An LSTM Framework For Modeling Network Traffic

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*—Forecasting fine-grained network traffic is crucial for many network management and optimization tasks such as traffic engineering, anomaly detection, network accounting, network analytics, load balancing, and traffic matrix estimation. However, building models that are able to predict a wide-variety of network traffic types is not a trivial task due to a) the diversity of network traffic, and b) the computational challenges in processing large datasets to train the prediction models. In this paper, we present a network traffic prediction framework that uses real network traces from a Tier-1 ISP to train a Long Short-Term Memory (LSTM) neural network and generate predictions at short time scales ($\leq$ 30 seconds). In order to reduce the number of models needed to capture the very diverse dynamics of the various traffic sources, we develop a feature-based clustering framework that acts as a preprocessing step in order to group similar time-series together and train a single model for each group. Our extensive experimental evaluation study shows that LSTMs can indeed be used to predict network traffic with low prediction errors.**

I. Introduction

Fine-grained network traffic predictions can enable a large variety of network management tasks such as network monitoring, traffic engineering, anomaly detection, network accounting, network analytics, performance diagnostics, load balancing, and Traffic Matrix (TM) estimation [14], [15], [16], [17], [18], [19], [20]. However, traditionally, prediction models for network traffic have only been developed for large aggregation time-windows ($>$ 15 minutes in most of the cases) due to a) the very volatile nature of network traffic in smaller time scales, b) the lack of computational resources to process packet level training data at scale, and c) the lack of efficient models that can predict network flow rates with high accuracy. For this, fine-grained traffic predictions have been substituted by network traffic measurement frameworks that fall into one of the following three categories, depending on their objective [19]: a) balance in overhead implications by using techniques like sampling, aggregation, efficient heuristics, etc., b) resource usage as a trade-off with measurement accuracy, and c) accurate measurements in real-time for decision making. However, none of the existing frameworks can provide a generic solution for granular traffic measurement, since either the framework will be application specific, or there is going to be some tradeoff that we are trying to balance. More recently, the development of Software-defined Networks (SDN) allowed more fine-grained measurement to be performed by providing statistics for each forwarding rule of an OpenFlow-enabled switch. However, commodity hardware switches use TCAMs to collect statistics for every forwarding rule installed, which have very limited size due to their high cost and power consumption (4K rules in most of the switches). For this, only a small fraction of the total number of flows that a switch forwards can be monitored at any time. As it is evident from the above, more efficient mechanisms are needed in order to be able to a) collect network measurements, and b) use the collected measurements in order to generate accurate traffic predictions for the traffic that cannot be measured, as well as traffic forecasts that characterize the future behavior of the network. Motivated by this, we proposed DeepFlow in [11], a framework that can increase the measurement capabilities of SDN by efficiently multiplexing measurements and predictions. In this paper, we extend our work from [11] and conduct an in-depth modeling study of backbone network traffic using a relatively recent dataset provided by CAIDA in [10] as well as the state-of-the-art deep-learning model for time-series predictions, namely Long Short-Term Memory (LSTM). The contributions of our work are summarized below:

1) We provide a generic architecture for analyzing large volumes of raw network traffic in order to model them in various time scales and various aggregation levels (megaflow level).
2) We provide an in-depth analysis of backbone network traffic that was captured relatively recently (i.e. 2016) and which contains new traffic dynamics that were unavailable in relevant studies two 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.
3) We provide an analysis of several variations of LSTMs for network traffic modeling, including vanilla LSTM, delta-based LSTMS (i.e. models that predict the consecutive flow-size deltas), and cluster-based LSTMs. This way we can shed more light on what type of model is good for what type of traffic.
4) We propose a set of features that can be used to cluster the traffic time-series and use a single model per cluster in order to reduce the number of distinct models needed to cover a large variety of traffic dynamics, and thus increase the scalability of the framework.

II. Background & Motivation

Before we present our analysis, we provide the following three definitions that will be used throughout this work:

Definition 1. A flow is defined as a set of IP packets passing an observation point in the network during a certain time

This work has been funded by the U.S. National Science Foundation under grant number CNS-1643351.

interval which share a set of common properties such as the source and the destination IP address (or 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.

