xNet: Modeling Network Performance With Graph Neural Networks

Sijiang Huang, Yunze Wei, Lingfeng Peng, Mowei Wang, Linbo Hui, Peng Liu, Zongpeng Du, Zhenhua Liu, and Yong Cui, *Member, IEEE*

***Abstract*— Today's network is notorious for its complexity and uncertainty. Network operators often rely on network models for efficient network planning, operation, and optimization. The network model is responsible for understanding the complex relationships between network performance metrics (e.g., delay and jitter) and network characteristics (e.g., traffic and configuration). 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). It is worth noting that xNet is not a dedicated network model designed for a specific network scenario 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 functions between time steps and rolls them out to obtain the full fine-grained prediction trajectory. We implement and instantiate xNet with three use cases. The experimental results show that xNet can accurately predict different performance metrics (i.e. temporal and steady-state QoS) in different scenarios, with performance comparable to state-of-the-art domain-specific models. Compared with traditional packet-level simulators, xNet achieves a speed improvement of more than two orders of magnitude, demonstrating its promising application in real-time optimization of network configurations.**

***Index Terms*— Network modeling, network performance inference, graph neural networks.**

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.

Manuscript received 23 February 2023; revised 21 August 2023 and 28 September 2023; accepted 27 October 2023; approved by IEEE/ACM Transactions on Networking Editor J. S. Sun. Date of publication 7 November 2023; date of current version 18 April 2024. This work was supported in part by NSFC Project of China under Grant 62132009 and Grant 62221003; and in part by Tsinghua University-China Mobile Communications Group Company Ltd., Joint Institute. An earlier version of this paper was presented in part at the IEEE INFOCOM 2022 [DOI: 10.1109/INFOCOM48880.2022.9796726]. *(Corresponding author: Yong Cui.)*

Sijiang Huang, Yunze Wei, Lingfeng Peng, Mowei Wang, Linbo Hui, and Yong Cui are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: cuiyong@tsinghua.edu.cn).

Peng Liu and Zongpeng Du are with the China Mobile Research Institute, Beijing 100032, China.

Zhenhua Liu is with Huawei Technologies Company Ltd., Beijing 100085, China.

Digital Object Identifier 10.1109/TNET.2023.3329357

To achieve this goal, practitioners often construct *network models* to provide guidance for network planning, operation, and optimization. A network model is tasked to predict how the network performance metrics (e.g., throughput, delay) 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 relying on 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 operators with predicted performance metrics under different network configurations, even if the network has yet been physically built. Benefiting from the development of deep learning, researchers have recently utilized Deep Reinforcement Learning (DRL) techniques to effectively optimize network performance [3], [4]. However, the aforementioned methods may perform unreliable trial-and-error explorations in real networks, potentially disrupting existing services and causing intolerable performance degradation. To avoid the impact on the performance of real networks, an accurate and real-time network model can serve as a secure environment for network optimization, enabling new methods including DRL.

In this context, much effort has been devoted to developing rapid and precise network performance models with acceptable costs. 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 real networks and will lead to inaccurate results, while the high computational costs of the latter make 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], [8], [9], [10], which is promising for building lightweight yet accurate network models. Previous learning-based attempts [11], [12], [13], [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.

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	✓	✓	✓	✓

However, state-of-the-art learning-based proposals [11], [12], [13], [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., the buffer size of switches) and network-wide (i.e., global) configurations (e.g., routing schemes). In order to obtain accurate predictions, the network model must be expressive enough to describe all the factors above and provide "knobs" to adjust the corresponding features to support the evaluation in "what-if" scenarios.

Furthermore, the prediction granularity of the network model should be as fine as possible. From the spatial view, the network model is much desired to provide 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 generally more interested in transient network performance anomalies (e.g., throughput drop-offs and delay spikes) [16], which can typically only be detected by continuous performance monitoring over time. The high-cost nature of performance monitoring 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 schemes on 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 combines the advantages of deep learning and neural networks to gain the potential benefits of accuracy and speed by learning directly from the raw data. 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 basic network model (§III) that can be configured and instantiated to model a variety of network scenarios with different focuses (§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 (GNN) [17] to model the intricate relationships among various network entities (e.g., switch and router) and different configurations (e.g., ECN and buffer size). GNNs carry strong relational inductive bias and therefore inherently support relational reasoning and combinatorial generalization [18]. Note that the reason for using GNN does not stem from the fact that the computer network topology can be represented visually as a graph, but rather comes from its ability to model relationships that can express the relationships between different network entities and attributes.

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 demonstrate the effectiveness (i.e., accuracy and speed) 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. We also dive deeply into xNet's long-term performance, statistical and ranking accuracy, and its performance in special circumstances to demonstrate its practice use (§V). 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 ∼0.9.
- For steady-state QoS inference, xNet slightly outperforms domain-specific approach in modeling path-level delay/jitter in 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. xNet can be used to predict longer scenarios without evident accumulation of errors, accurately rank different network parameters by their performance, and maintain considerable performance under previously unseen many-to-one incast traffic.

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 based on three use cases of different metrics and scenarios. (§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 part of efficient 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 close 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. 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 switch queue level, bandwidth and propagation delay at the link level, buffer management policy in shared memory switches [24] at the device level, and the topology and routing scheme at the global level, all of which 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 multiple 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. Achieving expressiveness not only increases the demands on the hard-to-collect datasets but also makes neural networks 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.

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 needs 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 various network scenarios. Fig. 1 presents an overview of our xNet framework. The typical workflow of xNet is summarized as follows.

- Prior to 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 into

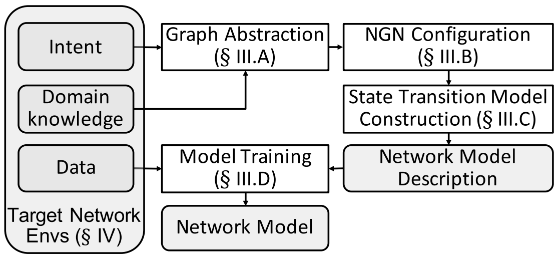

Intent
Domain knowledge
Data
Target Network Envs (§ IV)
Graph Abstraction (§ III.A)
NGN Configuration (§ III.B)
State Transition Model Construction (§ III.C)
Network Model Description
Model Training (§ III.D)
Network Model

Fig. 1. xNet overview.

a relation graph, representing the complex relationship between different network entities (§III-A).

- xNet uses a configurable GNN block (Networking Graph Network, NGN) to construct the network model and determines the form of the aggregation functions based on the properties of the relationships (§III-B). The aforementioned two steps are related to the *expressiveness* of the model (i.e., configuration modeling).
- xNet uses a recurrent form of the GNN to model transitions in network states by learning the difference between states at consecutive time steps (§III-C). This step is related to the modeling *granularity* (i.e., time series modeling).
- Finally, the model is trained on collected data (§III-D) and then can be used for performance inference.

