Performance Evaluation of Network Topologies using Graph-Based Deep Learning

Fabien Geyer
Technical University of Munich
fgeyer@net.in.tum.de

ABSTRACT

Understanding the performance of network protocols and communication networks generally relies on expert knowledge and understanding of the different elements of a network, their configuration and the overall architecture and topology. Machine learning is often proposed as a tool to help modeling complex protocols. One drawback of this method is that high-level features are generally used – which require expert knowledge on the network protocols to be chosen, correctly engineered, and measured – and the approaches are generally limited to a given network topology.

In this paper, we propose a methodology to address the challenge of working with machine learning by using lower-level features, namely only a description of the network architecture. Our main contribution is an approach for applying deep learning on network topologies via the use of Graph Gated Neural Networks, a specialized recurrent neural network for graphs. Our approach enables us to make performance predictions based only on a graph-based representation of network topologies. We apply our approach to the task of predicting the throughput of TCP flows. We evaluate three different traffic models: large file transfers, small file transfers, and a combination of small and large file transfers. Numerical results show that our approach is able to learn the throughput performance of TCP flows with good accuracies larger than 90%, even on larger topologies.

CCS CONCEPTS

• **Networks → Network performance modeling**; *Transport protocols*; • **Computing methodologies → Neural networks**;

KEYWORDS

Network performance evaluation, Graph Neural Network, Deep learning

ACM Reference Format:
Fabien Geyer. 2017. Performance Evaluation of Network Topologies using Graph-Based Deep Learning. In *VALUETOOLS 2017: 11th EAI International Conference on Performance Evaluation Methodologies and Tools, December 5–7, 2017, Venice, Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3150928.3150941

VALUETOOLS 2017, December 5–7, 2017, Venice, Italy

ACM ISBN 978-1-4503-6346-4/17/12…$15.00
https://doi.org/10.1145/3150928.3150941

1 INTRODUCTION

Understanding network performance is an important task for architecture design and Quality-of-Service in an increasing number of applications. Traffic engineering aims at bringing an answer to this need in order to avoid congestion and optimize network topologies to support an increasing number of applications. Network and traffic models are an important tool in order to predict how a given network architecture will behave. Different techniques have been developed for this purpose, such as mathematical modeling, simulations or measurements. While those techniques are usually accurate, they often require precise measurements of key performance indicators such as round-trip time or loss probability in order to be applied and generate realistic performance predictions. However, limited access to instrumentation of real networks make this measurement acquisition usually difficult.

One growing approach to tackle the challenge of performance modeling has been the use of machine learning. For instance, Tian and Liu applied in [27] the SVR-based (Support Vector Regression) TCP bandwidth prediction application from [19] to improve Quality-of-Service of media streaming over HTTP. Tariq et al. recently proposed WISE in [26], a framework for evaluating architecture changes in communication networks using Causal Bayesian Networks (CBNs). While those techniques and applications have been proven successful, they require high-level features about the studied network protocols and the trained models are often limited to a given network topology.

Our main contribution in this work is an approach for applying deep learning on a low-level graph-based representation of network topologies in order to predict network and protocol performance using only topology information as input. We propose to use Gated Graph Neural Networks (GG-NNs) [17] as a basis for our deep learning architecture. GG-NNs are a recently developed neural network architecture working on graph-structured inputs. The intuition behind our approach is to map network topologies and flows to graphs, and then train GG-NNs on those graphs. This enables us to avoid the task of engineering high-level protocol-specific input features such as round-trip time or drop probability, which usually require expect knowledge on the network protocol which is modeled. Another contribution in this work is the extension of GG-NNs with an alternative memory cell called LSTM (Long Short Term Memory) [13], which shows better performance than the initial architecture from [17].

As a concrete application of our approach, we address the task of performance evaluation of TCP flows, with the goal of predicting the average throughput of each flow in a given topology. We evaluate our approach against three types of traffic models: large file transfers, small file transfers, and a combination of small and large file transfers. As shown in previous studies about TCP, the average

throughput of TCP flows depends on various parameters such as the TCP version, the configuration of the TCP stack, round-trip times or drop probabilities. We show through a numerical evaluation that our approach is able to predict the average throughput of TCP flows without having direct access to those parameters, but only a low-level graph-based representation of the network topology and its flows. The results of our approach are also compared against a simpler recurrent neural network architecture.

This work is structured as follows. In Section 2, we present similar research studies. We describe in Sections 3 and 4 our modeling approach and the neural network architecture used here, with an introduction on Graph Neural Networks and Graph Gated Neural Networks, followed by the application of those concepts to network topologies and flows. We numerically evaluate our approach in Section 5 with the prediction of average flow throughput on three different use-cases. Finally, Section 6 concludes our work.

2 RELATED WORK

On the challenge of predicting the performance of TCP flows, analytical models have been proposed since the late 1990s. Mathis et al., and subsequently Padhye et al., modeled the throughput of a single flow using TCP Reno in [18] and [20] as a function of round-trip time, drop probability and some configuration parameters of TCP. This work was then extended by Cardwell et al. in [5] to take into account the slow-start phase of TCP. While those models address the mathematical modeling of a single flow, the interaction between multiple flows on a given topology is of greater interest for the problem addressed in this paper. Firoiu et al. proposed in [7] to reuse the results from [20] to achieve this. Those analytical models give great insights in the performance of TCP, but they usually suffer from poor applicability in real-world use-cases due to newer versions of TCP, simplifications of the mathematical model, or lack of modeling of non-intuitive behavior of TCP such as ACK compression or TCP Incast. Some later works have partially addressed those shortcomings, such as the work from Velho et al. in [29] and Geyer et al. in [8].

