DONS: Fast and Affordable Discrete Event Network Simulation with Automatic Parallelization

Kaihui Gao[★‡], Li Chen[‡], Dan Li[★‡], Vincent Liu[†], Xizheng Wang[★], Ran Zhang[‡], Lu Lu[◇]
[★]*Tsinghua University* [‡]*Zhongguancun Laboratory* [†]*University of Pennsylvania* [◇]*China Mobile Research Institute*

ABSTRACT

Discrete Event Simulation (DES) is an essential tool for network practitioners. Unfortunately, existing DES simulators cannot achieve satisfactory performance at the scale of modern networks. Recent work has attempted to address these challenges by reducing the traffic processed via novel approximation techniques; however, we argue in this paper that much of the slowdown of existing DES simulators is due to their underlying software architecture.

Using ideas from high-throughput simulation of virtual worlds in gaming, this paper presents a fundamental redesign of DES network simulator, DONS, that marries domain-specific aspects of packet-level network simulation with recent advances in data-oriented design. DONS can automatically parallelize simulation within and across servers to achieve high core utilization, low cache miss rate, and high memory efficiency. On a relatively weak ARM-based laptop (MacBook Air (M1, 2020)), DONS can simulate one second of a 100 Gbps, 1024-server data center in 22 minutes (a speedup of 21× compared to OMNeT++). On a cluster of CPU-based servers, DONS can achieve a speedup of 65×, matching the order of magnitude of recent GPU-accelerated deep learning performance estimators, but without any loss of accuracy.

CCS CONCEPTS

• **Networks** → **Network simulations**; • **Computing methodologies** → **Parallel computing methodologies**;

KEYWORDS

Network simulation, Data-oriented design, Automatic parallelization, Distributed computing

ACM Reference Format:
Kaihui Gao, Li Chen, Dan Li, Vincent Liu, Xizheng Wang, Ran Zhang, Lu Lu. 2023. DONS: Fast and Affordable Discrete Event Network Simulation with Automatic Parallelization. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3603269.3604844

1 INTRODUCTION

Network simulation is critical for network planning, operations, and innovations. Discrete Event Simulation (DES) [22, 29, 39, 51] is the predominant paradigm for network simulations. In a DES simulation, every event (*e.g.*, send or receive packet) is treated as a discrete event—a full simulation trace is simply the sequential execution of each event, ordered by timestamp. Unfortunately, existing DES simulators cannot provide satisfactory performance for modern networks due to their scalability issues [55, 57], as we confirm using experiments (§2.2). Parallel and distributed simulation frameworks exist, but as others [57] have noted, and we verify in §2.2, existing parallel DES frameworks also scale poorly and can sometimes perform worse than serial execution.

Improving the scalability of DES is difficult, but recent work has made progress in two primary axes. First, for specific research areas where flow-level analysis is useful and theoretically grounded, *e.g.*, congestion control or active queue management, researchers have developed flow-level continuous-time simulators (CTS) [5, 12, 26, 30, 31, 35, 40]. CTS can achieve higher scalability because it ignores packet-level events and works on a higher abstraction level than DES. Second, for applications where quick estimates of end-to-end performance metrics are sufficient, researchers have developed AI-powered performance approximators (APAs) [24, 43, 55, 57], which use ML to reduce the total amount of work done by the simulator and leverage the power of GPU hardware acceleration.

Although both approaches offer substantial speedups for their target use cases, network architects often still need full-fidelity packet-level results [27, 28, 53, 58]. Furthermore, many of the above techniques still rely on DES as a substrate (*e.g.*, simulating flow-level interactions [43] or collecting training data [57]). Thus, part of their performance is still bottlenecked by DES.

Are all of these simulation frameworks doomed to poor scalability by fundamental flaws in the DES paradigm? After a careful study of existing DES simulators, we answer this question in the negative. Specifically, we argue that the dismal performance of today's DES simulators is *not* due to the DES abstraction itself but, rather, the underlying software architecture of today's DES instantiations and their methods of parallelization. We summarize the problems (P1–3) of existing DES simulators:

P1 **Poor multi-core and cache efficiency:** Current DES simulators are not optimized for performance and scalability even on a single machine, which leads to poor multi-core utilization and cache efficiency (§2.2).

