Received: 17 November 2019 | Revised: 8 January 2020 | Accepted: 9 January 2020 | Published on: 12 February 2020

DOI: 10.1002/itl2.147

LETTER

WILEY

Comparison of HMM and RNN models for network traffic modeling

Radion Bikmukhamedov[1] | **Adel Nadeev**[1] | **Guido Maione**[2] | **Domenico Striccoli**[2]

[1]Radio-Electronic and Telecommunication Systems Department, Kazan National Research Technical University named after A.N.Tupolev, Kazan, Russia

[2]Department of Electrical and Information Engineering, Politecnico di Bari, Bari, Italy

Correspondence
Radion Bikmukhamedov, KNRTU-KAI, 10, K.Marx St., Kazan, Tatarstan 420111, Russia.
Email: radion.bikmukhamedov@pm.me

Major applications for statistical modeling of network traffic flows can be found in network testing and imitating of unavailable devices. Since packet-level modeling is considered, packet size (PS) and inter-arrival time (IAT) features are sufficient for accurate statistics. Two models are compared based on the hidden Markov model (HMM) framework and a recurrent neural network (RNN). In the RNN model, the feature space is encoded with latent components of a Gaussian mixture model (GMM). The comparison is carried out with a voice Skype call and traffic of an IoT device, and evaluated with the rolling entropy and Kulback-Leibler divergence (KLD) metrics that are derived from the generated PS and IAT parameters. The results show that the RNN is applicable for the packet-level modeling task, but it underperforms the HMM.

KEYWORDS

computer networks, hidden Markov model (HMM), recurrent neural networks (RNNs), traffic modeling

1 | INTRODUCTION

The evolution of computing power, new network devices, such as Internet of Things (IoT), and consequently unseen traffic patterns stimulate research on traffic modeling approaches that were inaccessible or inapplicable before. Network models can be broadly classified as:

1. Packet-level models. In this case, the objective is to provide the best possible statistical description, while the computational complexity is not so important. The approach finds its use-cases in modeling of expensive/inaccessible devices or when testing networks nodes that are sensitive to traffic properties, for example, load balancing gateways, firewalls, Deep Packet Inspection (DPI) systems.[1]
2. Aggregate traffic models. This approach aims at describing traffic patterns at macro level, for example, gateway devices, backbone links. Differently from the former models, a fine-granular description of end-devices can be replaced with aggregate statistics due to the law of large numbers and central limit theorem, thus simplifying models and making them scalable.[2] The models are often used within traffic generators in network stress-testing applications.

This paper considers modeling of distinct devices and flows at the packet level. Basically, one of the simplest approaches to handle the task is sampling from two independent, arbitrary distributions (including multi-modal) describing packet size (PS) and inter-arrival time (IAT) between packets.[3] Currently, this approach is one of the most popular solutions due to the simplicity, and finds its applications in existing traffic generators, for instance, D-ITG,[4] Ostinato,[5] T-Rex.[6] Basically, the approach can also serve as a basis for aggregate models. An example of IAT modeling with a Gaussian Mixture Model (GMM) is provided in.[7] However, relations between PS and IAT parameters are not considered, as well as temporal properties of the packet sequence.

The further development of the idea could be assessing the joint parameter density function, as[8] suggested. Given the complexity of the real traffic, single component models provide too coarse estimation, thus, an intuitive solution is approximating the distribution with multi-component models, that is, mixture distributions.[9] Even though such models can capture interrelationship between the parameters, sampled packets will have random properties in the time domain.

To generate and predict time sequences, several families of algorithms can be applied, among which the most common used are Markov models, auto-regressive moving-average (ARMA) and neural networks (NN).

The authors of[10] summarized experience of using ARMA models in traffic modeling and introduced a modeling framework that considers joint distributions of PS and IAT, auto- and cross-correlation properties, demonstrating a good approximation to real traffic. However, as the authors noted, the model cannot properly handle parameters of Gaussian or of a mixture nature.