Definition 3. Given a time-series F_i of length n, a subsequence $F_i^{(p)}$ of F_i is a sampling of length $w < n$ of contiguous positions from F_i, that is, $F_i^{(p)} = f_i^{(p-w+1)}, f_i^{(p-w+2)}, ..., f_i^{(p)}$ for $w \leq p \leq n$.

In this paper, and without loss of generality, we will focus on modeling the special case of Definition 1 where a *flow is defined as the traffic that shares the same source and destination IP prefix (i.e. megaflow), for the whole duration of our experiment*. To do so, we develop a framework that aggregates data both spatially and temporally using a big data processing framework, as it will be described in Section IV. The aggregation is done on source and destination IP prefix pairs that are characterized by their subnet mask size. Our goal is to use past measurements for the size of the traffic from an IP prefix over a given measurement period (e.g. the last 3 observations) and use that to predict the size for the next epoch. To train our model, we use the subsequence $F_i^{(p)}$ with $p = n/2$ and $w = n/2$ data points, and to test our model we use the subsequence $F_i^{(p)}$ with $p = n$ and $w = n/2$. In this paper, n is defined as the length of the dataset, after the packets have been grouped in time epochs.

One of the biggest challenges in modeling network traffic is its diversity in terms of the dynamics of the underlying traffic (e.g. multimedia, web content, file-sharing, etc.) that makes it hard for a single model to generate good predictions especially in small time-scales. For this, larger aggregation windows (usually > 15 min) have been used where network traffic exhibits hourly, daily, and weekly periodicities, and thus makes it easier to predict. However, this coarse grain predictions cannot always satisfy the requirements of many network management tasks. So, in our study, we focus on a 1-hour long trace containing large volumes of traffic from a Tier-1 ISP from [10], which is shown in Fig. 1 when aggregated across all prefixes. The trace contains 1.65 billion IPv4 packets with total size of 0.98 TB.

In order to better understand the trace characteristics, we show in Figs. 2(a), and 2(b) the histogram of packet size in the trace, and the histogram of the total traffic sizes for a measurement epoch of 1 second, respectively. As we can see from the graphs, the packet sizes are either small (e.g. TCP ACKS) or around 1500 bytes, which aligns with the fact that TCP packets are 88.95% of the trace and UDP is 10.73%. Another observation is that the aggregated traffic time-series appears to follow a bimodal distribution with a long left tail that corresponds to the frequent drops in the traffic which is not clear if it was due to the measurement equipment of CAIDA or it was inherent characteristic of the traffic. Finally, in Fig. 3 we show the autocorrelation coefficients of the aggregated trace for various lags from where we can see that all the first 100 lags have significant ACF, which is a good indicator that a model like LSTM that was designed specifically to effectively handle long-range dependencies ([8]) could work well.

III. 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 [12], [8]. 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. To the best of our knowledge, this is the first time that LSTM models are used for modeling fine-grained network flow sizes in short time scales.

LSTMs are characterized by the following recursive equations:

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

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

$$\tilde{\mathbf{c}}^{(t)} = \tanh\big(\mathbf{W}_c \mathbf{x}^{(t)} + \mathbf{U}_c \mathbf{h}^{(t-1)} + \mathbf{b}_c\big) \quad (3)$$

$$\mathbf{c}^{(t)} = \mathbf{i}^{(t)} \odot \tilde{\mathbf{c}}_{(t)} + \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} \quad (4)$$

$$\mathbf{o}^{(t)} = \sigma\big(\mathbf{W}_o \mathbf{x}^{(t)} + \mathbf{U}_o \mathbf{h}^{(t-1)} + \mathbf{b}_o\big) \quad (5)$$

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \tanh(\mathbf{c}^{(t)}) \quad (6)$$

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. Time-Series Clustering For Model Selection

Network traffic is very heterogeneous in nature, depending on the application that generates the data, the transport protocol used (e.g. TCP, UDP, etc.), the available bandwidth at the edge, the congestion in the network, the time of the day, the distance between source and destination, the end-user behavior and more. These can create flow-rate time-series that can have very different dynamics from prefix to prefix, and which would be hard to capture with a single machine learning model. On the other hand, using a different model for each flow prefix would not scale due to the large number of possible source - destination IP pairs. So, in order to be able to use a small set of models to model a large variety of flow prefixes with different dynamics, we use a time-series clustering framework that assigns time-series into a set of disjoint clusters and train a single machine learning model for each cluster.

