xNet: Improving Expressiveness and Granularity for Network Modeling with Graph Neural Networks

Mowei Wang*, Linbo Hui*, Yong Cui*‡, Ru Liang†, Zhenhua Liu†
*Tsinghua University, China
†Huawei Technologies, China

IEEE INFOCOM 2022 - IEEE Conference on Computer Communications | 978-1-6654-5822-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOM48880.2022.9796726

***Abstract*—Today's network is notorious for its complexity and uncertainty. Network operators often rely on network models to achieve efficient network planning, operation, and optimization. The network model is responsible for understanding the complex relationships between the network performance metrics (e.g., latency) and the network characteristics (e.g., traffic). However, we still lack a systematic approach to developing accurate and lightweight network models that are aware of the impact of network configurations (i.e., expressiveness) and provide fine-grained flow-level temporal predictions (i.e., granularity).**

In this paper, we propose xNet, a data-driven network modeling framework based on graph neural networks (GNN). Unlike the previous proposals, xNet is not a dedicated network model designed for specific network scenarios with constraint considerations. On the contrary, xNet provides a general approach to modeling the network characteristics of concern with relation graph representations and configurable GNN blocks. xNet learns the state transition function between time steps and rolls it out to obtain the full fine-grained prediction trajectory. We implement and instantiate xNet with three use cases. The experiment results show that xNet can accurately predict different performance metrics while achieving over two orders of magnitude of speedup compared with the conventional packet-level simulator.

I. INTRODUCTION

With the rapid growth of network traffic, operators continually strive to improve network performance to meet the service level agreements for various network services. To achieve this, practitioners often rely on *network models* to achieve efficient network planning, operation, and optimization. The network model is tasked to predict how the network performance metrics (e.g., throughput, latency) change for various hypothetical "what-if" scenarios [1], such as changes to traffic conditions and re-configurations of network devices.

Typically, network models can apply to the following two kinds of scenarios (§II-A). (i) For online performance monitoring, a **fast** network model can reduce the monitoring overhead by inferring quality of service (QoS) metrics directly from traffic statistics in real-time without the help of high-cost QoS measurements (e.g., probe delay for each path [2]). (ii) For offline network planning, an **accurate** network model can facilitate the design and optimization process by providing the operators with predicted performance metrics under different network configurations, even if the network has not yet been physically built. From the optimization perspective, benefiting from the deep learning advances, recent proposals start to leverage the Deep Reinforcement Learning (DRL) techniques to efficiently optimize the network performance [3], [4]. However, DRL may perform unreliable explorations in real networks, which will disrupt existing services and cause intolerable performance degradation. To avoid possibly wrong solutions in production networks, the accurate and real-time network model can be utilized as a fast and safe environment for DRL training and exploration, enabling new applications for performance-oriented network optimization.

In this context, much effort has been devoted to building network models able to accurately predict performance metrics at an acceptable cost and at high speed. Traditional network modeling methods either use analytic models with simplified assumptions (e.g., queuing theory, network calculus [5]) or use a network simulator to produce all the packet-level events (e.g., NS3 [6]). The assumptions used in the former may not be valid in the actual network and will lead to inaccurate network models, while the latter leads to high computational cost, which makes it infeasible to use in real-time.

With recent advances, deep neural networks show superior performance on complex data modeling directly from raw data [7]–[10], which is promising for building lightweight network models with good accuracy. Existing learning-based attempts [11]–[14] have great successes in their focuses. For example, Deep-Q [11] can produce accurate inferences on path-level delay distributions, while RouteNet [12] successfully makes accurate path-level performance estimations and generalizes to unseen topologies and routing schemes.

However, state-of-the-art learning-based proposals [11]–[14] have yet to fulfill their high expectations and still fall short of satisfying the requirements of **expressiveness** and **granularity** (§II-B). In most scenarios, the network performance is influenced by not only the traffic conditions but also many network characteristics of how real networks operate, in particular the configurations at different levels, such as local configurations of network devices (e.g., buffer size of switches) and network-wide (thus global) configurations (e.g., routing schemes). To make accurate predictions, the network model must be expressive enough to account for all of these factors and provide the "knob" to adjust the corresponding feature to allow evaluation in "what-if" scenarios.

Furthermore, the prediction granularity of the network model should be as fine-grained as possible. From the spatial view, the network model is much desired to provide

‡Yong Cui (cuiyong@tsinghua.edu.cn) is the corresponding author.

TABLE I
COMPARISON OF EXISTING SCHEMES ON THE ACHIEVED REQUIREMENTS.

Schemes	Expressiveness		Granularity	
	Global Config. (e.g. topology, routing)	Local Config. (e.g. buffer, ECN)	Spatial (e.g. flow level)	Temporal (e.g. time series)
Deep-Q [11]	×	×	×	✓
RouteNet [12]	✓	×	×	×
xNet	✓	✓	✓	✓

flow-level modeling ability, rather than coarse-grained path-level prediction, so that concerned services can be managed delicately, particularly in data center networks (DCN). [15]. From the temporal perspective, network operators are usually more interested in transient network performance anomalies (e.g., throughput decline and latency spikes) [16], which can only be recognized with continuous performance monitoring over time. Therefore, it calls for a general network modeling framework that is expressive enough to model a wide range of network configurations and can provide fine-grained (i.e., flow-level and time-series) predictions. A comparison of existing proposals to the achieved requirements is listed in Table. I.

In this paper, we propose xNet, a data-driven network modeling framework that simultaneously supports network configuration modeling and flow-level time-series prediction. Our high-level design goal is to provide a general approach to building the network models so that different network characteristics can be properly modeled and generalized in a unified way. xNet leverages the advantages of deep learning and neural networks to obtain the potential benefits of accuracy and speed by directly learning from data. Unlike the previous proposals, xNet is not a dedicated network model designed for specific network scenarios with constraint considerations. On the contrary, xNet offers a basic network model (§III) that can be configured and instantiated to model various network scenarios with different interests (§IV). xNet proposes the following innovative designs to meet the requirements of expressiveness and granularity.