Markov models appear to be used more extensively for traffic modeling than the regressive models. For example,[11] presented a framework for IoT traffic modeling that is based on Markov chains, where the chain represents different states of IoT devices. The authors of[12] used a first-order hidden Markov model (HMM) for flow-modeling task, where the latent components are described via Beta distributions. According to the results, the model showed its applicability for the task, and the performance could be further adjusted by tuning the number of latent components. Moreover, the HMM framework has practical difficulties to utilize more than one previous latent state[13] that limits its potential for the modeling task.

Taking into account the above-mentioned issues, this paper aims at studying the impact of the replacement of the HMM transition mechanism with a RNN[14] on the traffic model performance. This study is motivated by the analysis of the current literature on this topic, where hybrid approaches combining HMM and NN show improved performance, especially in speech recognition applications.[15] Moreover, despite the presence of numerous papers with applications of NN to network traffic forecasting,[16,17] as far as the authors are aware, there exists no publication applying a NN to model network flows or device generated traffic. The forecasting task aims at predicting future network load by the observed data flows, as opposed to modeling that recreates or mimics the load itself. The two problems are completely different and the tools used for forecasting and modeling are hardly comparable, although based on the same technique (NN).

Thus, the goals of this work are (a) to research applicability of the offered approach for network traffic modeling at the level of a single flow and device and (b) to compare the resulting model with the conventional HMM-based approach.

2 | BACKGROUND

2.1 | HMM

As it was mentioned before, transitions between latent components, encoding the PS/IAT parameter space, can be characterized by a Markov process. Usually, the information regarding observable traffic states is unavailable, and inferences can be made from parameter observations. Thus, it is necessary to estimate the model in a probabilistic manner resorting to the HMM framework.

The formal description of a HMM in the traffic modeling context can be presented as follows.[18] The set of hidden states (latent components) is $\mathbf{s} = \{S_1, S_2, \dots, S_N\}$, where N is the number of possible states and parameter observations can be seen as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M\}$, where $\mathbf{x}_m$ is a vector of features with $M = 2$, since only PS and IAT are considered.

The joint distribution of states is:

$$p(s_1, \dots, s_T) = \prod_{t=1}^{T} p(s_t | s_1, \dots, s_{t-1}) \approx p(s_1) \prod_{t=2}^{T} p(s_t | s_{t-1}), \tag{1}$$

where T is the total number of observations (eg, packets) and actual state at time t is denoted as s_t. Equation (1) can be expressed via the second order transition array $\mathbf{A}$ (shown below), which for a HMM of the T^{th} order will have $T+1$ dimensions.[13] The main difficulty when dealing with a high order $\mathbf{A}$ is its sparsity, even when the training dataset is large. Therefore, similarly to,[12] the current state is assumed to depend only on the previous one. Thus, the transition array can be viewed as a $N \times N$ matrix:

$$\mathbf{A} = \{p(s_t = S_j | s_{t-1} = S_i), \quad i,j = \overline{1,N}\}. \tag{2}$$

Symbol emission probability depends on the state as follows: $p(\mathbf{x}_1, \dots, \mathbf{x}_T) = \prod_{t=1}^{T} p(\mathbf{x}_t | s_t)$. Thus, the complete description of emission probabilities via matrix $\mathbf{B} \in N \times M$ (for Multinoulli distributions) is:

$$\mathbf{B} = \{p(\mathbf{x}_j | S_i), \quad i = \overline{1,N}, \quad j = \overline{1,M}\}. \tag{3}$$

In this paper, the states are represented as *multivariate* Gaussian distributions due to the two features.

In addition, the distribution of initial states can be configured via vector $\boldsymbol{\pi} = \{p(s_1 = S_i), \quad i = \overline{1, N}\}$.

Thus, a HMM can be fully described via the parameter set $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$. In general, there are three main tasks the HMM framework solves: (a) estimation of probability for the observed sequence given λ, (b) estimation of the optimal state sequence given the observations and λ, (c) estimation of λ via the Baum-Welch algorithm[18] given the observations. Herein, the focus is on the third task.

In fact, a HMM differs from a mixture model in one aspect: the former contains the state transition matrix **A**. Therefore, by knowing **B** and $\boldsymbol{\pi}$, matrix **A** can be retrieved either by the normalized count of state transitions similarly to Eqn. (2), or by execution of the Baum-Welch algorithm with pre-trained **B**, $\boldsymbol{\pi}$ parameters.

