An AI-based Traffic Matrix Prediction Solution for Software-Defined Network

Duc-Huy LE*, Hai-Anh TRAN*, Sami SOUIHI †, and Abdelhamid MELLOUK†
*School of Information and Communication Technology
Hanoi University of Science and Technology, Hanoi, Vietnam
Email: huy.ld141938@sis.hust.edu.vn, anhth@soict.hust.edu.vn
†LISSI-TincNET Research Team University Paris-Est Creteil, France
Email: sami.souihi@u-pec.fr, mellouk@u-pec.fr

ICC 2021 - IEEE International Conference on Communications | 978-1-7281-7122-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICC42927.2021.9500331

***Abstract*—Traffic Matrix (TM) clearly describes the volume and the distribution of traffic flows inside a network. TM plays an important role in many network management fields, such as traffic accounting, short-time traffic scheduling or re-routing, network design, anomaly detection, etc. Hence, an accurate TM prediction strategy is essential to handle those tasks effectively. Fortunately, Artificial Intelligence (AI) has been developing very strongly, thanks to computer technology developments such as GPU and TPU. That offers an opportunity to apply AI to TM prediction methods. However, applying Machine Learning techniques in traditional networks encounters some issues due to the distributed control and restricted local view of network nodes. For this purpose, a centralized control architecture, e.g., Software-defined Network (SDN), is a promising candidate. In this paper, we apply Long Short-Term Memory (LSTM) and its two variants, Bidirectional LSTM (BiLSTM) and Gate Recurrent Unit (GRU), for TM prediction mechanisms of an SDN architecture network. The prediction models have been evaluated using two datasets: the popular GÉANT backbone network traffic data and our dataset generated through a testbed. The experimental results show that our approach yielded promising traffic prediction accuracy.**

***Index Terms*—SDN, LSTM, Traffic Matrix Prediction**

I. Introduction

Traffic Matrix (TM) reflects the volume of traffic flowing between all possible pairs of original-destination (OD) nodes in a computer network at a specific time. A detailed network traffic matrix study is reported in [1]. In a modern network, the increasing demand of various network services over the Internet and the advent of new technologies and applications have led to an explosive growth of Internet traffic. This demand comes with the increasing importance of network management, provisioning, and traffic engineering. In this circumstance, TM prediction is considered as a proactive approach for many types of management applications. One example is energy saving: accurately predicted network traffic could enable efficient resource allocation in network devices; therefore, it could save a significant amount of power, which is extremely crucial in wireless networks [2]. Another application of TM prediction is dynamic routing. Based on the TM prediction results, the routing paths can be reasonably changed to avoid elephant traffic flowing through the same link. Therefore, it helps the network avoid congestion and decrease communication latency. In addition, TM prediction also offers a good solution in many other network management applications, such as Quality of Service (QoS), traffic accounting, long-term capacity planning, network anomaly detection, and so on.

The massive extension of network infrastructure nowadays requires efficient TM prediction techniques to deal with the enormous number of TM elements, which grows up to the square of the number of nodes. Machine Learning (ML) has been proved as the most relevant approach in the evolution of smart technology. However, providing accurate historical data of TMs for learning and predicting process is a struggle in traditional network architecture. That is explained by the fact that, in a traditional network, each network device only has restricted local view and control capacity. Consequently, TM could not be obtained directly by collecting every OD pair traffic volume, but it could be estimated via routing diagram and links' throughput gathered (e.g., using SNMP protocol). Although several methods have been introduced [3], the noticeable estimation error still affects TM prediction result. Also, the distributed-control restriction raises difficult to gather traffic information from network components and diminishes the ability to perform further network management. This limitation motivates us to come with a centralized solution: Software-Defined Network (SDN) [4]. The SDN architecture's main idea is to separate the control plane from the data plane of all network nodes, make it centralized controlled and directly programmable. Hence, the network observation and management can be performed easily at an SDN controller with a global view of the system. Various scenarios and backbone systems have proven the prominence of SDN in both academia and industry. In addition, taking advantage of OpenFlow (OF) protocol [5], which is commonly used in SDN network, flow statistics can be easily collected from every node and its traffic volume to the other ones can also be extracted, then accurately forming TM regardless of the underlying network structure.