Relation graph abstraction with configurable GNN: To achieve expressiveness, xNet leverages graph neural networks [17] to model the intricate relationships between various network entities and different configurations. GNNs carry strong relational inductive bias so that inherently support relational reasoning and combinatorial generalization [18]. Note that the reason for utilizing GNN does not stem from the fact that computer networks are inherently represented as graphs, but rather from its capability to model relationships.

Specifically, we represent the networking system as a relation graph. The graph's nodes represent network entities with their own configurations, and the graph's edges reflect the relations between nodes. The domain knowledge of the relationship between configurations can be embedded by configuring the edge connections (§III-A). The interactions are approximated by learned message-passing among nodes. During learning, the knowledge about interaction is encoded in the GNN's update function. Learned knowledge is shared across the same types of entities in the system, which supports generalization to different network systems composed of the same types of entities and configurations (§III-B).

State transition learning: To enable flow-level time-series prediction, xNet leverages the recurrent structure of the GNN model to learn the state transition function between discrete time steps. The model maintains the state of each flow and updates it with the transition function. The model can be trained with one-step supervision of the difference between states in adjacent time steps. After training, xNet can roll it out step by step to obtain the full prediction trajectory (§III-C).

To showcase the efficiency of our framework, we implement xNet and instantiate it for three use cases (§IV) with customization. Each case focuses on different properties of its target scenario, and the main results are listed below.

- For temporal QoS inference, xNet can make accurate time-series predictions of path-level latency under unseen queue-level configurations in a DCN scenario with an average accuracy of 93%.
- For flow completion time (FCT) prediction, xNet can accurately predict the FCT in DCN scenarios with a Pearson correlation of $\sim$0.9.
- For steady-state QoS inference, xNet slightly outperforms the existing approach in modeling the steady-state path-level latency in a wide area network (WAN) scenario.
- Compared with the conventional packet-level simulator, xNet achieves over two orders of magnitude of speedup in an FCT prediction scenario with thousands of flows.

To the best of our knowledge, xNet is the first data-driven network modeling framework that simultaneously supports network configuration modeling and flow-level time-series prediction. In summary, we make the following key contributions: 1) A system abstraction approach and a configurable GNN block enabling expressive network modeling (§III-A-III-B); 2) A state transition model enabling fine-grained performance prediction (§III-C); 3) The implementation and instantiation of xNet with three use cases. (§IV).

II. BACKGROUND AND MOTIVATION

In this section, we first introduce the background of network modeling and the related work (§II-A). Then we motivate the design of xNet by analyzing the requirements of network models and the challenges behind them (§II-B).

A. Background and related work

Network Modeling. Network modeling is a critical building block to achieve efficient network management spanning the entire network life cycle, including planning, operation, and optimization, especially in the context of the future self-driving network [19], [20] or Digital Twin Network paradigm [21]. The role of the network model is to understand the complex relationships between the network performance metrics (e.g., delay, utilization, FCT) and the network characteristics (e.g., topology, traffic, configurations). Once constructed, it can facilitate offline network planning with performance predictions of "what-if" scenarios or mitigate the cost of online performance monitoring with real-time inference.

Network modeling with deep learning. Recently, network modeling with deep learning has been discussed in the literature [19], [22] but with few attempts. Two representative works have the closest focus to ours. First, Deep-Q [11] uses the deep generative models to infer the network QoS with the traffic matrix as input. However, the authors only consider the influence of traffic while ignoring other factors that may affect the network QoS. To deal with this problem, RouteNet [12] leverages graph neural networks to understand the complex relationship between topology, routing, and input traffic to produce accurate estimates of the per-source/destination pair mean delay and jitter. In this way, RouteNet can generalize to topologies and routing schemes not seen during training.

Although these solutions have great success in their focus, they still fall short of meeting the operators' requirements for expressiveness and granularity (Tab. I). In the following, we will analyze these requirements and the challenges they pose.

B. Requirements and challenges of network models

Expressiveness: To produce accurate predictions, the network model must be expressive enough to encompass as many related influence factors relevant to the network performance metrics as possible. Otherwise, it will inevitably fail to generalize to a wide range of network configurations. Among these factors, the network configurations can span a wide range of different operating levels from the end-host to in-network devices, e.g. congestion control at the host level, scheduling policies and ECN marking thresholds [23] at the queue level, bandwidth and propagation delay at the link level, buffer management policy in shared memory switches [24] at the device level, and finally the topology and routing scheme at the global level. Furthermore, these factors also have complex interactions with each other. As for existing solutions, RouteNet only considers global configurations while Deep-Q totally ignores these influence factors.

Challenge: The key challenge behind the requirement is the large state space, i.e., the number of potential scenarios the network model faces. This is because networking systems often consist of tens to hundreds of network nodes, and each node may contain several configurations, which results in a combinatorial explosion of potential states. A naive solution to building the network model is to construct a large neural network that takes a flat feature vector containing all the configuration information as input. However, this approach cannot scale to process information from an arbitrary number of nodes and configurations since the input size of the neural network is fixed. The resultant complexity of the neural network will increase with the number of configurations, making the neural network difficult to train and generalize.

Granularity: In different network scenarios, the granularity of the operators' focus can be quite different. In the wide-area network (WAN) scenario, operators mainly focus on the long-term average performance of aggregated traffic, where path-level steady-state modeling is often sufficient to guide the planning process (e.g., traffic engineering). However, fine-grained network performance observation is the constant pursuit of network operators and cloud providers to provide precise information about when and which flow gets disturbed [16]. This requires the network model to support flow-level time-series performance predictions. In this context, RouteNet fails since it can only estimate steady-state performance metrics at the path-level, while Deep-Q only supports time-series QoS inference but falls short in flow-level prediction.

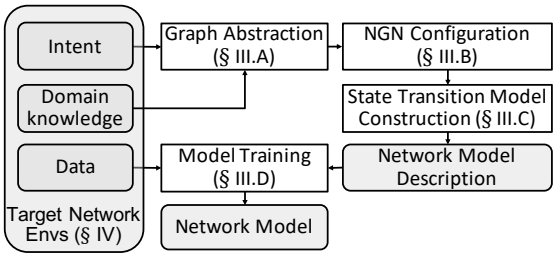


Fig. 1. xNet overview.