Adjacent to the mathematical modeling of TCP flows, machine learning methods were also applied to this problem, although less frequently than in other domains such as network intrusion detection. Mirza et al. used Support Vector Regression (SVR) in [19] using input features such as transfer duration of files over TCP and active measurements in order to measure queuing latency, loss probability and available bandwidth. Hours et al. used Causal Bayesian Networks (CBNs) in [14] to predict the throughput distribution of TCP flows, using similar features than [19]. Both works showed promising results regarding applicability to real-world use-cases, but are mainly specific to a given network topology or the studied protocol.

Various methods have been proposed for applying machine learning to graphs-based structures, either based on a spectral or spatial approach. Spectral approaches [4, 12] are usually based on the Graph Laplacian, an analogue to the Discrete Fourier Transform, which transforms graph signals to a spectral domain. The main limitation of those approaches is that their requires the input graph samples to be homogeneous.

Spatial approaches do not require a homogeneous graph structure, meaning that they can be applied to a broader range of problems. Gori et al. proposed in [10, 23] the Graph Neural Networks (GNNs) architecture, which propagates hidden representations of nodes to its adjacent nodes until a fixed point is reached. GNNs were applied on different tasks such as object localization, ranking of web pages, document mining, or prediction of graph properties [22]. This neural network architecture was subsequently refined in different works. Li et al. proposed an extension of GNNs in [16] through application of more modern practice of neural networks, namely by using Gated Recurrent Units (GRU) [6]. GG-NNs were applied to basic logical reasoning tasks and program verification in [16]. Relational Graph Convolutional Networks (R-GCNs) were recently proposed by Schlichtkrull et al. in [24], where hidden state information is also propagated across edges of the graph via convolutions, while taking into account the type and direction of an edge.

More general neural network architectures such as the Differentiable Neural Computer from Graves et al. in [11] were also applied to graph-based problems such as shortest path finding.

To the best of our knowledge, this is the first work on applying graph-based neural networks to the performance evaluation of network topologies and network protocols.

3 NEURAL NETWORKS FOR GRAPHS

The main intuition behind our approach is to map network topologies to graphs, with additional nodes for representing flows and additional edges for the path followed by the flows. Those graph representations are then used as input for a neural network architecture able to process general graphs.

In this section, we review the neural network architecture used for this purpose, namely Graph Neural Networks (GNNs) [10, 23] and one of its recent extension, Gated Graph Neural Networks (GG-NNs) [17]. We also introduce notation and concepts that will be used throughout this paper. The transformation between network topology and its graph representation will be detailed later in Section 4.

GNNs and GG-NNs are a general neural network architecture able to process graph structures as input. They are an extension of recursive neural networks which work by assigning hidden states to each node in a graph based on the hidden states of adjacent nodes. For the purpose of this work, our description of GNNs and GG-NNs is limited to undirected graphs. The concepts presented here can also be applied to directed graph, as presented in the original works on GNNs and GG-NNs [10, 17, 23].

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with nodes $v \in \mathcal{V}$ and edges $e \in \mathcal{E}$. Edges can be represented as pairs of nodes, such that $e = (v, v') \in \mathcal{V} \times \mathcal{V}$. The *hidden representation* for node v is denoted by the vector $\mathbf{h}_v \in \mathbb{R}^{\mathcal{D}}$. Nodes may also have features $l_v \in \{1, \dots, L_{\mathcal{V}}\}$ for each node v, and edges also $l_e \in \{1, \dots, L_{\mathcal{E}}\}$ for each edge e. Let $\textsc{Nbr}(v)$ denote the set of neighboring nodes of v.

3.1 Graph Neural Networks

In Graph Neural Networks (GNNs), each hidden representation $\mathbf{h}_v$ of a node v is based on the hidden state of its neighboring nodes. The following propagation model is used for expressing this

relationship:

$$\mathbf{h}_v^{(t)} = f^* \left(l_v, l_{\text{NBR}(v)}, \mathbf{h}_{\text{NBR}(v)}^{(t-1)} \right) \quad (1)$$

An example application of Equation (1) is given in Figure 1.

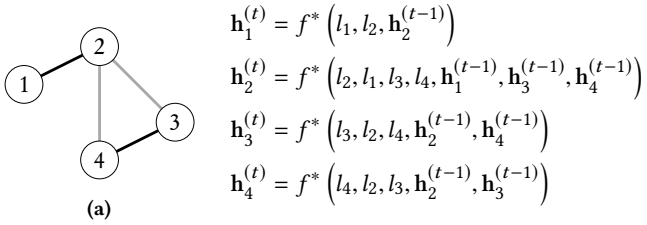

Figure 1: (a) Example graph. Edge colors denote edge types. (b) Application of Equation (1) to the graph.

As a concrete implementation, [23] recommends to decompose $f^*(\cdot)$ as the sum of per-edge terms such that:

$$\mathbf{h}_v^{(t)} = \sum_{v' \in \text{NBR}} f \left(l_v, l_{v'}, l_{(v,v')}, \mathbf{h}_{v'}^{(t-1)} \right) \quad (2)$$

with $f(\cdot)$ a linear function of $\mathbf{h}_v$ or a feed-forward neural network. For example, $f(\cdot)$ can be formulated as a linear function:

$$f \left(l_v, l_{v'}, l_{(v,v')}, \mathbf{h}_{v'}^{(t)} \right) = \mathbf{A}^{(l_v, l_{v'}, l_{(v,v')})} \mathbf{h}_{v'}^{(t-1)} + \mathbf{b}^{(l_v, l_{v'}, l_{(v,v')})} \quad (3)$$

with $\mathbf{A}$ and $\mathbf{b}$ learnable weight and bias parameters. The hidden node representations are propagated throughout the graph until a fixed point is reached. As explained in [23], it implies that $f(\cdot)$ has the property that a fixed point for Equation (2) can be reached.

