Deep Learning Models For Aggregated Network Traffic Prediction

Aggelos Lazaris and Viktor K. Prasanna
Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, CA 90089
Email: {alazaris, prasanna}@usc.edu

***Abstract*—The ability to generate network traffic predictions at short time scales is crucial for many network management tasks such as traffic engineering, anomaly detection, and traffic matrix estimation. However, building models that are able to predict the traffic from modern networks at short time scales is not a trivial task due to the diversity of the network traffic sources. In this paper, we present a framework for network-wide link-level traffic prediction using Long Short-Term Memory (LSTM) neural networks. Our proposed framework leverages link statistics that can be easily collected either by the controller of a Software Defined Network (SDN), or by SNMP measurements in a legacy network, in order to predict future link throughputs. We implement several variations of LSTMs and compare their performance with traditional baseline models. Our evaluation study using real network traces from a Tier-1 ISP illustrates that LSTMs can predict link throughputs with very high accuracy outperforming the baselines for various traffic aggregation levels and time scales.**

I. Introduction

Link-level network traffic predictions at short time-scales can provide information that is very crucial for various network management tasks such as traffic engineering, failure recovery, anomaly detection, performance diagnostics, load balancing, and Traffic Matrix (TM) estimation [6], [7], [8], [9], [10], [11], [12]. For example, in order for the network to react to a congestion event early on and prevent packet losses and increased delays, link level predictions can be used and reroute traffic accordingly without having to wait for the routing protocols to react with delay [2]. Another example is the use of predictions as baselines of normal behavior for the near future, and the treatment of any measured deviations from the baselines as detected anomalies [23], [22]. Finally, predictions can be used to substitute traffic measurements whenever the available resources are limited and network telemetry cannot be performed deterministically [4].

The prediction models that have been proposed in the relevant literature have been designed for large aggregation time-windows (i.e. 15 minutes in the vast majority of the cases) due to the challenges in regards to the very volatile nature of network traffic in smaller time scales that makes predictions hard. Another factor that poses significant challenges for the evaluation of statistical models in smaller time scales is the lack of *structured* datasets that the research community can easily use to better understand the modern network traffic. The development of Software-defined Networks (SDN) has enabled easier traffic measurement at short time-scales by leveraging the capabilities of the OpenFlow enabled switches to provide a centralized controller with statistics for each forwarding rule installed, as well as the traffic send/received by each link. This, makes SDN an ideal solution for bringing more visibility into the network and provide prediction models with the required data.

In this paper, we present an analysis of aggregated network traffic for various aggregation levels and short time scales, and propose the use of state-of-the-art deep-learning models for time-series predictions, namely Long Short-Term Memory (LSTM), in order to generate accurate traffic predictions. Due to the lack of publicly available structured datasets from both SDN and legacy networks at these time scales, we use the framework introduced in [26] that can process and aggregate traditional packet level logs (i.e. `pcap` files) at scale, and apply it to a relatively recent dataset from a Tier-1 ISP provided by CAIDA in [3].

The contributions of our work are summarized below:

1) We present an in-depth analysis of the capabilities of LSTMs to model aggregate network traffic at various traffic aggregation levels and short time scales, which can enable short-term decision making, unlike the easier to predict longer time-scales that traditional TM estimation frameworks are based on.
2) We propose a prediction framework that can be easily deployed to production SDN topologies without requiring any switch modifications since it leverages the port statistics that are available through the OpenFlow API in order to train the LSTM models.
3) We evaluate our framework using real backbone network traffic that was captured relatively recently (i.e. 2016) and which contains new traffic dynamics that were unavailable in relevant studies two or more decades ago, since in the recent years the multimedia content, the social networks, the mobile devices, the smart-TVs and more, have completely changed the traffic landscape.
4) We provide a comparison of several variations of LSTMs including vanilla LSTM, delta LSTM (which models the consecutive link throughput deltas), multi-variate LSTM (which models all the link throughput time series at once thus taking into account potential correlations), and compare with three versions of ARIMA-based models that have been traditionally used for network traffic modeling [20], [21].