To address the problem of TM prediction, several ML methods are proposed, generally classified into two main categories: linear prediction and nonlinear prediction. The most commonly used linear technique is ARMA/ARIMA model and HoltWinter (HW) algorithm [6], while Neural Networks is the most common solution for the nonlinear technique. In

[7], the network traffic evaluation on both methods claimed that nonlinear models as Neural Networks performed better in comparison to linear models. The authors also considered the Feed Forward Neural Network (FFNN) as the best forecasting technique for the problem. Recurrent Neural Network (RNN) [8], derived from FFNN, can use the internal memory to process variable-length sequences of units. That makes RNN suited for time-series predicting tasks, such as TM prediction (to predict a TM based on a sequence of previous TMs). However, RNN encounters gradient and exploding problems. In this paper, we applied three enhanced models of RNN: LSTM [9], Bidirectional-LSTM (BiLSTM) [10], and GRU [11].

Our contributions in this paper are summarized as follows:

- Developing an SDN framework to obtain the network TM using OpenFlow protocol.
- Constructing an SDN network testbed to generate traffic flows and capture TM data.
- Evaluating the performance of three proposed models on two different datasets.

We published all our source codes in [12].

The remainder of this paper is structured as follows. Section II discusses related works. In section III, we present our proposed TM obtaining method in SDN network and LSTM-RNN models. Section IV describes our experiments and analyzes obtained results.

II. Related Works

In [13], the authors attempt to use both statistical methods such as ARIMA and HW algorithm and Neural Network (NN) to model network traffic data. Based on their experimental results, NN attains a lower error rate comparing to the linear mechanisms.

In [14], Nie *el at.* apply a Deep Belief Network (DBN) model on both TM estimation and prediction. However, they still encounter a problem from a traditional network where they estimated TM based on links utilization. Despite how well the estimation method performs, the estimation process's incorrect information can lead to incorrectness in TM prediction.

In [15], the authors present OpenTM, a TM estimation system for SDN OpenFlow networks. Generally, they query the flow's statistical information from a switch along the routing path of an OD to directly measure the TM. Due to the chance of having some congestion point in the path, the TMs measured at different switches might be different. They analyze the result and performance of several switch selection schemes: first switch, least-loaded switch, a random switch, round-robin query, and last switch. According to their obtained results, the least-loaded scheme least affects the operation of switches. However, they claim that one might want to use methods that favor the last few switches along the path for higher accuracy requirement. In our work, we implement the TM observation system by collecting flow information of the destination switch for highly accurate measurement and don not have to keep track of every OD routing path. That has proven the scalability in large-scale networks.

In [16], Gunnar *el at.* use Simple Network Management Protocol (SNMP) to gather Links State information and apply several techniques to calculate full TM with the provided routing matrix. The results show that there still exists a significant variance between the estimated TM and the real data. To overcome this problem, our centralized architecture in using SDN is able to provide the exact TM and reduce the prediction error due to TM estimation inaccuracy.

The authors in [17] propose a LSTM-based TM prediction framework. They evaluate some linear prediction methods' performance with an LSTM approach and show that LSTM could outperform the traditional linear models in predicting accuracy. However, their work does not address the TM observation process in the network. Also, the lack of a centralized control solution reduces the effectiveness of their proposal.

This paper presents a method to address all the problems as mentioned earlier and apply different RNN variants.

III. Proposed Traffic Matrix Observation and Prediction Solution

The proposed system consists of two main components: the Traffic Matrix Observation module in SDN and the Traffic Matrix Prediction module using three RNN variants.

A. Traffic Matrix Observation module in a SDN

SDN architecture comes with the idea of separating the data plane and control plane. It consists of three layers : Infrastructure Layer, Control Layer, and Application layer. The first layer is composed of network devices, working as packet forwarding switches. The Application layer contains programs that can monitor and control network devices' behavior for incoming network flows via Control Layer. With this approach, many network management components, such as firewall, intrusion detection and load balancing, can be converted from traditional middle-boxes into applications in a centralized controller. In addition, a developer can arbitrarily monitor and manipulate SDN switches operation. That is supported by OpenFlow (OF) Protocol [18], which provides a secured communication framework between Control Layer and SDN switches. In this part, our idea is to deploy an SDN application, taking advantage of OF Protocol features, to capture the network's TM periodically. In this paper, we do not focus on the network's routing strategy in assuming that all flow paths are preinstalled and work well.