Once a fixed point $\mathbf{h}_v$ has been reached, a second model is then used to compute the output label o_v for each node $v \in \mathcal{V}$

$$o_v = g \left(\mathbf{h}_v, l_v \right) \quad (4)$$

Practically, $g(\cdot)$ is implemented using a feed-forward neural network. The neural network architecture is differentiable from end-to-end, so that all parameters can be learned using gradient-based optimization.

Learning of the parameters of $f(\cdot)$ and $g(\cdot)$ is done via the Almeida-Pineda algorithm [3, 21] which works by running the propagation of the hidden representation to convergence, and then computing gradients based upon the converged solution.

3.2 Gated Graph Neural Networks

Gated Graph Neural Networks (GG-NNs) [17] are an recent extension of GNNs using more recent neural network techniques, based on Gated Recurrent Units (GRU) [6]. In GG-NNs, each node aggregates the hidden representations it receives from all adjacent nodes, and uses that to update its own hidden representation using a GRU cell. More specifically, the propagation of the hidden representations among neighboring nodes for one time-step is formulated as:

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{(v)} \left[\mathbf{h}_1^{(t-1)} \cdots \mathbf{h}_{|V|}^{(t-1)} \right]^T + \mathbf{b}_a \quad (5)$$

$$\mathbf{z}_v^{(t)} = \sigma \left(\mathbf{W}_z a_v^{(t)} + \mathbf{U}_z \mathbf{h}_v^{(t-1)} + \mathbf{b}_z \right) \quad (6)$$

$$\mathbf{r}_v^{(t)} = \sigma \left(\mathbf{W}_r a_v^{(t)} + \mathbf{U}_r \mathbf{h}_v^{(t-1)} + \mathbf{b}_r \right) \quad (7)$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh \left(\mathbf{W} a_v^{(t)} + \mathbf{U} \left(\mathbf{r}_v^{(t)} \odot \mathbf{h}_v^{(t-1)} \right) + \mathbf{b} \right) \quad (8)$$

$$\mathbf{h}_v^{(t)} = \left(1 - \mathbf{z}_v^{(t)} \right) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^{(t)} \odot \widetilde{\mathbf{h}_v^{(t)}} \quad (9)$$

where $\sigma(x) = 1/(1+e^{-x})$ is the logistic sigmoid function and $\odot$ is the element-wise matrix multiplication. $\{\mathbf{W}_z, \mathbf{W}_r, \mathbf{W}\}$ and $\{\mathbf{U}_z, \mathbf{U}_r, \mathbf{U}\}$ are learnable weights matrices, and $\{\mathbf{b}_a, \mathbf{b}_r, \mathbf{b}_z, \mathbf{b}\}$ are learnable biases vectors. $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is a matrix determining how nodes in the graph $\mathcal{G}$ communicate with each other, as illustrated in Figure 2.

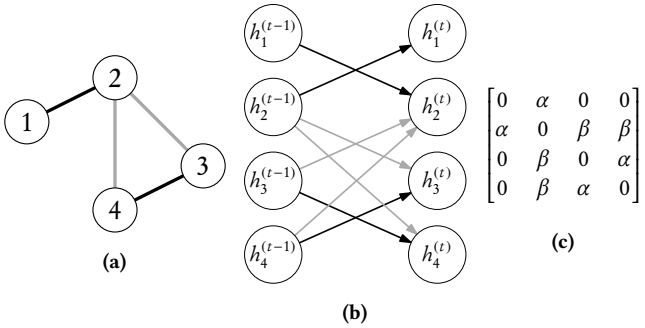

Figure 2: (a) Example graph. Edge colors denote edge types. (b) One time-step unrolling of Equations (5) to (9). (c) Matrix A corresponding to the graph. Parameters α and β encode the edge type.

Equation (5) corresponds to one time-step of the propagation of the hidden representation of neighboring nodes to node v, as formulated previously for Graph Neural Networks in Equations (1) and (2). Equations (6) to (9) correspond to the mathematical formulation of a GRU cell, with Equation (6) representing the GRU reset gate vector, Equation (7) the GRU update gate vector, and Equation (9) the GRU output vector. The initial hidden representation $h_v^{(0)}$ is based on the node's feature vector l_v, padded with zeros according to the dimensions of the hidden representation.

The output vector $\mathbf{o}_v$ for each node v is computed as in Equation (4) using a feed-forward neural network. The overall architecture of the GG-NN is summarized in Figure 3.

Learning of the weight matrices and bias vectors is performed using back-propagation through time in order to compute gradients, namely using standard gradient-based optimization algorithms such as RMSProp [28] or Adam [15].