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 various network configurations and the intricate relationships between them. We map the network entities relevant to performance as graph nodes with associated features. Graph edges connect nodes that are considered to be directly related. Note that although one could build the graph with a single type of node and use categorical features (e.g., one-hot vector) to represent distinct entities, using such a scheme would require the update function (§III-B) of the single node must learn to model various kinds of interactions between entities, which not only increases the burden of training, but also limits the scalability of the GNN model.

The heterogeneous nodes represent various 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., switches with buffer size). The **virtual** nodes represent performance-related entities (e.g., flow, path), to which the final performance metrics (e.g., FCT, path delay) can be associated within the graph. The edges reflect the relationships between entities and can be utilized to embed domain knowledge. For example, when the queue scheduling policy is activated, 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 DCNs as an example to illustrate how xNet builds the graph. To fully demonstrate the expressiveness of xNet, we take as many configurations that we can think of in this scenario into account, even though some of which are not always required. Once the graph is built, xNet can use it to create the network model with GNN(§III-B).

A DCN example. As shown in Fig. 2(a), we consider a data center network with a leaf-spine topology. Within each switch, the buffer is dynamically shared among all ports [24]. On each port, queues are managed by various scheduling policies (e.g., strict priority, weighted round robin). Four flows are ready to be transmitted. These flows are congestion controlled with DCTCP [23] at the end-host server, therefore 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.

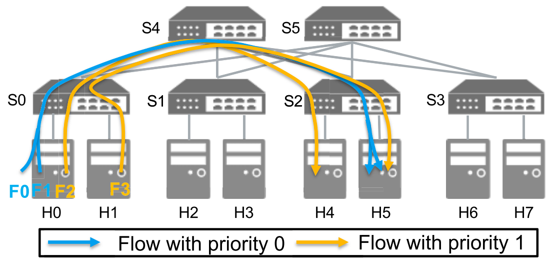
The graph abstraction of this DCN is shown in Fig. 2(b). To comprehensively model the network characteristics at different levels, we use five types of nodes, including switch $S = \{s_i\}$, port (link) $L = \{l_i\}$, queue $Q = \{q_i\}$, path $P = \{p_i\}$, and flow $F = \{f_i\}$. To model the local 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, as the buffer management policy operates at the port level. Additionally, the flow node F is connected to the path node P it travels through.

As a result, the local relationships can be represented as several independent tree-like structures (shaded blocks) without crossing 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, ensuring graph connectivity. Note that it is not necessary to instantiate all of the physical nodes when constructing a specific graph because some of them may have no effect on the desired performance, e.g., not all devices are traversed by flows.

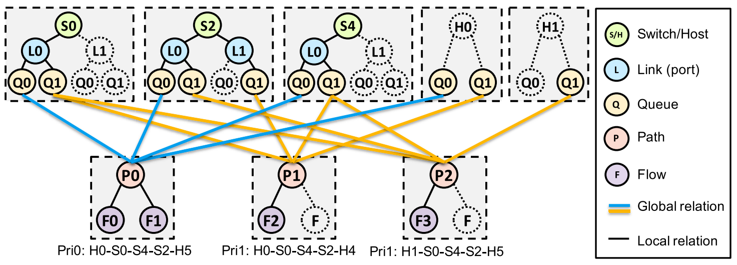
B. Message-Passing on the Heterogeneous Graph

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, 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 a single step of message-passing on the graph [7]. Typically, NGN first performs global updates, and then nodes in each tree structure



(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)).

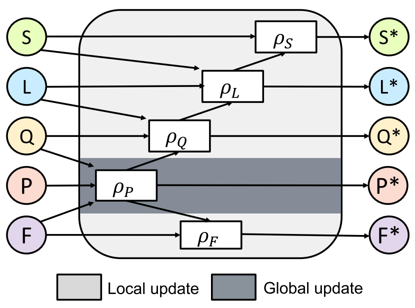


Fig. 3. Networking Graph Network block.

(see Fig. 2(b)) conduct the local updates independently. 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 comprises several configurable functions, each consisting of three types of sub-functions: the *aggregation* function, the *conversion* function, and the *update* function. These functions can be implemented with standard neural networks and shared among nodes of the same type.

We use the aforementioned DCN scenario as an example, as shown in Algorithm 1. Firstly, the node x gathers the messages from its neighbors and aggregates those of the same type with the corresponding aggregation function. Aggregation can take a simple form such as summation that models the neighbors' permutation-invariant property (e.g., ports as to a switch), or some complex functions like Recurrent Neural Network (RNN) for 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.

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).

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, $G^* = (S^*, L^*, Q^*, P^*, F^*)$
```

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 blocks 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 G_{in}^t 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.

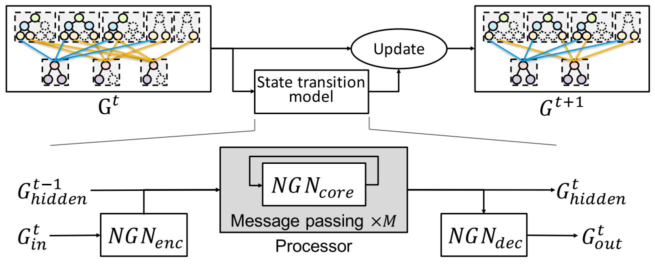


Fig. 4. State transition model of xNet.

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. In 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 determined 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.

As a data-driven model, xNet supports state transition modeling given different network characteristics, which should be selected based on their impact on network performance as well as whether obtaining such indicators is within the measurement capability of the system. xNet does not have stringent requirements on which input characteristics are mandatory, instead, it just makes the best with what it can get. On the other hand, the output characteristics are expected to be the network state information or quality of service predicted by the model. It is worth noting that the input and output characteristics vary in different scenarios. 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 system configurations as inputs, including physical node configurations (e.g., priority of queue, buffer size of switch), and virtual node configurations (e.g., flow size, start time). On the other hand, the dynamic graph G_d contains the temporary state of the system, including physical node states (e.g., link throughput, queue length), and virtual node states (e.g., remaining size and lifetime of the flow, 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.

Algorithm 2 NGN Forward Prediction Algorithm