P2 **Poor memory efficiency:** Existing DES simulators lack support for multi-threading. Process-based parallelization cannot share data without incurring context-switching overhead. Thus, data such as network topology and routing information are duplicated for each process, resulting in high memory consumption, which restricts the size of networks that can be simulated in a single machine.

P3 **Manual and inefficient partitioning:** For parallelization on a single multi-core machine or a cluster of servers, users of

This work is licensed under a Creative Commons Attribution International 4.0 License.
ACM SIGCOMM '23, September 10–14, 2023, New York, NY, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0236-5/23/09.
https://doi.org/10.1145/3603269.3604844

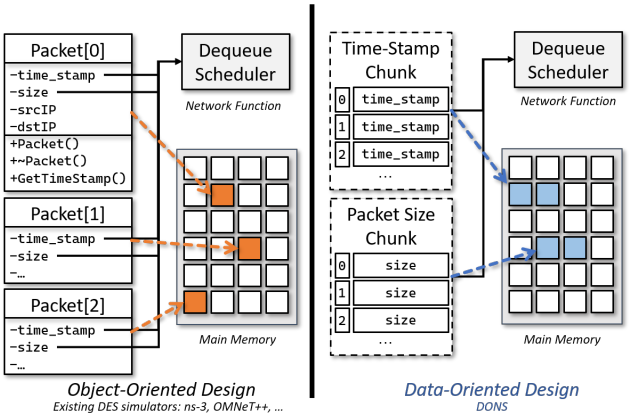


Figure 1: Object-Oriented Design vs. Data-Oriented Design.

existing network simulators must manually partition the network over available workers, which is coarse-grained and error-prone. Bad partitions also result in poor parallel execution performance, sometimes even worse than serial execution (§2.2).

To achieve scalable performance, we believe the DES simulation engine must be fundamentally redesigned. In particular, when video game developers met similar problems with cache efficiency and poor scalability more than two decades ago, they turned toward a paradigm called Data-Oriented-Design (DOD) [14], which is still widely used in game development since the 2000s [6, 13, 48, 49]. This paper presents the design and implementation of a Data-Oriented Network Simulator (DONS), a DES network simulator using a novel engine designed with DOD principles.

The key characteristic of DOD is putting data of the same type together. This vastly differs from Object-Oriented Design (OOD), which all existing DES simulators adopt. In OOD, data is encapsulated in objects. We use an example (Figure 1) to showcase why DOD is superior in terms of performance. Consider an output port with three packets in its buffer. The dequeue scheduler must look at their packet sizes and enqueue timestamps to determine which packet to send. For OOD, these two pieces of information are stored in different objects that are scattered in main memory at run-time, and the CPU core running the dequeue scheduler must fetch them one by one before making a decision. In contrast, DOD places the data of the same type together: the timestamps of all packets are placed contiguously, as well as the sizes. Thus, the dequeue scheduler can fetch the chunks containing the relevant information into the cache at once before beginning processing. The dequeue network function can directly access the packet's information in the chunk using the identifier (ID) of the packet. Doing so also enables efficient multi-core parallelization and cross-core load balancing. Once the data chunks are fetched into the L3 cache, the same network function of all devices can run in a data-parallel manner, because the L3 cache is shared by all cores. In addition, network devices usually have similar network functions (*e.g.*, forwarding and dequeue in switches). Thus, DOD is particularly well-suited for the software architecture of DES network simulator.

DONS features automatic parallelization at both server-scale and cluster-scale, comprehensively addressing P1–3. With DONS, we make the following contributions:

- We design a network-specific DES simulation engine using DOD principles that breaks the coupling between different network devices. This allows DONS to identify cross-device batching and parallelization opportunities that existing simulators ignore. This greatly improves multi-core utilization and significantly reduces cache misses. DONS also uses a thread-pool-based run-time environment, which enables thread-based parallelization and fine-grained automatic load-balancing on multiple cores. Taken together, DONS achieves up to 22× speedup against ns-3 in large-scale simulations, with high CPU utilization and efficient memory usage on a single machine (§6.1).
- Across multiple machines, we improve upon prior work by introducing computation capacity and traffic pattern into a *time-cost model*. Using this model, we develop a partitioning algorithm that enables automatic and efficient parallelization on a cluster. Using 8 servers to simulate a large-scale DCN, DONS achieves a 65× speedup compared to OMNeT++ (§6.2).
- We prove the correctness of DONS's DOD-based simulation engine. Our experiments also confirm that the simulation results of DONS are the same as that of ns-3 and OMNeT++.
- We release the source code of DONS[1] to promote further research in network performance evaluations. DONS is based on the Unity framework [47], and thus, it works on consumer laptops, CPU-based servers, and even mobile devices with ARM cores. Using a MacBook Air (M1, 2020) and within 22 minutes, DONS can finish simulating a 1024-server DCN with 100 Gbps interfaces running for 1000 milliseconds (a speedup of 21× compared to OMNeT++). DONS enables networking practitioners to simulate networks at today's scale with available and affordable hardware without loss of accuracy.

