# RouteNet-Erlang: A Graph Neural Network for Network Performance Evaluation

Miquel Ferriol-Galmés*, Krzysztof Rusek†, José Suárez-Varela*, Shihan Xiao‡,
Xiang Shi‡, Xiangle Cheng‡, Bo Wu‡, Pere Barlet-Ros*, Albert Cabellos-Aparicio*
*Barcelona Neural Networking Center, Universitat Politècnica de Catalunya, Spain
Corresponding email: miquel.ferriol@upc.edu
†AGH University of Science and Technology, Poland
‡Huawei Technologies Co., Ltd., China

***Abstract*—Network modeling is a fundamental tool in network research, design, and operation. Arguably the most popular method for modeling is Queuing Theory (QT). Its main limitation is that it imposes strong assumptions on the packet arrival process, which typically do not hold in real networks. In the field of Deep Learning, Graph Neural Networks (GNN) have emerged as a new technique to build data-driven models that can learn complex and non-linear behavior. In this paper, we present *RouteNet-Erlang*, a pioneering GNN architecture designed to model computer networks. RouteNet-Erlang supports complex traffic models, multi-queue scheduling policies, routing policies and can provide accurate estimates in networks not seen in the training phase. We benchmark RouteNet-Erlang against a state-of-the-art QT model, and our results show that it outperforms QT in all the network scenarios.**

***Index Terms*—Network Modeling, Graph Neural Network**

## I. Introduction

Network modeling is central to the field of computer networks. Models are useful in researching new protocols and mechanisms, allowing administrators to estimate their performance before their actual deployment in production networks. Network models also help to find optimal network configurations, without the need to test them in production networks.

Queuing Theory (QT) [1] is arguably the most popular modeling technique, where networks are represented as interconnected queues that are evaluated analytically. This represents a well-established framework that can model complex and large networks. Its main limitation is that it imposes strong assumptions on the packet arrival process, which typically do not hold in real networks [2]. Internet traffic has been extensively analyzed in the past two decades [3], [4], [5], [6], [7] and, despite the community has not agreed on a universal model, there is consensus that in general aggregated traffic shows strong autocorrelation and a heavy-tail [8].

Another network modeling alternative is computational models (e.g., network simulators), which provide excellent accuracy. State-of-the-art network simulators include a wide range of network, transport, and routing protocols, and are able to simulate realistic scenarios. However, this comes at a very high computational cost that depends linearly on the number of packets being simulated. As a result, they are impractical in scenarios with realistic traffic volumes or large topologies. In addition, and because they are computationally expensive, they do not work well in real-time scenarios.

Machine Learning (ML) [9] provides a new breed of mechanisms to model complex systems. In particular, Deep Learning (DL) [10] has demonstrated to extract deep knowledge from human-understandable descriptions of a system. This approach has proven to achieve unprecedented accuracy in modeling properties of complex systems, like proteins [11].

The main advantage of DL models is that they are *data-driven*. DL models can be trained with real-world data, without making assumptions about the system. This enables to build models with unprecedented accuracy by effectively modeling the entire range of non-linear and multidimensional system characteristics. Computationally, DL is based on linear algebra and can take advantage of massive parallelism leveraging dedicated hardware and compilers.

Within the field of DL, Graph Neural Networks (GNN) [12] have recently emerged as an effective technique to model graph-structured data. GNNs are tailored to understand the complex relationships between the elements of a graph. The main novelty of GNNs is that their internal architecture is dynamically assembled based on the elements and connections of input graphs, and this enables to learn universal modeling functions that are invariant to graph isomorphism. GNNs are thus able to *generalize* over graphs, which means that they can produce accurate estimates in different graphs not seen during training. As we will show in this paper, this is a critical advantage of GNNs in the context of network modeling.

The novel GNN paradigm finally allows the application of ML in domains where data is essentially represented as graphs. As a consequence, at the time of this writing, substantial research efforts are being devoted to applying GNNs to different fields where data is fundamentally represented as graphs, such as chemistry [13], physics [14] and others [15] [16].

We argue that GNNs represent a new network modeling language with attractive advantages and characteristics. GNNs are designed to learn graphs, and computer networks are fundamentally graphs of connected queues. However, GNNs are not a black-box that map data inputs to outputs, it is actually

This publication is part of the Spanish I+D+i project TRAINER-A (ref.PID2020-118011GB-C21), funded by MCIN/ AEI/10.13039/501100011033. This work is also partially funded by the Catalan Institution for Research and Advanced Studies (ICREA) and the Secretariat for Universities and Research of the Ministry of Business and Knowledge of the Government of Catalonia and the European Social Fund.

 http://dx.doi.org/10.1109/INFOCOM48880.2022.9796944

a *modeling tool* that needs to be researched and designed to account for the core behavior of computer networks. In contrast to more classical DL models, where the architecture is basically defined by the number of layers and neurons, GNNs are assembled ad-hoc, based on the elements and connections of the input graphs. These components represent the GNN modeling language, and they need to be carefully designed to reflect the relevant properties of the system being modeled.

In this paper, we present RouteNet-Erlang (RouteNet-E), a novel GNN-based architecture designed for performance evaluation of computer networks. RouteNet-E shares the same goals as QT models: it is also able to model a network of queues, with different sizes and scheduling policies, while providing accurate estimates of delay, jitter, and losses. Interestingly, RouteNet-E is not limited to Markovian traffic models as QT, but rather it supports arbitrary traffic models including more complex ones with strong autocorrelation and high variance, which better represent the properties of real-world traffic [8]. We also show that RouteNet-E overcomes one of the main limitations of existing ML-based models: *generalization*. RouteNet-E is able to make accurate estimates in samples of unseen topologies one order of magnitude larger than those seen during training.

We benchmark RouteNet-E against a state-of-the-art QT model, over a wide variety of network samples covering several different traffic models, from basic Poisson, to more realistic and complex models with strong autocorrelation and approximated heavy-tails. Our evaluation results show that the proposed model outperforms the QT benchmark in *all* the network scenarios evaluated, always producing accurate delay predictions with a worst-case error of $6\%$ (for QT is $68\%$). We also show RouteNet-E's remarkable performance in hundreds of random network topologies not seen during training. Lastly, we measure its inference speed, which is in the order of milliseconds, in line with the QT benchmark.

All datasets, code, and trained models of RouteNet-E used in this paper are publicly available at [17].

## II. Challenges of GNN-based network modeling

In this section, we describe the main challenges that GNN-based solutions need to address for network modeling. These challenges drove the core design of RouteNet-E.

**Traffic models:** A model is an abstraction of a system able to distill the essential aspects of the system. Networks transport millions of packets and, as a result, network models require a useful abstraction for packets, that is why supporting arbitrary stochastic traffic models is crucial. Experimental observations show that traffic on the Internet has strong autocorrelation and heavy-tails [8]. In this context, it is well-known that a main limitation of QT is that it fails to provide accurate estimates on realistic Markovian models with continuous state space, or non-Markovian traffic models. To the best of our knowledge, analytical models for queues with general arrival processes are limited to infinite buffers [18], or they make some sort of approximation (e.g., asymptotic), which greatly differs from the actual behavior of computer networks. The challenge for GNN-based modeling is, how to design an architecture that can accurately model realistic traffic models?

