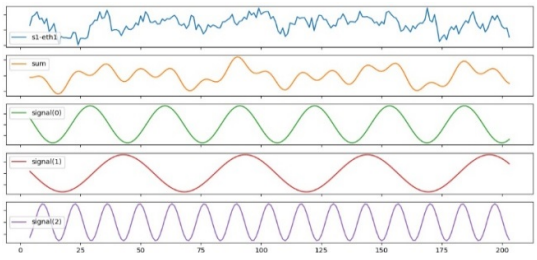


Fig. 8: First Three Harmonics, obtained by proposed FFT, and Their Sum for Traffic on Port s1-eth1

4.4 Proposed Harmony Search

Before creating a dataset, we first define a harmony for the Harmony Search algorithm. To avoid confusion with the Fourier harmonics, we write this harmony, which represents the solution in our Harmony Search algorithm, with a capital 'H'. In this project, a Harmony is composed of three numbers: the number of lags, the number of labels, and the number of shifts in the time series. Fig. 10 illustrates a Harmony in this project.

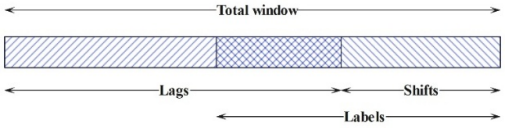


Fig. 9: Defining a Harmony in the Harmony Search Algorithm

As seen in Fig. 9, the size of the window depends on the lag and shift values. Consequently, independent values, namely lags, labels, and shifts, constitute the components of Harmony. Since changing the components of Harmony in each iteration of the search algorithm results in changes to the entire dataset, a new dataset must be created from the input time series for each Harmony. Table 2 presents the optimization operations for this project in pseudo-code.

Table 2. Pseudo-code of the Harmony Search for the Traffic Prediction Problem

Input: Harmony search parameters (PAR,HMCR, HM size), time series represents traffic for port s1-eth1
Output: the best parameters for lag, label, and shift, for time series

*Generate Harmony Memory and fill it with randomly generated Harmonies**
Sort Harmonies by fitness, obtained from Eq. 4.
Generate new Harmony by HS algorithm, explained in sections 3-4, and create a dataset from input time series with the parameters achieved by new Harmony.
Give the dataset to the learning machine, CNN or LSTM selected before, and train the machine
Calculate fitness and compare with the worst case in the HM
Is it better than the fitness of the worst Harmony of HM?
If yes: replace the new Harmony with it and then sort HM
If no: no change is required
If optimization criteria are satisfied or the iteration number exceeds the maximum iteration, stop the algorithm and go to step 8; else, go to step 3
Return the best harmony as the result
**: Harmony depicted by Fig. 9*

4.5 Fitness Function

In designing the fitness function, it is essential to consider that our time series has three components: the number of lags, the number of labels, and the shift. Standard error functions in neural networks consider errors obtained from all labels. However, some of the labels and inputs overlap concerning the number of shifts. Therefore, the error values of this portion are meaningless. For example, assuming the number of shifts is equal to zero, the system essentially takes past values from the input as labels and loses its predictive ability entirely. The larger the shift value, the less overlap there will be, indicating that the system predicts further into the future. Since the overlapping portion error may falsely indicate a lower error, we exclude this portion error from the fitness function calculation. Considering these observations, our fitness function will be as follows:

$$\begin{cases} n = \min\left(number\ of\ labels, number\ of\ shifts\right) \\ m = len(Err) - 1 \\ MSRE = \dfrac{\sqrt{\sum_{i=m-n}^{m} (Err_i)^2}}{n} \\ fitness = \dfrac{(1+sc)\times(1+s)\times MSRE}{n} \end{cases} \quad (4)$$

where Err is an array containing the errors related to the labels. Ans s is the variance of the error of the portion without overlap. The Mean Square Root for Errors (MSRE) is the same as s considers errors excluding the overlap. Additionally, sc (sign changed) is the number of sign changes in consecutive errors, which is obtained from the pseudo-code in Table 3.