Similar to a normal switch, an SDN OpenFlow switch contains multiple flow rules, usually called flows. Generally, each flow consists of three parts: *Matching Fields*, *Action*, and *Counters*. A switch uses its flow table to compare incoming packets with the *Matching Fields* and executes the corresponding behavior in *Action* field. The latter comprises actions such as forwarding to a specific physical port or dropping/blocking the packet. The *Matching Fields* have various parameters, and the widely-used ones are Source IP Address and Destination IP Address with IP packets. The flows can be modified, added,

or removed by the centralized controller through OF messages. However, unlike a normal switch, an OF switch monitors some basic activity statistics of the flows, including:

- **Duration**: the elapsed time of the flow since being installed in the switch.
- **Packet count**: the number of packets that correctly match with the flow.
- **Byte count**: the volume (in bytes) of traffic match with the flow.

OF protocol also allows the controller to gather that statistical information, named OFP_FLOW_STATS_REQUEST message. An application can proactively send the request to any switch of the network via SDN controller. Especially, each switch's reply contains all of its flows' information, including matching fields and the counter's value. Basing on the Source and Destination IP Address from the matching fields and the overall traffic amount captured by counters, we can aggregate the traffic status of a switch to all other nodes in the network.

By synthesizing all switches' status, an Overall Traffic Volume Matrix (OTVM) of the network is obtained. It stores the overall amount of traffic passed through every OD pairs at a certain point. This is the key part of our work, helping us accurately calculate the TM. The SDN TM Observation application is described in Fig. 1. This application is an infinitive loop with two phases, determined by the flag value (0 - Phase 1; 1 - Phase 2). In the first phase, the loop starts with an initiative flow stats request to all network switches. After receiving all the statistics, an OTVM is aggregated and saved temporarily for the next step. This phase ends with the change of the flag to 1 and sleeps for a short period. In the second phase, the program sends the second flow stats request's wave. Similar to the first phase, another OTVM is captured. The network's TM in this loop is the difference between two OTVM, divided by the elapsed time between two phases (Eq. 1).

$$TM = \frac{\Delta \text{OVTM}}{\Delta \text{t}} \tag{1}$$

The loop ends by changing the flag value to 0 and sleep for a short period, depending on the frequency of capturing the TM. The TM Observation module does not affect any network's operation in generating a small number of overhead. In addition, it does not consume much controller's resources.

B. Traffic Matrix Prediction module

1) Problem Modeling: Suppose that there are N nodes in the network. Each node is a router. The traffic matrix is denoted by a N-by-N matrix X such that an entry x_{ij} represents traffic volume from node i to node j.

To model the problem, the time-series dimension is added into the traffic matrix, creating a N-by-N-by-T time-series TM Y (a vector of TMs) that an entry y_{ij}^t represents the traffic volume from node i to node j at timestep t.

The problem is to solve the TM X^t (the TM at time t) via a time-series TM Y formed by T previous TMs (X^{t-T}, X^{t-T+1}, …, X^{t-3}, X^{t-2}, X^{t-1}).

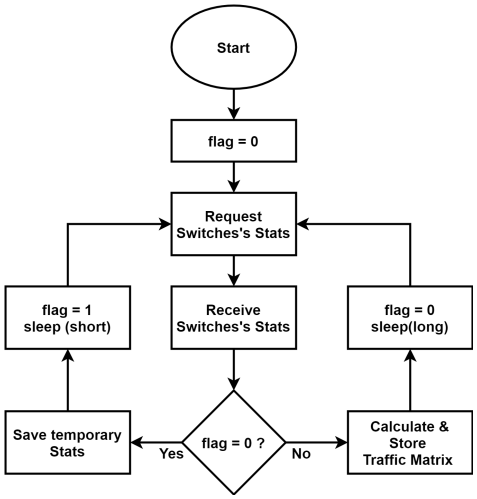

Fig. 1. TM Observation Working Flow

2) Machine Learning models: RNN, considered a class of artificial neural networks, can use its internal state to process sequences of input. That makes them appropriate to time-series problem, as TM prediction. However, training RNNs with the gradient-based back-propagation through time (BPTT) is difficult due to the vanishing and exploding gradient problem [19]. To cope with this issue, many RNN variants have been proposed. In this paper, three following LSTM variants are used:

- **LSTM**: It is the most popular variant of RNN and is widely used in sequential data problems. LSTM cell architecture consists of four main components: cell state, forget gate, input gate, and output gate. Each cell corresponds to a time step, uses the gates to decide which data is important to keep or throw away. By doing that, it only passes relevant information down a long chain of sequences to make a prediction. That makes LSTM be capable of learning long-term dependencies. LSTM uses cell state to control the exposure of memory content to other cells.
- **Bidirectional LSTM (BiLSTM)**: It is an extension of traditional LSTMs (based on the idea of Bidirectional RNNs [20]), which is able to improve the model performance. BiLSTM is the combination of two LSTMs on an input in two opposite directions, which means BiLSTM considers sequences in both forward and backward order. This can provide additional information on the network for a better prediction.
- **GRU**: It is introduced with a similar idea to LSTM to preserve important information in a sequence. Differently, GRU has a simpler architecture. It gets rid of the cell state and exposes the entire memory to other units with only two gates: reset gate and update gate. The reset gate combines the new input with the previous memory cell, and the update gate defines which information to keep.

The illustrations of LSTM and GRU cell architecture are described in Fig. 2.

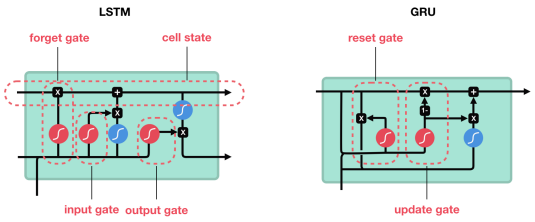

Fig. 2. LSTM and GRU cell architecture

Each of the three models has one hidden layer with 100 units. They are all trained through 200 epochs and the best parameters' set is kept for evaluation. It is important to note that a model's gates work with data based on a set of weights and biases, called parameters. The latter is updated in the training process via the back-propagation algorithm. The final parameters' set is considered as the trained model and it is used for prediction. The more parameters a model has, the more calculation time and resources it takes. Hence, the number of parameters represents the complexity of a model and can be used to evaluate the model's efficiency. The number of parameters of each model are presented in Tab.I.

TABLE I
MODEL'S NUMBER OF PARAMETERS

Model	Number of Params
Vanilla LSTM	44501
BiLSTM	89001
GRU	33701

3) Feeding data: To effectively train the models, we transform the traffic matrix X^t into a vector V_t with the size of $N*N$, called Traffic Vector (TV), by concatenating its N rows from top to bottom. By this, we can still get the traffic volume from node i to node j by accessing V^t with index = $i*N+j$. By this, the problem is redefined as solving the TV V^t via a time-series TV formed by T previous TVs (V^{t-T}, V^{t-T+1}, ..., V^{t-3}, V^{t-2}, V^{t-1}). In this work, we assume that each OD traffic is independent of all other ODs. Hence, we feed the models with one OD pair traffic at a time. Our training dataset contains multiple TMs ordered by time captured in ascending order. Before feeding the models, the data is normalized by dividing all the traffic volume to the highest one so that all of their values are in range of $[0, 1]$, which helps optimize the training process's performance. To prepare the data for the training phase, we transform the traffic volume sequence into records so that each record comprises two parts: (1) Input: a vector of T consecutive traffic volumes; and (2) Output: The next traffic timestep's volume.