Challenge: The flow transmission behavior, unlike the aggregated traffic, may experience a cascade effect since it is typically governed by some control loops (e.g., congestion control). Once the control-related configurations (e.g., ECN threshold, queue buffer size) change during the flow transfers, the resulting traffic measurements of flows will dramatically vary, and thus the afore-measured traffic status cannot reflect the changing results. Therefore, flow-level prediction could be more difficult than QoS inference from traffic measurements. The network model need to take the flow demands rather than the traffic measurements as input to predict flow-level performance (e.g., flow completion time) in "what-if" scenarios.

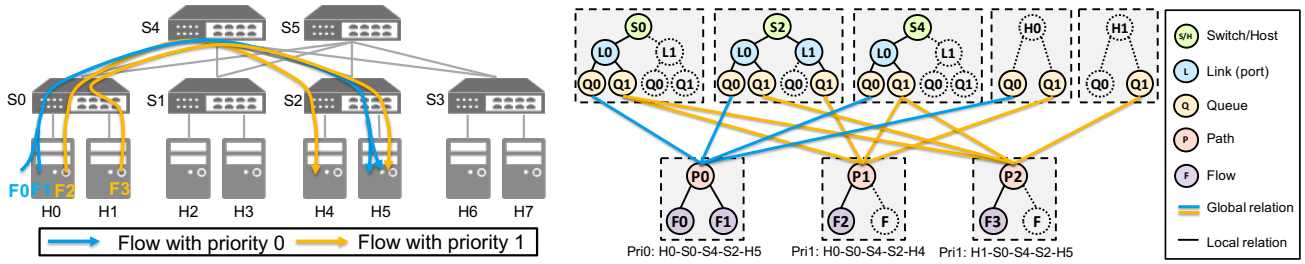
III. DESIGN

xNet meets the two requirements with a single framework and can be instantiated to model different network scenarios. Fig. 1 shows the overview of our xNet framework. A typical workflow of xNet is summarized as follows.

- Before the modeling, the operator needs to determine the concerned network configurations and modeling granularity based on the target network problem.
- According to the domain knowledge provided by the network expert, xNet abstracts the network system as a relation graph to represent the complex relationship between different network entities (§III-A).
- xNet uses a configurable GNN block to construct the network model and determines the form of the aggregation functions based on the properties of the relationships (§III-B). The above two steps relate to the expressiveness perspective (i.e., configuration modeling).
- xNet uses a recurrent form of the GNN to model network state transitions by learning the difference between states in adjacent time steps (§III-C). This step relates to the granularity aspect (i.e., time-series modeling).
- Finally, the model can be established by training with collected training data (§III-D).

A. Networking system as a relation graph

In xNet, we represent the networking system as a *heterogeneous relation graph* to provide a unified interface to model



(a) Target DCN environment in physical view: F0 and F1 have priority 0 while F2 and F3 have priority 1.

(b) Abstracted graph representation: The node of each color corresponds to a type of network entity; each shaded block represents a collection of nodes with local relationships; the colored edges between shaded blocks model the global relationships.

Fig. 2. A DCN example for graph abstraction: the target network environments (Fig. 2(a)) can be abstracted with a heterogeneous relation graph (Fig. 2(b)).

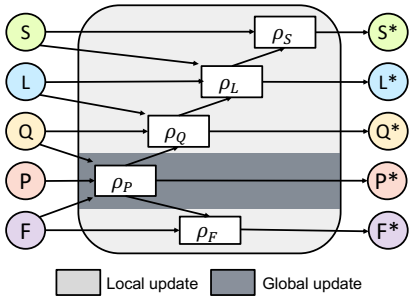


Fig. 3. Networking Graph Network block.

various network configurations and the intricate relationships between them. We map the network entities relevant to performance as graph nodes with the associated features. Graph edges connect nodes regarded to be directly related. Note that one could build the graph with a single type of node and use categorical features (e.g., one-hot vector) to represent distinct entities. However, in this way, the update function (§III-B) of the single node must learn to model various kinds of interactions between entities, which limits the combinatorial generalization ability of GNN.

The heterogeneous nodes represent different network entities with their own properties or configurations as the features. There are two kinds of nodes in our graph. The **physical** nodes represent the concrete network entities that have **local** configurations (e.g., switch with buffer size). The **virtual** nodes represent performance related entities (e.g., flow, path), so that the final performance (e.g., FCT, path latency) can be attached to this graph. The edges reflect the relationship between entities and may be utilized to embed the relational inductive bias from domain knowledge. For example, when the queue scheduling policy is active, the queue node can connect to its port but does not need to connect to other ports because they are not directly correlated. Similarly, the edges can be used to model the **global** configurations (e.g., topology, routing scheme). The path node, for instance, can connect to all of the queue nodes it passes through.

In the following, we take FCT prediction in DCN as an example problem to provide a concrete case to show how xNet builds the graph. To fully demonstrate the expressiveness of xNet, we take many configurations into account, which are usually not required though. Once the graph is built, xNet uses it to create the network model with GNN (§III-B).

A DCN example. As shown in Fig. 2(a), we'd like to consider a data center network with a leaf-spine topology. In each switch, the buffer is dynamically shared among all ports [24]. In each port, the queues are managed by some scheduling policies (e.g. strict priority, weighted round robin). Four flows are ready to be transmitted. The flows are congestion controlled with DCTCP [23] at the end-host server, and thus the ECN is enabled at each switch queue. The goal of the network model is to predict the flow completion time and path delay given the different configurations. We assume the path of each flow is known to the network model.

The graph abstraction of this DCN is shown in Fig. 2(b). To fully model the network characteristics at different levels, we use five types of nodes, including switch $S = \{s_i\}$, port(link) $L = \{l_i\}$, queue $Q = \{q_i\}$, path $P = \{p_i\}$, and flow $F = \{f_i\}$. To model the local relationships incurred by local configurations, we connect each node to other nodes that may have a direct impact on it (black edges). Specifically, the queue node Q, which has the features ECN threshold and priority weight, is linked to its port node L, which has the feature scheduling policy, because the scheduling policy controls the available bandwidth of queues. The port node P is connected to its switch node S whose feature contains the control factor α of dynamic buffer management policies since the buffer management policy operates at the port level. The flow node F is connected to the path node P it travels through.