3.3 GG-LSTM-NN: Extension of Graph Gated Neural Networks with LSTM

We propose in this section a new class of Graph Gated Neural Networks called GG-LSTM-NN based on the Long Short-Term Memory (LSTM) cell [13]. This neural network architecture is a variant of the

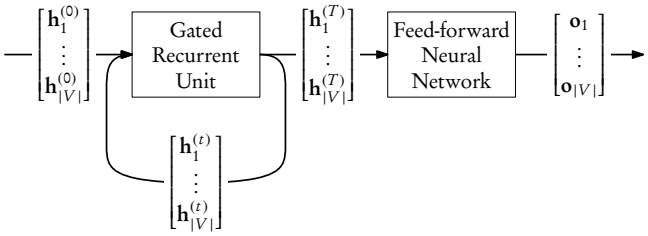

Figure 3: Representation of a Gated Graph Neural Network.

Graph Gated Neural Network architecture where the GRU memory cell is replaced with a LSTM cell. The overall architecture of the GG-LSTM-NN is similar to the GG-NN illustrated in Figure 3.

Similarly to Equations (5) to (9), the propagation of the hidden representations among neighboring nodes for one time-step in a GG-LSTM-NN is formulated as:

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{(v)} \left[\mathbf{h}_1^{(t-1)} \cdots \mathbf{h}_{|V|}^{(t-1)}\right]^T + \mathbf{b}_a \tag{10}$$

$$\mathbf{i}_v^{(t)} = \sigma\left(\mathbf{W}_i a_v^{(t)} + \mathbf{U}_i \mathbf{h}_v^{(t-1)} + \mathbf{b}_i\right) \tag{11}$$

$$\mathbf{f}_v^{(t)} = \sigma\left(\mathbf{W}_f a_v^{(t)} + \mathbf{U}_f \mathbf{h}_v^{(t-1)} + \mathbf{b}_f\right) \tag{12}$$

$$\mathbf{o}_v^{(t)} = \sigma\left(\mathbf{W}_o a_v^{(t)} + \mathbf{U}_o \mathbf{h}_v^{(t-1)} + \mathbf{b}_o\right) \tag{13}$$

$$\mathbf{g}_v^{(t)} = \tanh\left(\mathbf{W}_g a_v^{(t)} + \mathbf{U}_g \mathbf{h}_v^{(t-1)} + \mathbf{b}_g\right) \tag{14}$$

$$\mathbf{c}_v^{(t)} = \mathbf{c}_v^{(t-1)} \odot \mathbf{f}_v^{(t)} + \mathbf{g}_v^{(t)} \odot \mathbf{i}_v^{(t)} \tag{15}$$

$$\mathbf{h}_v^{(t)} = \tanh\left(\mathbf{c}_v^{(t)}\right) \odot \mathbf{o}_v^{(t)} \tag{16}$$

with $\{\mathbf{W}_i, \ldots\}$ and $\{\mathbf{U}_i, \ldots\}$ learnable weight matrices, and $\{\mathbf{b}_i, \ldots\}$ learnable bias vectors.

Equation (10) is the propagation of the hidden representations among neighbors, as in Equation (5). Equations (11) to (13) correspond respectively to the *input*, *forget* and *output* gates of the LSTM cell. Equation (14) is a *candidate* hidden representation, with an initial value $\mathbf{c}_v^{(0)}$ set to zero. Equation (15) is the internal memory of the LSTM cell.

Our motivation for proposing this new neural network architecture is motivated by better numerical performance than the GRU-based GG-NN presented in Section 3.2, as shown in the numerical evaluation in Section 5.

4 APPLICATION TO PERFORMANCE EVALUATIONS OF NETWORKS

We describe in this section the application of the deep learning architectures presented earlier to the performance evaluation of network topologies and network protocols. In other words, our goal is to represent network topologies and their flows as graphs which can be passed as input to a GG-NN. Compared to other works on the application of machine learning to performance evaluation, the main contribution is that this graph representation is a low-level input feature. This means that specific high-level features of the studied network protocol are not required and the trained machine learning algorithm is not restricted to a specific topology.

4.1 Input features definition

The main intuition behind the input feature modeling for the GG-NN is to use the queuing network as input graph $\mathcal{G}$, with additional nodes representing the flows in this network. An illustration of this queuing network is given in Figure 5, which is the queuing representation of the example network illustrated in Figure 4 with one switch or router interconnecting three PCs with three flows. Note that we illustrate on Figure 5 only the forward path of the flows. Figure 5 may also be extended to include the queues taken by the acknowledgement packets used by the flows if necessary, such as TCP ACK packets for example.

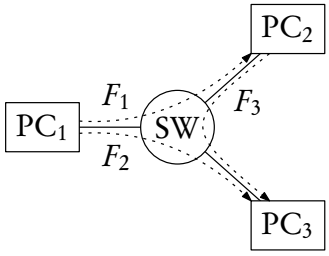

Figure 4: Example network topology with 3 flows.

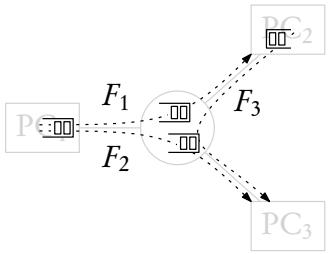

Figure 5: Associated queuing network of Figure 4 (here with only the forward path of each flow).

Regarding the constructed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, the nodes $\mathcal{V}$ correspond to the queues traversed by the flows in the network topology as well as specific nodes representing the flows. For the node features l_v, a vector encoding the node type (*i.e.* if a node represents a flow or a queue) with one-hot encoding is used. Namely l_v is a vector with two values, with $[1, 0]^T$ is for queue nodes, and $[0, 1]^T$ for flow nodes. Note that for simplification purpose, we assume here that every PCs and switches or routers to have the same behavior and all links in the topology to have the same capacity and latency. Additional features for distinguishing between different behaviors or node types may be used in case different configurations, types or link capacities are used. An example of such node-specific feature is given later in Sections 5.3 and 5.4.