Table 3: Algorithm to calculate the number of error signs changed

Input:
1 - Err, an array containing Errors between labels and the outputs
2 – number of shifts
3 – number of labels
Output*:*
sc: number of changing signs of consecutive errors

```
sc := 0
n:= min (number of shifts, number of labels)
k = len (Err)
For i: = k – n To K – 2 do:
    If Err [ i] * Err [ i+1] < 0 Then:
        sc := sc + 1
Return (sc)
```

4.6 Prediction-Based Routing Logic

In the proposed system, routing decisions are no longer made reactively based on instantaneous link states; instead, they are driven by predicted traffic trends obtained from the LSTM-based forecasting module. This predictive routing strategy enables the SDN controller to proactively reconfigure the network before congestion occurs, thus improving performance metrics such as delay, jitter, and packet loss.

As illustrated in Figure X (to be added), the Ryu controller collects traffic statistics at regular intervals from

OpenFlow switches. These statistics are converted into time series and passed to the prediction module, where an LSTM model trained on historical traffic data forecasts future traffic volumes for each port. The predicted values are then utilized in the routing module to calculate optimal paths between hosts.

- *Cost Function for Path Selection*

To select the most efficient path, the controller computes a composite cost function for each available path. This function incorporates multiple factors, including:

T_p : Predicted traffic load on each link

D : Estimated delay based on predicted congestion

C : Link capacity or residual bandwidth

J : Historical jitter (optional, for real-time applications)

The cost of a path P consisting of n links is defined by Eq. 5 as follows:

$$Cost(P) = \sum_{i=1}^{n} \left(\alpha . \frac{T_p(i)}{C(i)} + \beta . D(i) + \gamma . J(i) \right) \quad (5)$$

Where α, β, γ are weighting coefficients (set based on the QoS priority). If jitter is not considered, then $\gamma = 0$.

The path with the minimum total cost is selected as the preferred route.

- *Flow Rule Installation*

Once the optimal path is determined, the Ryu controller updates flow tables across all switches along the route. The controller ensures that low-cost paths are prioritized and frequently re-evaluates them as new predictions arrive. This dynamic adjustment helps maintain optimal network performance under both regular and bursty traffic patterns.

5. Results and Analysis

5.1 Simulation Setups

This section outlines the experimental setup and the environment in which the evaluations were performed. Since both the Ryu controller and Mininet operate on Linux platforms, Ubuntu 22.04 LTS was chosen as the operating system for deployment. All experiments were conducted on an HP ProBook 455 G7 laptop equipped with an AMD Ryzen 7 4700U processor, integrated Radeon Graphics, 16GB of RAM, and a 1TB solid-state drive, ensuring smooth and efficient operation. The subsequent sections detail the experimental results.

5.2 Traffic Prediction Evaluation

As mentioned earlier, this study's system input is a time series prepared in two different forms. The first is created from network traffic data when requests are sent at regular intervals, and the second is created from network traffic data when requests are sent at random rates. After conducting the Harmony Search according to the proposed method for 500 epochs, Figs. 10 and 11 show the changes in loss and fitness, respectively, for the best Harmony in each epoch when network traffic is regular. As evident in Figs. 10 and 11, in each iteration of the algorithm, fitness consistently decreases, but the loss value increases at times due to the overlap of labels and inputs, leading to false responses.