This work does not raise any ethical issues.

2 BACKGROUND AND MOTIVATION

In this section, we overview existing network simulation architectures, including DES. We then examine the scalability issues of existing DES simulators and motivate the design decisions of DONS.

2.1 Existing Network Simulators

Existing network simulators fall into three paradigms: continuous-time simulation (CTS), AI-powered performance approximation (APA), and discrete-event simulation (DES).

CTS. CTS takes a holistic view of the network. CTS seeks to describe the network state using sets of equations or models, estimating performance metrics by computing the evolution of system states. CTS can be further divided into control-theoretic CTS [5, 30, 31, 35], network calculus [12, 26], and queueing-theoretic CTS [40]. When the underlying theoretical foundation is clear for a networking scenario, such as congestion control, CTS can be of great use, as it works on a higher abstraction level than DES. However, high-fidelity CTS also suffers from scalability issues and, in some cases, cannot scale beyond a single device [55].

APA. Researchers have also recently explored the design of APAs [24, 43, 55, 57], which use deep neural networks to model end-to-end

[1] https://github.com/dons2023/Data-Oriented-Network-Simulator

network performance metrics. An APA recasts the network as a DNN model that is trained using real traffic traces and the corresponding network performance metrics. Given an embedding of facts about the simulation scenario, an APA can predict end-to-end performance metrics such as RTT, flow completion time (FCT), and packet drop rate. These models can then be optionally embedded into larger simulation architectures [57]. Because APAs eliminate the need to simulate billions of packets and, instead, leverage parallel execution using GPUs, APAs enjoy obvious performance advantages in terms of simulation completion time. Given the superior scalability of GPUs, APAs can be useful for quick end-to-end performance estimates. However, it is not always desirable to trade accuracy for performance, and even when it is, packet-level network simulation can still be a bottleneck, *e.g.*, when collecting training data or using the models in a larger simulation [43, 57].

DES. In contrast to CTS and APA, DES simulators provide accurate packet-by-packet traces in arbitrary networking scenarios. For this reason, most of the networking community still relies on them for performance evaluations [27, 28, 53, 58].

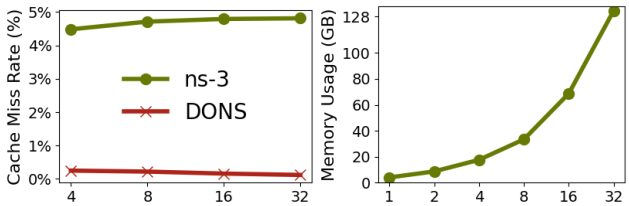
A DES simulator models the operation of a network as a discrete sequence of events in time, *e.g.*, packet enqueues and dequeues. Each event occurs at a particular instant in the simulation timeline and marks a state change in the network. As an example of their success, as of January 2023, ns-2/3 [22, 39] has garnered more than 4300 citations.

However, with networks growing larger and larger, DES simulators are failing to keep up with these hardware trends. Scalable parallel simulation has been studied extensively since the 1970s, and many parallel simulation synchronization algorithms have been proposed [16]. Despite these efforts, the current DES simulators still struggle to scale. For example, a simulation of a 65K-server data center network (DCN) takes more than 9 days to complete (§6.2) on a state-of-the-art DES simulator, OMNeT++ [51], even when using an 8-machine cluster. This is insufficient for today's "hyper-scalers" like Google and Amazon, which are building data centers containing tens of thousands of servers [37, 45].