IV. EXPERIMENTS

In this section, the experiments are presented in two different datasets:

1) GÉANT backbone network traffic dataset
2) The dataset generated by our network testbed

A. GÉANT backbone network

GÉANT backbone networks dataset is the most well-known publicly available traffic dataset [21]. The GÉANT network topology consists of 23 peer nodes and 38 links. The dataset contains 10772 traffic matrices, recorded in 4 months in 2004 with a 15-minutes interval between two consecutive capturing times. After choosing $T = 10$ from the matrices, we generate 10762 training records, divided into two parts: a training set with 8609 records and testing set with 2153 testing records. We use the training set to feed the training models and use the testing set to evaluate predicting results.

B. Our SDN Testbed

Besides the GÉANT backbone network dataset, we also construct an SDN network testbed to collect traffic matrices using the TM Observation module presented in section III. Our testbed's network topology is based on the BSO Network Solution topology taken from TopologyZoo [22], consists of 14 switches and 18 links, simulated in Mininet [23] environment. Each switch is connected to a virtual host. The hosts are assigned different IP addresses from different subnets. That configuration allows us to distinguish the traffics between switches. In order to deploy the SDN applications, we use POX Controller [24], a lightweight SDN framework, providing basic APIs for application development and communication between switches using OpenFlow protocol.

In order to generate traffic inside the network, we use pcap samples of Netresec [25] and Wireshark [26]. These samples are captured in the real network traffic. The traffic categories vary from web surfing, video streaming to intrusion traffic. We use *tcprewrite*, *editcap* and *bittwiste* to modify the pcap files's information to arbitrary values. Finally, *tcpreplay* is used to transmit packets via virtual hosts attached to the switches. The source codes of our program are all available in [12].

After running the testbed for 4 days with a 1-minute interval between two consecutive capture times, 6258 traffic matrices are collected. From this, a set of 6248 training records is generated and divided into two subsets: 4998 records for models training and 1250 records for evaluation.

C. Evaluation Metric

The Root Mean Square Error (RMSE) is used to estimate the prediction's accuracy (Eq. 2).

$$RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - x_i)^2} \quad (2)$$

where x_i and y_i are the true value of the traffic volume at a certain time and the predicted value, respectively. n is the testing set's size. The lower RMSE, the better prediction performance.

D. Experimental Results

In order to demonstrate the results, the prediction result in graph plots and RMSE of two sample OD pairs from each dataset are shown in Fig. 3 to Fig.10.