**Training and Generalization:** One of the main differences between analytical modeling (e.g., QT) and ML-based modeling is that the latter requires training. In ML, training involves obtaining a *representative* dataset with network measurements. The dataset needs to include a broad spectrum of network operational regimes. In practice, this means testing how different congestion levels affect performance metrics (delay, jitter, and losses), evaluating how different queuing policies affect performance or testing different routing policies, among others. Without this, the ML model is unable to learn and provide accurate estimates. Note that this is a common property of all neural network architectures. Generating this training dataset from networks in production is typically unfeasible, as it would require to artificially generate configurations (e.g., queue scheduling, routing) that lead to service disruption. A reasonable alternative is to create these datasets in controlled testbeds, where it is possible to use different traffic models and implement a broad set of configurations. Thus, the GNN model can be trained on samples from this testbed, and then be applied to real networks. Hence, the research challenge is: how to design a GNN able to provide accurate estimates in networks not seen during training? This includes topologies, traffic, and configurations (e.g., queue scheduling, routing) different from those seen in the training network testbed.

**Generalization to larger networks:** From a practical standpoint, the GNN model must also generalize to *larger* networks. Real-world networks include hundreds or thousands of nodes, and building a network testbed at this scale is typically unfeasible. As a result, the GNN model should be able to generalize from small network testbeds to considerably larger networks, by at least a factor of 10x. Generalizing to larger networks – or graphs, in general – is currently an open research challenge in the field of GNNs. We address this by using domain-specific knowledge from computer networks, and by proposing a novel GNN architecture that can effectively model relevant scale-independent features of networks that affect performance metrics.

**Queues and Scheduling policies:** A fundamental aspect when modeling networks is considering the behavior of queues (e.g., number, size), scheduling policies (e.g., WFQ, DRR), and the mapping of traffic flows to different Quality-of-Service classes if any. QT is a well-established technique, and models have been developed to support a wide range of scheduling policies [19], [20]. The challenge is, how to represent queues and scheduling policies inside the GNN architecture?

## III. Background: GNNs

Graphs are used to represent relational information. Particularly, a graph $G \in \{V, E\}$ comprises a set of objects $V$ (i.e., vertices) and some relations between them $E$ (i.e., edges).

GNN [12] is a family of NNs especially designed to work with graph-structured data. These models dynamically build their internal NN architecture based on the input graph. For this, they use a modular NN structure that represents

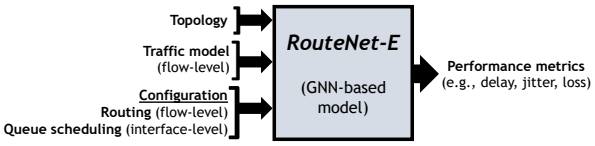



Fig. 1: Black-box representation of RouteNet-E

explicitly the elements and connections of the graph. As a result, they support graphs of variable size and structure, and their graph processing mechanism is invariant to node and edge permutation, which eventually endows them with strong generalization capabilities over graphs – also known as *strong relational inductive bias* [21].

Despite GNN covers a broad family of neural networks with different architectural variants (e.g., [12], [14], [22]), most of them share the basic principle of an iterative message-passing phase, where the different elements of the graph exchange information according to their connections, and a final readout phase uses the information encoded in graph elements to produce the final output(s). We refer the reader to [12], [13], [21] for a more comprehensive background on GNNs.

## IV. RouteNet-Erlang

This section describes RouteNet-E, a novel GNN-based solution tailored to accurately model the behavior of real network infrastructures. RouteNet-E implements a novel three-stage message passing algorithm that explicitly defines some key elements for network modeling (e.g., traffic models, queues, paths), and offers support for a wide variety of features introduced in modern networking trends (e.g., complex QoS-aware queuing policies, overlay routing).

Figure 1 shows a black-box representation of the proposed GNN-based network model. The input of RouteNet-E is a network state sample, defined by: a network topology, a set of traffic models (flow-level), a routing scheme (flow-level), a queuing configuration (interface-level). As output, this model produces estimates of relevant performance metrics at a flow-level granularity (e.g., delay, jitter, losses).

### A. Model Description

RouteNet-E has two main building blocks: (1) Finding a good representation for the network components supported by the model – e.g., traffic models, routing, queue scheduling –, and (2) Exploit scale-independent features of networks, in order to achieve good generalization power to larger networks than those seen during training, which is an important open challenge previously discussed in Section II.

*1) Representing network components and their relationships:* First, let us define a network as a set of links $L = \{l_i : i \in (1, ..., n_l)\}$, a set of queues on $Q = \{q_i : i \in (1, ..., n_q)\}$, and a set of source-destination flows $F = \{f_i : i \in (1, ..., n_f)\}$. According to the routing configuration, flows follow a source-destination path. Hence, we define flows as sequences with tuples of the queues and links they traverse $f_i = \{(q_{F_q(f_i,0)}, l_{F_l(f_i,0)}), ..., (q_{F_q(f_i,|f_i|)}, l_{F_l(f_i,|f_i|)})\}$, where $F_q(f_i, j)$ and $F_l(f_i, j)$ respectively return the index of the $j$-th queue or link along the path of flow $f_i$. Let us also define $Q_f(q_i)$ as a function that returns all the flows passing through queue $q_i$, and $L_q(l_i)$ as a function that returns the queues injecting traffic into link $l_i$ – i.e., the queues at the output port to which the link is connected.

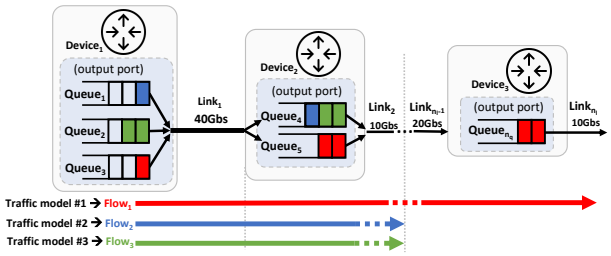



Fig. 2: Schematic representation of the network model implemented by RouteNet-E.

Following the previous notation, RouteNet-E considers an input graph with three main components: $(i)$ the physical links $L$ that shape the network topology, $(ii)$ the queues $Q$ at each output port of network devices, and $(iii)$ the active flows $F$ in the network, which follow some specific src-dst paths (i.e., sequences of queues and links), and whose traffic is generated from a given traffic model. Figure 2 shows a schematic representation of the network model internally considered by RouteNet-E, which is derived from the several mechanisms that affect performance in real networks. From this model, we can extract three basic principles:

(i) The state of flows (e.g., throughput, losses) is affected by the state of the queues and links they traverse (e.g., queue/link utilization).
(ii) The state of queues (e.g., occupation) depends on the state of the flows passing through them (e.g., traffic model).
(iii) The state of links (e.g., utilization) depends on the states of the queues at the output port of the link, and the queue scheduling policy applied over these queues.