2.2 | Recurrent neural network (RNN) model

As it was mentioned before, RNNs are used in various data-modeling tasks, such as generating discrete sequences (eg, text), and forecasting continuous variables.

A RNN can be viewed as an extension to a feedforward NN, where the output of a hidden state depends on the previous time, thus allowing to take into account the sequential nature of data. A usual approach to generate sequences with a recurrent NN is to treat continuous value prediction (generation) as a classification task, where the predicted "class" belongs to the discrete set of values.[19] The update for a hidden state of a plain RNN unit at time t is represented as follows:

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{e}_t + \mathbf{U}\mathbf{h}_{t-1}), \tag{4}$$

where **W**, **U** are the layer's weight matrices (inferred during the training stage), $\mathbf{e}_t \in N$ is one-hot encoded state s_t, g is an activation function.

The joint state distribution compares to HMM's Eqn. (1) as: $p(s_1, \dots, s_T) = \prod_{t=1}^{T} g(\mathbf{h}_t)$.

The number of inputs and outputs in the network is N to support the classification approach mentioned above. For this work, Gated Recurrent Units (GRU) are selected as the main building blocks, since this cell type was shown to be the most efficient for modeling of long-range dependencies in sequences.[20]

The output layer is usually a dense feed-forward layer with the *softmax* activation function that produces probabilistic predictions for each latent state, that is, probability mass function (PMF). The output $\mathbf{p}(\mathbf{e}_t^{pred})$ allows to apply various strategies for sequence generating, for example, by increasing or decreasing the entropy of the PMF via the *softmax temperature* parameter Q,[19] the prediction can be made more or less random, correspondingly:

$$\mathbf{p}_{mod}(\mathbf{e}_t^{pred}) = \frac{\exp(\ln(\mathbf{p}(\mathbf{e}_t^{pred}))/Q)}{\|\exp(\ln(\mathbf{p}(\mathbf{e}_t^{pred}))/Q)\|_1}. \tag{5}$$

3 | PROPOSED MODELING PROCEDURE

Since PS and IAT are continuous features, a quantization technique is required, however, a plain mapping to a discrete feature space will lead either to significant size of the NN or information loss. To cope with the problem, the feature space can be quantized with a generative GMM. The approach allows to map a continuous two-dimensional feature space to N ($\leq$20 in this work) discrete numbers, that is, latent components. Moreover, the number can be selected in an automated fashion by the minimum of Bayesian information criteria (BIC).[21] In this way, the required number of latent components is kept low, but sufficient enough to minimize the difference between the original and modeled feature spaces. The trained mixture model parameters (**B**, $\boldsymbol{\pi}$) allow to map a sequence of arbitrary packet features to a sequence of latent component numbers.

Essentially, the first-order state-transition mechanism from a HMM is replaced with an RNN. So the next step is to prepare training data for a NN, where the latter sequence has to be transformed to a tensor $\mathcal{S} \in K \times W \times N$, where K is the number of sequence batches, W is the window size for a sequence, N is the number of latent components, which are one-hot encoded to form the vector $\mathbf{e}_t$. Thus, the encoding transforms the source vector to a matrix, which is further rolled through the window and chunked to the tensor.

As for the network architecture, the recurrent layers consist of GRU blocks, where hidden layers have the same size as the outer ones, and their number ranges from 1 to 2.

The *categorical cross-entropy* training loss function is a common choice for classification problems, thus, it is selected for this work. The stop threshold for the training procedure is defined empirically, based on the performance on the validation dataset.

A sequence is required to instantiate the prediction process of the trained model, which can be an arbitrary k^{th} batch $\mathcal{S}_{k,:,:}$ from the training tensor. The predicted state becomes the first element of the sequence, while the last one is removed, thus, eventually replacing the old states with the new ones in initialization sequences. The process is repeated until a sufficient number of latent states is generated, from which the packet parameters will be further restored.