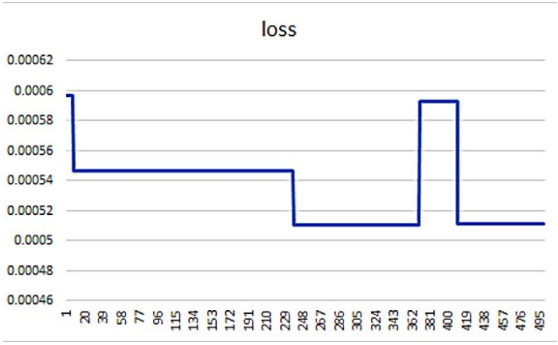


Fig. 10: The best harmony's loss for regular traffic

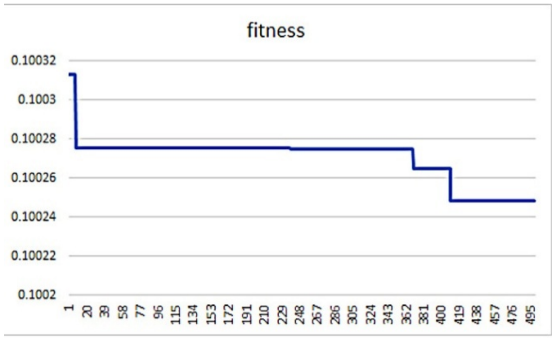


Fig. 11: The best harmony's fitness for regular traffic

The experiments for chaotic traffic were repeated, and the results are shown in Figs. 12 and 13. Fig. 12 shows the loss values, and Fig. 13 shows the fitness values of the best Harmony in each iteration for the chaotic traffic.

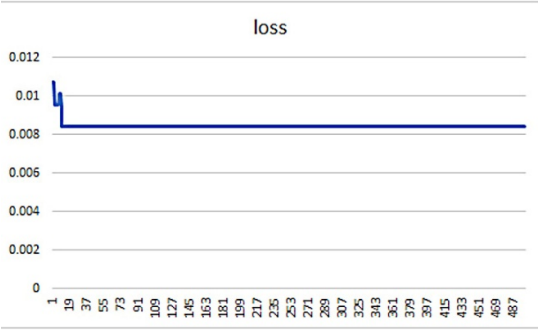


Fig. 12: The best harmony's loss for chaotic traffic

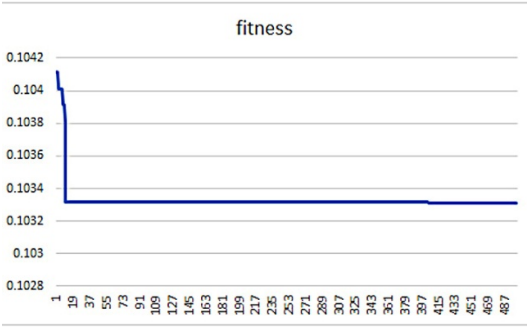


Fig. 13: The best harmony's fitness for chaotic traffic

5.3 The Effect of Nature and Input Form on System Performance

As seen in Figs. 11 and 13, the final fitness values, which measure prediction performance, indicate that when the nature of traffic is somewhat regular, the method exhibits acceptable performance. Although internet demands in the real world are not entirely regular, due to the influence of geographical and cultural factors, they show a relative order to some extent. For example, during certain hours of the day, requests increase, or they decrease at night. It can be said that network traffic in the real world falls within a spectrum between the two states. As shown in Figs. 13 and 14, the proposed method can accurately predict both regular (Fig. 13) and chaotic (Fig. 14) traffic.