For network researchers, the need to improve the scalability of DES simulators is urgent. The unsatisfactory performance has led to a curious inversion in networking experimentation: traditionally, simulations are used for large-scale experiments, and testbed evaluations are for small-scale prototyping. With the fast evolution of cloud and container technology and the slow improvement of simulators, nowadays, DES simulators are only fit for small-scale experiments; thus, people are skipping prototyping on DES simulators altogether and directly conducting real large-scale experiments on the cloud [9, 19, 38, 56]. Running large-scale experiments on the cloud is expensive, and this becomes a barrier to entry for networking practitioners with a small budget, especially in academia. Without a scalable DES simulator that can work on common hardware platforms, they are prevented from experimenting at the scale of current networks and cannot provide convincing validation of their ideas at scale.

2.2 Problems of Existing DES Simulators

Through extensive experiments, we identify that the fundamental limitation of existing DES simulators is their OOD-based software architectures. The fundamental structure of these programs gives rise to three primary problems (detailed below) that are difficult to mitigate using just engineering effort. Note that we conduct the experiments mostly on ns-3 but emphasize that other DES simulators share similar performance characteristics, and are also lacking in scalability [57].



(a) Cache miss rate **(b)** Memory usage of ns-3

Figure 2: Poor cache and memory efficiency of ns-3.

1. High cache miss rate: We use the default ns-3 (*i.e.*, with single process) to simulate an ordinary FatTree [3] network with a full-mesh dynamic flow pattern (§6), and record the miss rate of the L3 cache. As shown in Figure 2a, the L3 cache miss rates of ns-3 are always higher than 4% for different sizes of networks. We believe this is because current OOD-based DES simulators forgo data layout optimizations so that data of the same type may be scattered in memory and encapsulated in different objects. Thus, existing simulators fail to put relevant data into the cache at the same time and must access them in an object-by-object manner, resulting in a significant number of cache misses. As a comparison and a preview, in Figure 2a, we plot the L3 cache miss rates of DONS after optimizing the data layout, capping miss rates to less than 0.15% for all simulation scenarios.

2. Poor memory efficiency: By default, popular simulation frameworks like ns-3 and OMNeT++ are single-threaded[2,3]. To enable the usage of multiple cores, both simulators require users to manually partition their simulation topology into sub-graphs and execute them in different processes on different cores, communicating through protocols like MPI. In the parallel DES literature, each parallel simulation process is called a Logical Process (LP).

Because the different simulation processes would need to pay high IPC costs to share and maintain common data, the same data, such as network topology and routing information, are often duplicated for each process [33]. As shown in Figure 2b, using ns-3, we change the number of processes to simulate a FatTree topology with k=16 (FatTree16), which has 1024 servers, 320 switches, and 3072 links with 100 Gbps. We observe that ns-3 with 32 processes can saturate the cores, but its total memory usage is 132.5 GB. This high consumption limits the size of simulated networks on a single machine: simulating FatTree32 with 32 processes requires >5,000 GB of memory.

3. Manual partitioning is not optimal: In addition to the above, manual partitioning of network subgraphs to workers must be hand-tuned and may still be frequently imbalanced. An imbalanced workload among participating CPU cores or servers may prolong

[2]https://github.com/nsnam/ns-3-dev-git/blob/master/src/core/model/default-simulator-impl.cc#L197
[3]https://github.com/omnetpp/omnetpp/blob/master/src/sim/csimulation.cc#L1074

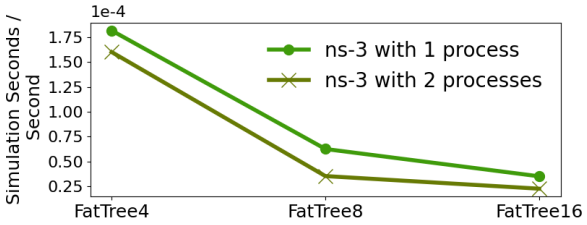


Figure 3: The performance of the parallel ns-3 is degraded.

simulation completion time. When a network is badly partitioned, a multi-process simulation can perform worse than a single-process simulation. Figure 3 shows a bad case of partitioning: the nodes in the network are randomly divided into the two ns-3 processes. We can see that the simulation speed of two processes in parallel is slower. This is because of the high synchronization overhead between processes. Formally, every LP must adhere to a *Local Causality Constraint* (LCC) [16], which refers to the fact that each LP processes events strictly in chronological order—the synchronization algorithm should guarantee that, before processing an event at simulated time t, no additional events with $t_e<t$ will ever occur. With bad simulation partitioning, synchronization between LPs introduces significant consistency overhead, making parallel processing slower than a single process [57].