II. Background & Motivation

Before presenting our analysis, we introduce the following definitions that will be used throughout this work.

978-3-903176-24-9

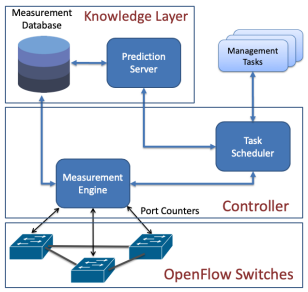
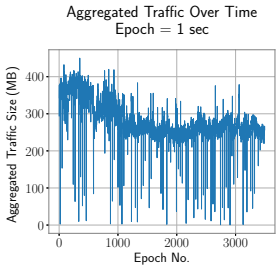
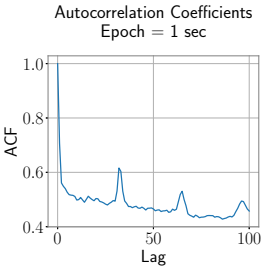


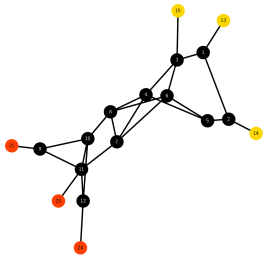
Fig. 1: SDN measurement and prediction architecture.



(a) CAIDA trace aggregated in 1 second periods.



(b) ACFs for various lags for the trace in Fig. 2(a).



(c) B4 topology (black) with ingress (yellow) and egress (orange) nodes attached.

Fig. 2: The CAIDA time-series aggregated across all prefixes, its autocorrelation coefficients, and the topology used to replay the trace.

Definition 1. A *flow* is a set of IP packets passing an observation point in the network during a certain time interval which share a set of common properties such as source and destination IP address or IP prefix.

Definition 2. A *flow-rate time-series*, or simply a flow time-series $F_i = f_i^{(1)}, f_i^{(2)}, ..., f_i^{(n)}$ is an ordered set of n real-valued variables that correspond to the total traffic volume of flow f_i over a measurement interval.

Since network links act as flow aggregators, we can extend the above definitions for the cases of directional link throughputs by summing the time series of the individual network flows that a link carries according to a routing protocol. For this, consider a set of flow-rate time-series F_i (potentially mega-flows) of length n (zero-padding can be used to achieve equal n across time-series), and a set of N switches that are connected with L directional links and forward traffic according to a routing matrix R with elements $r(j,i)$ that represent the fraction of traffic of flow i that is forwarded through link j (assuming links have unique identifiers). Then:

Definition 3. A *link-throughput time-series* for link j, or simply a link time-series $L_j = l_j^{(1)}, l_j^{(2)}, ..., l_j^{(n)}$ is an ordered set of n real-valued variables such that:

$$l_j^{(t)} = \sum_{\substack{i \in \{1,\dots,N\} \\ r_{j,i} > 0}} f_i^{(t)}, \tag{1}$$

for $t \in \{1, \dots, n\}$.

Definition 4. Given a link-throughput time-series L_j of length n, a *link-throughput subsequence* $L_j^{(p)}$ of L_j is a sampling of length $w < n$ of contiguous positions from L_j, that is, $L_j^{(p)} = l_j^{(p-w+1)}, l_j^{(p-w+2)}, ..., l_j^{(p)}$ for $w \leq p \leq n$.