Formally, these principles can be formulated as follows:

$$h_{f_k} = g_f(h_{q_{k(0)}}, h_{l_{k(0)}}, ..., h_{q_{k(|f_k|)}}, h_{l_{k(|f_k|)}}) \quad (1)$$

$$h_{q_i} = g_q(h_{p_1}, ..., h_{p_m}), \quad q_i \in p_k, k = 1, ..., j \quad (2)$$

$$h_{l_j} = g_l(h_{q_1}, ..., h_{q_m}), \quad q_m \in L_q(l_j) \quad (3)$$

Where $g_f$, $g_q$ and $g_l$ are some unknown functions, and $h_f$, $h_q$ and $h_l$ are latent variables that encode information about the state of flows $F$, queues $Q$, and links $L$ respectively. Note that these principles define a circular dependency between the three network components ($F$, $Q$, and $L$) that must be solved to find latent representations satisfying the equations above.

Based on the previous network modeling principles, we define the architecture of RouteNet-E (see Algorithm 1). Our GNN-based model implements a custom three-stage message-passing algorithm that combines the states of flows, queues and links according to Equations (1)-(3), thus aiming to resolve the circular dependencies defined in such functions. First, the hidden states $h_l$, $h_q$, and $h_f$ – represented as n-element vectors – are initialized with some features (lines 1-3), denoted respectively by $x_{l_i}$, $x_{q_j}$ and $x_{f_k}$. In our case, we set the initial

**Algorithm 1** Internal architecture of RouteNet-E

**Input:** $F$, $Q$, $L$, $x_f$, $x_q$, $x_l$
**Output:** $h_q^T$, $h_l^T$, $h_f^T$, $\hat{y_f}$, $\hat{y_q}$, $\hat{y_l}$

1: **for each** $l \in L$ **do** $h_l^0 \leftarrow [x_l, 0...0]$
2: **for each** $q \in Q$ **do** $h_q^0 \leftarrow [x_q, 0...0]$
3: **for each** $f \in F$ **do** $h_f^0 \leftarrow [x_f, 0...0]$
4: **for** t = 0 to T-1 **do** ▷ Message Passing Phase
5: **for each** $f \in F$ **do** ▷ Message Passing on Flows
6: **for each** $(q, l) \in f$ **do**
7: $h_f^t \leftarrow FRNN(h_f^t, [h_q^t, h_l^t])$ ▷ Flow: Aggr. and Update
8: $\widetilde{m}_{f,q}^{t+1} \leftarrow h_f^t$ ▷ Flow: Message Generation
9: $h_f^{t+1} \leftarrow h_f^t$
10: **for each** $q \in Q$ **do** ▷ Message Passing on Queues
11: $M_q^{t+1} \leftarrow \sum_{f \in Q_f(q)} \widetilde{m}_{f,q}^{t+1}$ ▷ Queue: Aggregation
12: $h_q^{t+1} \leftarrow U_q(h_q^t, M_q^{t+1})$ ▷ Queue: Update
13: $\widetilde{m}_q^{t+1} \leftarrow h_q^{t+1}$ ▷ Queue: Message Generation
14: **for each** $l \in L$ **do** ▷ Message Passing on Links
15: **for each** $q \in L_q(l)$ **do**
16: $h_l^t \leftarrow LRNN(h_l^t, \widetilde{m}_q^{t+1})$ ▷ Link: Aggr. and Update
17: $h_l^{t+1} \leftarrow h_l^t$
18: $\hat{y_f} \leftarrow R_f(h_f^T)$ ▷ Readout phase
19: $\hat{y_q} \leftarrow R_q(h_q^T)$

features of links ($x_l$) as: $(i)$ the link capacity ($C_i$), and $(ii)$ the scheduling policy at the output port of the link (FIFO, SP, WFQ, or DRR [23]), using one-hot encoding. For the initial features of queues ($x_q$) we include: $(i)$ the buffer size, $(ii)$ the priority level (one-hot encoding), and $(iii)$ the weight (only for WFQ and DRR). Lastly, the initial flow features ($x_f$) are a descriptor of the traffic model used in the flow ($T_i$). Once the states are initialized, the message-passing phase is iteratively executed $T$ times (loop from line 4), where $T$ is a configurable parameter. Each message-passing iteration is in turn divided in three stages, that respectively represent the message passing and update of the hidden states of flows $h_f$ (lines 5-9), queues $h_q$ (lines 10-13), and links $h_l$ (lines 14-17).

Finally, functions $R_f$ (line 18) and $R_q$ (line 19) represent independent readout functions that can be respectively applied to the hidden states of flows $h_f$ or queues $h_q$. In our experiments in Section VI, we use $R_f$ and $R_q$ to predict the flow-level delay, jitter and losses – as described later in this section.

The main motivation to use data-driven methods, such as RouteNet-E, instead of traditional QT is to achieve accurate modeling of complex traffic models that better reflect real-world traffic – as previously introduced in Section II. Hence, in RouteNet-E the representation of the traffic model descriptors ($T_i$) is central to achieve accurate modeling of different traffic patterns, and capturing their intrinsic properties. Particularly, we define $T_i$ as an n-element vector that includes the specific parameters that shape each traffic model. Find more details about the parameters of each model in section IV-B.

*2) Scaling to larger networks: scale-independent features*

As previously discussed in Section II, generating datasets directly from networks in production would imply testing configurations that may break the correct operation. As a result, GNN-based network models should be typically trained with data from network testbeds, which are usually much smaller than real networks. In this context, it is essential for our GNN to effectively scale to larger networks than those of the training dataset – by at least a 10x factor.

GNNs have shown an unprecedented capability to generalize over graph-structured data [21], [15]. In the context of generalizing to larger graphs, it is well known that these models keep good generalization capabilities as long as the spectral properties of graphs are similar to those seen during training [24]. In the case of RouteNet-E, its message-passing algorithm can analogously generalize to graphs with similar structures to those seen during the training phase – e.g., similar number of queues at output ports, or similar number of flows aggregated in queues. In this vein, generating a representative dataset for RouteNet-E in small networks, covering a wide range of graph structures, does not imply any practical limitation to then achieve good generalization properties to larger networks. It can be done by simply adding a broad combination of realistic network samples with a wide variety of traffic models, routing schemes, and queuing policies as in the process described later in section IV-B.

However, from a practical standpoint, scaling to larger networks often entails a broader definition beyond the topology size and structure. In particular, there are two main properties we can observe as networks become larger: $(i)$ *higher link capacities* (as there is more aggregated traffic in the core links of the network), and $(ii)$ *longer paths* (as the network diameter becomes larger). This requires devising mechanisms to effectively scale on these two features.

**Scaling to larger link capacities:** If we observe the internal architecture of RouteNet-E (Algorithm 1), we can find that the link capacity $C$ is only represented as an initial feature of links' hidden states $x_{l_i}$. The fact that $C$ is encoded as a numerical feature in the model introduces inherent limitations to scale to larger capacity values. Indeed, scaling to out-of-distribution numerical values is widely recognized as a generalized limiting factor among all neural networks [25], [26]. Thus, our approach is to exploit particularities from the network domain to find scale-independent representations that can define link capacities and how they relate to other link-level features that impact performance (e.g., the aggregated traffic in the link), as the final goal of RouteNet-E is to accurately estimate performance metrics (e.g., delay, jitter, losses). Inspired by traditional QT methods, we aim to encode in RouteNet-E the relative ratio between the arrival rates on links (based on the traffic aggregated in the link), and the service times (based on the link capacity), thus enabling the possibility to infer the output performance metrics of our model from scale-independent values. As a result, we define link capacities ($Cap_{link}$) as the product of a *virtual reference link capacity* ($C_{ref}$) and a *scale factor* ($S_f$) – i.e., $Cap_{link} = C_{ref} * S_f$.