Given the set of all the active flows $F_i^{(p)}$ for $i \in \{i, \ldots, K\}$, the goal of the time-series clustering framework is to find a partition $A_1, A_2, \ldots A_k$ where k is the total number of clusters and K the total number of flows. Let $z_i \in \{1, 2, \ldots, k\}$ be the cluster to which time-series $F_i^{(p)}$ is assigned. Then, the clustering algorithm needs to find the optimal $z_i^* = argmax_z \; p\big(z_i = z | \mathbf{x}_i, \mathcal{D}\big)$ where $\mathbf{x}_i$ is the feature vector that characterizes the time-series $F_i^{(p)}$, and $\mathcal{D}$ is the rest of the input data. In the analysis above, the number of clusters k can be set

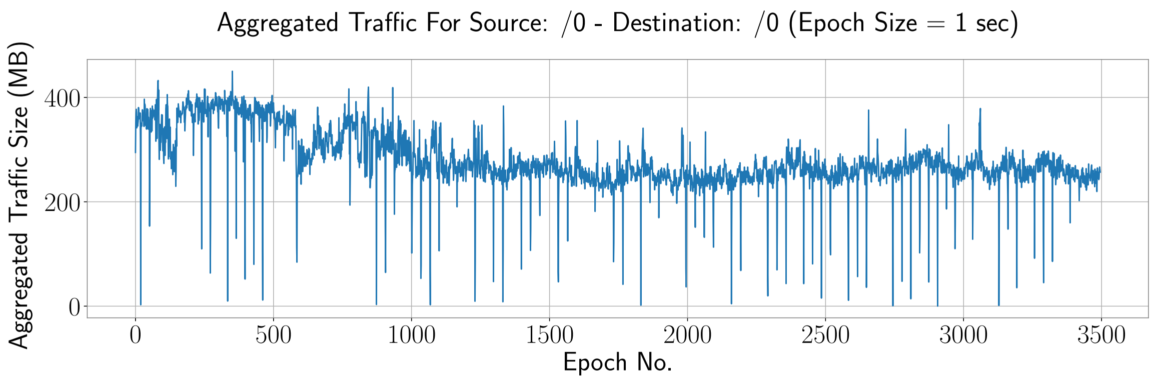
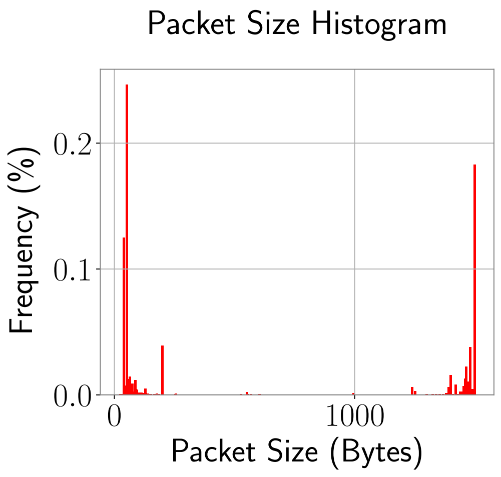
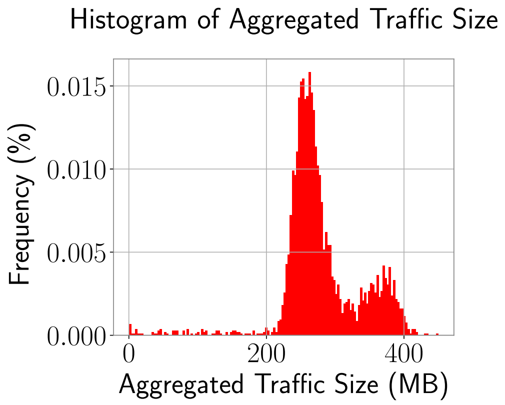


Fig. 1: Network traffic time-series from a Tier-1 ISP link (aggregated) over 1-sec measurement intervals.



(a) Histogram of packet sizes for the CAIDA trace.