As a result, the local relationships can be expressed as several independent tree-like structures (shaded blocks) without cross edges. To model the global relationships (i.e., routing), the path nodes P are connected with the queue nodes Q they traverse (colored edges). These edges connect different local trees, so that ensures the connectivity of the graph. Note that it is not required to instantiate all of the physical nodes because some of them may have no effect on the desired performance, i.e., not all devices are traversed by flows.

B. Message-passing on the heterogeneous graph

In xNet, we use Networking Graph Networks (NGN) as the basic building block for our network model, which is a customization of graph networks [18]. As shown in Fig.3, the NGN block is defined as "graph-to-graph" modules with heterogeneous nodes. The NGN block takes an attributed graph as input and, after a series of message-passing procedures,

Algorithm 1: Networking graph network, NGN

Input: Graph, $G = (S, L, Q, P, F)$
// Global relationship update
for each path p_i in P **do**
  Aggregate flows $\hat{f} = \Sigma_{f_j \in N(p_i)} f_j$
  Aggregate queues $\hat{q} = \text{RNN}_{q_j \in N(p_i)}(q_j)$
  Update $p_i^* = \phi_P(p_i, \psi^{F \to P}(\hat{f}), \psi^{Q \to P}(\hat{q}))$
// Local relationship update
for each flow f_i in F **do**
  Update $f_i^* = \phi_F(f_i, \psi^{P \to F}(p_j^*)), p_j^* \in N(f_i)$
for each queue q_i in Q **do**
  Aggregate paths $\hat{p} = \Sigma_{p_j^* \in N(q_i)} p_j^*$
  Update
  $q_i^* = \phi_Q(q_i, \psi^{L \to Q}(l_j), \psi^{P \to Q}(\hat{p})), l_j \in N(q_i)$
for each link l_i in L **do**
  Aggregate queues $\hat{q} = \Sigma_{q_j^* \in N(l_i)} q_j^*$
  Update
  $l_i^* = \phi_L(L_i, \psi^{S \to L}(s_j), \psi^{Q \to L}(\hat{q})), s_j \in N(l_i)$
for each switch s_i in S **do**
  Update $s_i^* = \phi_S(s_i, \psi^{L \to S}(l_j^*)), l_j^* \in N(s_i)$
Output: Graph, $G^* = (S^*, L^*, Q^*, P^*, F^*)$

outputs another graph with changed attributes. These attributes are the features of nodes, expressed in fixed-length tensors. This block can also be used in recurrent forms (§III-C).

A single feedforward NGN pass can be viewed as one step of message-passing on the graph [7]. Typically, NGN first performs global updates, and then nodes in each tree structure (see Fig. 2(b)) conduct the local updates independently. The circular dependencies among different update operations can be resolved with multiple rounds of message passing [17].

In each round of message passing, each node x of type X aggregates the messages from its neighbors $N(x)$ and updates its internal states according to a configurable function ρ_X. An NGN block contains several configurable functions, each of which has three kinds of sub-functions: the aggregation function, the conversion function, and the update function. These functions can be implemented using standard neural networks and shared with nodes of the same type.

We use the above DCN scenario as an example shown in Algorithm 1. First, the node x gathers the messages from its neighbors and aggregates those of the same type with the corresponding aggregation function. Aggregation can take the form of summation or RNN, which models the neighbors' permutation-invariant property (e.g., ports as to a switch) or sequential dependency (e.g., queues traversed by the path).

Then, to deal with heterogeneous nodes, the aggregated messages of type Y are transformed with a type-to-type conversion function $\psi^{Y \to X}$ before being put into the update function. By doing this, information from heterogeneous nodes can be mapped into the same hidden space that corresponds to the target type of node, so that they can be operated in a unified way in the update function without limiting the modeling capability of GNN.

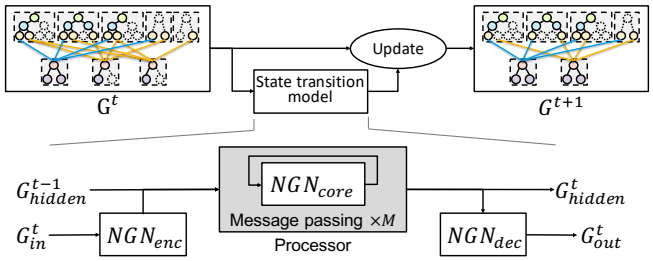


Fig. 4. State transition model of xNet.

Algorithm 2: NGN forward prediction algorithm.

Input: trained NGN_{core}, NGN_{enc} and NGN_{dec}.
Input: dynamic graph G_d^t at the current time step.
Input: static graph G_s and hidden state G_h^{t-1}.
Input: maximum message-passing number M.
Build input graph $G_{in} = \text{concat}(G_s, G_d^t)$
Obtain encoded input graph $G_{enc} = NGN_{enc}(G_{in})$
Obtain graph hidden state G_h^{t-1} **if** *not exist* **then**
  Obtain EMPTY graph hidden state
Obtain graph $G^0 = \text{concat}(G_h^{t-1}, G_{enc})$
for $i = 1 : M$ **do**
  Obtain updated graph $G^i = NGN_{core}(G^{i-1})$
Obtain graph after message-passing G^M
Update graph hidden state $G_h^t = G^M$
Obtain decoded output graph $G_{out} = NGN_{dec}(G^M)$
Obtain predicted delta of dynamic nodes
$\Delta N_d = G_{out}.\text{nodes}$
Obtain next graph G_d^{t+1} by updating N_d^t with ΔN_d
Output: next dynamic graph G_d^{t+1}.

Finally, the node takes the transformed messages and its own state in the last round as input to update the state with the update function ϕ_X. After the given rounds of message passing, we can use a readout function to predict the final performance metric using as input the hidden state of related nodes (e.g., path node as to end-to-end delay).

C. State transition learning

The network models are required to support fine-grained prediction at a short time scale and transient state prediction (e.g., flow state). To achieve this, xNet uses the recurrent form of the NGN block to learn to predict future states from the present ones. It operates on one time-step and has the "encode-process-decode" structure. These three components are NGN blocks with the same graph but different neural network parameters. They process the input feature sequentially, where the output of the first block becomes the input of the second. Fig. 4 depicts the structure of xNet's state transition model.