A link-throughput time-series and all its subsequences can be easily derived by collecting link-level measurement data using the port statistics from any OpenFlow-enabled switches. Of course, in hybrid deployments (i.e. SDN and legacy topologies), a combination of both OpenFlow and SNMP statistics can be used in order to build a more complete picture of the network. In this paper, and without loss of generality, we will be focusing on the SDN use-case only which can be summarized in the architecture shown in Fig. 1. As we can see, in order for an SDN controller to perform a given management task, it retrieves directional link counters from OpenFlow switches and stores them to a database in order to calculate the desired subsequences that will used by the prediction server. In order for the SDN controller to calculate the link-throughput time-series, it collects the cumulative incoming/outgoing traffic counters of each switch port, and converts them to differentials using the counters from the previous measurement epoch. Finally, the layer that stores, analyzes, models, and predicts the traffic can also be used to enable a knowledge plane, as described in [14], and can further support network-wide data-driven decision making. In our modeling study, or when implementing a prediction server as described above, we use a subsequence $L_j^{(p)}$ with $p = n/2$ and $w = n/2$ data points for model training, and a subsequence $L_j^{(p)}$ with $p = n$ and $w = n/2$ for model testing. In this paper, n is defined as the full length of the dataset (described in more detail in Section III), after the traffic has been grouped in time epochs. In a production deployment, n can be capped using a rolling time-window (i.e. the last n samples).

III. Aggregated Network Traffic Analysis

One of the biggest challenges in modeling aggregated network traffic is the ever-changing nature of the content generation and consumption landscape which can make old datasets obsolete almost every decade. For example, the constant consumption of multimedia content, the frequent web browsing from mobile devices, the file-sharing applications, are examples that were barely available more than a decade ago. So, all these new traffic types and patterns need to a) be better understood, and b) be modeled accurately using a scalable architecture.

The available traffic matrix datasets also suffer in terms of the issues described above (i.e. $\geq$ 15 min time scales from data collected in 2004 or 2006 [24], [25]). In addition, in such cases, network traffic exhibits hourly, daily, and weekly periodicities, and thus makes it easier to predict [2]. In our study, we use a 1-hour long trace containing large volumes of packet level traffic from a Tier-1 ISP from [3] which was captured in 2016. The trace contains 1.65 billion IPv4 packets with total size of

0.98 TB. The aggregated traffic time-series is shown in Fig. 2(a) when aggregated across all flow prefixes with 1 second aggregation epoch, from where we can see that the traffic is quite volatile with a visible downward trend that is part of the daily traffic periodicities, as well as significant autocorrelation coefficients for various lags, as shown in Fig. 2(b) that motivate the use of LSTMs for modeling such timeseries [1].

The CAIDA dataset contains traffic collected from the high-speed "equinix-chicago" monitor which is located at the Equinix datacenter in Chicago, IL, and is connected to a backbone 10GigE link of a Tier1 ISP that connects Chicago, IL and Seattle, WA. Each direction of the bidirectional link is monitored and logged separately and labeled as "direction A" (Seattle to Chicago) and "direction B" (Chicago to Seattle). However, since CAIDA is aware that some data in this dataset contain more than trivial amounts of packet loss (especially direction B) due to the way the monitoring equipment is set up and the high network speeds, in this work we are focusing only on direction A. The extracted dataset was approximately 120GB in size, and in order to aggregate it in various time scales and subnet mask sizes, we use a modern big-data processing framework, namely Google BigQuery [13] similar to [26].

In order to analyze the characteristics of the traffic when grouped together in network links, we proceed to implement Google's B4 backbone topology [1] from [15] and assign the source and destination prefixes from the CAIDA trace to ingress and egress routers as shown in Fig. 2(c). Specifically, we group the source and destination IPs of each micro-flow into one of the following four aggregation levels, i.e. /8, /10, /12, and /14 both at the source and the destination IP, and assign these aggregated prefixes at random to one of the 3 ingress and egress nodes shown in Fig. 2(c). To calculate the link-throughput subsequences observed by each link in our topology, we assign source and destination prefix pairs to links using shortest path routing from each source to its destination. This way, each link observes various traffic aggregations simulating a real network scenario.