```
Input: trained $NGN_{core}$, $NGN_{enc}$ and $NGN_{dec}$.
Input: dynamic graph $G^t_d$ at the current time step.
Input: static graph $G_s$ and hidden state $G^{t-1}_h$.
Input: maximum message-passing number $M$.
Build input graph $G_{in} = \text{concat}(G_s, G^t_d)$
Obtain encoded input graph $G_{enc} = NGN_{enc}(G_{in})$
Obtain graph hidden state $G^{t-1}_h$ if not exist then
    Obtain EMPTY graph hidden state
Obtain graph $G^0 = \text{concat}(G^{t-1}_h, G_{enc})$
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 $G^t_h = G^M$
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 $G^{t+1}_d$ by updating $N^t_d$ with $\Delta N_d$
Output: next dynamic graph $G^{t+1}_d$.
```

IV. Evaluation With Use Cases

xNet can be instantiated to model various 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

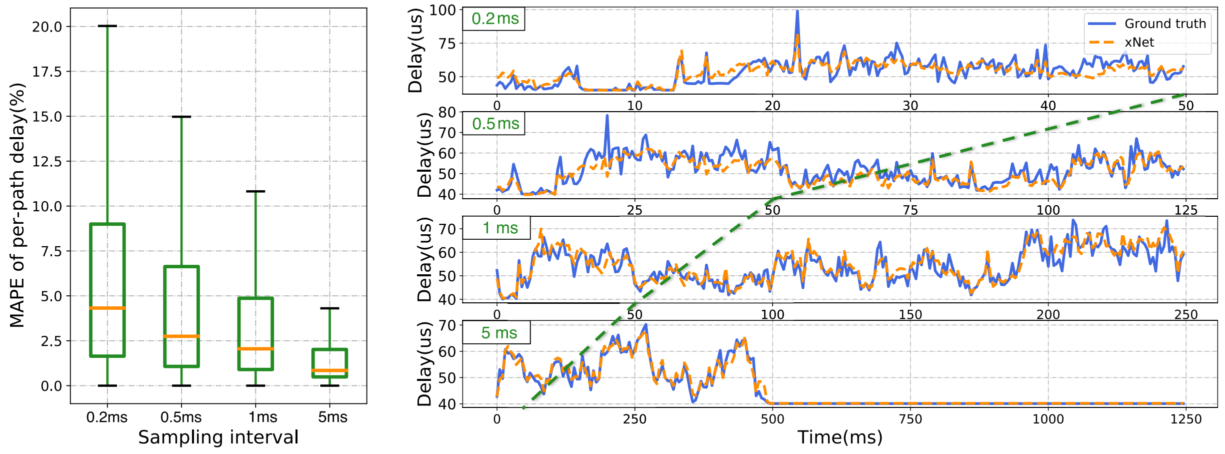
We implement xNet using Tensorflow [25] and the Graph Nets library [26], where the components of the xNet framework are implemented with standard deep learning building blocks. Unless otherwise specified, all three network models are built with the following specifications. The RNN used in aggregation 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

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 delay 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.

configurations, reflecting the application of online performance monitoring [11]. The network model is required to predict the time-series evolution of path-level delays based on real-time measurements of traffic volumes along 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], encompassing a total of 20 paths. The capacity of each link is 10 Gbps, accompanied by a 10 μs propagation delay, resulting in a 40 μs maximum base Round-Trip Time. In terms of 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 buffer sizes range from 0.05 to 0.5MB, while the ECN marking thresholds range from 6 to 60KB. 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 randomly chose the three configurations from a uniform distribution. We constrain the buffer size to always exceed the ECN threshold. To measure the path delay, we calculate the average packet delay along the path during a predefined sampling interval which determines 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. We choose these four typical intervals to showcase xNet's capability to model network scenarios of different granularities that xNet can model transient network behavior (with smaller sampling intervals) as well as steady network states (with larger sampling intervals). We contend that taking a smaller interval will result in too little data in one interval and lead to a potential loss of statistical significance. And since xNet has already achieved nearly impeccable performance on the 5ms scenario, it's not necessary to further expand the 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 delay and the ground truth over all paths as the evaluation metric.

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

[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.

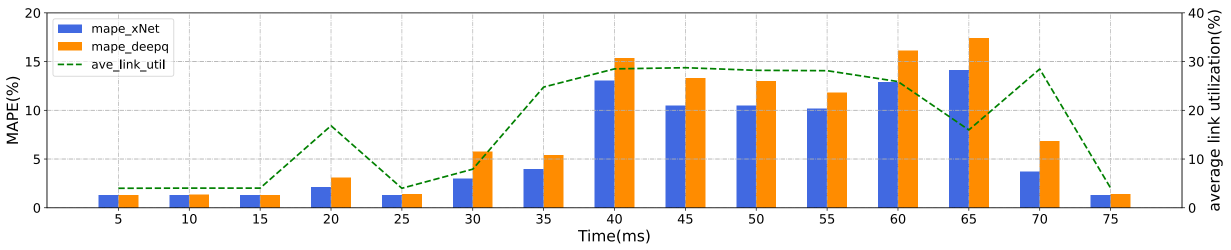


Fig. 6. Comparison with Deep-Q: MAPE of time-series path delay prediction with corresponding average link utilization. The sampling interval is 1ms and prediction errors are calculated every 5ms.

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 path delay of the next step. 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 delay prediction in four experiments with different sampling intervals. Firstly, 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 potent generalization capability and robustness for worst-case of xNet. Secondly, the prediction error systematically 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 illustrate this, we present the time series of xNet's delay prediction alongside the ground truth of a randomly selected path in Fig. 5(b). We can see that a delay spike of approximately 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 predict the evolution of path delay over time, which includes both the fluctuation and the steady-state.

To put the performance of xNet into perspective, we reproduce the Deep-Q [11] model for this scenario and compare the performance of the CVAE-based generative model (Deep-Q) and the GNN-based model (xNet). The original purpose of Deep-Q is to predict the distribution of end-to-end delay, as opposed to the targeted per-path delay in this scenario. In order to directly compare these two models in this specific case, we modify the Deep-Q model to output numerical predictions of per-path delay and replace its loss function from the *Cinfer-loss* to the *Mean Square Error (MSE)* loss. The time series prediction error (MAPE) comparison of the models is shown in Fig. 6, where the green dashed line shows the average link utilization over time. The experimental results indicate that xNet outperforms Deep-Q under various link load conditions, and its advantage is more evident under higher loads.

C. FCT Prediction in DCN

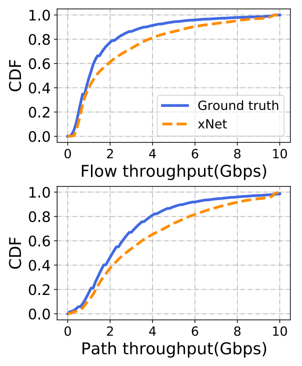
Task: With this use case, we aim to answer the question of whether xNet can provide the flow-level time-series modeling capabilities across various configurations. Unlike the previous ones, in this case, the network model behaves like a simulator, which needs to predict FCT by taking as input only the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

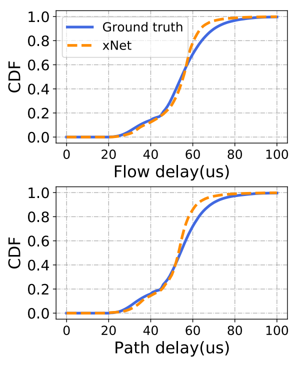
Model: Nodes of the queue, path, and flow types are required to construct the graph. The features on flow nodes include the flow size, start time, remaining size, and lifetime, where the latter two are dynamic states. The model predicts the FCT by predicting the received data volume between time steps 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 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 for prediction. This model is trained by the supervision using received bytes and per-step performance metrics at both path and flow levels. We use the state of the preceding 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 Cumulative Distribution Function (CDF) of per-step performance predictions with the corresponding ground truth. As shown in Fig. 7(a) and Fig. 7(d), the distribution of predictions generated by xNet closely aligns with the ground truth, especially for the delayed predictions, which validates the xNet's temporal prediction ability. Note that since 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 rollouts, we present the CDF of FCT in Fig. 7(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.