Encoder. The "encoder" NGN block embeds the input state Gt_in into a latent graph by encoding different nodes independently. It ignores the relationship between nodes and does not perform message passing. After encoding, the input state of each node is transformed into a fixed-dimension vector.

Processor. The "processor" NGN block performs M rounds of message-passing steps. The input to the processor is the

concatenation of the encoder's output and the previous output of the processor (i.e., the hidden state G^{t-1}_{hidden}).

Decoder. The "decoder" NGN block, i.e., the readout function, extracts the dynamic information of the final hidden graph by independently decoding different nodes. It also ignores the relationship between nodes as the encoder. After decoding, the output state G^t_{out} contains both the current performance metric and the state delta Δ used to update the next-step state.

xNet in this form can be treated as a learnable network simulator Θ, which computes the dynamics information with a parameterized function approximator. At each step, it takes the current (potentially historical) state G^t of the networking system and tries to predict the current performance and the delta Δ between the current and next state. Then the next state can be obtained using this state difference. To summarize, it behaves as $G^{t+1} = \Theta(G^t)$. The forward procedure of this state transition model is shown in Algorithm 2.

To support state transition modeling, we distinguish the static and dynamic characteristics of the network system, and express them as different graphs. The static graph G_s contains the static configurations of the system, including physical node configurations (e.g., priority of queue, buffer size of switch), and virtual node configurations (e.g., flow size, start time). The dynamic graph G_d contains the temporary state of the system. This is mainly related to the virtual nodes, which include the remaining size and lifetime of the flow or the end-to-end delay of the path. Additionally, when considering the dynamic configurations (e.g., time-varying ECN threshold), the actions taken (i.e., new configuration) should be put into the dynamic graph and inputted at each time step.

D. Training

We train our model by supervising the per-node output features produced by the decoder using an L2 loss between the predicted value and the corresponding ground truth values. To generate a long-range rollout trajectory, we iteratively feed updated absolute state predictions back into the model as input. As data pre and post-processing steps, we normalized the inputs and outputs to the NGN model.

IV. Evaluation with use cases

xNet can be instantiated to model different network scenarios with different concerns. In this section, we apply xNet to three concrete use cases to show its modeling capability and wide application scope. Note that the network models used for each case are different, and are trained and evaluated separately. The key properties and requirements of the three use cases are summarized in Table II.

A. Implementation

We implement xNet using Tensorflow [25] and the Graph Nets library [26], where the components of the xNet framework are implemented with standard deep learning building blocks. Unless otherwise specified, all three network models are built with the following specifications. The RNN used in aggregation is a 1-layer GRU [27] with 64 neurons. The conversion function is a 1-layer multi-layer perceptrons (MLP). The update functions in NGN are all 3-layer MLP with 64 neurons followed by layerNorm [28]. The activation function is Leaky Relu [29]. We use the Adam [30] optimizer to minimize the loss with a learning rate of 0.001. The maximum message-passing number M is set to 3. The state transition model of xNet works in a "dynamic" graph manner, where the number of nodes (e.g., the number of flows) in the graph can be different between time steps. We train our models with an Nvidia RTX 3080 GPU.

B. Temporal QoS inference in DCN

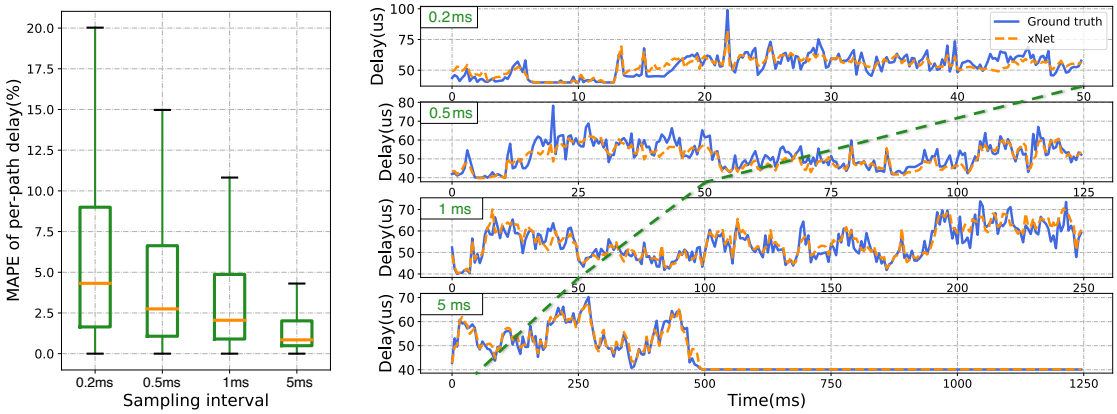
Task: This case intends to validate whether xNet can accurately perform time-series inference and generalize to unseen configurations, reflecting the application of online performance monitoring [11]. The network model is required to predict the path-level latency evolution in time series given the real-time measurements of traffic volumes on a path, where the configurations of traffic loads, queue buffer size, and ECN marking threshold can be varied.

Experiment setup: We leverage the data generated by the packet-level simulator NS3 [6] as the ground truth. We consider a data center environment with the topology of a PoD of a 4-port Fat-Tree [31], where there are 20 paths in total. The capacity of each link is 10 Gbps. All links have a 10 μs propagation delay, which gives a 40 μs maximum base RTT. As for traffic workloads, we use widely accepted and publicly available data center traffic traces Web Search [23] to generate flows following the all-to-all pattern. We adjust the flow generation rates to set the average link loads ranging from 20% to 80%. The flows are congestion controlled by the host using DCTCP [23]. For switch configurations, we only consider the buffer size and ECN marking threshold at the queue level. The values range from 0.05 to 0.5MB and 6 to 60KB, respectively. Note that the ECN threshold here ranges from around 10% to 100% of the bandwidth-delay production, which is consistent with the typical suggested configurations [23], [32], [33].