(b) Histogram of network traffic size for epoch = 1 sec.

Fig. 2: Network trace histograms for packet size and aggregate volume over 1 second periods.

manually (e.g. in case of k-Means) or derived automatically by the clustering algorithm (e.g. in DBSCAN).

In general, time-series clustering can be done using three kinds of approaches [13]: a) raw data based methods, b) feature-based methods, and c) model-based methods. Here, we use a feature based method to cluster the time-series $F_i^{(p)}$ since a) this type of approach works well for heterogeneous time-series, b) it requires a smaller set of dimensions compared to

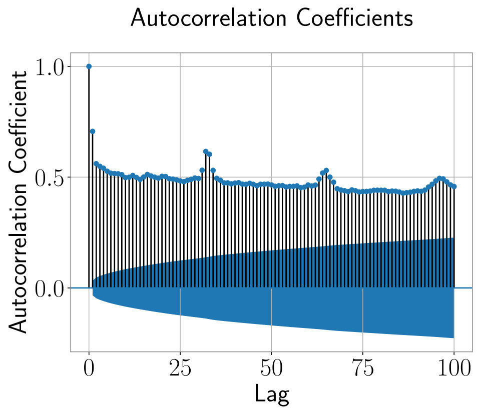


Fig. 3: ACFs for various lags for the trace in Fig. 1. The shaded area corresponds to statistically non-significant ACF values.

a raw-data approach, and c) it does not depend on a specific model of the data. However, the challenge here is to find a set of features that can work well. For this, we use the features shown in Table I that were derived since they a) all capture the dynamics of the time-series, b) can produce relatively good clusters, as indicated by various cluster quality metrics such as homogeneity, completeness, and silhouette coefficient.

The clustering algorithms used are k-Means and DBSCAN. For k-Means, we specify the number of clusters k (20 was used after experimenting with various values and deriving clustering quality metrics, but depending on the data we want to model this can be tweaked accordingly) that need to be used to partition the time-series features such that the within-cluster sum of squares is minimized as follows: $z_i^* = argmin_z \|\mathbf{x}_i - \mu_z\|_2^2$ where μ_z is the cluster's center and it is defined as $\mu_z = \frac{1}{N_k}\sum_{i:z_i=z}\mathbf{x}_i$ and N_z the total number of points in cluster z. On the other hand, DBSCAN only needs to be specified the maximum distance between two samples for them to be considered as belonging to the same neighborhood, and also the minimum number of samples (10 was used as a minimum) in a neighborhood for a point to be considered as a core point (including the point itself).

C. 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

TABLE I: Features for network time-series clustering.

Feature Name	Description
mean	mean traffic size
variance	variance of traffic size
median	median of traffic size
min	the smallest traffic size seen
max	the largest traffic size seen
ptp	the range of values (peak-to-peak)
skew	skewness of the time-series
kurtosis	fourth central moment divided by the square of the variance
acf1	lag-1 autocorrelation coefficient
acf10	lag-10 autocorrelation coefficient
entropy	sample entropy of the data
mask	the mask size used to aggregate the traffic
epoch	the epoch size used to aggregate the traffic over time.

this work, we apply the following transformations in various combinations in order to assess their effectiveness: a) normalize the time-series, 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 (this also changes the distribution of the data and may achieve better performance), and c) model the logarithm of the time-series values.

IV. Evaluation

In order to evaluate the proposed approach, we analyzed real network logs provided by CAIDA [10] that contain traces collected from the high-speed "equinix-chicago" monitor in 2016. The "equinix-chicago" monitor 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 network traces are available in large `pcap` files which were processed to extract the `(timestamp, ip_version, source_ip, destination_ip, protocol, source_port, destination_port, packet_size)` of each packet. Since 99.68% of the packets in the traces are TCP or UDP, we are explicitly focusing on extracting only these two protocols form the trace. In addition, due to the large size of the resulting files (since they contain packet level information), traffic aggregation at various time scales and mask sizes was performed using a modern big-data processing framework, namely Google BigQuery [21]. To leverage BigQuery's fully managed backend, we first import the packet-level processed logs to the database (in CSV format), and then use a Python SQL query generator to create and execute the SQL aggregation queries that will create the final dataset. The overall process is shown in Fig. 4.