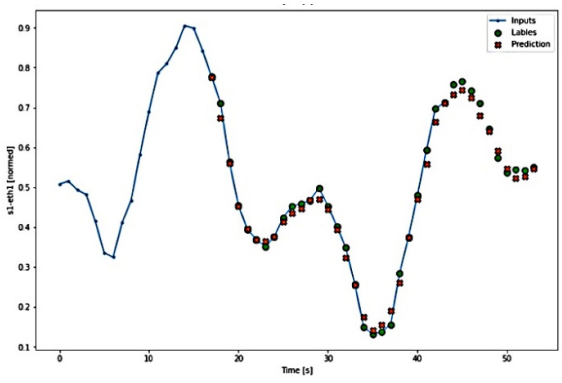


Fig. 14: A prediction example of regular traffic

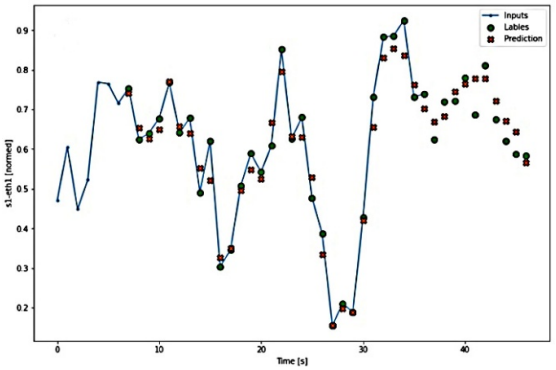


Fig. 15: A prediction example of chaotic traffic

5.4 The Effect of Harmony Memory Size on Performance

In another experiment, we investigated the impact of varying the size of the harmony memory. In our experiments, four different values for memory size were considered. Initially, we conducted experiments with a memory size of 5. Then, we repeated the experiments with memory sizes of 20, 50, and 100. In these experiments, all other harmony search parameters were kept constant. The values of the HMCR and PAR parameters were both set to 0.3. The results obtained from this experiment are shown in Figs. 16 and 17, respectively, for regular and chaotic traffic. As seen in the figures, increasing the memory size not only does not lead to better results but also results in poorer performance. To explain these results, it can be said that the reason for the decrease in performance with larger memories is the small number of iterations. Because a new dataset is created with new parameters in each iteration, and the neural network is trained based on it, we cannot significantly increase the number of iterations without compromising the model's accuracy. In our problem, the number of iterations has been set to 500.

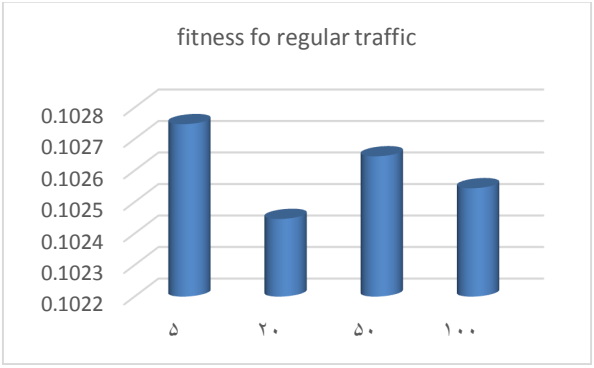


Fig. 16: fitness versus Harmony memory size for regular traffic

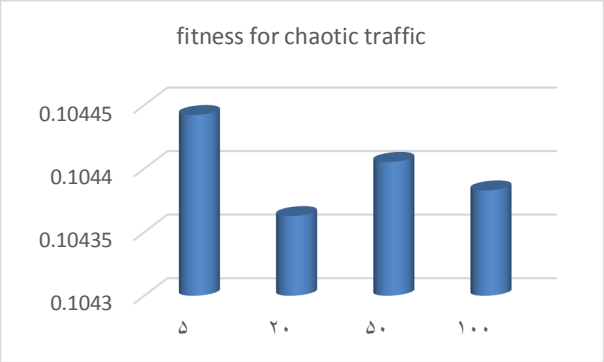


Fig. 17: fitness versus Harmony memory size for chaotic traffic

5.4 The Effect of HMCR Parameter

In the next experiment, we kept the size of the harmony memory constant and changed other parameters. Fig. 18 shows that in a harmony memory of 10, if the HMCR changes, increasing HMCR will result in a lower fitness value. Moreover, as the HMCR value approaches 1, the fitness value rises significantly. To explain this issue, it can be said that when HMCR is close to zero, the algorithm creates the next Harmony completely randomly.

Consequently, the harmony memory is practically unused, and the harmony search algorithm loses its effectiveness. Now, if we increase HMCR, we will see that fitness decreases. By continuing to increase HMCR to near 1, in this case, the next Harmonies will be selected from memory, and the contribution of random harmonies will be almost zero. This causes the algorithm to get stuck in local minima, resulting in higher fitness. In investigating the effect of the HMCR value, we found that if the memory size is chosen appropriately, increasing the HMCR value will reduce the fitness value. Such a reduction will continue to some extent and then increase. The reason for the decrease in fitness is that as the number of iterations increases, the memory becomes filled with the best solutions, and the likelihood of these solutions participating in generating subsequent solutions gradually improves, resulting in lower fitness. However, when the size of HMCR increases excessively, since very few random responses are generated and almost all responses are created from memory, fitness rises steadily.