For each simulation, we chose the three configurations at random from a uniform distribution. We constrain the buffer size to exceed the ECN threshold. To measure the path latency, we calculate the average packet delay along the path during a predefined sampling interval, thus determining the length of a time step. To show the necessity of temporal performance modeling, we run the simulations with four sampling intervals $\{0.2,\ 0.5,\ 1,\ 5\}$ ms and train a dedicated model for each interval. For training/testing, we use a collection of samples from 30/10 simulations with more than 30000/10000 flows generated. With the different sampling intervals, the total number of samples varies, ranging from around 176000/83000 to 7000/3300. Note that the flows and configuration combinations generated in the testing are **unseen** by the model during the training process. We report the mean absolute percentage error (MAPE) between the predicted latency and the ground truth over all paths as the evaluation metric.

TABLE II
PROPERTIES AND REQUIREMENTS OF THREE USE CASES.

Use case	Target environment	Metrics & Granularity	Configurations	Temporal prediction	Traffic measurement	Application scenario
Temporal QoS inference	DCN	Path-/flow-level delay	Traffic, buffer size, ECN	✓	w/	Online monitoring
FCT prediction	DCN	Path-/flow-level delay/throughput, FCT	Traffic, buffer size, ECN	✓	w/o	Simulation for "What-if" scenarios
Steady-state QoS inference	WAN	Path-level delay	Traffic, topology, routing, queue scheduling policy	×	w/	Offline planning



(a) Per-path delay predicted by xNet under different sampling intervals.

(b) An example: time-series of path latency with different sampling intervals under the same traffic trace. The same time period (i.e., 0-50ms) is marked with the green dotted line. For the sampling interval of 5ms, the trace is finished after around 500ms.

Fig. 5. Results of temporal QoS inference in the DCN scenario.

Model: To build the graph, only the nodes in the type of queue and path are required. The features on queue nodes include the buffer size, ECN threshold, bandwidth, and link propagation delay[1]. The features on the path include the traffic volume and a one-hot vector that indicates the number of hops on this path. To enable time-series prediction, we leverage the state transition model to predict the changes in path latency between time steps. In particular, we use the state of the last five steps to predict the next state, to improve accuracy. The training process with a batch size of 64 takes about 2 hours.

Results: Fig. 5(a) shows the MAPE distribution of xNet's latency prediction in four experiments with different sampling intervals. First, xNet achieves high accuracy and maintains the mean/50th percentile MAPE below 7%/5%. Even for the tail performance, the error is limited to 20%, which shows the high generalization ability and worst-case performance of xNet. Second, the prediction error gradually decreases when the sampling interval becomes larger. This is because the larger the sampling interval, the more stable the network performance and thus the easier it is to predict. However, large sampling intervals would lose detailed information about transient network performance.

To show this, we present the time series of xNet's delay prediction and the ground truth of a randomly selected path in Fig. 5(b). We can see that a latency spike of near 100us (at around 22ms) in the resolution of a 0.2ms interval can only be found as 60us with a 5ms interval. Therefore, fine-grained time-series performance prediction is quite necessary, which confirms our motivation (§II-B). What's more, our model can successfully predict the evolution of path latency over time, which includes both the fluctuation and the steady-state.

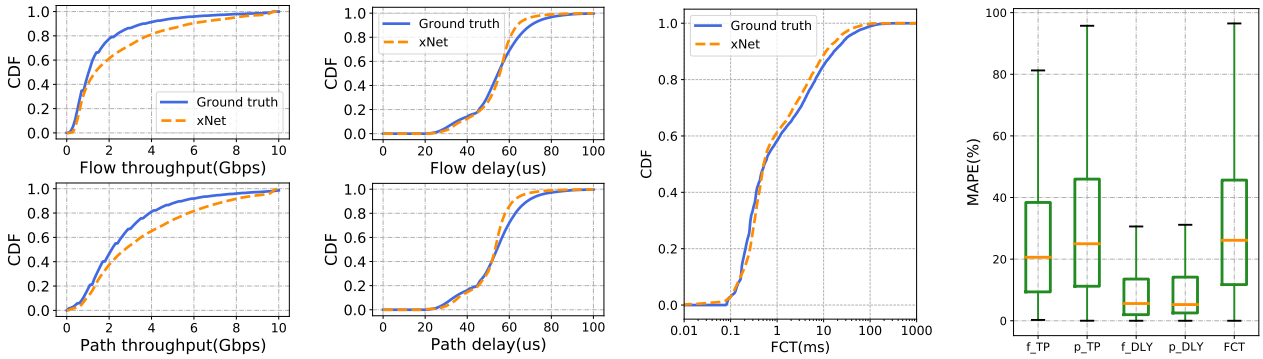
[1]Here, the queue and link are in a one-to-one correspondence, so we use the single node with their configurations to represent them.

C. FCT prediction in DCN

Task: With this use case, we aim to answer the question of whether xNet can provide the flow-level time-series modeling ability along with different configurations. Unlike the previous ones, in this case, the network model behaves like a simulator, which needs to predict FCT by taking as input only the flow descriptions without traffic measurements. Besides, it is also required to predict the path-level latency and throughput.

Experiment setup: The experimental setup is the same as that in §IV-B except that the sampling interval is set to 0.1 ms. We calculate throughput by dividing the received bytes during the interval by the interval length. For training/testing, we use the same 30/10 traffic traces as in the previous case, and the total number of samples is around 352000/166000. For the evaluation metric, we report the MAPE of predicted per-step performance metrics (i.e., delay and throughput) at both path and flow levels, as well as FCT.

Model: Nodes of the queue, path, and flow types are required to construct the graph. The features on flow nodes include the flow size, start time, remaining size, and lifetime, where the latter two are dynamic states. The model predicts the FCT by predicting the received data volume between time steps and updating the state of the remaining size and lifetime on each step. Once the remaining size is below zero, the model considers that the flow is complete and uses the lifetime as

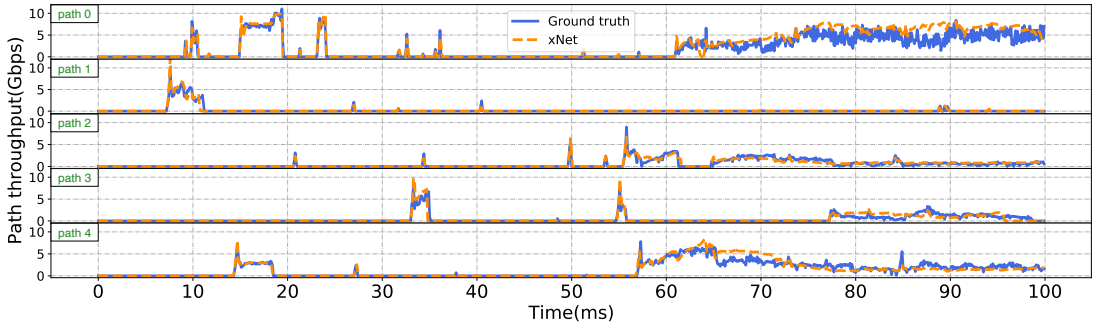


(a) CDF of path-level and flow-level throughput.