2.3 Motivating DONS's Design Choices

With the above observations, we see plenty of headroom for improving the scalability of DES simulations. DONS centers around three key ideas.

1. Adopt DOD to optimize the data layout. To enable efficient parallelization and improve cache efficiency, we adopt DOD and fundamentally restructure the storage and management of simulation data.

To the best of our knowledge, all existing DES simulators adopt OOD to design their engines. Although OOD claims to "organise code around data", it actually organises source code around data types rather than physically grouping individual fields and arrays in an efficient format for access by specific functions [54]. For example, in all existing DES simulators, a packet is encapsulated as an object, containing all its information, such as size, timestamps, source and destination addresses, protocol, *etc.* At run-time, objects of packets are scattered in heap memory. Many network functions, such as packet scheduling, generally do not need all of the information about a packet but only specific pieces of information (*e.g.*, timestamps) from all relevant packets. After our careful investigation, we found that network functions of this nature account for more than 78% of the total network functions and protocols in the current ns-3 [44]. When a function is incurred, current simulators fetch the relevant packet objects one by one into the CPU cache and access only one field in each of them. This random access pattern results in inefficient cache line usage, ineffective prefetching, and high cache miss rates, which we witness in §2.2.

We adopt DOD for its focus on data layout. DOD advocates storing data of the same type (*e.g.*, the same attribute of all objects) together, akin to the strategy taken by columnar databases. In DOD, all processing logic considers only the transformations of data, which can be parallelized easily. In this way, the attributes of interest to the running network function can be fetched simultaneously to the cache and processed in parallel on multi-core CPUs.

DOD also breaks the coupling between network devices. In existing DES simulators, a device's functions are executed sequentially for each packet. In contrast, DONS (on a single machine) executes the same functions of all devices together. This allows DONS to exploit cross-device parallelization opportunities, which existing simulators ignore. We prove the correctness of this parallel execution model (§3.3) and confirm that packet traces produced by this model are the same as those of ns-3/OMNeT++ (§6.1).

DONS uses a popular DOD software framework, the Entity-Component-System (ECS) [49] framework, and we describe the ECS-based network modeling in §3.2.

2. Use multi-threaded parallelization with a novel batch-based threading model. Within a machine, DONS employs a thread-pool-based run-time environment to achieve automatic multi-threaded parallelization, making full use of the multi-core CPU. Compared to process-based parallelization, multi-threaded parallelization is superior in at least two aspects: efficient memory usage and low cross-thread communication overhead, because data can be shared easily between threads within a process. We also develop a batch-based threading model. Within a batch, the processing logic of the network function can be executed in a data-parallel fashion without synchronization algorithms, which we elaborate in §3.3.

3. Design an automatic partitioning algorithm for multi-server parallel simulation. Across machines, we cannot directly generalize the above single-machine, batch-based threading model design because the communication latency between servers is much larger than that between cores, and the bandwidth is smaller. To enable parallel DES simulation among servers, we instead layer the existing methodology of DES simulators (partition the simulation topology into sub-graphs and execute each on a different server) on top of the single-machine DONS execution model, with the following improvements (§4):

- We design a new time-cost model for partitioning the topology. To the best of our knowledge, this is the first time computation capacity and the traffic pattern are considered in a time-cost model for parallel DES simulations.
- With the new model, we develop an automatic heuristic partitioning algorithm that can balance the load among servers. With this algorithm, users no longer need to iteratively tune the partition for parallel DES to achieve satisfactory load-balancing and can program the simulation as if it is on a single machine.

3 DONS DESIGN

In this section, we first overview the architecture of DONS and then describe its ECS-based modeling of network simulation. We proceed to the design of the batch-based threading model, and finally, we provide proof of the correctness of DONS's execution model.

3.1 Architecture

DONS adopts a simple Client-Server architecture, which is shown in Figure 4. The DONS Manager accepts a user's submission of simulation settings and orchestrates the simulation on one or more

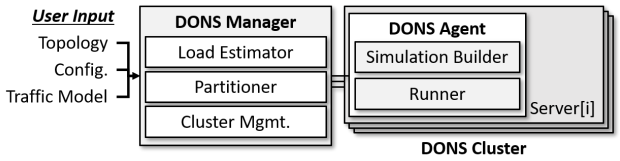


Figure 4: DONS's system architecture.

machines. Each participating machine runs a single DONS Agent, and it executes the simulation tasks assigned by the Manager. The Manager and the Agent can run on the same machine.