IV. Methodology

A. Long Short-Term Memory Model

A Long-Short-Term-Memory (LSTM) model is a form of a recurrent neural network that has gained popularity in the recent years due to its effectiveness in modeling complex time-series with time lags of unknown size that separate important events [5], [1]. The main idea of LSTM is the use of self-loops where the gradient can flow for long durations without vanishing or exploding. This, in combination with the use of a forget-gate, allows the LSTM to accumulate knowledge that can be "forgotten" later depending on the input data. LSTMs are characterized by the following recursive equations:

$$\mathbf{f}^{(t)} = \sigma\big(\mathbf{W}_f\mathbf{x}^{(t)} + \mathbf{U}_f\mathbf{h}^{(t-1)} + \mathbf{b}_f\big) \quad (2)$$

$$\mathbf{i}^{(t)} = \sigma\big(\mathbf{W}_i\mathbf{x}^{(t)} + \mathbf{U}_i\mathbf{h}^{(t-1)} + \mathbf{b}_i\big) \quad (3)$$

$$\tilde{\mathbf{c}}^{(t)} = \tanh\big(\mathbf{W}_c\mathbf{x}^{(t)} + \mathbf{U}_c\mathbf{h}^{(t-1)} + \mathbf{b}_c\big) \quad (4)$$

$$\mathbf{c}^{(t)} = \mathbf{i}^{(t)} \odot \tilde{\mathbf{c}}_{(t)} + \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} \quad (5)$$

$$\mathbf{o}^{(t)} = \sigma\big(\mathbf{W}_o\mathbf{x}^{(t)} + \mathbf{U}_o\mathbf{h}^{(t-1)} + \mathbf{b}_o\big) \quad (6)$$

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \tanh(\mathbf{c}^{(t)}) \quad (7)$$

[1]We also tried random topologies with very similar results.

where $\mathbf{f}^{(t)}, \mathbf{i}^{(t)}, \tilde{\mathbf{c}}^{(t)}, \mathbf{c}^{(t)}, \mathbf{o}^{(t)}, \mathbf{h}^{(t)}$ are the forget gate, input gate, candidate state, current state, output gate, and hidden state, respectively, $\mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_c, \mathbf{W}_o$ are the input weights for the forget gate, input gate, candidate state gate, and output gate, respectively, and $\mathbf{U}_f, \mathbf{U}_i, \mathbf{U}_c, \mathbf{U}_o$ are the recurrent weights for the forget gate, input gate, current state, and output gate, respectively. In addition, $\odot$ is the (element-wise) Hadamard product, and σ is the sigmoid function.

B. Data Transformations

One very common approach when modeling data in practice is to apply several transformations to the data depending on their distribution, in order to achieve certain properties. In this work, we apply the following transformations in various combinations in order to assess their effectiveness: a) normalize the time-series, and b) model the deltas (i.e. $f_i^{(j)} - f_i^{(j-1)}$, $f_i^{(j-1)} - f_i^{(j-2)}, \ldots$) instead of the actual values. The transformations aim to help the model focus on the relative feature importance rather than their absolute magnitudes.

C. Error Metric

In order to evaluate the ability of LSTMs to model the network traffic described in Section III, assess their effectiveness by calculating the Mean Absolute Percentage Error (MAPE), as defined below:

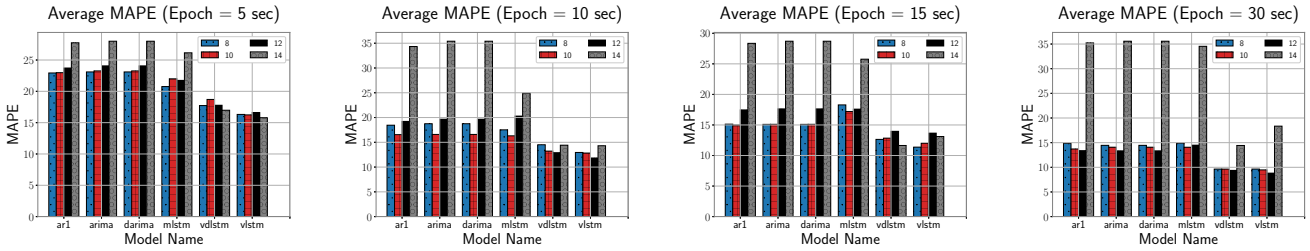
$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\frac{|f_i^{(t)} - u_i^{(t)}|}{|f_i^{(t)}|} \quad (8)$$

where $f_i^{(t)}$ is the actual flow-rate value and $u_i^{(t)}$ the estimated flow-rate value for a given flow i.