(b) CDF of path-level and flow-level latency.

(c) CDF of FCT.

(d) MAPE with boxplot.



(e) Path-level throughput trajectory predicted by step-by-step state transition.

Fig. 6. Results of the FCT prediction scenario in DCN. xNet behaves as a learned simulator that iteratively predicts states of path and flow.

the FCT prediction. A flow node exists in the graph only if the flow is thought to be alive by the model. The model also takes the per-step metrics at the path and flow level as dynamic states to predict. This model is trained by the supervision of the received bytes and the per-step performance metrics at both path and flow levels. We use the state of the five steps to infer the state of the next step. The training process with a batch size of 256 takes about 48 hours.

Results: Firstly, we present the CDF of per-step performance prediction with the corresponding ground truth. As shown in Fig. 6(a) and Fig. 6(b), the distribution of predictions from xNet can well match that of ground truth, especially for the delayed predictions, which validates the xNet's temporal prediction ability. Note that there are no traffic volume measurements provided, so the network model needs to infer the traffic conditions with the rollouts from scratch. To investigate the accuracy of long rollout, we present the CDF of FCT in Fig. 6(c). We can find that the distribution of FCT predictions well matches that of the ground truth with a Pearson correlation of 0.9 (not shown), which confirms that xNet has the ability to model flow dynamics.

Except for the distribution, to provide a faithful comparison, we also report the MAPE of each metric shown in Fig. 6(d). The errors of throughput and FCT are much higher than in the last case, but still lie within an acceptable range such that the 50th percentile error is all below 30%. This is because the predictions are obtained from the step-by-step state transition, where the errors will accumulate along with the rollout.

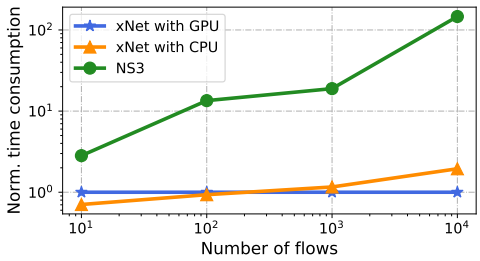


Fig. 7. Speedup of xNet compared to the NS3 simulator with different flow numbers. For clarity, we normalize the time consumption of each scheme against that of xNet with GPU.

Finally, we present an example of rollout trajectories of five randomly selected paths in Fig. 6(e). As expected, the model can make near-perfect predictions for traffic that lasts for a short time since the model does not need to deal with the long-time dependencies and cumulative errors. When the traffic lasts for a longer time, the prediction becomes more inaccurate. Nevertheless, xNet successfully achieves flow-level temporal state prediction and can generalize to different configurations, at least from the perspective of the overall distribution.

Inference performance: To show the efficiency of xNet, we compare the time consumption of the NS3 simulator and that of xNet to simulate the same traffic traces. We vary the flow number in the simulation, ranging from 10 to 10,000 while maintaining the basic experiment setup unchanged. Note that even with the same network topology, the number of flows will influence not only the number of time steps but also the number of nodes in the graph, thus impacting the per-step time

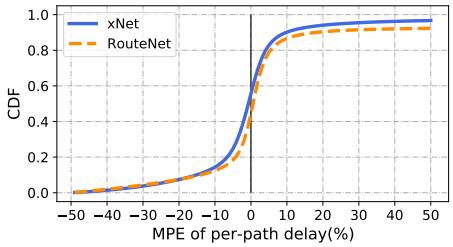


Fig. 8. MPE of steady-state QoS inference in a WAN scenario.

consumption. xNet and NS3 are tested on an Intel i9-10980XE 3.00GHz CPU with a single core. xNet is also tested on the GPU. For inference, we set the batch size to 1.

As shown in Fig. 7, the time consumption of each scheme is normalized by that of xNet with GPU, so that it can directly reflect the relative speedups. As expected, xNet is much faster than NS3 and achieves up to two orders of magnitude of acceleration. This is because NS3 needs to simulate all the packet-level events whose number will tremendously increase with more flows. On the contrary, the network model of xNet operates on the flow segments with a fixed time length and can process multiple flows/paths in a single forward pass, thus being less influenced by an increased number of flows.

D. Steady-state QoS inference in WAN.

Task: In this last use case, we intend to show that xNet can efficiently model and generalize to both global and local configurations, which is designed to reflect the usage of offline network planning [11]. The network model needs to infer the end-to-end per-source/destination mean delay at a steady state in a wide variety of network topologies, routing schemes, and traffic intensities. In particular, the devices are configured with different scheduling policies.

Experiment setup: We use the public network modeling dataset [34], [35] generated by OMNET++ simulator [36]. The training dataset contains samples simulated in the NSFNET (14 nodes) and GEANT2 (24 nodes) network topologies, while the test dataset is simulated in the RedIRIS (19 nodes) topology. Each switch port is implemented with one of the following scheduling policies, i.e., Strict Priority, Weighted Fair Queueing, or Deficit Round Robin, with three priority queues. Each sample of the dataset includes a traffic matrix where flows may have 3 different Types of Services (ToS) associated with one of the three priority queues. All the packets in a path have the same ToS. Packets are generated following a Poisson distribution. We use a variant of RouteNet [37] as the comparison baseline. We report the mean percentage error (MPE) between the predicted delay and the ground truth as the evaluation metric.

Model: To build the graph, the nodes need have the type of queue, link (port), and path. We attach the features of each network entity to the corresponding nodes. Although this scenario does not require temporal inference, we still leverage the encoder-processor-decoder architecture of our state transition model for better performance.