Edges connect the queues which are used by the flows according to the physical topology of the network. In order to encode flow routing in the graph, edges between the nodes representing flows and their traversed queues are used. Figure 6 is an example of such graph modeling applied to the topology presented in Figure 4. A labeling of the edge type may be used in order to distinguish

between queues representing the forward path of flows and the path used for acknowledgement packets, as illustrated in Figure 2.

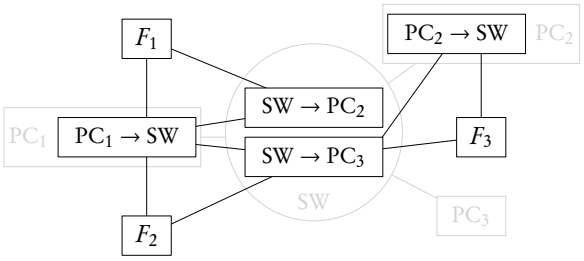

Figure 6: Graph representing the queuing network from Figure 5.

We note that the graph and feature representations described here are independent of any studied network protocols.

4.2 Task-specific modeling

For the scope of this work, we focus on the challenge of evaluating the performance of TCP flows, namely predicting their average throughput. As shown in previous works on TCP Reno such as [18], the average throughput of a TCP flow can be modeled as:

$$\frac{C}{RTT\sqrt{p}} \tag{17}$$

with RTT the round-trip time and p the probability of packet loss on the flow's path, and C a constant value depending on a configuration of the TCP stack. In the case of small file transfers, additional parameters such as file size distributions are also important.

This is an interesting problem since TCP adapts its throughput according to the perceived level of congestion in the network or other factors depending on the version of TCP which is used. Since the flows also contribute themselves to the overall congestion in the network, the sharing of bandwidth at a given bottleneck is not trivial to predict.

Based on this task description, the output vector $\mathbf{o}_v$ of node v will then be the average throughput of the TCP flow for nodes representing flows. Practical experimentations showed that using discretized values across N bins provided better accuracy than using continuous values for modeling the throughputs. Hence the regression task essentially becomes a classification task.

The neural network is then trained against a log softmax cross entropy loss function, as usually done in classification tasks:

$$\mathcal{L}_v = -y_v + \log \sum_{i=1}^{N} e^{\mathbf{o}_{v:i}} \tag{18}$$

where y_v corresponds to the index of the binned average throughput value, and $\mathbf{o}_{v:i}$ to the i-th element of the output vector $\mathbf{o}_v$.

5 NUMERICAL EVALUATION

We present in this section a numerical evaluation of the concepts presented in Sections 3 and 4. We focus here on the evaluation of Ethernet networks with 100 Mbit/s links. The evaluated topologies and flows are randomly generated as follows using [9]. A random number of Ethernet switches is first selected using a uniform distribution and connected in according to a daisy chain as illustrated in Figure 7. A random number of nodes is then generated using a uniform distribution and connected to a randomly selected Ethernet switch. For each node, a TCP flow is generated with a randomly selected destination among the other nodes.

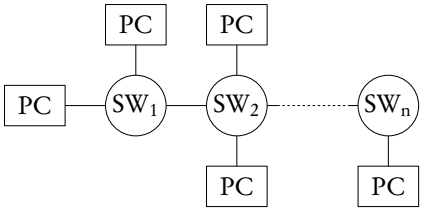

Figure 7: Daisy chain topology used for the numerical evaluation.

In order to build our datasets for learning, each random topology is evaluated using the *ns-2* simulator [1] until the steady-state of the flows throughput is reached. The defaults parameters of *ns-2* for the TCP stack are used, meaning that TCP Reno is used as a congestion control algorithm. The results of the simulations is used as a basis for the learning process of the neural network.

We evaluate our approach against three different traffic use-cases, namely:

(1) Infinite flows, where clients always have data to send, with results presented in Section 5.2;
(2) Finite flows, where clients follow an ON/OFF loop behavior where a random amount of data is sent, followed by a random idle time, with results presented in Section 5.3;
(3) A combination of the two previous traffic models, with results presented in Section 5.4.

5.1 Implementation

The GG-NN architectures presented in Sections 3.2 and 3.3 was implemented using Tensorflow [2]. The recurrent part of the GG-NN and GG-LSTM-NN were respectively implemented according to Equations (5) to (9) and Equations (10) to (16). The function $g(\cdot)$ in Equation (4) was implemented using a feed-forward neural network with two dense layers. Additional dropout layers [25] between each time-step of the GG-NN were added in order to avoid over-fitting.

For each studied use-case, the model was trained multiple times using the parameters listed in Table 1 and different seeds for the random number generators. Randomization of the node indexes was also performed for each mini-batch. The neural network producing the best result was then selected for the numerical results presented in the rest of this section.

As a comparison basis, we also evaluated a simple version of Graph Neural Network from section 3.1, using a simple Recurrent Neural Network (RNN) architecture similar. The hidden node representation is driven by:

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{(v)} \left[\mathbf{h}_1^{(t-1)} \cdots \mathbf{h}_{|V|}^{(t-1)}\right]^T + \mathbf{b}_a \tag{19}$$

$$\mathbf{h}_v^{(t)} = \tanh\left(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\mathbf{h}_v^{(t-1)} + \mathbf{b}\right) \tag{20}$$

Parameter	Value
Learning algorithm	RMSProp [28]
Size of hidden representation	64
Learning rate	10^{-4}
Mini-batch size	32
Training iterations	20 000

Table 1: Parameters used for the training phase of the neural networks.

with **W** and **U** learnable weight matrices, and **b** a learnable bias vector.