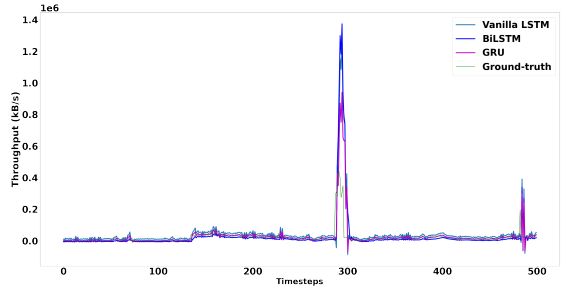

Fig. 3. GÉANT network 12th OD prediction comparision

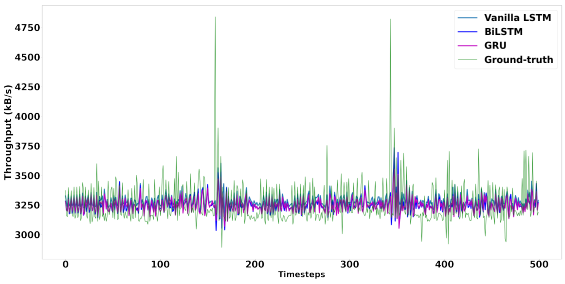

Fig. 7. Simulated Testbed 5th OD prediction comparision

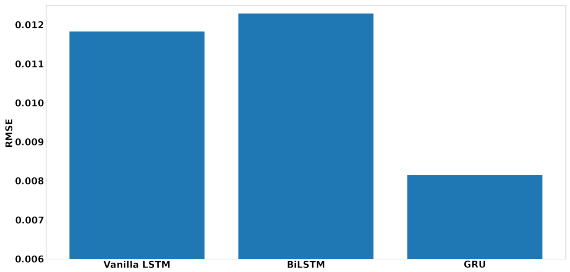

Fig. 4. GÉANT network 12th OD RMSE

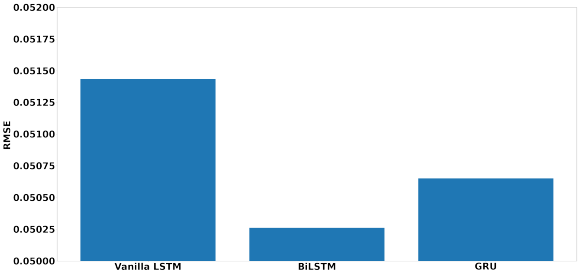

Fig. 8. Simulated Testbed 5th OD RMSE

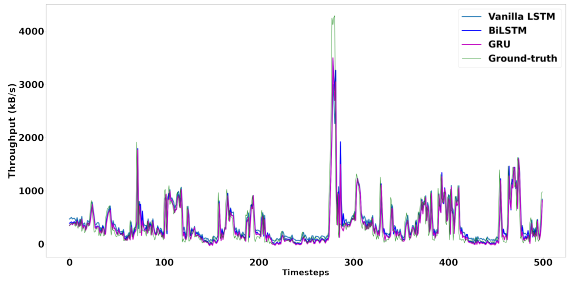

Fig. 5. GÉANT network 102nd OD prediction comparision

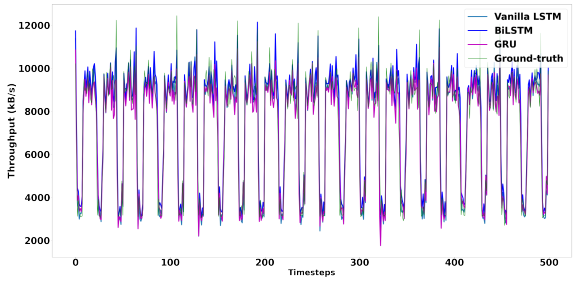

Fig. 9. Simulated Testbed 52nd OD prediction comparision

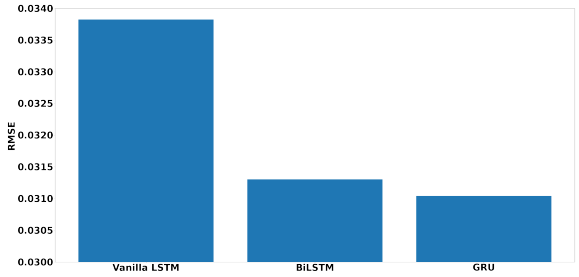

Fig. 6. GÉANT network 102nd OD RMSE

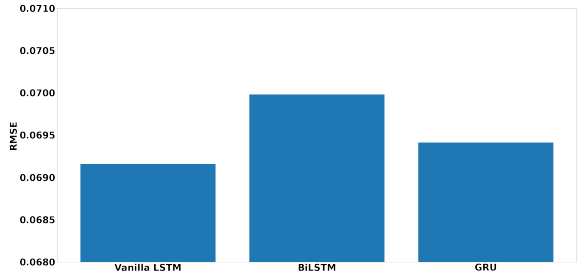

Fig. 10. Simulated Testbed 52nd OD RMSE

Theoretically, LSTM and GRU are quite similar. They are both solutions to short-term memory and use gates to regulate flow information. According to the difference, GRU has a simpler cell structure, without a memory unit. Hence, GRU uses less parameters (as shown in Table I) and memory for a faster execution. LSTM, in theory, remembers longer sequences and provides a better long-distance relations. However, a long sequence could increase complexity and reduce efficiency. Meanwhile, BiLSTM is the combination of sequence forward and backward LSTM. So, BiLSTM requires approximately double the number of parameters. Hence, BiLSTM consumes the most resources in training and predicting.

As we can see, with mostly stable traffic over time as 12th OD in GÉANT dataset (Fig. 3), all the three models perform well but still produce significant predicting errors when the traffic suddenly shot up or collapsed. Although the traffic fluctuates more quickly as in 102nd OD (Fig. 5), the results are better in terms of predicting local extreme points. In both cases, GRU gives better performance.

