Nüwa: Efficient Generative Control Plane for AI Network Simulation

Wenkai Li
Xi'an Jiaotong University
Xi'an, China
Microsoft Research
Beijing, China
wkli24@stu.xjtu.edu.cn

Ran Shu
Microsoft Research
Beijing, China
ran.shu@microsoft.com

Peng Zhang
Xi'an Jiaotong University
Xi'an, China
p-zhang@xjtu.edu.cn

Yongqiang Xiong
Microsoft Research
Beijing, China
yqx@microsoft.com

Abstract

Network simulation plays a critical role in improving the efficiency of AI supercomputers for new design validation, parameter tuning, and network protocol development. However, high-fidelity network simulation is very slow at scale. We observe that the significantly inefficient initialization of the network control plane is one of the key reasons for the slow simulation speed. The existing network simulation searches for the available routing at simulation initialization, which takes hours or even days. Moreover, the large routing table involves redundant information which occupy a high memory volume and makes routing lookup very slow. In this paper, we present Nüwa[1], an efficient generative control plane for AI network simulation. Nüwa leverages the layered network architecture of the AI network to express routing information in a formula for each layer. The formulas are generated directly from the topology description with an extremely simple transformation. Evaluations show that Nüwa can reduce routing table generation from hours to only 20 seconds for 64K nodes. For data plane execution, Nüwa can reduce the overall simulation time over 100x by almost eliminating the forwarding calculation.

CCS Concepts

• **Networks** → **Network simulations**; **Data center networks**; • **Computing methodologies** → **Discrete-event simulation**; **Simulation tools**.

Keywords

Network Simulation, Data Center Networks

[1]In Chinese mythology, Nüwa (pronounced [nỳ.wá]) created humanity from clay according to her shadow. We use the name Nüwa to signify that our work directly generates control plane information.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
APNET 2025, Shang Hai, China
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1401-6/25/08
https://doi.org/10.1145/3735358.3735365

ACM Reference Format:
Wenkai Li, Ran Shu, Peng Zhang, Yongqiang Xiong. 2025. Nüwa: Efficient Generative Control Plane for AI Network Simulation. In *9th Asia-Pacific Workshop on Networking (APNET 2025), August 07–08, 2025, Shang Hai, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3735358.3735365

1 Introduction

To support the rapid evolution of Artificial Intelligence (AI), leading companies are building AI supercomputers at extremely large scale [16]. xAI is training the successor of Grok3 using 200,000 GPU cluster [23]. To optimize the performance of AI training, operators need to tune training framework parameters, optimize software stack, and even develop network protocols. However, running a 10,000 GPU for a day costs more than 1 million dollars [5]. Even top companies cannot afford such extremely expensive tests on real clusters. As a result, simulation has become a crucial tool in evaluating the design of training clusters [20, 22].

Network simulation plays a key role in achieving high-fidelity large scale AI training simulation [20]. Discrete-event simulation (DES) is the type of simulator desired for high-accuracy AI simulation due to the exact processing of packet-level behaviors. Typical DESs include ns-3 [15], OMNeT++ [14] and ns.py [12]. However, DES is slow due to the need for executing all packet-level events, in which the number of events can be trillions per simulated second in a 100,000 GPU cluster. Network simulation dominates the total simulation time, and the efficiency of network simulation is the key to optimizing AI simulation speed.

Many approaches have been proposed to speed up DES, and the approaches can be categorized into two types. Increasing parallelism and improving computation efficiency. Parallelism was introduced to DES in 1990 [7]. Recent work DONS [9] and Unison [2] further improve the parallelism of DES by fine-grained automatic partition and scheduling. SimAI further improves Unison by eliminating the global locks [20]. For computation efficiency, DONS [9] introduces data-oriented concept to network simulation which improves the cache hit rate to speed up the computation.

Despite these efforts that have improved the simulation speed by up to two orders of magnitude, hyperscale AI supercomputer network simulation is still very slow. Running an 8,192-GPU simulation with 0.002 seconds simulated time requires 100,000 seconds in ns-3.

This is 50,000,000 times slower than that in real time. Through detailed profiling, we find a very interesting thing that the inefficient control plane is one of the major bottlenecks in DES.