Each set of simulations was split into training and test datasets. The training dataset was used for training the neural network, while the test dataset was used for evaluating the prediction performances of the neural network.

5.2 Evaluation on infinite TCP flows

In this first use-case, we assume an infinite traffic model for TCP flows. This model illustrates the case of large file transfers over TCP. Multiple sets of simulations were generated, with different parameters regarding the size of the network, namely the maximum number of switches used for the line topology presented in Figure 7 and the maximum number of flows. Statistics about the dataset with the largest topologies are given in Table 2.

Property	Mean	Min.	Max.	Std. dev.
Number of flows	16.38	2	30	8.21
Number of queues	35.54	4	66	16.75
Number of edges	250.47	20	596	133.10

Table 2: Parameters of the dataset with the largest network topologies.

Numerical results of the average accuracy of the predictions of the trained neural network are presented on Figure 8. As mentioned in Section 4.2, the output vector corresponds to a discretized value of the average throughput of the evaluated TCP flows. The accuracy is hence defined as the correct prediction from the neural network of the bins associated to the average throughputs.

We notice in Figure 8 that the GG-NN described in Section 3.2 – labeled *GG-GRU-NN* in the plot – is able to reach accuracies higher than 85 %, even on topologies with a larger number of flows and number of hops. On small topologies, the neural network is able to reach accuracies higher than 95 %. We notice that the average prediction accuracy decreases slightly with the complexity of the network, namely according to the number of hops traversed by the TCP flows and overall number of flows in the network.

The results of the GG-LSTM-NN architecture, our proposition for a modification of the GG-NN architecture, performs better than the original GG-NN architecture, with overall accuracies higher than 90 %. Finally, as a comparison, the RNN architecture from Equation (20) is only able to reach accuracies higher than 90 %, even on the smaller topologies.

Those numerical results motivates our choice of the GG-LSTM-NN architecture over the original GG-NN and RNN architectures for our application.

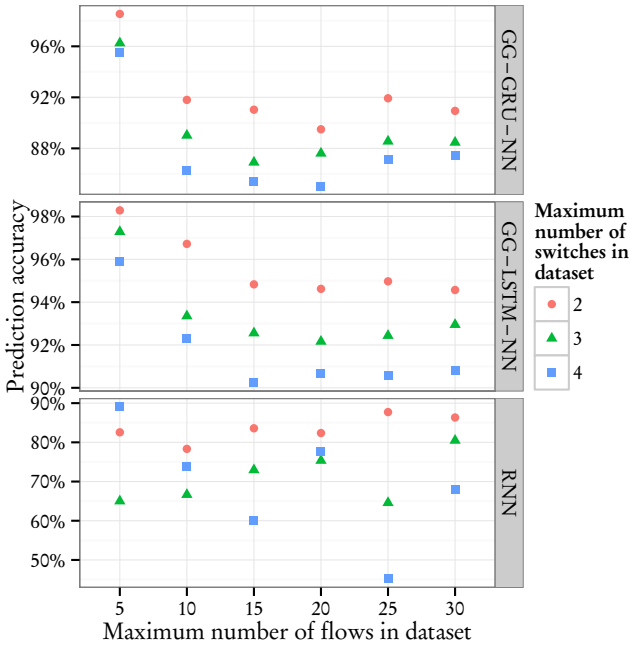

Figure 8: Average prediction accuracy on topologies with infinite TCP flows. Each data point correspond to another dataset where the neural networks were trained and evaluated.

5.3 Evaluation on finite TCP flows

In this second use-case, we assume an ON/OFF traffic model for TCP flows, where clients repeatedly send a random amount of data followed by a random idle period. It represents the case of small file transfers over TCP. This traffic model is illustrated in Figure 9 where three flows are sharing the same link. We restrict here the line topology to a maximum of two switches.

An exponential distribution is used for file sizes, where each flow is randomly assigned a different mean value between 1 MB and 5 MB for the file size distribution. Similarly, an exponential distribution is used for idle periods, with a mean value of 1 s for all flows. The feature vector l_v is extended here to take into account the mean of the file size distribution, using one-hot encoding.

Numerical results are presented in Figure 10. Despite less accurate predictions compared to Figure 8, we notice that the neural networks are still able to learn the bandwidth sharing of the ON/OFF flows and use the file size distributions. As in the previous results, the GG-LSTM-NN architecture outperforms the GRU-based GG-NN architecture.

In order to illustrate the impact of file size distribution on the prediction accuracy, we also trained the neural network without this information. Results are presented in Figure 11. As expected, the prediction accuracy decreases, showing that the network was able to use the file size distribution previously in Figure 10.

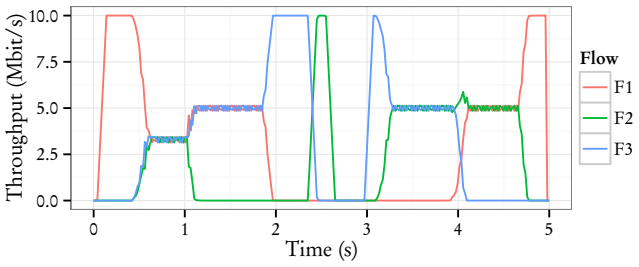

Figure 9: Illustration of the bandwidth sharing of three ON/OFF TCP flows using the same bottleneck. Each data point correspond to another dataset where the neural networks were trained and evaluated.