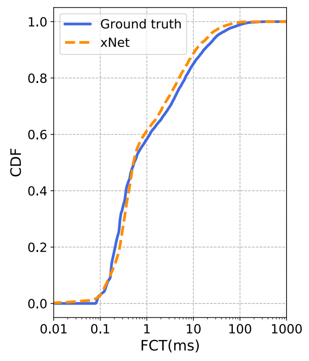
Except for the distribution, to provide a faithful comparison, we also report the MAPE of each metric shown in Fig. 7(d). The errors of throughput and FCT are much higher than in



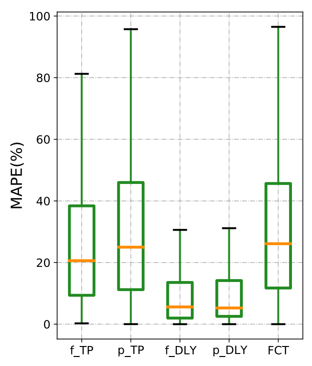
(a) CDF of path-level and flow-level throughput.



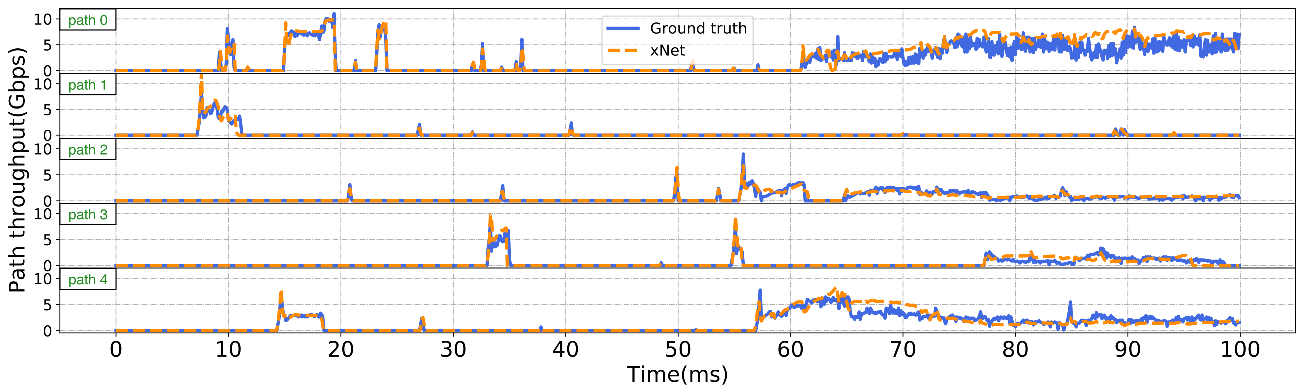
(b) CDF of path-level and flow-level delay.



(c) CDF of FCT.



(d) MAPE with boxplot.



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

Fig. 7. Results of the FCT prediction scenario in DCN. xNet behaves as a learned simulator that iteratively predicts states of path and flow.

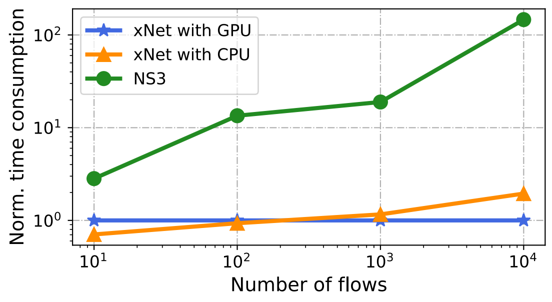


Fig. 8. Speedup of xNet compared to the NS3 simulator with different flow numbers. For clarity, we normalize the time consumption of each scheme against that of xNet with GPU.

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. Finally, we present an example of rollout trajectories of five randomly chosen paths in Fig. 7(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 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. 8, 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.

To ascertain the reasons behind this speedup of xNet, we measure the time for a single forward and backward propagation of the xNet model. The results show that in the most intricate DCN scenario for predicting FCT (§IV-C), the average time taken for forward propagation of the model is 41ms and the backward propagation takes an average time of 102ms. Note that only forward propagation is needed during the actual deployment of the model. Considering that each inference of the model corresponds to a real-world time interval, 0.1ms in this case, xNet requires approximately four hundred seconds to perform FCT prediction for a scenario involving one thousand flows (equivalent to a real-world time of 1s). Moreover, in the case of §IV-B, where the model is simpler and possesses fewer parameters, its inference speed is even higher, achieving a range of 100us-1ms for forward propagation and therefore satisfies the real-time prediction speed requirements proposed in the §IV-B.

D. Steady-State QoS Inference in WAN

Task: This use case aims to verify that the xNet can function in Wide Area Network (WAN) scenarios. We intend to show that xNet can effectively model and generalize to both global and local configurations, which is designed to reflect the usage of offline network planning [11].

It is worth noting that there is a significant difference between the topology of the WAN scenario and the data center network scenario. Unlike the highly structured topologies found in DCNs, such as Fat-Tree, the topology in WAN is more arbitrary, which leads to wide fluctuations of node neighbors, path length, and base value of metrics (such as base delay of paths). Therefore, the WAN Scenario tests the generalization ability of the model at the level of network topology. In this use case, the xNet model predicts end-to-end per source/destination mean delay and jitter at a steady state in the WAN scenario and generalizes to multiple scheduling policies and network topologies. We compare the performance with the **specifically-designed** RouteNet model [12] [34], demonstrating the generality and effectiveness of our **general** xNet model in this special scenario.

Experiment setup: We use the public network modeling dataset [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 sample of the dataset includes the topology, a routing matrix, a traffic matrix, and a performance matrix. Each switch node is implemented with one of the following scheduling policies, i.e., Strict Priority (SP), Weighted Fair Queueing (WFQ), or Deficit Round Robin (DRR), with three priority queues. The scheduling weights of WFQ and DRR are chosen randomly according to four scenarios [35]. Each flow may have 3 different Types of Services (ToS) associated with one of the three priority queues. Particularly, all the packets within the same path have the same ToS. Packets are generated following a Poisson distribution on each flow. The traffic matrix provides measurements of traffic-related information, such as average bandwidth and packet generation at the level of source-destination pairs. The performance matrix provides performance measurements, such as average delay and jitter on paths. All the measurements are relative to time units. We use a variant of RouteNet [34] as the comparison baseline. We use 400000/50000 samples for training/testing, and each sample has more than 300 active paths. For the evaluation metrics, we report both percentage error (PE) and absolute percentage error (APE) of per-path performance metrics (i.e., delay and jitter) between the predicted value and the ground truth. We also report mean absolute percentage error (MAPE) under different traffic loads.

Model: To build the graph, the nodes to need have the type of link (port), queue, and path. The features on link nodes include the bandwidth and scheduling policy. The features on queue nodes include the scheduling policy, scheduling weights, and queue sizes. The features on path nodes include the average bandwidth and packets generated in a time unit and the ToS of flows on this path. Although this scenario does not require temporal inference, we still leverage the encoder-processor-decoder architecture of our state transition model for better performance. The training process with a batch size of 16 takes about 12 hours.

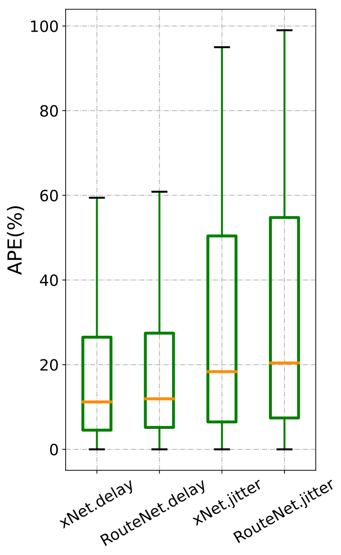
Results: Firstly, we present the boxplot of absolute percentage error (APE) and the CDF of percentage error (PE). As shown in Fig. 9(a), both xNet and RouteNet demonstrate relatively low median APE of delay at around 10%. APEs of jitter are much higher than delay, but still lie within an acceptable range such that the median APE is below 20%. Fig. 9(b) also shows highly similar curves of CDF of PE between xNet and RouteNet, indicating that xNet can perform nearly the same (in the scenario of delay inference) as RouteNet or a little bit better (in the scenario of jitter inference).

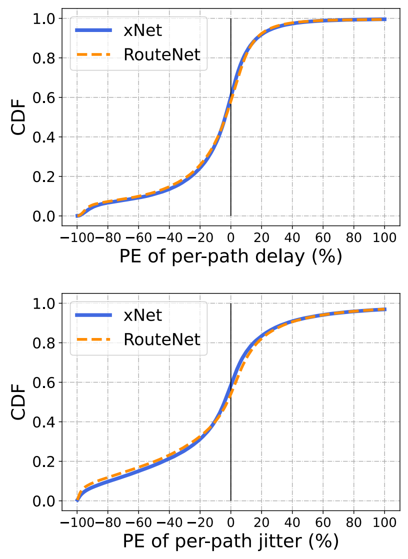