This representation enables to define arbitrary combinations

of scale factors and reference link capacities to define the actual capacity of links in networks. Hence, in RouteNet-E we introduce the capacity feature ($C_i$) as a 2-element vector defined as $C_i$=$[C_{ref}, S_f]$, which is included in the initial feature vector of links ($x_l$). Note that this feature will eventually be encoded in the hidden states of links ($h_l$). In the internal architecture of RouteNet-E (Algorithm 1), this factor will mainly affect the update functions of flows and links (lines 7 and 16), as they are the only ones that process directly the hidden states of links ($h_l$). As a result, the RNNs approximating these update functions can potentially learn to make accurate estimates on any combination of $C_{ref}$ and $S_f$ as long as these two features are within the range of values observed *independently* for each of them during the training phase (i.e., $S_f \in [s_{f_{min}}, s_{f_{max}}]$ and $C_{ref} \in [C_{ref_{min}}, C_{ref_{max}}]$). Thus, we exploit this property to devise a custom data augmentation method, where we take samples from small networks with limited link capacities and generate different combinations of $C_{ref}$ and $S_{factor}$ that enable us to scale accurately to considerably larger capacities. Note that in this process, the numerical values seen by RouteNet-E ($C_{ref}$ and $S_{factor}$) are kept in the same ranges both in the training on small networks and the posterior inference on larger networks, thus overcoming the practical limitation of out-of-distribution predictions [25], [26]. More details about the proposed data augmentation process are given in Sec. IV-C.

The previous mechanism enables to keep scale-independent features along with the message-passing phase of our model (loop lines 4-17 in Algorithm 1), while it is still needed to extend the scale independence to the output layer of the model. Particularly, in this paper, we use RouteNet-E to predict the flows' delay, jitter, and losses. Note that the distribution of these parameters can also vary for flows traversing links with higher capacities, thus leading again to out-of-distribution values. Based on the fundamentals of QT, we overcome this potential limitation by inferring delays/jitter indirectly from the occupation of queues in the network $O_{q_i} \in [0, 1]$, using the $\hat{y_q}$=$R_q(h_q)$ function of RouteNet-E (Algorithm 1). Then, we infer the flow delay/jitter as a linear combination of the waiting times in queues (inferred from $O_{q_i}$) and the transmission times of the links the flow traverses. Note that a potential advantage with respect to traditional QT models is that the queue occupation estimates produced by RouteNet-E can be more accurate, especially for complex traffic models resembling real-world traffic – as shown later in our experimental results of Section VI. Likewise, for packet loss, RouteNet-E predicts directly the percentage of packets dropped with respect to the packets that were sent by the source of the flow, thus producing a bounded value $D_{f_i} \in [0, 1]$, that is estimated with the $\hat{y_f}$=$R_f(h_f)$ function of Algorithm 1.

**Scaling to longer paths:** In the internal architecture of RouteNet-E, the path length only affects to the RNN function of line 7 (Algorithm 1), which collects the state of queues ($h_q$) and links ($h_l$) to update flows' states ($h_f$). The main limitation here is that this RNN can typically see during training shorter link-queue sequences than those it can find then in larger networks, that can potentially have longer paths. As a result, we define $L_{max}$ as a configurable parameter of our model that defines the maximum sequence length supported by this RNN. Then, we split flows exceeding $L_{max}$ into different queue-link sequences that are independently digested by the RNN. To keep the state along with the whole flow, in case it is divided into more than one sequence, we initialize the initial state of the RNN with the output resulting from the previous sequence.

### B. Simulation Setup

To train, validate and test RouteNet-E we use as ground truth a packet-level network simulator (OMNeT++ v5.5.1 [27]), where network samples are labeled with performance metrics, including the flows' mean delay, jitter and losses, and queue-level statistics (e.g., occupation, packet loss). To generate these datasets, for each sample we randomly select a combination of input features (traffic model, topology, and queuing configuration) according to the descriptions below:

*1) Traffic models:* Traffic is generated using five different models with increasing levels of complexity, which ranges from a basic Poisson generation process to more realistic traffic models with strong autocorrelation and heavy-tails [8]. We define below the implementation details of these models (except for the well-known Poisson and Constant Bitrate, whose only configurable parameter is the traffic intensity level):

*a) On-Off:* This model defines two possible states (On or Off). The lengths of On and Off periods are randomly selected $[5, 15]$ seconds. Likewise, during the On period, packets are generated using an exponential distribution.

*b) Autocorrelated exponentials:* This model generates autocorrelated exponentially distributed traffic staring from the following auto-regressive (AR) process: $z_{t+1}$=$az_t+\varepsilon$, $\varepsilon \sim \mathrm{N}(0, \sigma^2)$ where $a \in (-1, 1)$ controls the level of autocorrelation. The marginal distribution of $z$ is $\mathrm{N}(0, s^2 = \sigma^2/(1-a^2))$, so $z$ can be negative. In order to construct positive inter-arrival times, $z$ is mapped by a nonlinear transformation: $\delta_t = F_E^{-1}\left(\lambda, F_N\left(0, s^2, z_t\right)\right)$, where $F_N(0, s^2, \cdot)$ and $F_E(\lambda, \cdot)$ are respectively a CDF of the normal distribution with $\mu = 0$ and variance $s^2$=$[3, 12]$, and an exponential distribution with parameter $\lambda$=$[40, 2000]$. The first transformation changes the distribution from normal to uniform on $(0, 1)$, while the second maps it into an exponential distribution. As a result, $\delta_t$ follows an exponential distribution with autocorrelation. Such a model can be interpreted as a copula [28].

*c) Modulated exponentials:* This model represents an alternative autocorrelated model with higher complexity for QT than the one above and is inspired by observation from [4]. Particularly, the inter-arrival times are set by a hierarchical model. Inter-arrivals follow an exponential distribution (Exp) whose rate is controlled by the value of a hidden AR model. Formally, we can describe the model as $\delta_t | z_t \sim \mathrm{Exp}(\lambda_t = Ae^{z_t})$, where $A$ is scaling constant and $z$ is an AR model as in the previous traffic model.

In all the previous models, average traffic rates on src-dst flows are carefully set to cover low to quite high congestion

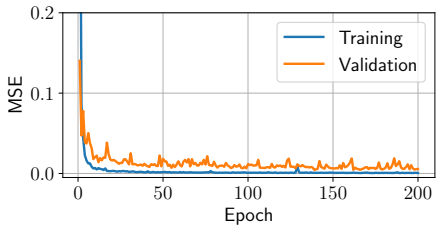



Fig. 3: Training and evaluation losses over time.

levels across different samples, where the most congested samples have ≈3% of packet losses.

*2) Queuing configuration:* Each forwarding device is configured with a different scheduling policy that depends on the particular scenario of our evaluation (more details in Sec. VI). Overall, we use four different queue scheduling policies: First In First Out (FIFO), Strict Priority (SP), Weighted Fair Queueing (WFQ), and Deficit Round Robin (DRR) [23]. We consider three queues per output port (except for FIFO, with only one queue), and queues have a size of 16 or 32 packets. For WFQ and DRR, we define random queue weights.