It was observed that the model's performance heavily depends on the training configuration. The RNN model has many degrees of freedom, some of which include: window size for the state tensor, stop threshold for training, entropy of the output PMF. To simplify the model synthesis procedure, a heuristic was developed in this work to adjust the softmax temperature, which is the inverse of the initialization sequence's entropy:

$$Q = -\frac{1}{\sum_{i=1}^{N} p(\mathcal{S}_{k,:,i} = 1) \log p(\mathcal{S}_{k,:,i} = 1)}, \tag{6}$$

where $p(\mathcal{S}_{k,:,i} = 1)$ is the probability of the i^{th} latent state within the sequence.

Overall, the modeling procedure is split into the following steps:

1. Feature extraction from the source traffic recording.
2. Feature quantization by the latent components of a GMM.
3. Model training with the state sequences. In case of the HMM, it corresponds to inferring **A**, while **B**, $\boldsymbol{\pi}$ are taken from the GMM. As for RNN, the model is trained given tensor $\mathcal{S}$ on the input and its shifted by one time-step version on the output.
4. Sequence generation by the trained models.
5. Parameter sampling from **B** to restore the packet feature space.

It is important to make sure the HMM and RNN models have the same **B**, $\boldsymbol{\pi}$ parameters, in order to provide a common ground for their comparison, since Baum-Welch/Expectation-Maximization (EM) algorithms do not guarantee converging to the global minimum, often producing various local optima on the same data, that is, different descriptions of the components.

The implementation of the model is available in.[22]

4 | MODEL EVALUATION

To evaluate the models, two kinds of traffic are considered in this work:

1. a User Datagram Protocol (UDP) flow containing a Skype voice call between two mobile terminals with 1 minute duration.
2. a bi-directional traffic recording (20 hours) from an Amazon Echo device, as an example of more complex modeling patterns with long-term dependencies.

Traffic is processed and modeled in each direction independently: (a) the PS (payload size) and IAT features of the target device/flow are extracted, where packets without payload are dropped (eg, TCP-ACK messages); (b) packets with at least one feature below the first and above the 99*th* percentiles are considered the outliers and filtered as well.

To assess the models' effectiveness quantitatively the generated traffic is evaluated and compared with the source via the following performance measures:

1. *Kulback-Leibler divergence* (KLD), which allows to measure the distance between the source and generated distributions for one feature at a time. The result is a-dimensional with values within the (0, 1) range.
2. *Rolling entropy*, which evaluates how "random" a feature is within the time-window. For instance, the entropy of a generated feature is high when it changes rapidly over the time, and vice versa, for constant values the entropy is zero. The metric allows monitoring the model's fidelity and accuracy during traffic generating, not only at the final point as the KLD does. In this work, the entropy is calculated every 50 packets.

The models were trained and evaluated for the two scenarios independently.

4.1 | Skype voice call

To select the W parameter for an RNN, it is important to estimate temporal properties of state sequences. Hence, setting the parameter lower than the auto-correlation lag of a state vector may not allow the model catch its linear dependency. According to the auto-correlation function of the state vector (not shown here), the sequence does not have significant

TABLE 1 The Kulback-Leibler divergence (KLD) metric

		Skype voice call		Amazon Echo	
Model	**Feature**	**From**	**To**	**From**	**To**
HMM	IAT	0.0056	0.0115	0.022	0.013
	PS	0.0147	0.0153	0.0024	0.001
RNN	IAT	0.0248	0.013	0.0374	0.02
	PS	0.0329	0.0096	0.0905	0.0288