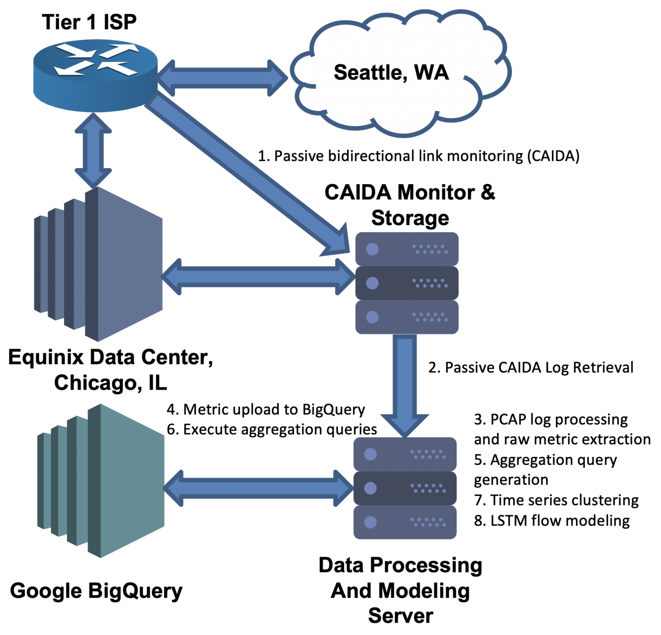

Tier 1 ISP
Seattle, WA
1. Passive bidirectional link monitoring (CAIDA)
CAIDA Monitor & Storage
Equinix Data Center, Chicago, IL
2. Passive CAIDA Log Retrieval
4. Metric upload to BigQuery
6. Execute aggregation queries
3. PCAP log processing and raw metric extraction
5. Aggregation query generation
7. Time series clustering
8. LSTM flow modeling
Google BigQuery
Data Processing And Modeling Server

Fig. 4: Architecture of the data collection and processing pipeline.

In order to evaluate the effectiveness of the model, we calculate the Mean Absolute Percentage Error (MAPE) across all the time-series, as defined below:

$$MAPE = \frac{100}{n} \sum_{t=1}^{n} \frac{|f_i^{(t)} - u_i^{(t)}|}{|f_i^{(t)}|} \tag{7}$$

where $f_i^{(t)}$ is the actual flow-rate value and $u_i^{(t)}$ the estimated flow-rate value for a given flow i. Since MAPE can produce large errors when the actual time-series values are close to 0, we remove from the analysis time-series that appear to be 0 for more than 50% of the trace. In a real system, we can use a framework like DeepFlow from [11] to detect active flows and model those only.

We implemented the following four variations of LSTM:

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, 20 training epochs (not to be confused with the aggregation epoch used during the dataset creation), batch size 8, 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.
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 deltas.
3) **Cluster LSTM (clstm):** This is an LSTM architecture that consists of 20 individual LSTM models (equal to the number of clusters) that are trained using data from a given cluster that the time-series are grouped into. Each model has an LSTM layer with 50 units, followed by a dense layer with 50 units, dropout of 30%, look back window 3, 20 training epochs, batch size 128, and standard scaler on the time-series data.
4) **Cluster Delta LSTM (cdlstm):** This is exactly the same architecture as in 3) above, with the only difference that

the input data have been pre-processed to calculate the deltas.

In addition to the above models, some more variations were also tested (e.g. using log-transform) but are not included here since they did not provide any better results.

The data are aggregated using epochs of (5, 10, 15, 30) seconds, as well as subnet mask sizes of (0, 1, ..., 7) in all the possible combinations of the two. Due to the large number of the resulting active flows, we randomly sample up to 100 active flows per combination, run the models, and then repeat the process 10 times to calculate the final MAPE averages. The results are shown in Figs. 5, and 6. As we can see from Fig. 5, smaller mask sizes and higher epoch sizes can be predicted more easily, however, even at larger mask sizes, the average MAPE can be less than 30%. In addition, it is important to mention here that this does not mean that LSTM cannot predict more fine grained flows, since our experimentation showed that this increase in the MAPE is due to new traffic patterns being observed during the testing phase, something that can be easily resolved if we use more historical data. Another interesting observation is that the delta transformation does not give a big improvement over vanilla LSTM in most of the cases, however when it gives, it gives by quite a lot, as it is evident in Fig. 6. This shows that there is a certain pattern that is better captured by the delta transformation. Fig. 6 also suggests that each cluster can be further optimized by assigning a separate model architecture instead of using the same architecture with different training data across clusters. Finally, Fig. 6 shows that when all the individual time-series have been grouped into similarity groups, the model generalizes better since more data with similar patterns are taken into account.