Results: As shown in Fig. 8, xNet and RouteNet achieve low prediction MPE of around 12%. Note that the topologies used in the test set were never included in the training. The high accuracy reveals the ability of xNet to generalize well to both global configurations (i.e., unseen topologies) and local configurations (i.e., complex scheduling policies).

V. Discussion and Future Work

Data collection. The accuracy and generalization ability of the learning-based network models strongly depend on the amount, quality, and diversity of the training data. Operators can quickly collect enough data to accurately reflect real-world distribution with production network systems.However, operators do not frequently adjust configurations on different network elements, leading to limited combinations of traffic patterns and configurations in the collected dataset. It is also unrealistic to build dedicated testbeds due to their high cost.

In this context, simulation data can be a good alternative. However, the obstacle of the well-known "simulation-to-reality" gap stands in front of all the learning-based network models that care for network configurations, including RouteNet, xNet, and potential future ones. Recent advances in transfer learning [38] and few-shot learning [39] can hold the key to a cure. One can first train the network model with plenty of diverse data (may not be of high quality) from simulation, and then transfer the model with few but high-quality data from the real-world collection. We leave this for future work.

Optimization. Learning-based network models can quickly produce accurate evaluations of the network performance with different configurations, so they can be used to explore better schemes to optimize the traffic transmission. This optimization procedure is often driven by some search-based methods (e.g., grid/random search). They first generate some candidate configurations and then evaluate them with the network model until the predefined optimization objective is met. Indeed, xNet can also be applied to this paradigm. What's more, xNet opens a new door for efficient sequential decision-making in networking systems. The state transition model can serve to predict the next state so that enables model-based control (e.g., model predictive control [40]) or model-based reinforcement learning [41]. We leave this for our future work.

VI. Conclusion

In this paper, we propose xNet, a GNN-based data-driven network modeling framework. Our high-level goal is to provide a general method for network modeling that allows varied network properties to be properly described and generalized consistently. xNet represents the networking entities and their configurations as nodes in a relation graph and learns the relationship between them via message passing. We implement xNet and instantiate it with three use cases. The results indicate that xNet can learn to efficiently model various networking scenarios while achieving up to a 100x speedup over the traditional packet-level simulator.

Acknowledgment

This work was supported in part by the NSFC Project of China (No. 6213000078 and No. 61872211).

REFERENCES

[1] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar, "Answering what-if deployment and configuration questions with wise," in *ACM SIGCOMM*, 2008.

[2] "Ip-sla." https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipsla/configuration/15-mt/sla-15-mt-book/sla_icmp_echo.html, accessed Jan 10, 2022.

[3] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM*, 2018.

[4] P. Almasan, J. Suárez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case," *arXiv preprint arXiv:1910.07421*, 2019.

[5] F. Ciucu and J. Schmitt, "Perspectives on network calculus: no free lunch, but still good value," in *ACM SIGCOMM*, 2012.

[6] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010.

[7] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning (ICML)*, 2017.

[8] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *International Conference on Machine Learning (ICML)*, 2018.

[9] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, "Learning mesh-based simulation with graph networks," *arXiv preprint arXiv:2010.03409*, 2020.

[10] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, "Learning to simulate complex physics with graph networks," in *International Conference on Machine Learning (ICML)*, 2020.

[11] S. Xiao, D. He, and Z. Gong, "Deep-q: Traffic-driven qos inference using deep generative network," in *Proceedings of the 2018 Workshop on Network Meets AI & ML*, 2018, pp. 67–73.

[12] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in sdn," in *ACM SOSR*, 2019.

[13] A. Mestres, E. Alarcón, Y. Ji, and A. Cabellos-Aparicio, "Understanding the modeling of computer network delays using neural networks," in *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, 2018, pp. 46–52.

[14] M. Wang, Y. Cui, S. Xiao, X. Wang, D. Yang, K. Chen, and J. Zhu, "Neural network meets dcn: Traffic-driven topology adaptation with deep learning," *ACM SIGMETRICS*, 2018.

[15] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," in *ACM SIGCOMM*, 2006.

[16] Y. Zhou, C. Sun, H. H. Liu, R. Miao, S. Bai, B. Li, Z. Zheng, L. Zhu, Z. Shen, Y. Xi *et al.*, "Flow event telemetry on programmable data plane," in *ACM SIGCOMM*, 2020.

[17] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[18] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[19] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.

[20] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," *arXiv preprint arXiv:1710.11583*, 2017.

[21] "Concepts of digital twin network," https://tools.ietf.org/html/draft-zhou-nmrg-digitaltwin-network-concepts-06, accessed Jan 10, 2022.

[22] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Network*, 2017.

[23] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *ACM SIGCOMM*, 2010.

[24] A. K. Choudhury and E. L. Hahne, "Dynamic queue length thresholds for shared-memory packet switches," *IEEE/ACM Transactions On Networking (ToN)*, 1998.

[25] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *USENIX OSDI*, 2016.

[26] "Graph nets library, deepmind, 2018." https://github.com/deepmind/graph_nets, accessed Jan 10, 2022.

[27] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[28] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[29] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM*, 2008.

[32] W. Bai, S. Hu, K. Chen, K. Tan, and Y. Xiong, "One more config is enough: Saving (dc) tcp for high-speed extremely shallow-buffered datacenters," *IEEE/ACM Transactions on Networking (ToN)*, 2020.

[33] M. Alizadeh, A. Javanmard, and B. Prabhakar, "Analysis of dctcp: stability, convergence, and fairness," *ACM SIGMETRICS*, 2011.

[34] "Network modeling datasets." https://github.com/knowledgedefinednetworking/NetworkModelingDatasets, accessed Jan 10, 2022.

[35] "Graph neural networking challenge 2020." https://bnn.upc.edu/challenge2020/, accessed May 25, 2020.

[36] A. Varga, "Omnet++," in *Modeling and tools for network simulation*. Springer, 2010, pp. 35–59.

[37] M. Ferriol-Galmés, J. Suárez-Varela, P. Barlet-Ros, and A. Cabellos-Aparicio, "Applying graph-based deep learning to realistic network scenarios," *arXiv preprint arXiv:2010.06686*, 2020.

[38] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering (TKDE)*, 2009.

[39] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele, "Meta-transfer learning for few-shot learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[40] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[41] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.