To explore the performance of the model across varying traffic loads, we divided the data into ten groups according to the load rate of the bottleneck link of the path, as shown in Fig. 9(c). Each point in the figure represents the relative value of MAPE over corresponding traffic load (TL) range ([TL-5%,TL+5%)) with RouteNet as 100%. It can be seen that xNet and RouteNet have similar performance under medium and high load situations, and xNet outperforms RouteNet significantly under low load situations (approximately 50% at [0,10%) traffic load).

The experimental results show that our **general** model xNet can provide similar or slightly improved performance than the **specific model** RouteNet in the WAN scenario, which proves the effectiveness of xNet in this scenario and demonstrates the generalization ability of our model.

V. Deep Dive xNet

In addition to assessing accuracy across diverse use cases, we also dive deeper into the practical dimensions of xNet, including an exploration of its long-term performance and its potential applicability in optimization scenarios (i.e., choosing the best ECN setting). We begin by conducting experiments to determine whether the state transition model of xNet accumulates errors, followed by an analysis of the underlying causes of the observed outcomes. Then, we investigate other types of prediction accuracy of the model with clear practical implications, including statistical and ranking accuracy that supports downstream tasks such as configuration recommendation. At the end of this section, we test how the xNet model work under an unseen special situation, i.e., facing particular traffic such as many-to-one incast.


(a) APE with boxplot.


(b) CDF of PE.

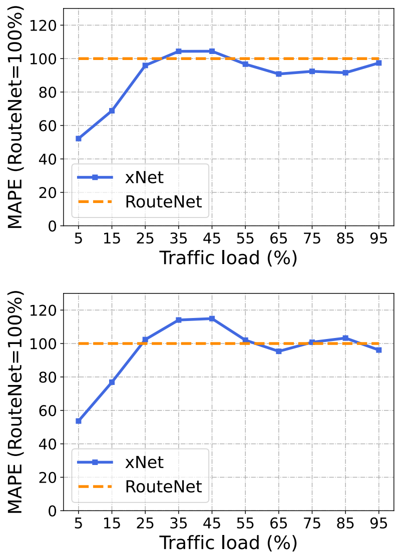

(c) MAPE under different traffic loads.

Fig. 9. Multiple metrics of steady-state QoS inference in a WAN scenario.

A. Long Term Performance

The state transition design of xNet allows for prediction at a finer grain compared to existing "end-to-end" approaches. However, this introduces a new concern related to accumulative errors. As the model "rolls out", there's a potential for prediction errors to accumulate and lead to "growing" errors in FCT predictions. In this case, our objective is to verify whether xNet can accurately simulate scenes over extended periods of time.

Experiment setup: The experimental setup and used model are the same as that in §IV-C. Note that the model trained on "shorter" traces is directly used to predict longer traces without retraining. For testing, we generate 3000 flows for each scene to prolong the scenes, that is 30000 flows and more than 263000 samples in total for the 10 test scenes. In each scene, we equally divide the total simulation time (∼2s) into 10 periods, and for each period, we focus on those flows that end during it and report the prediction errors.

Results: The results are shown in Fig. 10. All 10 periods have similar MAPEs and the overall MAPE is close to the level of the "shorter" test scene in Fig. 7(d), which indicates that xNet has consistent performance in the previously unseen long-term scenes. There is no evident increase in FCT prediction errors over time which provides compelling evidence that the state transition model of xNet remains immune to cumulative error. This outcome can be attributed to two factors. Firstly, xNet is free from systematic errors, i.e., it does not constantly over/under-estimate certain network states. And more importantly, the errors of network state predictions cannot accumulate as they are often "reset" due to the "on-off" nature of the network dynamics. Influenced by traffic demands and congestion control, in common network scenarios with moderate loads (<100%), the network states are observed to empty from time to time, thus clearing out cumulative errors. With these results, we show that xNet can be used to predict scenes with arbitrary length without concerns of error accumulation.

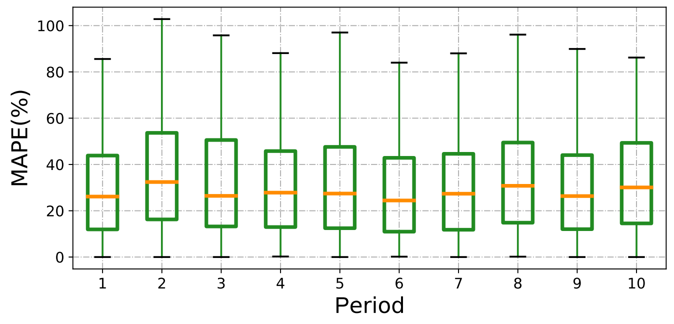
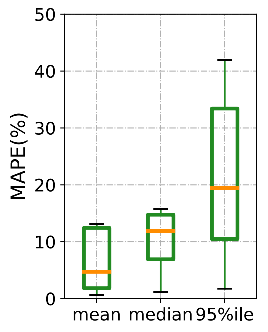


Fig. 10. Long-term Accuracy of xNet.

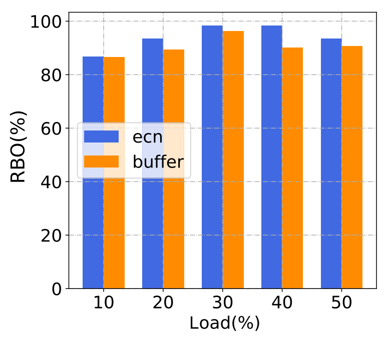
B. Revisiting Model Accuracy

Statistical prediction performance. Up until now, we have adopted the Mean Absolute Percentage Error (MAPE) metric as the primary metric for evaluating the model's predictive capabilities at the per-flow level. To make predictions at such fine grain is inherently error-prone, even with considerable cost. In this paper, we strive to make the best of the GNN-based model to achieve a fairly good prediction performance at a reasonable cost. However, in practical scenarios with more specific goals, there might not be a stringent requirement for fine-grained flow-level predictions. Instead, in some circumstances, we would prefer the statistical performance of a group of flows (e.g., the average or the tail performance), which allows the user to acquire a more comprehensive view of the network performance rather than focusing on single flows.

The experimental setup and used model are identical to that in §IV-C. We tasked the model with predicting FCT across ten distinct scenarios with different traffic traces. We normalize the FCT value with their corresponding flow size and compute the



(a) Statistical accuracy.



(b) Ranking accuracy.

Fig. 11. Accuracy of xNet in other forms.

trace-level prediction accuracy of mean, median, and the 95^{th} percentile FCT within each scenario. As shown in Fig. 11(a), the accuracy of trace-level prediction accuracy is evidently better than that of flow-level prediction. Notably, xNet has a $\sim$ 20% error on tail performance prediction, which is quite useful for many optimization tasks whose main focus is to improve the tail latency of the network.

Ranking performance. As shown in §IV-C, xNet achieves a median error of less than $\sim$30% when predicting FCT. Notwithstanding its remarkable speed-up compared with traditional simulators, some might wonder whether a (really) **fast** but (somewhat) **inaccurate** model has practical utility. In this section, we further dig into the practical use of xNet to show that an "inaccurate" model can be especially helpful to downstream optimization tasks, parameter selection, in particular. In this case, we use xNet to determine the best ecn/buffer setting when other configurations are fixed. In other words, given a series of different settings of the same scene, we want to know whether xNet can rank their performance in accordance with the ground truth and pick the best setting (s).