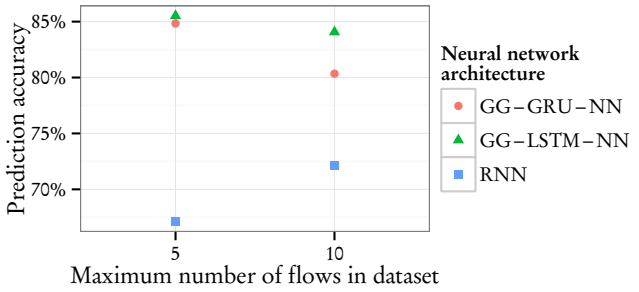

Figure 10: Average prediction accuracy on topologies with finite ON/OFF TCP flows. Each data point correspond to another dataset where the neural networks were trained and evaluated.

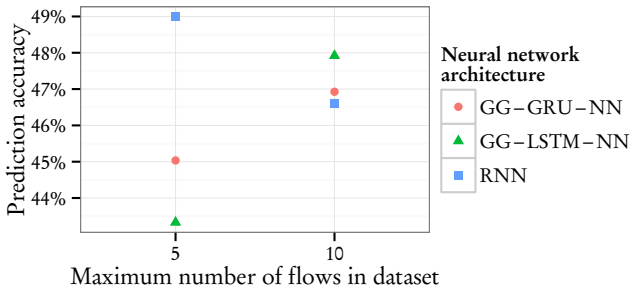

Figure 11: Average prediction accuracy on topologies with finite ON/OFF TCP flows, where the neural network is trained without information on the file size distributions.

5.4 Evaluation on combined infinite and ON/OFF TCP flows

In this third use-case, we assume a combination of both previous traffic models on the same network, where some flows are infinite and some flows are finite. This combined traffic model is often referred as *"mice and elephants"* in the literature. The same parameters as in Section 5.3 were used for the file size distributions and idle time distributions. Topologies were generated such that $1/6^{\text{th}}$ of the flows were infinite flows, and the rest finite ON/OFF flows as presented in Figure 9. We also restrict here the line topology to a maximum of two switches.

Numerical results are presented in Figure 12. We notice here similar results than in Figure 10. The neural networks are able to predict the average throughputs, although with less accuracy compared to the two previous use-cases.

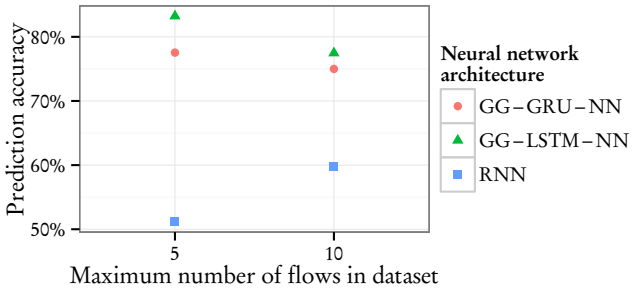

Figure 12: Average prediction accuracy on topologies with a combination of finite and infinite TCP flows.

5.5 Interpreting GG-NNs

An important subject when working with neural network is the interpretability of the learned weights. We propose here to visualize Equation (5) as t increases, namely visualize how the hidden representation of a node evolves at different time-steps. This is illustrated in Figure 13 on a sample network with 4 flows and 10 queues.

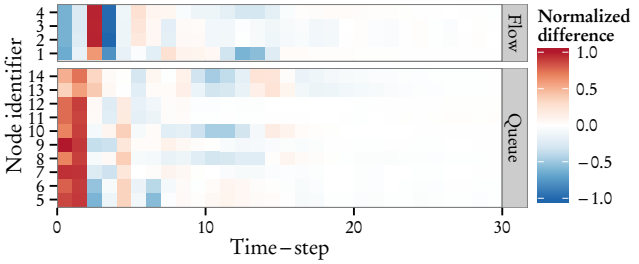

Figure 13: Visualization of the hidden representations' propagation. The color of each cell corresponds to $\sum_{\mathcal{D}} \mathbf{h}_v^{(t)} - \mathbf{h}_v^{(t-1)}$ for each node v in the graph.

We observe that a fixed point for Equation (5) is indeed reached since there is almost no difference between $\mathbf{h}_v^{(t)}$ and $\mathbf{h}_v^{(t-1)}$ in the last time-steps for all nodes. It is interesting to notice that the fixed point for each queue is reached at different time-steps, meaning that some queues (eg. node 13 or 14) have a larger impact on flow performance than other queues (eg. node 5 or 6).

6 CONCLUSION

We presented in this paper a novel approach for the performance evaluation of network topologies and flows by using graph-based deep learning. Our approach is based on the use of a modified Gated

Graph Neural Networks called GG-LSTM-NN and a low-level graph-based representation of queues and flows in network topologies. Compared to other approaches using machine learning for performance evaluation of computer networks, the trained model is not specific to a given topology and high-level input features requiring more advanced knowledge on the studied protocol are not required.

We applied our approach to the performance evaluation of TCP flows with the task of predicting the average throughput for each flow. This is an interesting task since the throughput of TCP flows is dependent on the network architecture and network conditions (*i.e.* congestion and delays). Different traffic models for the flows were evaluated: large file transfers, small file transfers, and a combination of large and small file transfers. We showed via a numerical evaluation that our approach is able to reach good accuracies, even on large network topologies with multiple hops. We compared the chosen neural network architecture against a simpler recurrent neural network architecture, motivating our choice for GG-NNs. Finally we also visualized the internal working of the neural network in order to give some insights on which queues have an influence on protocol performances.