V. Evaluation

We implemented three variations of LSTM, as well as three variations of ARIMA baselines [27]. In order to choose the best performing hyper-parameters, we implemented random search [30] where random combinations of the model parameter values are chosen (i.e. look-back window, number of units, number of layers, dropout ratio, epochs, and batch size) and the one that performs the best is finally selected. The details of each model are shown below:

1) **Vanilla LSTM (vlstm):** This is a simple LSTM architecture with an LSTM layer with 50 units, followed by a dense layer with 50 units, dropout of 20%, look back window 3, 50 training epochs (not to be confused with the aggregation epoch used during the dataset creation), batch size 4, and standard scaler on the time-series data. The model completed its training in less than a minute in a regular PC, with a 50-50 train/test split. The model was used to model each link's traffic separately and calculate an average of all the MAPE errors across links.
2) **Delta LSTM (dlstm):** This is exactly the same architecture as in 1) above, with the only difference that the input data have been pre-processed to calculate the time-series deltas. The model was used to model each link's traffic separately and calculate an average of all the MAPE errors across links.
3) **Multivariate LSTM (mlstm):** This is an LSTM architecture that consists of an LSTM layer with 3×(#links) units (for the last 3 observations of each link), followed by a dense layer of #links units, dropout of 30%, 50 training



(a) Average MAPE for various mask sizes for epoch = 5 sec.

(b) Average MAPE for various mask sizes for epoch = 10 sec.

(c) Average MAPE for various mask sizes for epoch = 15 sec.

(d) Average MAPE for various mask sizes for epoch = 30 sec.

Fig. 3: Average MAPE for each model across various mask sizes and epoch durations.

epochs, batch size 4, and standard scaler on the time-series data. This model produces predictions for all the links at once.

4) **ARIMA model (arima):** This is an ARIMA model with $p = 3, q = 0$, and $d = 0$ parameters, that was found to perform well for a variety of traces. The parameter p corresponds to the number of lag observations included in the model, d corresponds to the number of times that the input data are differenced, and q corresponds to the size of the moving average window.
5) **Delta ARIMA model (darima):** This is an ARIMA model with $p = 3, q = 0$, and $d = 1$ parameters that implements differentiation to improve stationarity.
6) **First-Order Autoregressive ARIMA (ar1):** This is an ARIMA model with $p = 1, q = 0$, and $d = 0$ parameters that generates predictions based on the last value seen.

We run each model 10 times to calculate the final MAPE averages. For the models that operate on each link separately, we calculate the average MAPE across links. All the model training times range between 5-30 seconds in a server with 32GB of RAM and an Intel CPU with 16 cores. The average MAPEs for each mode across all epochs are shown in Figs. 3(a), 3(b), 3(c), and 3(d). As we can see from the figures, all the LSTM models perform much better than the ARIMA ones in all the scenarios under consideration, which validates our hypothesis that LSTM is a good candidate for link-level traffic predictions at time-scales below 30 seconds. The difference becomes much bigger (i.e. more than 2x) for all time scales in the case of /14 network assignment due to the more challenging nature of the resulting aggregation that separated volatile low rate time series with non volatile high rate ones. The same pattern but in smaller scale is evident for smaller aggregation masks (i.e. /8, /10, /12). The time epoch played also an important role in the model performances, with larger time epochs producing better results for all the models, which is expected due to the reduced variance of the traffic at higher aggregation scales. Among all the LSTM models, we can observe that vlstm (per link) performs better overall, followed by vdlstm and mlstm. The reason for this is due to the fact that vlstm and vdlstm are optimized for each link separately, whereas the mlstm underfits when trying to capture the traffic patterns across the network. Finally, the vdlstm performs relatively similar to vlstm, especially for higher epoch sizes. In the future, we are planning to further optimize mlstm and use more data in order to reduce underfitting. Finally, we can observe that all three ARIMA models exhibit very similar performance.