DES processing can be divided into two phases, initialization and execution. It exceeded our expectations that control plane inefficiency is shown not only in the initialization phase but also in the execution phase. During initialization, DES establishes topology and computes routing tables by simulating routing protocols such as OSPF and BGP. Generating routing table can take about 5 hours for an 8192-GPU Fat-tree topology in ns-3. In the execution phase, DES processes every packet-level event. Specifically, simulators lookup routing tables for each packet to generate all candidate next hops. The switch then uses routing policies like ECMP or packet spraying to choose one output port from the possible next hops. We find that 90% time is cost by generating the next hops in a 1024-GPU fat-tree running 1MB-Allreduce workload, and this is due to the inefficient control plane.

After detailed analysis, we get the following conclusions of the inefficient control plane: In the initialization phase, network simulators often simulate routing protocols such as OSPF, BGP to generate routing tables that are used for packet forwarding. The simulation is costly since it needs to model detailed routing protocol behaviors. Advanced control plane verifiers like Batfish [4, 6], which can only scale to 2,000 routers [13]. Ns-3-datacenter [19] optimizes the routing computation with BFS and speeds up the lookup by optimizing routing entry storage data structure. However, we observe that for AI training cluster simulation, detailed routing protocol behaviors are unnecessary, but only the routing results are required. Here, routing results mean the calculated next hops for each node. For example, in a clos topology, the group of up and down ports are enough for switches to calculate the next hop.

To guide the forwarding of packets, network simulators need to maintain a routing table for each node. When there are a large number of nodes, the memory cost will be quite high. For example, as our estimation on a fat tree of 27,468 nodes, the total memory consumption of route entries can take up over 150GB memory.

In this paper, we provide Nüwa , an efficient generative control plane for AI network simulation. Our key insight is that routing tables can be replaced with routing rules in datacenter simulation.

We make the following contributions:

- We used detailed profiling to show that one bottleneck of simulation of large-scale training clusters is the efficiency of the control plane.
- We proposed Nüwa, a new control plane abstraction for network simulation, which captures essential behaviors with less computation and memory cost.
- We implemented Nüwa based on ns-3, and used experiments to show that it achieves a 100× speedup over vanilla ns-3.

2 Background and Motivation

In this section, we first introduce the architecture and workflow of DES network simulators. We then introduce optimizations for the efficiency of the DES and our observations for inefficient control plane computation. Finally, we introduce our motivation to redesign the control plane for AI network simulation.

2.1 DES Network Simulation

Discrete-event simulation (DES) tools including ns-3 [15], OMNet++ [14], etc., are the most prevalent due to their high fidelity. These tools can simulate traffic at the packet level to obtain an accurate estimate of network performance. As shown in Figure 2a, a DES simulator typically contains two stages, *initialization stage* and *execution stage*. The initialization stage mainly does the work of control plane functionalities, such as building topology and computing routing tables. The execution stage mainly corresponds to the data plane functionalities and is the core of DES. In this stage, the simulator maintains a timeline, schedules network events, and executes events in strict chronological order. Common events including packet sending, queuing, forwarding, and receiving.

Improving DES execution speed has a long history in the literature, which falls in two directions.

(1) Making simulation parallel. Parallel DES (PDES) [7] introduces parallel processing to the DES field. Mainstream DES ns-3 [15] and OMNeT++ [14] both support the PDES mode. Unison [2] breaks network into finer granularity and performs thread-level parallelism, avoiding the high overhead for process synchronization in existing PDES. DONS [9] re-architects the DES using data-oriented design and breaks DES processing into fine-grained network functions for efficient parallel scheduling.

(2) Improving computation efficiency. Nix-vector routing [21] introduces source routing to speed up packet forwarding in ns-3. However, source routing cannot fit for switch-decided packet-level or flowlet load-balancing, which limits usage scenarios. Ns-3-datacenter [19] find that computing routing entries to every interface of every device leads to an unimaginable number of routing entries. Instead, concentrating on the destination node rather than the destination interface is enough. Htsim [10] is a fast DES optimization that targets the simulation of the behavior of the congestion control algorithm. It optimizes for specific datacenter topologies. However, datacenter topologies have many variants, such as dragonfly [11] to dragonfly+ [18], and writing a specific optimization for every topology is so tedious.