V. 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 [1], [2], [3]. These models are traditionally good for coarse grain traffic matrix predictions, since they leverage long-range dependencies in order to predict how the overall volume seen by an observation point will behave in the future. On the other hand, there have been some efforts on modeling aggregated flow-sizes in shorter time scales, such as [4], [5], [6]. A in-depth discussion of all the previous work on the topic is out of the scope of this paper. Here, we emphasize on the main differences of the existing approaches with our framework and motivate the need of a new effort to accurately model fine grained flows in short time scales. Specifically, most of the prior research was done more than 2 decades ago, with the flow datasets being significantly different compared to now due the significantly lower network speeds, the limited amount of multimedia traffic (e.g. video, VOIP, etc.), and the different traffic dynamics overall. Second, in the case of more recent examples such as [9], only aggregated traffic at the link (port) level was modeled in large timescales (i.e. 15 minutes), which differs significantly from what our framework aims to model. For this reason, in this work we follow a different approach, and inspired by the state-of-the-art deep learning models for time-series predictions [8], [12], we proceed to validate their effectiveness for predicting network traffic at small time-scales.

VI. Conclusion And Future Work

In this paper, we presented several variations of LSTM that can effectively model backbone network traffic. In addition, we presented a big data architecture where analysis like this can be conducted at scale by processing large PCAP files containing packet level network logs. The results obtained look very promising and validate the hypothesis that LSTM is a good candidate for network traffic modeling. In the near future, we are planning to further investigate this possibility by optimizing more the models used by each cluster, as well as trying different data transformations, neural network architectures, and error metrics.

References

[1] 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.

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

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

[4] 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.

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

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

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

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

[9] A. Azzouni and G. Pujolle, ”A Long Short-Term Memory Recurrent Neural Network Framework for Network Traffic Matrix Prediction”, CoRR abs/1705.05690, June 2017.

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

[11] 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.

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

[13] R. Xu and D. Wunsch Ii, ”Survey of Clustering Algorithms,” IEEE Trans. NEURAL NETWORKS, vol. 16, no. 3, 2005.

[14] Augustin Soule, Kavé 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.

[15] 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.

[16] 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.

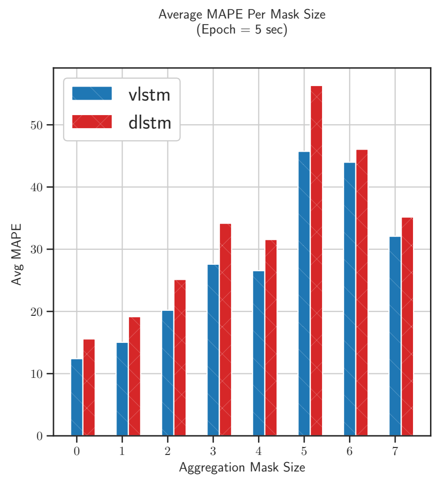
[17] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. 2011. “DevoFlow: scaling flow management for high-performance networks”. SIGCOMM Comput. Commun. Rev. 41, 4 (August 2011), 254-265.

[18] Cristian Estan and George Varghese. 2002. “New directions in traffic measurement and accounting”. SIGCOMM Comput. Commun. Rev. 32, 4 (August 2002), 323-336.