This issue is illustrated in Fig. 18. As shown in the figure, fitness decreases for HMCR up to 0.3, and then it increases. The irregularity of the graph in Fig. 18 is due to the uncertainty and randomness of the initial population that makes up the content of the harmony memory. To analyze it, consider a situation where HMCR is close to 1, and a local minimum is randomly entered into the initial population. In this case, after many iterations, the responses converge to the local minimum, and due to the high value of HMCR, the chance of new random values entering the memory and consequently escaping from the local minimum decreases. In the second scenario, when HMCR is very low and close to zero because the memory does not contribute to generating new Harmonies and their random generation in each iteration, the optimization algorithm will completely lose its efficiency.

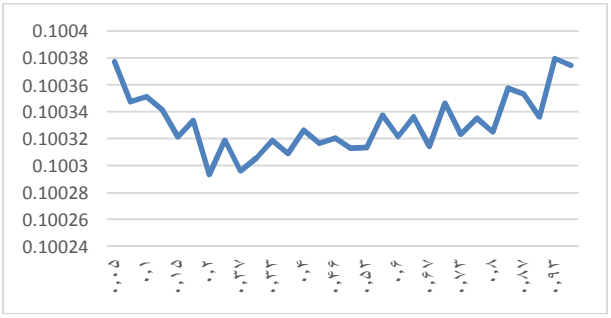


Fig. 18: Changes in fitness versus HMCR value.

5.5 Obtained results for QoS

In this study, three key performance metrics are used to evaluate the efficiency of the proposed routing method: delay, jitter, and throughput.

- **Delay** (D) measures the average time taken for a packet to travel from the source to the destination. It can be defined by Eq. 6 as follows:

$$D = \frac{1}{N}\sum_{i=1}^{N}\left(t_i^{receive} - t_i^{send}\right) \qquad (6)$$

where t_i^{send} and $t_i^{receive}$ denote the sending and receiving times of the i^{th} packet, and N is the total number of packets.

- **Jitter** (J) quantifies the variation in packet delay, representing the consistency of packet arrival times. It can be expressed as Eq. 7 as follows:

$$J = \frac{1}{N-1}\sum_{i=1}^{N-1}\left|(D_{i+1} - D_i)\right| \qquad (7)$$

where D_i is the delay of the i^{th} packet.

- **Throughput** (T) is the average rate of successful data delivery over the network, which can be calculated by Eq. 8 as follows:

$$T = \frac{N_{success} \times Packet\ size}{Total\ time} \qquad (8)$$

where $N_{success}$ is the number of packets successfully received.

Table 4 presents comparative results of these QoS

metrics between the proposed method and a basic routing algorithm, as well as those presented in Gunavathie et al.'s work.

Table 4: Comparisons between routing methods.

	Throughput (MB/s)	Jitter (µs)	Delay (µs)
Default shortest path	2.72	6991.8	4428.9
GRU-based routing[36]	2.66	8050.9	4875.6
Proposed method	2.84	5791.8	4318.2

6. Conclusion

In this study, we proposed an intelligent traffic-aware routing framework for Software-Defined Networking (SDN) that integrates deep learning and metaheuristic optimization. By leveraging LSTM networks for traffic prediction, enriched with FFT-based feature extraction, and fine-tuned using the Harmony Search algorithm, our method enables proactive routing decisions based on forecasted traffic patterns rather than reactive responses to current states.

Experimental results, conducted under both regular and chaotic traffic scenarios, demonstrated that the proposed approach effectively reduces delay and jitter while improving throughput, compared to baseline methods. Furthermore, the detailed analysis revealed that parameters such as harmony memory size and HMCR have a significant influence on prediction performance.

Overall, the combination of predictive intelligence and optimization techniques proved to be effective in enhancing network performance and stability in SDN environments. Future work could focus on extending this framework to multipath routing, exploring additional deep learning architectures, and evaluating its scalability in larger and more complex network topologies.

References

[1] Z. Shu *et al.*, "Traffic engineering in software-defined networking: Measurement and management, " *IEEE Access,* vol. 4, pp. 3246-3256, 2016, doi: 10.1109/ ACCESS. 2016.2582748.