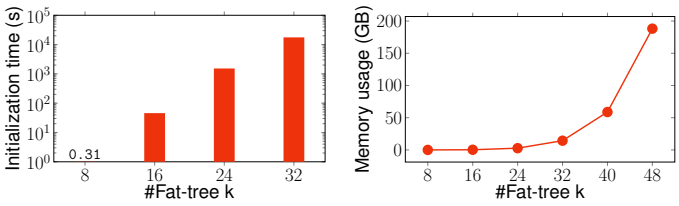
2.2 DES in the Era of AI Training

Although there have already been several optimizations for DES in networks, their performance is still lag behind when simulating AI training clusters. A modern training cluster can consist of more than 100K GPUs, which generate a large volume of traffic. For example, a 100,000 GPU cluster with 400 Gbps generates trillions of events per simulated second. To process such a large number of packet-level events, DES can take days to complete.

To mitigate such a huge mismatch between simulation speed and real traffic rate, SimAI [20] further enhances Unison by eliminating the global locks and achieves a lock-free design which improves parallelism. However, network simulation still dominates the end-to-end AI task simulation and it struggles to perform simulation in today's cluster scale.

2.3 Observations

After conducting an in-depth analysis of the time distribution across different parts of the simulation, we observe that beyond the execution stage, computation of routing tables in the initialization stage



(a) Initialization time (b) Memory usage of routing tables

Figure 1: Performance of ns-3

Table 1: The table lookup time and total execution time for fat-tree in ns-3

Fat-tree (k)	8	16	24
Lookup time (s)	21.06	1930.95	23256
Execution time (s)	42.90	2100.16	23808
Ratio (%)	49.09	91.94	97.68

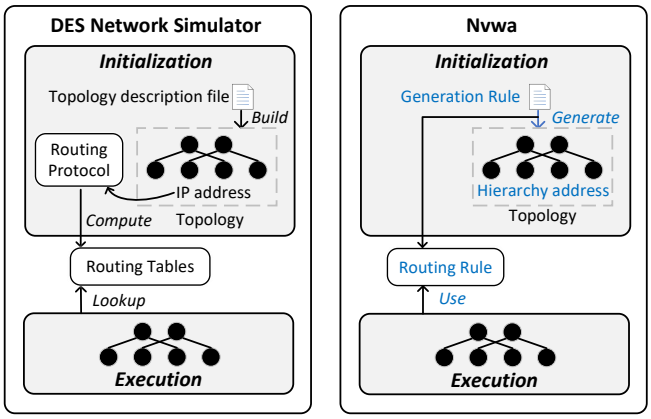
has become a critical bottleneck. In our experiment, initialization of routing tables for an 8,192 GPU fat-tree topology takes 5 hours to complete in ns-3. Optimization of the ns-3-datacenter [19] is helpful but still insufficient. Without routing tables, switches have no idea how to forward packets to the next hop. That is, data plane execution can only begin after initialization is complete. Therefore, before further improving the efficiency of execution, we must first address the challenges in initialization.

We summarize the reasons why control plane simulation becomes a bottleneck as follows:

Long computation time. Real datacenter clusters run routing protocols such as OSPF, BGP in the control plane. Popular DES simulators also implement traditional routing protocols to be consistent with the real environment. On the other hand, routing protocols consider a wide variety of network conditions, which introduces a significant computational overhead in software. When network scale continuously grows, this overhead increases drastically, even requiring days to compute.

Large routing table. Another key challenge is scalability. Real datacenter networks carefully assign the IP address [1] so that they can aggregate routes to hosts that are in the same subnet into a single network route, reducing the number of routing entries in the routing table. However, the current simulation pays little attention to IP address assignment and routing aggregation, and thus their routing tables are considerably large. As we estimate in Figure 1b, the routing tables will cause a large amount of memory resource.

Inefficient Lookup. Additionally, we notice that initialization can also slow down the data plane execution stage. This is counterintuitive, because initialization only provides the routing tables for execution. The secret lies in the efficiency of the cache. It is well known that packet forwarding is one of the most frequent operations in the data plane. First, switch looks up the routing table for all matched routes with minimal cost, then it chooses a route from them according to ECMP or packet spray policy. Lookup efficiency is affected by cache efficiency. We now understand that large routing tables trigger more frequent data exchanges between the cache



(a) DES network simulator (b) Nüwa

Figure 2: Architecture of DES network simulator and Nüwa

and main memory, which in turn slows down lookup operations and ultimately impedes data plane execution, Table 1 shows that lookup accounts for over 90% of time in execution.