Since the network topology is directly taken as input of the neural network, applications such as network planning and architecture optimization may benefit from the method developed in this paper. As our approach is not specific to the performance evaluation of TCP flows, future work may include evaluations and extensions of our approach to other congestion control algorithms, performance measure such as latency or other network protocols.

Acknowledgments This work has been supported by the German Federal Ministry of Education and Research (BMBF) under support code 16KIS0538 (DecADe).

REFERENCES

[1] 2017. ns-2, Network Simulator (ver. 2.35). (2017). Retrieved July 28, 2017 from https://www.isi.edu/nsnam/ns/

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

[3] Luis B. Almeida. 1990. Artificial Neural Networks. IEEE Press, Piscataway, NJ, USA, Chapter A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment, 102–111.

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR'2014)*.

[5] Neal Cardwell, Stefan Savage, and Thomas Anderson. 2000. Modeling TCP Latency. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)*, Vol. 3. IEEE, 1742–1751. https://doi.org/10.1109/INFCOM.2000.832574

[6] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. (June 2014). arXiv:1406.1078

[7] Victor Firoiu, Ikjun Yeom, and Xiaohui Zhang. 2001. A Framework for Practical Performance Evaluation and Traffic Engineering in IP Networks. In *Proceedings of the IEEE International Conference on Telecommunications*.

[8] Fabien Geyer, Stefan Schneele, and Georg Carle. 2013. Practical Performance Evaluation of Ethernet Networks with Flow-Level Network Modeling. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2013)*. 253–262. https://doi.org/10.4108/icst.valuetools.2013.254367

[9] Fabien Geyer, Stefan Schneele, and Georg Carle. 2014. PETFEN: A Performance Evaluation Tool for Flow-Level Network Modeling of Ethernet Networks. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2014)*. https://doi.org/10.4108/icst.valuetools.2014.258166

[10] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A New Model for Learning in Graph Domains. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks (IJCNN'05)*, Vol. 2. IEEE, 729–734. https://doi.org/10.1109/IJCNN.2005.1555942

[11] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* 538, 7626 (Oct. 2016), 471–476. https://doi.org/10.1038/nature20101

[12] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep Convolutional Networks on Graph-Structured Data. (June 2015). arXiv:1506.05163

[13] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[14] Hadrien Hours, Ernst W. Biersack, and Patrick Loiseau. 2016. A Causal Approach to the Study of TCP Performance. *ACM Trans. Intel. Syst. Tech.* 7, 2 (Jan. 2016), 25:1–25:25. https://doi.org/10.1145/2770878

[15] Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR'2015)*. https://arxiv.org/abs/1412.6980

[16] Cheng Li, Xiaoxiao Guo, and Qiaozhu Mei. 2016. DeepGraph: Graph Structure Predicts Network Growth. (Oct. 2016). arXiv:1610.06251

[17] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated Graph Sequence Neural Networks. In *Proceedings of the 4th International Conference on Learning Representations (ICLR'2016)*.

[18] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. 1997. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM Comput. Commun. Rev.* 27, 3 (June 1997), 67–82. https://doi.org/10.1145/263932.264023

[19] Mariyam Mirza, Joel Sommers, Paul Barford, and Xiaojin Zhu. 2010. A Machine Learning Approach to TCP Throughput Prediction. *IEEE/ACM Trans. Netw.* 18, 4 (Aug. 2010), 1026–1039. https://doi.org/10.1109/TNET.2009.2037812

[20] Jitendra Padhye, Victor Firoiu, Don F. Towsley, and James F. Kurose. 2000. Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation. *IEEE/ACM Trans. Netw.* 8, 2 (April 2000), 133–145. https://doi.org/10.1109/90.842137

[21] Fernando J. Pineda. 1987. Generalization of back-propagation to recurrent neural networks. *Phys. Rev. Lett.* 59 (Nov. 1987), 2229–2232. Issue 19. https://doi.org/10.1103/PhysRevLett.59.2229

[22] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. Computational Capabilities of Graph Neural Networks. *IEEE Trans. Neural Netw.* 20, 1 (Jan. 2009), 81–102. https://doi.org/10.1109/TNN.2008.2005141

[23] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Trans. Neural Netw.* 20, 1 (Jan. 2009), 61–80. https://doi.org/10.1109/TNN.2008.2005605

[24] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2017. Modeling Relational Data with Graph Convolutional Networks. (March 2017). arXiv:1703.06103

[25] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 1 (Jan. 2014), 1929–1958.

[26] Mukarram Bin Tariq, Kaushik Bhandankar, Vytautas Valancius, Amgad Zeitoun, Nick Feamster, and Mostafa Ammar. 2013. Answering "What-If" Deployment and Configuration Questions With WISE: Techniques and Deployment Experience. *IEEE/ACM Trans. Netw.* 21, 1 (Feb. 2013), 1–13. https://doi.org/10.1109/TNET.2012.2230448

[27] Guibin Tian and Yong Liu. 2012. Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*. ACM, 109–120. https://doi.org/10.1145/2413176.2413190

[28] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning* 4, 2 (2012), 26–31.

[29] Pedro Velho, Lucas M. Schnorr, Henri Casanova, and Arnaud Legrand. 2011. *Flow-level network models: have we reached the limits?* Technical Report 7821. INRIA.