The experimental setup and used model are the same as that in §IV-C. We rank a series of configurations by running the same scene use those configurations independently and sort them in ascending order by the mean FCT of small flows ($\leq$ 100KB). For testing, we range the traffic load from 10% to 50% and enumerate all possible ecn/buffer settings when other parameters are fixed. Particularly, when setting the ecn, we set the buffer to 0.25MB and run simulations using ecn values ranging from 10% to 90% of 60KB, using 10% as step size. And when setting the buffer we set the ecn to 30KB and varying buffer values from 10% to 90% of 0.5MB. We report the rank-biased overlap (RBO) between the predicted rankings and the ground truths generated by the simulator.

Fig. 11(b) illustrates the RBO between the xNet predictions and the ground truths with different load settings. Remarkably, xNet consistently achieves high ranking accuracy with the worst RBO above 0.85, which means xNet possesses strong performance isotonicity. It is worth mentioning that in all scenes and settings, xNet successfully picks the setting which achieves the best (smallest) FCT results. The ability to rank different configurations with their performance, combined with its remarkable order-of-magnitude speed-up, makes xNet a promising tool for downstream parameter tuning tasks.

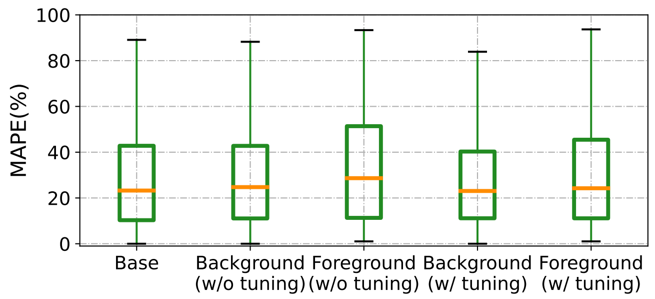


Fig. 12. Performance of xNet under incast traffic.

C. Performance Under Incast Traffic

As shown in §IV-C, xNet demonstrates promising performance in predicting the FCT of flows following an all-to-all pattern in DCN, which corresponds to the traffic pattern encountered during the model's training phase. While this pattern is prevalent and widely used by researchers, one would wonder how well the model performs under some special situations whose traffic pattern is previously unseen during model training. The efficacy of xNet in such intricate scenarios will decisively influence its practical utility. With a series of experiments, we aim to illustrate the adaptability and extensibility of xNet across varied traffic patterns. Using the performance of xNet under the all-to-all traffic pattern as the *Base*, we extend our experiment to encompass additional many-to-one incast traffic alongside the predominant all-to-all communication. We refer to the former as the foreground traffic and the latter as the background traffic. Since the model is expected to have increased error facing previously unseen foreground incast traffic, we also opt to fine-tune the model with a small amount of data collected over this new hybrid pattern and compare the results.

Experiment setup: For the new pattern, 90% of the flows follow the all-to-all pattern, while the remaining 10% follow a three-to-one incast pattern. The remaining configuration settings remain the same as in §IV-C. For fine-tuning, we use 10 simulations with a total of 10000 flows generated, and we train the model on this corpus for only 1 hour. We report the MAPE between the truth and the prediction of xNet.

Results: Fig. 12 shows the MAPE of FCT between the xNet predictions and the ground truths. On the base setting, xNet achieves a median error of 23% which confirms the results of §IV-C. Without fine-tuning, xNet demonstrates a median error of 24% for background flows and 29% for foreground flows. With one hour of fine-tuning, xNet yields a median error of 23% and 24%, respectively, which is close to the base results. The above results show the adaptivity of xNet to new traffic patterns. When confronted with previously unseen traffic patterns, xNet maintains a reasonable level of accuracy without encountering significant degradation. With moderate fine-tuning, we can apply the model to a previously unseen scenario and expect comparable performance.

VI. Discussion and Future Work

In this section, we discuss the limitation and future research directions of xNet.

Data collection. As an application of deep neural networks, the accuracy and generalization ability of the learning-based network models highly depend on the quantity, quality, and diversity of the training data. Operators can quickly collect sufficient data that reflect the distribution of the real world with the production networking systems. However, operators do not frequently adjust their configurations on different network elements, which leads to limited combinations of various traffic patterns and different configurations in the training dataset. In addition, in order to collect diverse training data, it is often unrealistic to require operators to frequently change configurations or build dedicated testbeds, because this will lead to potential performance degradation or high costs.

In this context, leveraging the simulation data can be a good alternative. However, it is well known that there is "simulation-to-reality" gap between the simulated in-lab data and the real-world ones. This obstacle stands before all the learning-based network models that care about network configurations including RouteNet [12], xNet, and potential future ones. In our view, recent advances in transfer learning [37] and few-shot learning [38] can be a cure. One can first train the network model with plenty and diverse data from simulation (may not be of high quality), and then transfer the model with few but high-quality data from the real-world collection. The assumption behind this approach is that the simulation and real-world network systems follow the same basic principles, that these principles can be learned from simulated data, and the transfer process simply attempts to correct systemic errors.

One more thing that calls for further discussion is the cost of data collection. To generate "labeled" data for the model in §IV-C, we run the NS3 simulator for approximately 44 hours. At first glance, some might find it confusing that xNet aims to accelerate network simulation by training with data generated with time-consuming simulation. However, we find that generating a certain amount of training data with simulation can be a good bargain for the following reasons. Firstly, of all the possible combinations of traffic and parameters, the training data only covers an extremely small portion. That means the model trained on such a small portion of data actually provides its user with remarkable generalizability, coupled with notable speedup. Secondly, data collection can leverage the parallelization capabilities inherent in modern multi-core servers, since the sampled inputs are independent of each other, which means pre-collected data could come at a lower time cost. Finally, as shown in §V-C, xNet can be fine-tuned with a rather small amount of data, which indicates that after the initial base model is trained, the data collection cost when applying it to other scenarios would become more and more acceptable.