In fact, some studies on network verification have proposed methods for fast control plane simulation. For example, Batfish [4, 6] supports many routing protocols (OSPF, BGP, IS-IS etc.), computes routing tables at a convergent steady state. FastPlane [13] targets the datacenter network, develops a generalized dijkstra's algorithm, replacing the number path weight with route advertisements, and achieving scalability. Although routing protocol simulation has been accelerated and simplified, it is difficult to scale to a network of hundreds of thousands of devices. Other works targeting fast routing in the networks. Primus [24] notes that routing tables can be generated in DCNs that have regular topologies, and maintain a path table and a link table at each switch for routing. Even these tools can generate routing tables faster, the routing table can still consume a large amount of memory during simulation, and the lookup is still inefficient.

3 Nüwa Design

3.1 Basic Idea

Our key insight is that the training clusters have regular topologies and simple policies, such that faithfully simulating the control plane to generate a large routing table becomes an over-kill.

Topology. The topologies used for training clusters are highly structured. Switches are organized hierarchically and symmetrically. This provides opportunities for more efficient route computation, storage, and lookup.

Policy. The policies configured in the training clusters are quite simple. Even BGP has been widely used for routing in datacenters [1], the policies are mainly focused on valley-free or shortest-path routing, and equal-cost multi-path. Taking the classic fat-tree topology as an example, packets are supposed to be forwarded upward first and then downward to the destinations to avoid valley on forwarding paths, which may cause problems like PFC-deadlock in datacenter. Such routing policies can be obtained without complex routing protocol computation in simulation. Specifically, a fat-tree

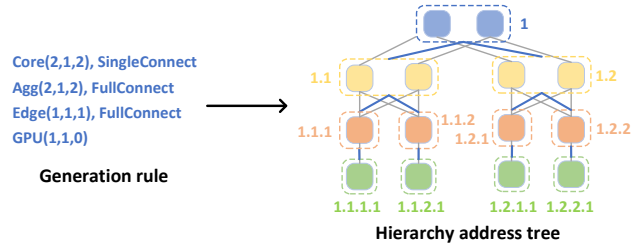


Figure 3: An example of generating topology with generation rule. The generated topology forms a hierarchy address tree.

switch just needs to judge whether the current switch and destination are in the same subnet, and it can determine whether to forward the packet upward or downward.

We propose a unified topology generation and forwarding framework for the datacenter topology, enabling more efficient and scalable hyperscale network simulation.

Suppose that routing in the clos topology, which is the most widely adopted topology [8, 17], always follows the shortest paths. The key insight of Nüwa is that the routing paths follow a fixed pattern, called the up-and-down routing, as they always go upward first and then downward to the destinations. This pattern helps prevent PFC deadlocks in the datacenter. We argue that simulation for datacenter networks can use routing rule to replace routing table for forwarding, and thus the computation of routing tables can be avoided, saving the increasingly longer initialization time.

3.2 Architecture

The architecture of Nüwa is shown in Figure 2. It consists of three main components: topology generator, hierarchy address system, and routing rule. The topology generator reads the generation rules as input, generates the topology and routing rule accordingly. During the construction process, the generator will assign each node a hierarchy address, which will be discussed in §3.4. Based on the hierarchy address, the switches use the routing rule to determine the forwarding direction.

3.3 Topology Generator

A topology structure describes the arrangement of nodes and the connections between them. In existing network simulators, the topology is described mainly as an undirected graph consisting of a specific number of nodes and links. While this topology specification is general, it fails to take advantage of the hierarchical structure and symmetry present in datacenter topologies.

We propose a concise topology specification for network simulation to (1) generate topology directly and (2) capture hierarchy and symmetry.

The AI training datacenter is usually constructed from the bottom to the top. Typically, the lowest layer consists of GPUs, and they are connected by ToR switches in the upper layer, which is called the ToR subnet. The ToR switches are then connected by a set of aggregation switches to form a larger network, known as a pod. Finally, to satisfy distributed hyperscale training, the pods are connected by core switches, forming the entire topology. Therefore, a datacenter network can be viewed as being constructed bottom-up, layer by layer, from smaller unit networks. Note that a single GPU at the leaf of the topology is considered as a subnet of edge layer, but only contains one node and has no subnets.

The way in which the upper layer switches connecting the units together are simple, either *full connection* or *single connection*. For example, in Figure 3, switches in the core layer connect only to one switch in each unit, while switches in the aggregation layer connect to all nodes of each unit. These connection patterns benefit load-balancing. There may be several blocks within a layer, each of which is a set of nodes that connect to the same nodes in lower units. The core layer in Figure 3 contains two blocks of size 1.