DONS Manager. The Manager has three core components, the Load Estimator, the Partitioner, and the Cluster Controller. The Load Estimator and Partitioner work together to produce parallel execution plans when there is more than one machine in the cluster. We describe them in detail in §4. The Cluster Controller maintains a connection with all the Agents, monitors the health and simulation progress of the DONS Cluster, and aggregates the simulation results.

DONS Agent. Each machine runs only one instance of DONS Agent. The Agent has two core components, the Simulation Builder and the Runner. The Simulation Builder constructs the simulation scenario locally based on the setting information from the DONS Manager. The Runner is a thread-pool-based run-time environment that executes the simulation.

The Agent also sets up and manages the communication channels between the DONS Manager, as well as other Agents. During distributed simulation across machines, the Agents perform clock synchronization among themselves using their direct channels.

3.2 Network Modeling

As discussed in §2.3, we use the ECS framework to implement DONS. ECS is a popular framework for realizing DOD principles and has been adopted in many large-scale gaming and virtual reality software projects [6, 13]. In the following, we describe how we model network simulations using concepts in ECS.

First, we introduce the concepts in ECS. As the name indicates, ECS comprises three concepts:

- **Entity**: The entities are general-purpose objects in the simulation scenarios. An entity is not an object in OOD. Entities serve to identify which pieces of data belong together logically. Thus, it is usually implemented as a unique identifier (UID).
- **Component**: A component labels an entity as possessing a particular aspect or attribute and holds the data needed to model that aspect. For example, in a physics engine, every entity has a mass component associated with it. Data for all instances of a component are stored together in physical memory, indexed by the UIDs of entities. This enables efficient memory access for systems that operate over many entities.
- **System**: A system contains all logic of a process. It acts on all entities with the desired components and transforms the component data from its current state to the next state. For example, a physics system may query entities having mass, velocity, and position components and iterate over the results by doing physics calculations on the sets of components for each entity. A system can be considered a behavioral description of an aspect of the simulation.

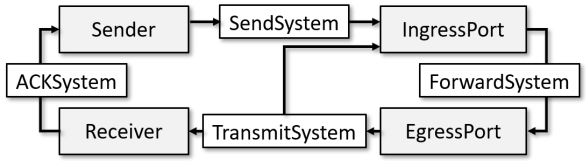


Figure 5: DONS modeling of network simulation.

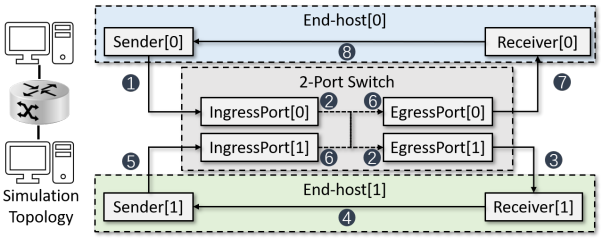


Figure 6: Example of DONS modeling: 2 hosts connected by a 2-port switch.

With the ECS framework for DOD, we analyze the life cycles of all packets in the network data plane and abstract out the entities, components, and systems in network simulations. Our abstraction is shown in Figure 5. A packet originates from a Sender Entity and then traverses a data plane forwarding path comprised of consecutive IngressPort Entities and EgressPort Entities. Finally, the packet arrives at the Receiver Entity, which triggers its corresponding Sender Entity to send an acknowledgment (ACK) if needed. Based on this abstraction, we design a novel network simulation engine that conforms to ECS framework. The engine currently encompasses four entities and four systems. Due to space limits, we introduce the components along with their associated entities, and we refer interested readers to our released source code, where the definitions of all components can be found.

DONS Entities. DONS has four entities:

- **Sender**: it abstracts all the traffic generators. It is associated with components such as ID, source and destination addresses, flow demands, and congestion windows.
- **Receiver**: it represents the destination of the traffic. It is associated with components such as ID, source and destination addresses, and receive window.
- **IngressPort**: it models the input ports of network devices, such as an end-host or a switch. A switch has multiple IngressPorts, and each host has one IngressPort. The IngressPort Entity contains components such as a forwarding table and buffer.
- **EgressPort**: it models the output port of the network devices. A switch can have multiple EgressPorts, and it contains components such as scheduling strategy and buffer.