[2] E. Stapf, "Predicting Traffic Flows for Traffic Engineering in Software-Defined Networks," M.Sc., TECHNISCHE UNIVERSITAT DARMSTADT, 2016.

[3] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *2013 Proceedings IEEE INFOCOM*, 14-19 April 2013 2013, pp. 2211-2219, doi: 10.1109/INFCOM. 2013.6567024.

[4] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: dynamic flow scheduling for data center networks, " in *Nsdi*, 2010, vol. 10, no. 8: San Jose, USA, pp. 89-92.

[5] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *2011 Proceedings IEEE INFOCOM*, 10-15 April 2011 2011, pp. 1629-1637, doi: 10.1109/INFCOM.2011. 5934956.

[6] F. Francois and E. Gelenbe, "Towards a cognitive routing engine for software defined networks," in *2016 IEEE International Conference on Communications (ICC)*, 22-27 May 2016 2016, pp. 1-6, doi: 10.1109/ICC.2016.7511138.

[7] H. T. Zaw and A. Maw, "Traffic management with elephant flow detection in software defined networks (SDN)," *International Journal of Electrical and Computer Engineering,* Research Article vol. 9, no. 4, p. 9, 2019-08-01 2019, doi: 10.11591/ijece.v9i4.pp 3203-3211.

[8] G. Wassie Geremew and J. Ding, "Elephant Flows Detection Using Deep Neural Network, Convolutional Neural Network, Long Short-Term Memory, and Autoencoder," *Journal of Computer Networks and Communications,* vol. 2023, no. 1, p. 1495642, 2023, doi: https://doi.org/10.1155/2023/ 1495642.

[9] P. K. Hoong, I. K. Tan, and C. Y. Keong, "Bittorrent network traffic forecasting with ARMA," *arXiv preprint arXiv:1208.1896,* 2012.

[10] N. Hoong, P. Hoong, I. Tan, N. Muthuvelu, and L. Seng, "Impact of utilizing forecasted network traffic for data transfers," in *13th International Conference on Advanced Communication Technology (ICACT2011)*, 2011: IEEE, pp. 1199-1204.

[11] Y. Yu, M. Song, Y. Fu, and J. Song, "Traffic prediction in 3G mobile networks based on multifractal exploration," *Tsinghua Science and Technology,* vol. 18, no. 4, pp. 398-405, 2013, doi: 10.1109/TST.2013.6574678.

[12] B. Vujicic, C. Hao, and L. Trajkovic, "Prediction of traffic in a public safety network," in *2006 IEEE International Symposium on Circuits and Systems (ISCAS)*, 21-24 May 2006 2006, p. 4 pp., doi: 10.1109/ISCAS.2006.1693165.

[13] N. C. Anand, C. Scoglio, and B. Natarajan, "GARCH — non-linear time series model for traffic modeling and prediction," in *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*, 7-11 April 2008 2008, pp. 694-697, doi: 10.1109/NOMS. 2008.4575191.

[14] P. Cortez, M. Rio, M. Rocha, and P. Sousa, "Internet Traffic Forecasting using Neural Networks," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, 16-21 July 2006 2006, pp. 2635-2642, doi: 10.1109/IJCNN.2006.247142.

[15] S. Chabaa, A. Zeroual, and J. Antari, "Identification and prediction of internet traffic using artificial neural networks," *Journal of Intelligent Learning Systems and Applications,* vol. 2, no. 3, pp. 147-155, 2010.

[16] D. C. Park and D. M. Woo, "Prediction of Network Traffic Using Dynamic Bilinear Recurrent Neural Network," in *2009 Fifth International Conference on Natural Computation*, 14-16 Aug. 2009 2009, vol. 2, pp. 419-423, doi: 10.1109/ICNC.2009.662.

[17] S. Chabaa, A. Zeroual, and J. Antari, "ANFIS method for forecasting internet traffic time series," in *2009 Mediterrannean Microwave Symposium (MMS)*, 15-17 Nov. 2009 2009, pp. 1-4, doi: 10.1109/MMS.2009. 5409834.