However, if traffic is unsteady and irregular as 5th OD of our Testbed (Fig. 7), the models perform the prediction with less accuracy. In the same circumstance, BiLSTM gives slightly better performance over the other two models (Fig. 10). On the contrary, if the traffic fluctuates regularly (Fig. 9), the predicted values fit accurately with real traffic value.

To sum up, our experiments show that the three RNN variants could accurately predict the TM in most cases. Overall, GRU model demonstrates its prominence over LSTM and BiLSTM with a lower error rate. With less complexity in structure, GRU could outperform LSTM models in prediction accuracy and efficiency.

V. Conclusion

In this paper, we have proposed an AI-based TM prediction application for an SDN network. The proposed application consists of two modules: TM Observation and TM Prediction. The former provides the SDN network's current TM information, while the latter predicts the future TM information. Concretely, we applied LSTM and its two variants, GRU and BiLSTM, to the TM prediction module. Experimental results have been performed to evaluate and compare the performance of these RNN algorithm variants. The obtained experimental results showed that our proposed RNN variants could generate accurate future traffic information for further network management activity. With the evaluation above, we recommend GRU as the most promising approach to coping with modern networks' complexity. This paper's obtained results may offer potential applications, such as network planning, resource management and network security.

Acknowledgment

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.02-2019.314.

References

[1] P. Tune and M. Roughan, "Internet traffic matrices: A primer," 2013.
[2] M. A. Togou, D. A. Chekired, L. Khoukhi, and G. Muntean, "A hierarchical distributed control plane for path computation scalability in large scale software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1019–1031, 2019.
[3] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic matrix estimation: existing techniques and new directions," 01 2002, pp. 161–174.
[4] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.
[5] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.
[6] M. V. De Assis, A. H. Hamamoto, T. Abrão, and M. L. Proença, "A game theoretical based system using holt-winters and genetic algorithm with fuzzy logic for dos/ddos mitigation on sdn networks," *IEEE Access*, vol. 5, pp. 9485–9496, 2017.
[7] M. Barabas, G. Boanea, A. B. Rus, V. Dobrota, and J. Domingo-Pascual, "Evaluation of network traffic prediction based on neural networks with multi-task learning and multiresolution decomposition," in *2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing*, 2011, pp. 95–102.
[8] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.
[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.
[10] Y. Fan, Y. Qian, F.-L. Xie, and F. K. Soong, "Tts synthesis with bidirectional lstm based recurrent neural networks," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.
[11] K. Cho, B. van Merriënboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 06 2014.
[12] D. H. LE and H. A. TRAN, *The paper source repository*, 2020. [Online]. Available: https://github.com/duchuyle108/SDN-TMprediction
[13] 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*, 2006, pp. 2635–2642.
[14] L. Nie, D. Jiang, L. Guo, S. Yu, and H. Song, "Traffic matrix prediction and estimation based on deep learning for data center networks," in *2016 IEEE Globecom Workshops (GC Wkshps)*, 2016, pp. 1–6.
[15] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "Opentm: Traffic matrix estimator for openflow networks," 04 2010, pp. 201–210.
[16] A. Gunnar, M. Johansson, and T. Telkamp, "Traffic matrix estimation on a large ip backbone: a comparison on real data," 01 2004, pp. 149–160.
[17] A. Azzouni and G. Pujolle, "Neutm: A neural network-based framework for traffic matrix prediction in sdn," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–5.
[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *Computer Communication Review*, vol. 38, pp. 69–74, 04 2008.
[19] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," *30th International Conference on Machine Learning, ICML 2013*, 11 2012.
[20] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.
[21] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *Computer Communication Review*, vol. 36, pp. 83–86, 01 2006.
[22] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.
[23] Mininet. [Online]. Available: http://mininet.org/
[24] S. Kaur, J. Singh, and N. S. Ghumman, "Network programmability using pox controller," in *International Conference on Communication, Computing & Systems (ICCCN'2014)*, 2014, pp. 134–138.
[25] netresec. [Online]. Available: https://www.netresec.com/
[26] Wireshark sample captures. [Online]. Available: https://wiki.wireshark.org/SampleCaptures