We do not define packets as entities because of the sheer amount of packets in modern networks. Instead, DONS treats packets as plain data to be transformed by systems and put the packet data in the components of different entities.

We use an example to help the understanding of DONS's entity modeling. In Figure 6, we show a simple topology with 2 end-hosts connected by a 2-port switch. For each of the two ports, DONS's

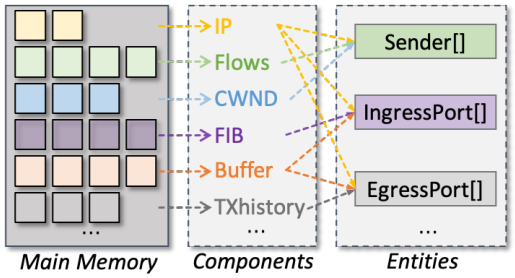


Figure 7: DONS's component data layout.

modeling divides it into two logical entities: the IngressPort and the EgressPort. Within the switch, the IngressPorts are connected to all EgressPorts in a cross-bar.

DONS Systems. DONS has four systems operating on component data, corresponding to four categories of network behavior, and each system governs one aspect of the simulation. We argue that all network functionality can be split in this way. The SendSystem manages all the packet generation behavior; the ForwardSystem is responsible for packet-moving within a device, *i.e.*, forwarding; the TransmitSystem controls the packet-moving across devices; and the ACKSystem handles all the processing regarding the termination of the packets. Specifically, using the same example in Figure 6:

- **SendSystem**: it generates packets from all the Senders and places them in the buffer component of the connected IngressPort Entities. It also maintains the state machines for transport layer congestion control protocols, *e.g.*, TCP, for all the Senders. In the example, Steps ❶ & ❺ are processed by this system.
- **ForwardSystem**: For all the IngressPorts in the simulation, this system forwards packets to the corresponding EgressPorts according to the forwarding information base (FIB), which is determined in the build phase of the simulation. It can also enable Equal Cost Multi-Path (ECMP). Steps ❷ & ❻ are processed by this system.
- **TransmitSystem**: it first sorts the packets from all the EgressPorts' buffer components in chronological order and then moves them to the linked IngressPorts or Receivers according to the pre-defined scheduling strategies of the EgressPorts. Steps ❸ & ❼ are processed by this system.
- **ACKSystem**: it is responsible for processing the packets received by all the Receivers. For example, if a flow uses TCP, the ACKSystem checks the packet sequence number and then registers an ACK packet to its paired Sender Entity. Steps ❹ & ❽ are processed by this system.

Component data layout. As shown in Figure 7, DONS places component data of the same type together. The data belongs to and is indexed by different entities. For large components that cannot fit in a memory page, DONS splits them into chunks and manages the indexing metadata. This improves cache-friendliness and helps multi-core parallelization. Take the TransmitSystem as an example. This system processes all the cross-device packet transmission for all the links in a simulation scenario. It can fetch the relevant chunk(s) into the cache at once and begin processing. Once the data chunks are in the L3 cache, the TransmitSystem can run in a data-parallel manner across multi-cores because the L3 cache is shared by all cores.

3.3 Batch-based Threading Model

With the ECS-based network modeling, we next explain how we execute the simulation correctly using multi-core parallelization. For parallel DES, the synchronization overhead always increases with the degree of parallelism, *i.e.*, how many LPs participate in the simulation. Existing synchronization algorithms incur high overhead [8, 10, 16].

Insights. For DONS, we observe two properties of network simulation that help to improve parallelization performance:

1. Use link delay as lookahead time. Lookahead time is a period of time in which the LP can safely execute the events without worrying about synchronization with other LPs [16]. In modern networks, link delay (*e.g.*, ~1 µs) is at least an order of magnitude larger than packet processing time (*e.g.*, ~100 ns)[4]. This indicates that DONS can safely execute a batch of packet-level events during the link delay without worrying about violating LCC.

However, merely using lookahead time in LPs is not enough, and existing parallel DES [39, 51] are already treating link delay as the lookahead. It seems imperative to implement synchronization algorithms in DONS, since some entities exhibit minuscule delays in their interaction, such as the Receiver and Sender within a host. Next, we show why this is unnecessary for DONS.