As the basic construction units at the lower layer, such as pods, are identical, the datacenter topologies exhibit a high degree of symmetry. Therefore, it is sufficient to describe a single unit at each layer, from which all other units can be generated accordingly.

Our topology specification is a set of *structure rules* and *connection rules*.

- A structure rule specifies the arrangement of the nodes in a specific layer with a 3-tuple $rule(n, b, m)$, where n is the number of nodes in the layer, b is the number of nodes in each block, and m is the number of the subnets unit.
- A connection rule specifies how the nodes are connected. For the clos topology, we define two connection rules: *Full Connection* or *Single Connection*.

Figure 3 shows an example of four generation rules, based on which Nüwa generates the four-layer topology.

3.4 Hierarchy addressing

In real networks, each device has a unique IP address to identify itself in the network and facilitate routing between devices. Although we use IP addresses in the simulation, they are not quite necessary since the IP address will not affect the simulation results. In Nüwa, we define a new hierarchy addressing method. Intuitively, hierarchy addresses express the hierarchy and symmetry of the topology and imply the relative position of the nodes.

Unlike an IP address, a hierarchy address is not defined for each node, but rather for a set of nodes. All nodes in the top layer of a subnet unit have the same hierarchy address.

We organize the hierarchy addresses of a topology as a *hierarchy address tree*. As shown on the right of Figure 3, each node of the tree (represented as a dashed box) consists of a set of network devices with the same hierarchy address. Notice that the GPU is considered as a single subnet, thus it has a unique hierarchy address. The advantage of the hierarchy address tree is that we can easily resolve a unique routing path at each network device by mapping the device to a node in the tree.

3.5 Routing Rule

Packet forwarding is one of the most frequent operations in the simulation. At each hop, forwarding of a packet has several steps, (1) extracting the packet's destIP address; (2) looking up in the routing table for a set of ECMP routing entries; (3) selecting one of the routes based on hash values; (4) send the packet out through the corresponding ports.

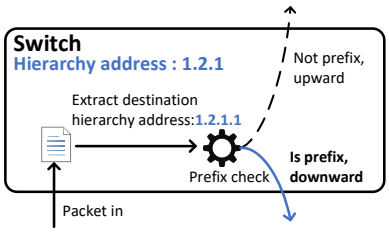


Figure 4: An example of forwarding packet in switch. The switch's hierarchy address is 1.2.1, the packet's destination is 1.2.1.1, whose prefix is 1.2.1, so forward downward.

Algorithm 1: `Up-down routing rule`(did, cid, l)

Input: did: the unit id of destination end point.
cid: the unit id of current node.
l: the layer of current node.
Output: $Outp$: the out ports to the next hop.

```
// Check if current id is prefix of destination id
isPrefix ← CheckPrefix(did, cid);
// Forward upward
if isPrefix == true then
    Outp ← PortsToUpwardNodes
// Forward downward
else
    // Get the i-th level id of the destination id
    nextSubnet ← GetLevelId(l, did);
    Outp ← GetPortsToSubnet(nextSubnet);
```

Among these steps, routing table lookup dominates the execution time, accounting for more than 90% of execution time for large-scale networks, as shown in Table 1. Although ns-3-datacenter optimizes lookup performance, it still maintains large routing tables, which take up a lot of memory resources, and limits its scalability to hyperscale cluster simulation. Luckily, we found that routing rules can be abstracted in clos topology and propose a lookup-free forwarding method.

As mentioned in §3.4, any pair of nodes has only one routing path in the hierarchy address tree. Specifically, a packet is always first forwarded upward and then downward to the destination in the tree. The forward direction is determined by the positional relationship between the device and the destination in the tree. If the current device is an ancestor of the destination, then the packet should be forwarded downward. Otherwise, the packet goes up towards the root node of their sub-tree.

As Algorithm 1 shows, in the tree, if a node u is an ancestor of node v, i.e., u's address is a prefix of v's address, the packet will be forwarded upward, else downward. In the clos topology, all upward ports are used for ECMP, and downward ports to a specific subnet are fixed. Prefix checking is simply a trivial bit-wise operation. Figure 4 shows an example of forwarding a packet targeting destination 1.2.1.1 at device whose address is 1.2.1. Because 1.2.1 is prefix of 1.2.1.1, so the packet should be forwarded downward to the 1.2.1.1 subnet.