[18] Y. Liang and L. Qiu, "Network traffic prediction based on SVR improved by chaos theory and ant colony optimization," *International journal of future generation communication and networking,* vol. 8, no. 1, pp. 69-78, 2015.

[19] P. Rai and H. K. Deva Sarma, "A Survey on Application of LSTM as a Deep Learning Approach in Traffic Classification for SDN," in *Machine Learning in Information and Communication Technology*, Singapore, H. K. Deva Sarma, V. Piuri, and A. K. Pujari, Eds., 2023// 2023: Springer Nature Singapore, pp. 161-173.

[20] H. Yao, C. Liu, P. Zhang, S. Wu, C. Jiang, and S. Yu, "Identification of Encrypted Traffic Through Attention Mechanism Based Long Short Term Memory," *IEEE Transactions on Big Data,* vol. 8, no. 1, pp. 241-252, 2022, doi: 10.1109/TBDATA.2019. 2940675.

[21] K. M. R. Mokoena, R. L. Moila, and P. M. Velempini, "Improving Network Management with Software Defined Networking using OpenFlow Protocol," in *2023 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, 3-4 Aug. 2023 2023, pp. 1-5, doi: 10.1109/icABCD59051.2023. 10220519.

[22] O. N. Fundation, "Software-defined networking: The new norm for networks," *ONF White Paper,* vol. 2, no. 2-6, p. 11, 2012.

[23] N. McKeown *et al.*, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.,* vol. 38, no. 2, pp. 69–74, 2008, doi: 10.1145/ 1355734.1355746.

[24] Z. Motazedian and A. A. Safavi, "Nonlinear and Time Varying System Identification Using a Novel Adaptive Fully Connected Recurrent Wavelet Network," in *2019 27th Iranian Conference on Electrical Engineering (ICEE)*, 30 April-2 May 2019 2019, pp. 1181-1187, doi: 10.1109/IranianCEE. 2019.8786669.

[25] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich, "A survey on long short-term memory networks for time series prediction," *Procedia Cirp,* vol. 99, pp. 650-655, 2021.

[26] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[27] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078,* 2014.

[28] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555,* 2014.

[29] D. Britz, A. Goldie, M.-T. Luong, and Q. Le, "Massive exploration of neural machine translation architectures," *arXiv preprint arXiv:1703.03906,* 2017.

[30] D. Manjarres *et al.*, "A survey on applications of the harmony search algorithm," *Engineering Applications of Artificial Intelligence,* vol. 26, no. 8, pp. 1818-1831, 2013.

[31] H. J. Sadaei, R. Enayatifar, A. H. Abdullah, and A. Gani, "Short-term load forecasting using a hybrid model with a refined exponentially weighted fuzzy time series and an improved harmony search," *International Journal of Electrical Power & Energy Systems,* vol. 62, pp. 118-129, 2014.

[32] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: harmony search," *simulation,* vol. 76, no. 2, pp. 60-68, 2001.

[33] Z. W. Geem and K.-B. Sim, "Parameter-setting-free harmony search algorithm," *Applied Mathematics and Computation,* vol. 217, no. 8, pp. 3881-3889, 2010.

[34] K. Mirzaei Talarposhti and M. Khaki Jamei, "A secure image encryption method based on dynamic harmony search (DHS) combined with chaotic map," *Optics and Lasers in Engineering,* vol. 81, pp. 21-34, 2016/06/01/ 2016, doi: https://doi.org/10.1016/j. optlaseng.2016.01.006.

[35] P. Duhamel and M. Vetterli, "Fast fourier transforms: A tutorial review and a state of the art," *Signal Processing,* vol. 19, no. 4, pp. 259-299, 1990/04/01/ 1990, doi: https://doi.org/10.1016/0165-1684(90)90158-U.

[36] M. Gunavathie and S. Umamaheswari, "Traffic-aware optimal routing in software defined networks by predicting traffic using neural network," *Expert Systems with Applications,* vol. 239, p. 122415, 2024.