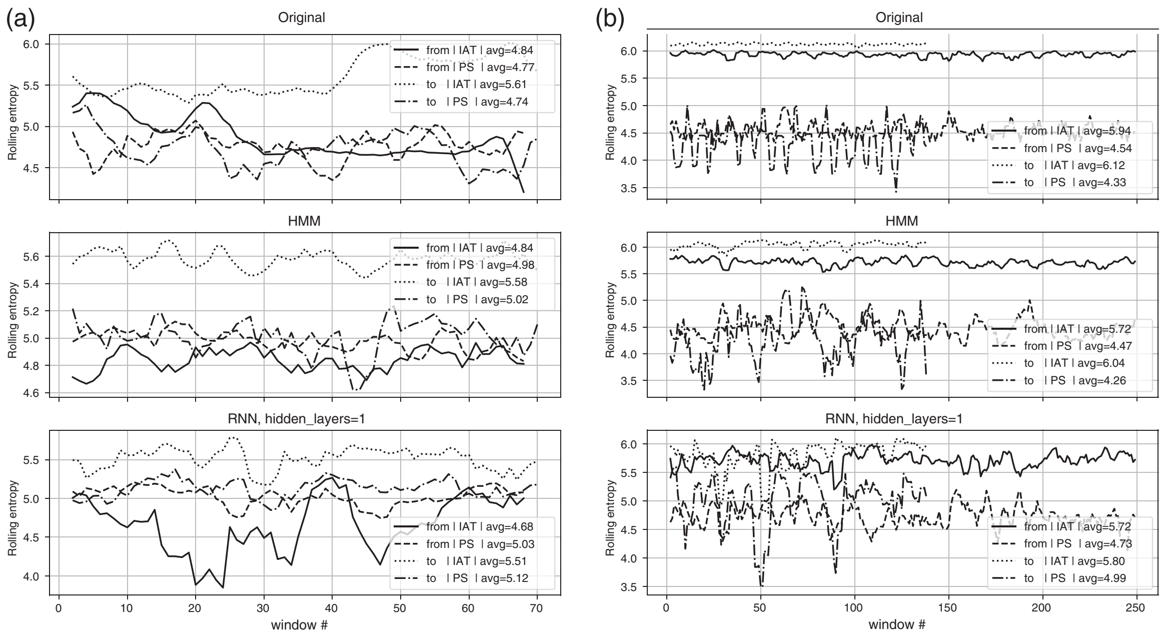

FIGURE 1 The rolling entropy for a) Skype voice call and b) Amazon Echo recording

temporal linear dependencies. Also, some experiments with $W = (20\ldots 300)$ did not reveal the influence on the results (not shown here), thus $W = 200$ was arbitrarily selected.

According to the KLD metric, the models closely approached the source distributions of PS/IAT, yielding to the distance on average <0.02, however, the RNN handled traffic *from* the source slightly worse (see Table 1).

As for the rolling entropy, the HMM and RNN models showed similar averaged values (see Figure 1), with the randomness for the PS parameter up to 15% above the expected, contrasting to the RNN's IAT case.

4.2 | Amazon Echo

On contrary to the voice traffic case, the state sequences of Amazon Echo have an emphasized temporal pattern that corresponds to periodic receiving sessions by the device (not shown for the sake of brevity). Due to the pattern, $W = 50$ was selected for the RNN, with the window covering the first two auto-correlation spikes.

The KLD metric shows that the RNN had worse performance in comparison with the HMM (see Table 1), especially the traffic *from* the device that diverged by ~0.06 from the original, as compared with ~0.025 for the HMM. One of the reasons is the difficult training procedure leading to a model that generates state sequences with distributions quite distant from the expected.

The rolling entropy metric suggests the same results as above (see Figure 1). As for the source traffic, the metric resembles a stationary process due to the traffic nature, where the HMM-derived traffic had overall similar values to the source, except for the PS destined to the device. Finally, the RNNs produced features with a weaker pattern, especially when modeling traffic *to* the device.

5 | CONCLUSIONS

The paper offered to model network traffic with a modified version of the HMM framework, such that the state transition mechanism was substituted by a RNN. The comparison with the reference HMM was provided by modeling a Skype voice call session and the traffic of an IoT device - Amazon Echo. The two approaches demonstrated their suitability for flow and device modeling tasks according to objective metrics, namely, the KLD and the rolling entropy.

Thus, the PS and IAT distributions produced by the HMM models diverged from the source not more than 0.02, whereas the RNNs demonstrated the 0.04 difference on average. Moreover, the rolling entropy had a better stability for the first model as well.

Nevertheless, the RNNs are capable of reproducing statistically similar traffic, given a proper training and model selection procedures, since the performance heavily depends on patterns of the traffic being modeled. However, a HMM may be a preferred choice due to fewer degrees of freedom and more robust performance.