Configurability. In principle, learning-based network models can only generalize within the distribution described by the training data. This means existing network models can not be used to evaluate the performance of new protocols or policies unless they can be expressed by the combination of existing configurations or policies that are considered by the network model and the training dataset. In this context, the discrete-event simulators are still necessary to be the first step used to implement and do the proof test for the new proposals. The learning-based model, in this situation, plays the role of accelerator to support large-scale evaluation. Therefore, the key to expanding the application scope of learning-based network model is to improve the granularity and range of configurability. xNet is designed to stand in this line of work. For example, the range of configurable features is gradually larger, i.e. traffic for Deep-Q, additional topology and routing for RouteNet, and extra configurations for xNet.

Optimization. Learning-based network models can quickly produce accurate evaluations of the network performance with varying configurations, so they can be used to explore better schemes to optimize traffic transmission. This optimization procedure is often driven by some search-based methods (e.g., grid/random search). These methods first generate candidate configurations and then evaluate them using 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 be used to predict the next state, thus enabling model-based control (e.g., model predictive control [39]) or model-based reinforcement learning [40]. We leave this for our future work.

Scalability. Our solution provides an expressive approach capable of modeling diverse network elements across various levels. However, this will result in a larger graph with more nodes (e.g. hundreds to thousands of devices and flows) that require more computations to update, which could raise concerns about scalability. From the modeling perspective, to provide fine-grained performance prediction (e.g. at flow level) and embed relational inductive bias from domain knowledge, we think this is a reasonable and acceptable trade-off between accuracy and efficiency. In the view of computation, with the growth of web search and social networks [41], a lot of graph computing engines [42], [43], [44] have been built to support efficient computation on large-scale graphs (e.g. millions to billions of nodes and edges [41]). As a result, our graph is rather small compared to these huge graphs and thus can be easily handled with these well-designed systems.

One appealing attribute of the used Graph Neural Network is its inherent extendability. Graph neural networks have a strong relational inductive bias that a GNN model mainly learns the interactive relationship between nodes, regardless of the scale of the network topology. Therefore, theoretically speaking, the xNet model based on the graph neural network does not limit the network size of the input, and the model is inherently extendable. This property has been partially confirmed by the experimental results in §V-C that the GNN-based model remains effective in previously unseen scenarios. However, GNN models are not a panacea and we wouldn't expect it can miraculously scale to completely new scenarios or extremely large-scale topologies. Besides fine-tuning, we are working on several other possibilities to improve model extendability and hope to present them in the future.

VII. Conclusion

In this paper, we propose xNet, a GNN-based data-driven network modeling framework. Our overarching goal is to provide a versatile approach to network modeling that enables the accurate description and consistent generalization of diverse network properties. xNet represents the networking entities and their configurations as nodes within a relation graph and learns the relationship between them via message passing. We implemented xNet, instantiated it into three use cases, and explored some features of xNet in depth. The results indicate that xNet can learn to efficiently generalize to various networking scenarios while achieving up to a 100x speedup over traditional packet-level simulator.

REFERENCES

[1] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar, "Answering what-if deployment and configuration questions with wise," in *Proc. ACM SIGCOMM*, 2008, pp. 99–110.

[2] *IP-SLA*. Accessed: Jan. 10, 2022. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipsla/configuration/15-mt/sla-15-mt-book/sla_icmp_echo.html

[3] Z. Xu et al., "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 1871–1879.

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

[5] F. Ciucu and J. Schmitt, "Perspectives on network calculus: No free lunch, but still good value," in *Proc. ACM SIGCOMM*, 2012, pp. 311–322.

[6] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and Tools for Network Simulation*. Cham, Switzerland: Springer, 2010.

[7] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 1263–1272.

[8] A. Sanchez-Gonzalez et al., "Graph networks as learnable physics engines for inference and control," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 4470–4479.

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

[10] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, "Learning to simulate complex physics with graph networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 8459–8468.

[11] S. Xiao, D. He, and Z. Gong, "Deep-Q: Traffic-driven QoS inference using deep generative network," in *Proc. Workshop Netw. 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 *Proc. ACM SOSR*, 2019, pp. 140–151.

[13] A. Mestres, E. Alarcón, Y. Ji, and A. Cabellos-Aparicio, "Understanding the modeling of computer network delays using neural networks," in *Proc. Workshop Big Data Anal. Mach. Learn. Data Commun. Netw.*, 2018, pp. 46–52.