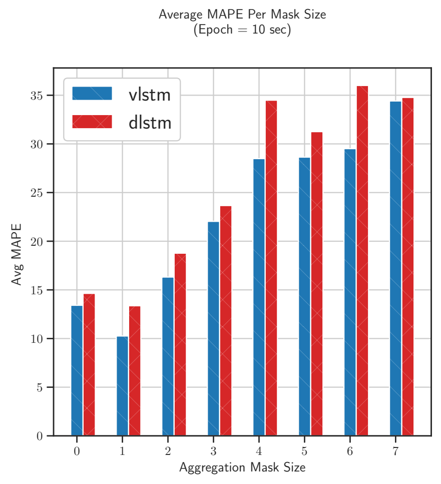
[19] A. Yassine, H. Rahimi and S. Shirmohammadi. 2015. “Software defined network traffic measurement: Current trends and challenges,” in IEEE Instrumentation & Measurement Magazine, vol. 18, no. 2 (April 2015), 42-50.

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

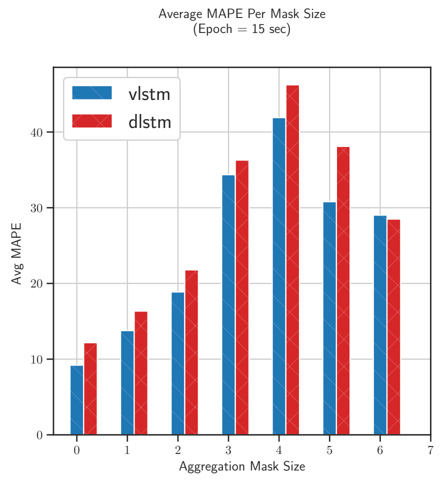
[21] Google BigQuery cloud data warehouse, https://cloud.google.com/bigquery/



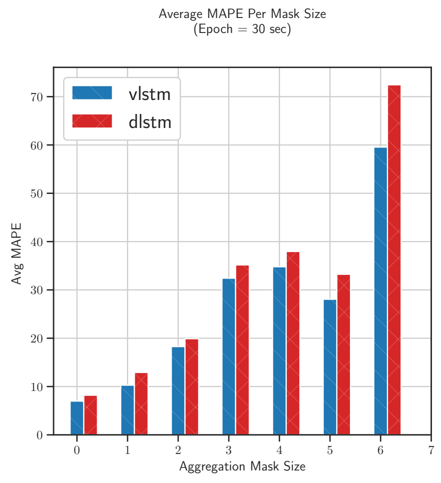
(a) Average MAPE for various mask sizes for VLSTM, and DLSTM, for epoch = 5 sec.



(b) Average MAPE for various mask sizes for VLSTM, and DLSTM, for epoch = 10 sec.



(c) Average MAPE for various mask sizes for VLSTM, and DLSTM, for epoch = 15 sec.



(d) Average MAPE for various mask sizes for VLSTM, and DLSTM, for epoch = 30 sec.

Fig. 5: Average MAPE for various mask sizes and epoch durations.

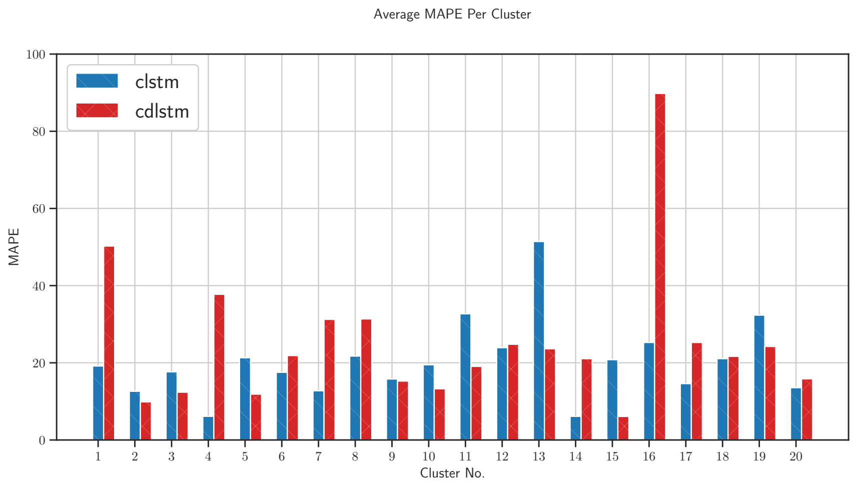


Fig. 6: Average MAPEs for each of the 20 clusters used for CLSTM and CDLSTM.