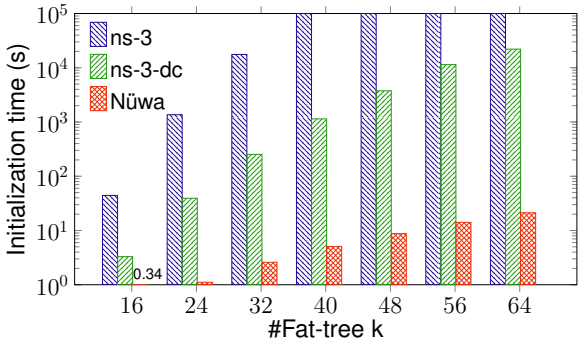


Figure 5: Initialization time of ns-3, ns-3-dc and Nüwa.

Table 2: Lookup time in all-reduce execution

Fat-tree (k)	16	24	32	40	48	56	64
ns-3	1930.95	23256	1 day[2]	1 day	1 day	1 day	1 day
ns-3-dc	2.01	9.11	22.88	59.36	100.87	269.13	OOM[3]
Nüwa	0.012	0.02	0.12	0.19	0.43	0.93	1.12

4 Evaluation

We implemented a prototype of Nüwa based on ns-3 with ~3k LOC of C++. In this section, we evaluate the efficiency and scalability of Nüwa, in comparison with ns-3 [15] and ns-3-datacenter [19]. We show that Nüwa: (1) Achieves negligible initialization time. (2) Reduces a lot of memory resources. (3) Improve the efficiency of the execution stage.

4.1 Workload and setup

Topology. We choose two types of topologies: (1) fat-tree topologies of various sizes, and (2) the Meta AI training clos topology [8], which contains 24,576 GPUs. All links are 100Gbps and the link latency is $2\mu s$.

Workload. We simulate with two types of workloads: (1) all-reduce and (2) all-to-all, two common traffic patterns in distributed AI training tasks. For the all-reduce workload, each GPU sends 1MB of data. For the all-to-all workload, each GPU sends 400KB of data. As the simulation results indicate, these two workloads take approximately $2ms$ to run in a fat-tree AI training cluster.

Setup. All evaluations run on a Linux server, with two 32-core Intel(R) Xeon(R) Gold 6338 CPU @ 2.00GHz and 1T memory.

4.2 Results for Fat-trees

Efficiency As shown in Figure 5, Nüwa significantly reduces the initialization time to a nearly negligible level. Figure 6 and Figure 7 show that Nüwa's execution efficiency is higher than ns-3 and ns-3-datacenter, achieving up to 20% speed-up compared to ns-3-datacenter. Table 2 shows Nüwa's lookup takes very little time.

The speed up is attributed to the reduction in the overhead of routing tables and lookup operations. As shown in Figure 10, Nüwa reduces up to 60% cache reference times during simulation compared to the ns-3-datacenter.