*3) Topologies:* To train the GNN model we used two different real-world topologies: NSFNET (14 nodes) [29], and GEANT (24 nodes) [30]. Then, we validate the accuracy of RouteNet-E in GBN (17 nodes) [31].

### C. Training

We implement RouteNet-E in TensorFlow. All the datasets, the code, and the trained models are publicly available [17]. To train the model, we use a custom data augmentation approach that, given a link capacity ($Cap_{link}$), covers a broad combination of $S_f$ and $C_{ref}$ values, in order to eventually make the model generalize over samples with larger link capacities. Particularly, given a link capacity, in some samples, we use low values of $S_f$ with higher values of $C_{ref}$, while in other samples we make it in the opposite way. As an illustrative example, if the model is trained over samples with 1Gbps links, we can represent these capacities in different samples as $Cap_{link}$=10*100$Mbps$=1$Gbps$, or $Cap_{link}$=1*1$Gbps$=1$Gbps$. Thus, after training the model should be able to make accurate inferences on samples that combine the maximum $S_f$ and $C_{ref}$ values seen during training – i.e., $Cap_{link}$=10*1$Gbps$=10$Gbps$. In practice, this means that the model can be trained with samples with a maximum link capacity of 1 Gbps, and then scale effectively to samples with link capacities up to 10 Gbps. Note that these numbers are just illustrative, while this data augmentation method is sufficiently general to produce in the training dataset wider ranges of $S_f$ and $C_{ref}$ given a maximum link capacity. Thus, it can be potentially exploited to represent combinations leading to arbitrarily larger capacities.

After making some grid search experiments, we set a size of 32 elements for all the hidden state vectors ($h_f$, $h_q$, $h_l$), and $T$=8 message-passing iterations. We implement $FRNN$, $LRNN$, and $U_q$ as Gated Recurrent Units (GRU) [32], and functions $R_f$ and $R_q$ as 2-layer fully-connected neural networks with ReLU activation functions. Here, it is important to note that the whole neural network architecture of RouteNet-E (Algorithm 1) constitutes a fully differentiable function, so it is possible to train the model end to end. Hence, all the different functions that shape its internal architecture are jointly optimized during training based on RouteNet-E's inputs (network samples) and outputs (performance metrics).

We use a training dataset with 200,000 samples from the NSFNET and GEANT topologies (100,000 samples each), including a variety of traffic model descriptors ($T_i$), routing schemes, and queue scheduling configurations – following the descriptions in Section IV-B. For the validation and test datasets, we generate 2,000 samples from the GBN topology (1,000 samples for each dataset). We train RouteNet-E for 200 epochs – with 4,000 samples per epoch – and set the Mean Squared Error (MSE) as loss function, using an Adam optimizer with an initial learning rate of 0.001. Figure 3 shows the evolution of the loss during training on delay estimates (for the training and validation samples), which shows stable learning along the whole training process.

## V. Baseline: Queuing Theory

In this section, we describe the state-of-the-art model we use to benchmark RouteNet-E. QT applied to networking results in a model as a function of graph-structured data. The network is represented as a directed multigraph of queues (buffers) while edges correspond to virtual channels along with the physical link. The general description of Equations (1)-(3) holds, however, the exact relations are derived analytically from the common assumption that a system is approximated by a Markov chain. This makes it a perfect benchmark for RouteNet-E.

In the holistic approach, the network is modeled as a single system, like in Jackson Networks [33] or more general BCMP queuing networks [34]. For those systems, the product form of the stationary distribution greatly simplifies the solution, however, the assumption of infinite buffers makes those models unrealistic and unable to estimate packet loss ratio.

In our approach, all the queues along the path are modeled independently. Further, we assume that arrival to each queue is approximated by the Poisson process. Service times are assumed to be independent and exponentially distributed. Under those assumptions, we can derive analytical results for queue throughput, delay distribution, and blocking probability.

The aforementioned model also suffers from circular dependency. Packet loss on a particular queue depends on its load so it also depends on the throughput of other queues feeding this particular one. The throughput, however, depends on packet loss so we end up with circular dependence. We fixed this problem by a *map-reduce* inspired algorithm that also emphasizes the resemblance between GNN and QT.

The algorithm consists of there functions: *map_queues*, *map_paths* and *reduce*. The *map_queues* function updates packet loss for each queue, given the total traffic (external demands plus within network transfer). The function also computes the remaining QoS parameters (jitter and delay). The *map_paths* function updates the traffic knowing the packet

loss on every queue. Finally, the *reduce* function aggregates per path delay, jitter, and packet loss. In the first iteration, we assume no packet loss. Given the first approximation, we can compute the loss probability (*map_queues*) and update the intensities to account for the losses (*map_paths*). After a few iterations, the algorithm converges to a fixed point and the final values are reduced (*reduce*). Notice how this approach is similar to RouteNet-E forward pass. In QT, we use known analytical relations while in GNN those relations are approximated by a neural network and learned from the data.

For an $M/M/1/b$ system, we used known formulas for blocking probabilities and delay distribution to get average delay and jitter. For a network with scheduling, we designed *map_queues* functions based on the Markov chain model described below. Because scheduling couples the queues, the corresponding *map_queues* operates on groups of queues assigned to the same link.

Let us begin with a strict priority scheduler. Consider $p$ priority class customers arriving at rate $\lambda_i$ and requiring service time with mean $1/\mu_i$. Each class waits in the independent virtual queue limited by $b_i$ and served in non-preemptive FIFO order. Such a system can be modeled as a continuous-time Markov chain on the state space $\mathcal{S}_{SP} = \{(s, q = (q_1, q_2, \ldots, q_p)\}$, where $s$ denotes the priority class currently being served or 0 if the system is empty. The remaining part of the state: $p$-tuple $q$ encodes the number of customers for each priority. For convenience let us define $q_{i+} := (q_1, \ldots, q_i + 1, \ldots, q_p)$, $q_{i-} := (q_1, \ldots, q_i - 1, \ldots, q_p)$ and $q^0 = (0, \ldots, 0)$. The model is based on [20] and modified to allow for per-priority class buffer size. Time evolution of the CTMC is characterized by the generator matrix $\boldsymbol{Q}$ whose elements follow the rules:

$$\boldsymbol{Q}[(0, q), (i, q_{i+})] = \lambda_i \quad 0 < i \leq p \tag{4}$$

$$\boldsymbol{Q}[(s, q), (s, q_{i+})] = \lambda_i I_{q_i < b_i} \quad 0 < i \leq p \tag{5}$$

$$\boldsymbol{Q}[(s, q^0_{s+}), (0, q^0)] = \mu_s \quad 0 < s \leq p \tag{6}$$

$$\boldsymbol{Q}[(s, q), (\min\{i : q_i > 0\}, q_{s-})] = \mu_s \tag{7}$$

where $I_A$ is an indicator function and $\boldsymbol{Q}[.,.]$ denotes entry in generator matrix. If neither rule matches states pair a general rules $\boldsymbol{Q}[s, s'] = 0, \quad s \neq s'$ and $\boldsymbol{Q}[s, s] = -\sum_{s' \neq s} \boldsymbol{Q}[s, s']$ apply. A similar model can be constructed for WFQ and DRR.