[14] M. Wang et al., "Neural network meets DCN: Traffic-driven topology adaptation with deep learning," *ACM SIGMETRICS*, vol. 2, no. 2, pp. 1–25, 2018.

[15] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," in *Proc. ACM SIGCOMM*, 2006, pp. 59–62.

[16] Y. Zhou et al., "Flow event telemetry on programmable data plane," in *Proc. ACM SIGCOMM*, 2020, pp. 76–89.

[17] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

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

[19] A. Mestres et al., "Knowledge-defined networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 3, pp. 2–10, Sep. 2017.

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

[21] *Concepts of Digital Twin Network*. Accessed: Jan. 10, 2022. [Online]. Available: https://tools.ietf.org/html/draft-zhou-nmrg-digitaltwin-network-concepts-06

[22] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Netw.*, vol. 32, no. 2, pp. 92–99, Mar. 2018.

[23] M. Alizadeh et al., "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, 2010, pp. 63–74.

[24] A. K. Choudhury and E. L. Hahne, "Dynamic queue length thresholds for shared-memory packet switches," *IEEE/ACM Trans. Netw.*, vol. 6, no. 2, pp. 130–140, Apr. 1998.

[25] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. USENIX OSDI*, 2016, pp. 265–283.

[26] (2018). *Graph Nets Library, Deepmind*. Accessed: Jan. 10, 2022. [Online]. Available: https://github.com/deepmind/graph_nets

[27] K. Cho et al., "Learning phrase representations using RNN encoder–decoder for statistical machine translation," 2014, *arXiv:1406.1078*.

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

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

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

[31] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 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 Trans. Netw.*, vol. 29, no. 2, pp. 489–502, Apr. 2021.

[33] M. Alizadeh, A. Javanmard, and B. Prabhakar, "Analysis of DCTCP: Stability, convergence, and fairness," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 39, no. 1, pp. 73–84, 2011.

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

[35] *Network Modeling Datasets*. Accessed: Jan. 19, 2023. [Online]. Available: https://github.com/BNN-UPC/NetworkModelingDatasets

[36] A. Varga, "OMNeT++," in *Modeling and Tools for Network Simulation*. Cham, Switzerland: Springer, 2010, pp. 35–59.

[37] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[38] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele, "Meta-transfer learning for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 403–412.

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

[40] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *J. Intell. Robot. Syst. Theory Appl.*, vol. 86, no. 2, pp. 153–173, May 2017.

[41] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One trillion edges: Graph processing at Facebook-scale," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1804–1815, 2015.

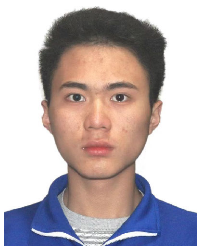
[42] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed GraphLab: A framework for machine learning in the cloud," 2012, *arXiv:1204.6078*.

[43] W. Xiao et al., "TuX^2: Distributed graph computation for machine learning," in *Proc. USENIX NSDI*, 2017, pp. 669–682.

[44] H. Liu, S. Lu, X. Chen, and B. He, "G^3 when graph neural networks meet parallel graph processing systems on GPUs," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 2813–2816, 2020.

Sijiang Huang received the B.E. degree in information engineering from the Beijing University of Posts and Telecommunications, Beijing, China. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, Tsinghua University, Beijing. His current research interests include data center networks and programmable hardware.

Yunze Wei received the B.E. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His current research interests include machine learning for network modeling and digital twin networks.



Lingfeng Peng is currently pursuing the B.E. degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His current research interests include data center networks and digital twin networks.

Mowei Wang received the bachelor's degree in communication engineering from the Beijing University of Posts and Telecommunications in 2017 and the Ph.D. degree in computer science and technology from Tsinghua University in 2022. He is currently a Researcher with Huawei Technologies Company Ltd. His research interests include datacenter networks and data-driven networks.

Linbo Hui received the M.E. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2022. His research interests include machine learning for network modeling.

Peng Liu received the master's degree in computer science and technology from Beijing Jiaotong University, Beijing, China, in 2017. He is currently with the China Mobile Research Institute. His research interests include computing force networks, deterministic networks, and industrial internet.

Zongpeng Du received the B.E. degree in computer science and technology from the Beijing Institute of Technology in 2006 and the Ph.D. degree in information and communication engineering from the Beijing University of Posts and Telecommunications in 2012. He is currently working on the future network at the China Mobile Research Institute. His research interests include the future IP networks and intelligent networks.

Zhenhua Liu received the B.E. degree in information engineering from Xidian University, Xi'an, China, in 1997. He is currently a Technology Strategy Planning Expert with Huawei Technologies Company Ltd. He has rich experiences of large software system, telecom equipment system architecture, and development. His research interests include autonomous driving networks and next generation mobile operating systems.

Yong Cui (Member, IEEE) received the B.E. and Ph.D. degrees in computer science and engineering from Tsinghua University, China, in 1999 and 2004, respectively. He is currently a Full Professor with the Computer Science Department, Tsinghua University. His research interests include mobile cloud computing and network architecture. He served or serves on the Editorial Boards for IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Cloud Computing, and the IEEE Internet Computing.