ORCID

Radion Bikmukhamedov https://orcid.org/0000-0002-3755-3309

REFERENCES

1. Molnár S, Megyesi P, Szabó G. How to validate traffic generators?. In: *2013 IEEE International Conference on Communications Workshops (ICC)*. 2013:1340-1344.
2. Hoßfeld T, Metzger F, Heegaard PE. Traffic modeling for aggregated periodic IoT data. In: *Proc. Internet and Networks and Workshops (ICIN) 2018 21st Conf. Innovation in Clouds*. 2018:1-8.
3. Wu Y, Yu W, Griffith D, Golmie N. Modeling and performance assessment of dynamic rate adaptation for M2M communications. *IEEE Trans Network Sci Eng*. 2018; 1: 1–16.
4. Botta A, Dainotti A, Pescapè A. A tool for the generation of realistic network workload for emerging networking scenarios. *Comput Networks*. 2012;56(15):3531-3547.
5. Ostinato Packet Generator. https://ostinato.org. Accessed January 4, 2020.
6. TRex. https://trex-tgn.cisco.com. Accessed January 4, 2020.
7. Grigoreva E, Laurer M, Vilgelm M, Gehrsitz T, Kellerer W. Coupled Markovian arrival process for automotive machine type communication traffic modeling. In: *Proc IEEE International Conference Communications (ICC)*. 2017:1-6.
8. Salvador P, Pacheco A, Valadas R. Modeling IP traffic: joint characterization of packet arrivals and packet sizes using BMAPs. *Comput Networks*. 2004;44(3):335-352.
9. Luo S, Marin GA. Realistic internet traffic simulation through mixture modeling and a case study. In: *Proceedings of the Winter Simulation Conference*. 2005:9.
10. Laner M, Svoboda P, Rupp M. Parsimonious network traffic modeling by transformed ARMA models. *IEEE Access*. 2014;2:40-55.
11. Nikaein N, Laner M, Zhou K, Svoboda P, Drajic D, Popovic M, Krco S. Simple traffic modeling framework for machine type communication. In: *Proc. ISWCS 2013; the Tenth Int. Symp. Wireless Communication Systems*. 2013:1-5.
12. Dainotti A, Pescapé A, Rossi PS, Palmieri F, Ventre G. Internet traffic modeling by means of Hidden Markov Models. *Comput Networks*. 2008;52(14):2645-2662.
13. Thede SM, Harper MP. A Second-order hidden Markov model for part-of-speech tagging. In: *Proc. of the 37th Meeting of the Association for Comp. Linguistics*. 2002.
14. Bourlard H, Wellekens CJ. Links between Markov models and multilayer perceptrons. *IEEE Trans Pattern Anal Mach Intell*. 1990;12(12):1167-1178.
15. Renals S, Morgan N, Bourlard H, Cohen M, Franco H. Connectionist probability estimators in HMM speech recognition. *IEEE Trans Speech Audio Process*. 1994;2(1):161-174.
16. Madan R, Mangipudi PS. Predicting computer network traffic: a time series forecasting approach using DWT, ARIMA and RNN. In: *2018 Eleventh International Conference on Contemporary Computing (IC3)*. IEEE; 2018:1-5.
17. Chen Z, Hu J, Min G, Zomaya A, El-Ghazawi T. Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning. *IEEE Trans Parallel Distrib Syst*. 2019; 31: 1-1.
18. Rabiner L. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc IEEE*. 1989;77:257-286.
19. Chollet F. Deep Learning with Python. Manning Publications Company 2017.
20. Chung J, Gulcehre C, Cho KH, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv: 1412.3555 [cs.NE]*. 2014. https://arxiv.org/abs/1412.3555.
21. Claeskens G, Hjort NL. *Model Selection and Model Averaging*. 1st ed. Cambridge University Press, Cambridge, UK; 2008.
22. Bikmukhamedov R. Statistical estimator of network traffic. https://github.com/RadionBik/Statistical-estimator-of-network-traffic. Accessed January 4, 2020.

How to cite this article: Bikmukhamedov R, Nadeev A, Maione G, Striccoli D. Comparison of HMM and RNN models for network traffic modeling. *Internet Technology Letters*. 2020;3:e147. https://doi.org/10.1002/itl2.147