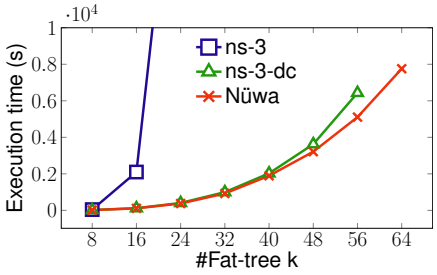


Figure 6: Execution time of all-reduce

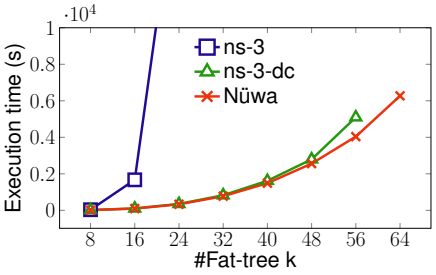


Figure 7: Execution time of all-to-all

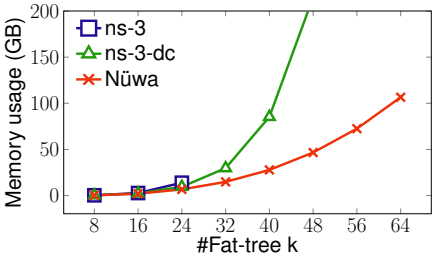


Figure 8: All-reduce memory usage

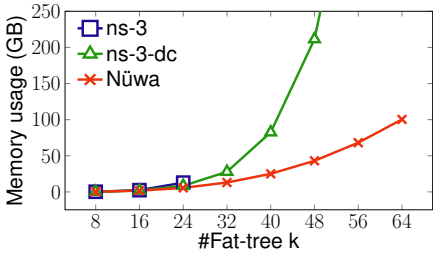


Figure 9: All-to-all memory usage

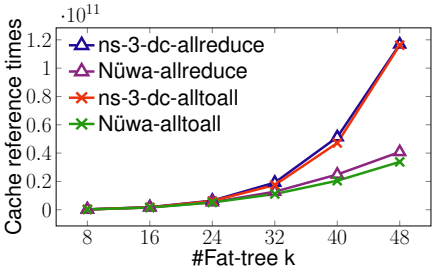


Figure 10: Cache reference times

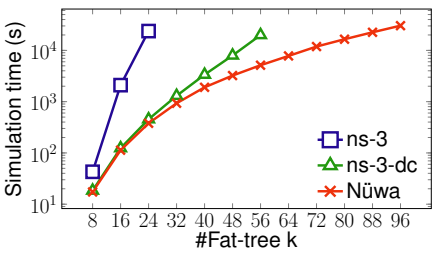


Figure 11: Total simulation time at scale

Scalability Nüwa saves a lot of memory by replacing routing tables with routing rule, achieving good scalability. As shown in Figure 8 and Figure 9, Nüwa's memory usage increases much slower than ns-3 and ns-3-datacenter, the memory usage is only about 20% of that of the datacenter in large-scale simulation.

We evaluate Nüwa on a fat-tree network with up to 222,184 GPUs. Nüwa takes only 76 seconds for initialization. The total simulation time results of 1MB-allreduce workload in ns-3, ns-3-dc and Nüwa are presented in Figure 11, Nüwa completes the 222,184 GPU fat-tree simulation in less than 9 hours.

4.3 Results for Meta AI training cluster

We continue to evaluate Nüwa's performance on Meta's 3-layer clos AI training cluster. Since ns-3 fails to simulate such a large-scale cluster, we only compare Nüwa with ns-3-datacenter. As shown in Table 3, Nüwa is more efficient in both time and memory. Nüwa reduces the initialization time from 1 hour to just 8 seconds and up to 20% faster for the execution stage. In terms of memory, Nüwa only costs 20% memory of the ns-3-datacenter,

Table 3: Meta cluster simulation results

	ns-3-dc	Nüwa
Initialization time (s)	3377.98	8.06
AllReduce time (s)	3978.34	3169.49
AllToAll time (s)	2808.47	2559.61
AllReduce Memory (GB)	180.66	43.16
AllToAll Memory (GB)	175.51	38.01

[2]Denotes simulation time is more than 1 day.
[3]Denotes simulation out of memory.

5 Discussion

There are several open questions to be solved to extend the capability of Nüwa to a comprehensive solution.

5.1 Supporting Other Topologies

Low-diameter topologies such as Dragonfly [11], Dragonfly+ [18], Slimfly [3] are also popular choices for supercomputer network due to lower cost. The hierarchy of low-diameter topologies is more complicated than the Clos topology. For example, Dragonfly connect a group of ToR switches with a full mesh, and groups are interconnected with global links. The intra-group mesh links and global links both connect two nodes from the same hierarchy level. Our solution is based on the strict hierarchical structure of the Clos topology. How to extend the forwarding rules beyond the simple up-down rule is one of the key open questions.

5.2 Simulating Failures in Network

Failures are common in the hyper-scale AI training cluster [8]. Simulating network behaviors under failure scenarios are one of the key use cases of network simulation. Existing network simulators like ns-3 can simulating failures but with a time-consuming recomputation of whole routing tables. It is desired to let Nüwa support simulating failures with the efficient rule-based approach. However, current Nüwa only supports strictly symmetrical topology which the symmetry will be broken by the failure. Supporting simulation with network failures efficiently is an important future direction of Nüwa.

5.3 Supporting Other Simulators

The advantages of Nüwa also apply to other network simulators like OMNeT++ [14] and DONS [9]. However, how to make the

Nüwa adopt to multiple simulators without high porting effort is also a problem we want to tackle in the future.

6 Conclusion

The initialization of the control plane has become a bottleneck for scalable network simulation. This paper introduces a generative control plane for AI network simulation, Nüwa, which is efficient in both time and memory. Nüwa achieves efficiency and scalability by replacing the routing table with high-level routing rule. Our evaluation shows that Nüwa achieves negligible initialization time, saves a lot of memory consumption, and speeds up execution.