Since both scheduling policies approximate an ideal Generalized Processor Sharing the same model is used for WFQ and DRR. The CTMC is similar to (4)-(7) with exception that the queue $i$ is served at rate $\mu_i$ if other queues are empty, otherwise the rate scales proportionally to the positive weight $w_i$. State space $\mathcal{S}_{GPS}$ is also simplified and it is formed solely of $p$ tuples $q$ defined as for SP. The resulting CTMC is based on [19] and evolves according to the following generator:

$$\boldsymbol{Q}[q, q_{i+}] = \lambda_i I_{q_i < b_i} \quad 0 < i \leq p \tag{8}$$

$$\boldsymbol{Q}[q, q_{i-}] = \frac{I_{q_i > 0} w_i}{\sum_{q_i > 0} w_i} \mu_i \quad 0 < i \leq p \tag{9}$$

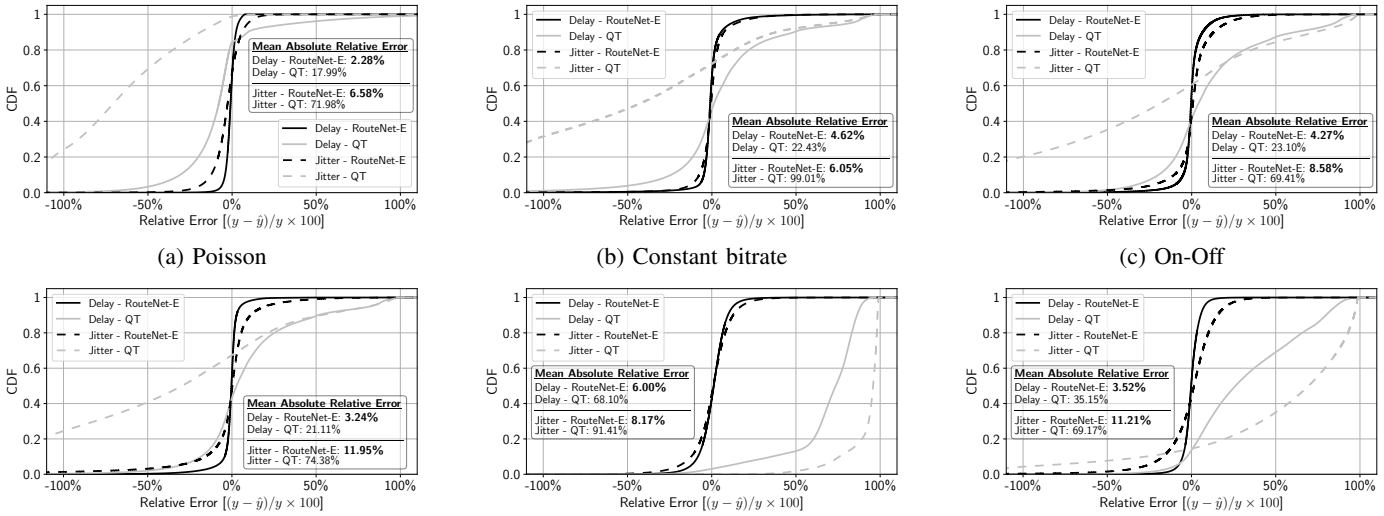
Given the generator matrix $\boldsymbol{Q}$, we can develop either an analytical solution for queue characteristics as in [19], [20] or use a direct approach and obtain them numerically. We chose the latter and compute packet loss delay and jitter assuming the CTMC has reached stationary distribution $\boldsymbol{\pi}$ computed from global balance equations (GBE) [33] that form a sparse linear system. We obtained $\boldsymbol{\pi}$ from sparse eigenvalue decomposition via Arnoldi method [35] with a general sparse linear solver as a fallback in case of numerical instabilities. Given the $\boldsymbol{\pi}$, the packet loss ratio for class $i$ ($\boldsymbol{p}_b[i]$) is the sum of all state probabilities where queue $i$ is full. The delay is computed from average queue size (with respect to $\boldsymbol{\pi}$) using Little's law. Computation of jitter requires a more sophisticated approach. We pose this as the first passage time problem in CTMC [36]. The delay of a class $i$ customer is the first passage time to any state where the queue $i$ is empty provided that no new customers can arrive so $\lambda_i = 0$ for GPS or $\lambda_j = 0, j \leq i$ for SP. Its conditional distribution can be calculated from $\boldsymbol{Q}$ using Laplace transform [36]. The final delay distribution and jitter are obtained from the total probability theorem. It is assumed that a packet of class $i$ observing state $s$ at his arrival experiences delay equal to the first passage time from the state just after his arrival $s_{i+}$. From PASTA property, the probability of such an event is $\boldsymbol{\pi}[s]/(1 - \boldsymbol{p}_b[i])$, here we condition of the event that packet is not dropped.

## VI. Evaluation

In this section, we evaluate the performance of RouteNet-E in a wide range of relevant scenarios. We seek to understand:
1) Can RouteNet-E model complex traffic models? What is the accuracy with realistic models with strong autocorrelation and heavy-tails?
2) Is RouteNet-E able to understand more complex multi-queue scheduling policies? What is the accuracy compared to QT?
3) Is RouteNet-E able to generalize to unseen network configurations and traffic loads? Also, can it generalize to *larger* networks?
4) How fast is RouteNet-E compared to the QT benchmark? Does it allow for real-time operation?

### A. Evaluation Methodology

To analyze the accuracy of RouteNet-E (Sec. IV) and benchmark it against the state-of-the-art queuing theory model (Sec. V), we use the following methodology. In all the experiments the ground-truth is obtained with a packet-level simulator (see Sec. IV-B for details). Unless noted otherwise, in each evaluation we perform 50k experiments with a random configuration (src-dst routing, traffic intensity, per-interface scheduling policy, and queue length) and compute the mean average delay, jitter, and losses. Then, we compute the error of RouteNet-E's and QT's estimates. For a fair comparison, we use samples of the GBN topology, which is not included during training (see Sec. IV-C for training details). Finally, depending on the experiment we use different traffic models (Sec. IV-B) and a wide range of realistic topologies.




(a) Poisson (b) Constant bitrate (c) On-Off

(d) Autocorrelated exponentials (e) Modulated exponentials (f) All traffic models multiplexed

Fig. 4: CDF of the relative error for RouteNet-E and QT with different traffic models. Top row shows models with discrete state space, bottom row includes continuous state space. Each figure also shows numbers of the mean absolute relative error.

### *B. Traffic Models*

This section focuses on analyzing the accuracy of RouteNet-E in a wide range of traffic models. The experiment is organized such that we add complexity to the traffic model by changing its first and second-order statistics (i.e., variance and autocorrelation). With this, we use challenging models that are good approximations to those seen in Internet links.

Figure 4 shows the CDF of the relative error (in %) for all the traffic models under evaluation. We plot the error for the delay and jitter estimates of both, RouteNet-E and QT. As we can observe, RouteNet-E achieves excellent results, producing very accurate estimates of delay and jitter in all traffic models, with a worst-case error below $6\%$ for delay and $12\%$ for jitter (mean absolute relative error).

As expected, QT results in unacceptable performance in continuous-state traffic models (up to $68\%$ for delay), while it achieves moderate accuracy for discrete-state models. Interestingly, QT shows poor accuracy across all the experiments estimating jitter. The reason for this is that QT assumes independence between queues in the network. Hence, the estimator used for jitter is the sum of the individual delay variance of queues along flow paths, which ignores possible covariance effects between queues.