Based on the above results, we can conclude that per-link LSTMs can provide the best results with quick training times ($\leq$ 30 seconds), and up to 2x reduction in MAPE compared to ARIMA models.

VI. Related Work

The problem of modeling network flow time-series is not new in the relevant literature. Most of the previous works have focused on modeling the aggregate size of a number of flows over time windows of several minutes [16], [17], [18], [28], [29]. On the other hand, there have been some efforts on modeling aggregated flow-sizes in shorter time scales, such as [19], [20], [21], [28] but none of them provides a generic and scalable solution for modern networks, since most of the works rely on traces that are more than 15 years old that differ significantly from the modern traffic dynamics. From these works, [28] is the most relevant to our approach. However, [28] has some major differences as follows: a) it uses the 2004 GEANT/ABILENE datasets (we use the 2016 CAIDA) that contain 15 minute aggregates which are easier to model and not suitable for short-term/fast decision making, b) it also uses a small dataset with 5 second intervals collected from a single link sending artificial traffic between 2 virtual machines and which contains only 25K packets (we use 1.65 Billion packets from real ISP) and thus cannot provide any conclusions about backbone traffic modeling at short time-scales. In addition to the above, our previous works in [4], [26] use small time-scale predictions as in here. However, none of them models network traffic at the link level, neither they apply multivariate models as a way to capture network-wide correlations.

VII. Conclusion And Future Work

In this paper, we presented several variations of LSTM that can effectively model backbone network traffic at the link level and for various time-epochs and compared with several ARIMA baseline models. The results obtained look very promising and validate the hypothesis that LSTM is a good candidate for link-level network traffic modeling. In the near future, we are planning to further investigate this possibility by optimizing more the models used, as well as trying different data transformations, neural network architectures, and error metrics.

REFERENCES

[1] Sepp Hochreiter and Jrgen Schmidhuber. 1997. “Long Short-Term Memory”. Neural Comput. 9, 8 (November 1997), 1735-1780.

[2] A. Azzouni and G. Pujolle, ”NeuTM: A neural network-based framework for traffic matrix prediction in SDN,” NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, 2018, pp. 1-5.

[3] CAIDA Anonymized Internet Traces 2016. http://www.caida.org/data/passive/passive_2016_dataset.xml

[4] Aggelos Lazaris and Viktor K. Prasanna. 2017. DeepFlow: a deep learning framework for software-defined measurement. In Proceedings of the 2nd Workshop on Cloud-Assisted Networking (CAN ’17). ACM, New York, NY, USA, 43-48.

[5] Ian Goodfellow and Yoshua Bengio and Aaron Courville, ”Deep Learning”, MIT Press, 2016.

[6] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A. Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, Changhoon Kim, and Naveen Karri. 2013. Ananta: cloud scale load balancing. SIGCOMM Comput. Commun. Rev. 43, 4 (August 2013), 207-218.

[7] Augustin Soule, Kav Salamatian, and Nina Taft. 2005. “Combining filtering and statistical methods for anomaly detection”. In Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement (IMC ’05). USENIX Association, Berkeley, CA, USA, 31-31.

[8] Matthew Roughan, Mikkel Thorup, and Yin Zhang. 2003. “Traffic engineering with estimated traffic matrices”. In Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC ’03). ACM, New York, NY, USA, 248-258.

[9] Theophilus Benson, Ashok Anand, Aditya Akella, Ming Zhang. 2011. “MicroTE: fine grained traffic engineering for data centers”. In Proceedings of the 2011 Conference on Emerging Networking Experiments and Technologies (Co-NEXT ’11). ACM, Tokyo Japan.