Acknowledgments

The authors gratefully acknowledge the anonymous reviewers for their valuable and constructive comments. This work is partially supported by the National Natural Science Foundation of China (No. 62272382)

References

[1] Anubhavnidhi Abhashkumar, Kausik Subramanian, Alexey Andreyev, Hyojeong Kim, Nanda Kishore Salem, Jingyi Yang, Petr Lapukhov, Aditya Akella, and Hongyi Zeng. 2021. Running BGP in Data Centers at Scale. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 65–81.

[2] Songyuan Bai, Hao Zheng, Chen Tian, Xiaoliang Wang, Chang Liu, Xin Jin, Fu Xiao, Qiao Xiang, Wanchun Dou, and Guihai Chen. 2024. Unison: a parallel-efficient and user-transparent network simulation kernel. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 115–131.

[3] Maciej Besta and Torsten Hoefler. 2014. Slim fly: A cost effective low-diameter network topology. In *SC'14: proceedings of the international conference for high performance computing, networking, storage and analysis*. IEEE, 348–359.

[4] Matt Brown, Ari Fogel, Daniel Halperin, Victor Heorhiadi, Ratul Mahajan, and Todd Millstein. 2023. Lessons from the evolution of the Batfish configuration analysis tool. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 122–135.

[5] Google Cloud. 2025. VM instance pricing | Google Cloud. https://cloud.google.com/compute/vm-instance-pricing?hl=en#section-5.

[6] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. A general approach to network configuration analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 469–483.

[7] Richard M Fujimoto. 1990. Parallel discrete event simulation. *Commun. ACM* 33, 10 (1990), 30–53.

[8] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, et al. 2024. Rdma over ethernet for distributed training at meta scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 57–70.

[9] Kaihui Gao, Li Chen, Dan Li, Vincent Liu, Xizheng Wang, Ran Zhang, and Lu Lu. 2023. Dons: Fast and affordable discrete event network simulation with automatic parallelization. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 167–181.

[10] Mark Handley. 2023. htsim. https://github.com/Broadcom/csg-htsim.

[11] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. 2008. Technology-driven, highly-scalable dragonfly topology. *ACM SIGARCH Computer Architecture News* 36, 3 (2008), 77–88.

[12] Huawei 2012 Labs. 2020. ns.py. https://github.com/TL-System/ns.py.

[13] Nuno P Lopes and Andrey Rybalchenko. 2019. Fast BGP simulation of large datacenters. In *Verification, Model Checking, and Abstract Interpretation: 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13–15, 2019, Proceedings 20*. Springer, 386–408.

[14] OpenSim Ltd. 2018. OMNeT++. https://omnetpp.org/.

[15] nanam. 2017. ns-3. https://www.nsnam.org/.

[16] Dylan Patel and Daniel Nishball. 2024. 100k H100 Clusters: Power, Network Topology, Ethernet vs InfiniBand, Reliability, Failures, Checkpointing. https://www.semianalysis.com/p/100000-h100-clusters-power-network.

[17] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, et al. 2024. Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 691–706.

[18] Alexander Shpiner, Zachy Haramaty, Saar Eliad, Vladimir Zdornov, Barak Gafni, and Eitan Zahavi. 2017. Dragonfly+: Low cost topology for scaling datacenters. In *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*. IEEE, 1–8.

[19] vamsiDT. 2022. ns-3-datacenter. https://github.com/inet-tub/ns3-datacenter.

[20] Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Dan Li, Li Chen, Heyang Zhou, Linkang Zheng, Sen Zhang, Yikai Zhu, et al. 2025. SimAI: Unifying Architecture Design and Performance Tunning for Large-Scale Large Language Model Training with Scalability and Precision. (July 2025).

[21] Janaka Wijekoon, Rajitha Tennekoon, Erwin Harahap, and Hiroaki Nishi. 2015. Introducing a distance vector routing protocol for ns-3 simulator. In *Proceedings of the 8th International Conference on Simulation Tools and Techniques*. 38–46.

[22] William Won, Taekyung Heo, Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. 2023. Astra-sim2. 0: Modeling hierarchical networks and disaggregated systems for large-model training at scale. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 283–294.

[23] xAI. 2025. Grok 3 Beta — The Age of Reasoning Agents. https://x.ai/news/grok-3.

[24] Guihua Zhou, Guo Chen, Fusheng Lin, Tingting Xu, Dehui Wei, Jianbing Wu, Li Chen, Yuanwei Lu, Andrew Qu, Hua Shao, et al. 2021. Primus: Fast and robust centralized routing for large-scale data center networks. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 1–10.