It is remarkable that RouteNet-E is also accurate even with non-Markovian traffic models (On-Off, Figure 4c) and with challenging models that approximate strong autocorrelation (Autocorrelated Exponentials, Figure 4d). For the latter, it has been shown in the literature that the TCP protocol generates traffic with autocorrelation for a finite range of time-scales [37]. In this scenario, RouteNet-E estimates the delay with a mean error of $3.24\%$.

Figure 4e plots the accuracy for the Modulated Exponentials model, this emulates observations found at Internet links [8] by approximating a heavy-tail. In this scenario, RouteNet-E still produces very accurate estimates. It is worth noting that this traffic model could be made even more difficult for QT by increasing both the variance and the autocorrelation factor.

The key to RouteNet-E's performance is that it has been trained for such traffic models. As discussed in Section IV, we have parameterized the models and trained the GNN to learn the interaction between the traffic, the queues, and the resulting performance metrics. The experiments depicted in Figure 4 show that RouteNet-E can generalize to traffic, providing good accuracy even for traffic models with parameters not seen in training. RouteNet-E is designed to be an extensive model, adding a new traffic model is as simple as pasteurizing it and including it in training.

To showcase this, consider the experiment shown in Figure 4f, where we run 100k experiments with samples where each src-dst pair uses a *random traffic model* with random parameters. Effectively, we multiplex all traffic models in a single network topology. As the figure shows, RouteNet-E is able to model this scenario in the presence of complex interactions of various multiplexed traffic models.

### *C. Scheduling Policies*

With this experiment, we aim to validate that RouteNet-E is able to model the behavior of queues. For this we use 100K samples of the GBN topology, each router port is configured with three different queues and with a randomly selected scheduling policy (FIFO, WFQ, DRR, SP). For WFQ and DRR, the set of weights is also randomly assigned. Moreover, each src-dst path is assigned a Quality-of-Service class that maps traffic flows to specific queues. In order to provide a fair benchmark with QT, we use only Poisson traffic.

Table I summarizes the results, which are grouped for various traffic intensities, from low-loaded to highly-congested scenarios, where the mean packet loss rate is around $3\%$. As

| | Delay | | | Jitter | | | Loss | | |
|---|---|---|---|---|---|---|---|---|---|
| | Low | Med | High | Low | Med | High | Low | Med | High |
| **RouteNet-E** | 2.0% | 2.2% | 3.3% | 4.8% | 6.2% | 10.6% | 12.61% | 12.7% | 12.66% |
| **QT** | 13.0% | 17.3% | 25.1% | 49.0% | 53.2% | 59.6% | 61.83% | 59.3% | 57.9% |

TABLE I: Results for Scheduling Policies

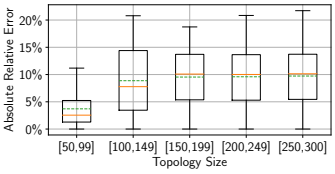



Fig. 5: Absolute relative error vs. topology size.

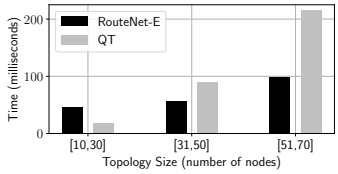



Fig. 6: Execution time vs. topology size.

we can observe, RouteNet-E outperforms QT, obtaining highly accurate estimates for all the evaluated metrics.

### D. Generalization to larger topologies

The previous experiments have shown that RouteNet-E achieves remarkable accuracy in performance evaluation under different traffic models (Sec. VI-B) as well as complex scheduling policies (Sec. VI-C). As we have discussed in Section II, ML-based network models must generalize to unseen and *larger* networks to become a practical solution. In this vein, RouteNet-E was carefully designed to address this challenge (see Sec. IV-A for details).

In this set of experiments, we evaluate RouteNet-E in a wide range of networks considerably larger than the ones seen during training. Specifically, the model has been trained with topologies between 25 and 50 nodes and tested with topologies from 50 to 300 nodes. All these networks have been artificially generated using the Power-Law Out-Degree algorithm described in [38], where the ranges of the $\alpha$ and $\beta$ parameters have been extrapolated from real-world topologies of the Internet Topology Zoo repository [39]. Link capacities and the generated traffic volume is scaled accordingly.

Figure 5 shows how RouteNet-E generalizes to larger topologies not seen in training. Specifically, the boxplots show the absolute relative error with respect to the topology size. As expected, RouteNet-E obtains better accuracy in topologies that are closer to the ones seen during the training phase (50 to 99 nodes), achieving an average error of $4.5\%$ (green line). As the topology size increases, the average error stabilizes to $\approx 10\%$. Note that this value is even lower than the one obtained by the QT model, which achieves a mean error of $12.6\%$ in samples with Poisson traffic (Fig. 4a). We could not test larger topologies ($>$300) in our cluster (180 nodes), as packet-level simulations – used for the ground-truth – are sequential in nature, and, with our traffic configurations, have exponential complexity with respect to the topology size.

Generalization is an open challenge in the field of GNN. As discussed in Sec. IV, we have addressed this by using domain-specific knowledge and data augmentation. Particularly, we infer delay/jitter from queues' occupation and apply our scale-independent method to generalize to larger topologies.

### E. Inference Speed

Finally, in this section, we evaluate the inference speed of RouteNet-E. Fast models are especially appealing for network control and management, as they can be deployed in real-time scenarios. For this, we have measured the execution times [Intel(R) Xeon(R) Gold 5220 CPU @ 2.20GHz] of the experiments in the previous section, for both QT and RouteNet-E. The results (Figure 6) show that both models operate in the order of milliseconds. In particular, RouteNet-E goes from a few milliseconds for small topologies to a few hundred for the larger ones.

## VII. Related Work

The use of Deep Learning (DL) for network modeling has recently attracted the interest of the networking community. This idea was initially suggested by *Wang, et al.* [40]. The authors survey different techniques and discuss data-driven models that can learn real networks. Initial attempts to instantiate this idea use fully connected neural networks (e.g. [41], [42]). Such early attempts do not generalize to networks not seen in training, are not tested with realistic traffic models, and do not model queues. More recent works propose more elaborated neural network models, like Variational Auto-encoders [43] or ConvNN [44]. However, they have similar limitations.

Finally, some early pioneering works use GNN in the field of computer networks [45], [46], [47]. However, they use a basic GNN architecture that considers a simplified model of the network, ignoring traffic models, queuing policies, and the critical property of generalizing to larger networks.

## VIII. Discussion and Concluding remarks

In this paper, we have presented RouteNet-E, a new tool for network modeling. RouteNet-E has shown remarkable accuracy in all the scenarios, outperforming a state-of-the-art QT model. RouteNet-E also overcomes the main limitation of QT, and it is able to model challenging traffic models. More importantly, the proposed model addresses the main drawback of existing ML-based models, and it is able to provide accurate estimates in larger networks ($\approx$10x).

RouteNet-E provides unprecedented accuracy in network performance evaluation. However, in contrast to QT, it does not help understand the behavior of the network being modeled. The knowledge learned by RouteNet-E during training is not human-understandable. This is a common issue for all ML-based models, and substantial research efforts are being devoted to producing explainable ML models [47]. However, this is still an open research problem.