[10] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. 2011. “DevoFlow: scaling flow management for high-performance networks”. SIGCOMM Comput. Commun. Rev. 41, 4 (August 2011), 254-265.

[11] A. Yassine, H. Rahimi and S. Shirmohammadi. 2015. “Software defined network traffic measurement: Current trends and challenges,” in IEEE Instrumentation & Measurement Magazine, vol. 18, no. 2 (April 2015), 42-50.

[12] Tune, Paul, and Matthew Roughan. ”Internet traffic matrices: A primer.” Recent Advances in Networking 1 (2013).

[13] Google BigQuery cloud data warehouse, https://cloud.google.com/bigquery/.

[14] Albert Mestres, et al. 2017. Knowledge-Defined Networking. SIGCOMM Comput. Commun. Rev. 47, 3 (2017), 2-10.

[15] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hlzle, Stephen Stuart, and Amin Vahdat. 2013. B4: experience with a globally-deployed software defined wan. In Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM (SIGCOMM ’13). ACM, New York, NY, USA, 3-14.

[16] K. Papagiannaki, K. Papagiannaki, N. Taft, N. Taft, Z. Zhang, Z. Zhang, C. Diot, and C. Diot, ”Long-Term Forecasting of Internet Backbone Traffic: Observations and Initial Models”, vol. 0, no. C, pp. 1178-1188, 2003.

[17] K. U, Z.-L. Zhang, and S. Bhattacharyya, ”Profiling internet backbone traffic”, ACM SIGCOMM Comput. Commun. Rev., vol. 35, no. 4, p. 169, 2005.

[18] You, C. and Chandra, K., ”Time Series Models for Internet Data Traffic”, In Proc. of IEEE LCN 1999.

[19] C. Barakat, P. Thiran, G. Iannaccone, C. Diot, and P. Owezarski, ”Modeling Internet backbone traffic at the flow level,” IEEE Trans. Signal Process, vol. 51, no. 8, pp. 1-12, 2003.

[20] S. Basu and A. Mukherjee, ”Time Series Models for Internet Traffic”, in 24th Conf. on Local Computer Networks, Oct. 1999, pp. 164-171.

[21] A. Sang and S. Li, ”A Predictability Analysis of Network Traffic”, in INFOCOM, Tel Aviv, Israel, Mar. 2000.

[22] Min Cheng, Qian Xu, Jianming Lv, Wenyin Liu, Qing Li and Jianping Wang, ”MS-LSTM: A multi-scale LSTM model for BGP anomaly detection,” 2016 IEEE 24th International Conference on Network Protocols (ICNP), Singapore, 2016, pp. 1-6.

[23] G. Qin, Y. Chen and Y. Lin, ”Anomaly Detection Using LSTM in IP Networks,” 2018 Sixth International Conference on Advanced Cloud and Big Data (CBD), Lanzhou, 2018, pp. 334-337.

[24] [Online] https://www.geant.org/Projects/GEANT_Project_GN4

[25] Uhlig, Steve, et al. ”Providing public intradomain traffic matrices to the research community.” ACM SIGCOMM Computer Communication Review 36.1 (2006): 83-86.

[26] A. Lazaris and V. K. Prasanna, ”An LSTM Framework For Modeling Network Traffic,” 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Arlington, VA, USA, 2019, pp. 19-24.

[27] Robert H. Shumway and David S. Stoffer. 2005. Time Series Analysis and its Applications (Springer Texts in Statistics). Springer-Verlag, Berlin, Heidelberg.

[28] N. Ramakrishnan and T. Soni, ”Network Traffic Prediction Using Recurrent Neural Networks,” 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, 2018, pp. 187-193.

[29] V. A. Le, P. Le Nguyen and Y. Ji, ”Deep Convolutional LSTM Network-based Traffic Matrix Prediction with Partial Information,” 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Arlington, VA, USA, 2019, pp. 261-269.

[30] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. J. Mach. Learn. Res. 13 (February 2012), 281-305.