RouteNet-E's performance enables network optimization, planning, and operation in real-time scenarios. It also represents an open-source extensible model. We hope that the community will use it as a baseline to incorporate additional network components, such as other scheduling policies, traffic models, etc.

## REFERENCES


[1] R. B. Cooper, "Queueing theory," in *Proceedings of the ACM'81 conference*, 1981, pp. 119–122.

[2] Z. Xu *et al.*, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM*, 2018, pp. 1871–1879.

[3] A. Arfeen, K. Pawlikowski, D. McNickle, and A. Willig, "The role of the weibull distribution in modelling traffic in internet access and backbone core networks," *Journal of network and computer applications*, vol. 141, pp. 1–22, 2019.

[4] T. Karagiannis *et al.*, "A nonstationary poisson view of internet traffic," in *IEEE INFOCOM*, vol. 3, 2004, pp. 1558–1569.

[5] T. Karagiannis, M. Molle, and M. Faloutsos, "Long-range dependence ten years of internet traffic modeling," *IEEE internet computing*, vol. 8, no. 5, pp. 57–64, 2004.

[6] E. Kresch and S. Kulkarni, "A poisson based bursty model of internet traffic," in *IEEE International Conference on Computer and Information Technology*, 2011, pp. 255–260.

[7] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *IEEE/ACM Transactions on networking*, vol. 3, no. 3, pp. 226–244, 1995.

[8] J. Popoola and R. Ipinyomi, "Empirical performance of weibull self-similar tele-traffic model," *International Journal of Engineering and Applied Sciences*, vol. 4, no. 8, p. 257389, 2017.

[9] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] Y. LeCun *et al.*, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[11] J. Jumper *et al.*, "Highly accurate protein structure prediction with AlphaFold," *Nature*, 2021, (Accelerated article preview).

[12] F. Scarselli, M. Gori *et al.*, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[13] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," *arXiv preprint arXiv:1704.01212*, 2017.

[14] P. Battaglia *et al.*, "Interaction networks for learning about objects, relations and physics," in *Advances in neural information processing systems*, 2016, pp. 4502–4510.

[15] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[16] O. Lange and L. Perez, "Traffic prediction with advanced Graph Neural Networks," 2020. [Online]. Available: https://deepmind.com/blog/article/traffic-prediction-with-advanced-graph-neural-networks

[17] M. Ferriol-Galmés *et al.*, "Routenet-erlang," https://github.com/BNN-UPC/Papers/wiki/RouteNet_Erlang, 2021.

[18] I. Norros, "A storage model with self-similar input," *Queueing Systems*, vol. 16, no. 3, pp. 387–396, 1994. [Online]. Available: https://doi.org/10.1007/BF01158964

[19] A. Al-Sawaai *et al.*, "Performance evaluation of weighted fair queuing system using matrix geometric method," in *IFIP NETWORKING*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 66–78.

[20] A. S. Kapadia, M. F. Kazmi, and A. Mitchell, "Analysis of a finite capacity non preemptive priority queue," *Computers & Operations Research*, vol. 11, no. 3, pp. 337–343, 1984.

[21] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[22] D. Raposo *et al.*, "Discovering objects and their relations from entangled scene representations," *arXiv preprint arXiv:1702.05068*, 2017.

[23] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on networking*, vol. 4, no. 3, pp. 375–385, 1996.

[24] L. Ruiz, L. Chamon, and A. Ribeiro, "Graph neural networks and the transferability of graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[25] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "Exploring the landscape of spatial robustness," in *International Conference on Machine Learning*, 2019, pp. 1802–1811.

[26] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.

[27] A. Varga, "Discrete event simulation system," in *European Simulation Multiconference (ESM)*, 2001, pp. 1–7.

[28] R. Nelsen, *An Introduction to Copulas*, ser. Lecture notes in statistics. Springer, 1999.

[29] X. Hei, J. Zhang *et al.*, "Wavelength converter placement in least-load-routing-based optical networks using genetic algorithms," *Journal of Optical Networking*, vol. 3, no. 5, pp. 363–378, 2004.

[30] F. Barreto *et al.*, "Fast emergency paths schema to overcome transient link failures in ospf routing," *arXiv preprint arXiv:1204.2465*, 2012.

[31] J. Pedro, J. Santos, and J. Pires, "Performance evaluation of integrated otn/dwdm networks with single-stage multiplexing of optical channel data units," in *International Conference on Transparent Optical Networks*, 2011, pp. 1–4.

[32] J. Chung *et al.*, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[33] F. P. Kelly, *Reversibility and Stochastic Networks*. Cambridge University Press, 2011.

[34] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *J. ACM*, vol. 22, no. 2, p. 248–260, Apr. 1975. [Online]. Available: https://doi.org/10.1145/321879.321887

[35] W. J. Stewart, *Numerical Methods for Computing Stationary Distributions of Finite Irreducible Markov Chains*. Boston, MA: Springer US, 2000, pp. 81–111. [Online]. Available: https://doi.org/10.1007/978-1-4757-4828-4_4

[36] M. Kijima, *Markov Processes for Stochastic Modeling*, ser. Stochastic Modeling Series. New York, NY: Springer-Science+Business Media, B.V., 1997.

[37] D. R. Figueiredo, B. Liu, V. Misra, and D. Towsley, "On the autocorrelation structure of tcp traffic," *Computer Networks*, vol. 40, no. 3, pp. 339–361, 2002.

[38] C. R. Palmer and J. G. Steffan, "Generating network topologies that obey power laws," in *IEEE Global Telecommunications Conference (GLOBECOM)*, vol. 1, 2000, pp. 434–438.

[39] S. Knight, H. Nguyen *et al.*, "The internet topology zoo," *IEEE JSAC*, vol. 29, no. 9, pp. 1765 –1775, 2011.

[40] M. Wang *et al.*, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Network*, vol. 32, no. 2, pp. 92–99, 2017.

[41] A. Valadarsky *et al.*, "Learning to route," in *ACM workshop on hot topics in networks*, 2017, pp. 185–191.

[42] A. Mestres *et al.*, "Understanding the modeling of computer network delays using neural networks," in *ACM SIGCOMM BigDAMA workshop*, 2018, pp. 46–52.

[43] S. Xiao *et al.*, "Deep-q: Traffic-driven qos inference using deep generative network," in *ACM SIGCOMM Workshop on Network Meets AI & ML*, 2018, pp. 67–73.

[44] X. Chen *et al.*, "Deep-rmsa: A deep-reinforcement-learning routing, modulation and spectrum assignment agent for elastic optical networks," in *2018 Optical Fiber Communications Conference and Exposition (OFC)*, 2018, pp. 1–3.

[45] F. Geyer and S. Bondorf, "Deeptma: Predicting effective contention models for network calculus using graph neural networks," in *IEEE INFOCOM*, 2019, pp. 1009–1017.

[46] K. Rusek, J. Suárez-Varela *et al.*, "Unveiling the potential of graph neural networks for network modeling and optimization in sdn," in *ACM Symposium on SDN Research*, 2019, pp. 140–151.

[47] Z. Meng *et al.*, "Interpreting deep learning-based networking systems," in *ACM SIGCOMM*, 2020, pp. 154–171.