2. Break the coupling between network devices. In large-scale networks, there are a lot of devices, and in each device, we find the same set of network functions (or behavioral aspects). In our ECS-based modeling, we summarize four aspects and define them as different systems. As we prove at the end of this section, if these systems are executed in the following order, ("ACKSystem, SendSystem, ForwardSystem, TransmitSystem"), the correctness of the simulation can be guaranteed.

We also observe that, within an aspect, there is no dependency between devices. Take the ForwardSystem as an example. In a window of the lookahead time, this system moves all packets in all the IngressPorts to the corresponding EgressPorts, according to the FIB component. Thus, the ForwardSystem concerns only the buffer component and the FIB component associated with all the IngressPorts, and these two components are determined in the previous lookahead window by the SendSystem (for host-to-device links) and the TransmitSystem (for device-to-device links). There is no dependency between devices when the ForwardSystem is running, so it can be safely parallelized on multi-cores without violation of LCC. This independence means that we can safely execute the same behavioral aspect of all simulated devices in a data-parallel manner across multiple cores.

On a server, DONS has only one LP because DONS uses a single-process, multi-threaded run-time model. Unlike existing parallel DES on a multi-core server, DONS does not bind a core to a partition of the network. In DONS, all cores simulate the whole network together, just that only one aspect of the simulation is executed at

[4]We also expect this discrepancy to enlarge with the introduction of 200/400 GbE interfaces in modern networks because the link bandwidth grows exponentially while wire lengths stay relatively stable.

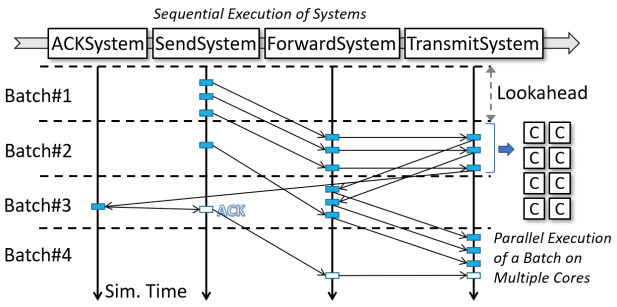


Figure 8: Batch-based threading model of DONS.

the same time. This frees DONS from synchronization overhead, which contributes greatly towards its performance and scalability on a single multi-core machine.

Threading model: With these two insights, we redesign the DES simulation engine. DONS uses a batch-based threading model, as shown in Figure 8, where vertical is the time order, and horizontal is the system execution order. The length of a batch is the lookahead time, which is based on the smallest link delay in the simulation scenario.

DONS sequentially executes the 4 systems within a batch. Note that a naïve execution order would be: "SendSystem, ForwardSystem, TransmitSystem, ACKSystem," which packs packet generation and retrieval at the beginning of the batch and packet processing at the end. With this order, however, when the current batch ends and the next batch begins, the simulation clock must increase, and the ACKSystem may trigger the SendSystem to execute events in the current batch, causing the SendSystem to violate LCC. Therefore, we instead use "ACKSystem, SendSystem, ForwardSystem, TransmitSystem" as the execution order[5].

Within a machine, DONS employs a thread-pool-based run-time environment. Each task in the thread pool is a parallel part of the execution of a system. For the simulation of modern-size networks, the tasks are usually large and computation-intensive. So we recommend configuring the size of the thread pool to be the number of cores of the CPU for full utilization. Using thread-pool, DONS achieve automatic multi-threaded parallelization and fine-grained load-balancing on multiple cores. Compared to process-based parallelization, multi-threaded parallelization has more efficient memory usage and low cross-thread communication overhead, because data can be shared easily between threads within a process.

Correctness of DONS. The following theorem is proved by prior work in distributed and parallel DES [16]:

Theorem 1. *If each LP adheres to the LCC, execution of the simulation program on a parallel computer will produce exactly the same results as execution on a sequential computer.*

We intend to prove the following:

Theorem 2. *DONS's execution model on a multi-core CPU produces exactly the same results as execution on a sequential computer.*

[5]SendSystem as the last system and ForwardSystem as the beginning system should also work, but is less intuitive.

Since the LP of DONS is the whole server, if we can prove that all systems process events strictly in chronological order, then DONS's execution is correct.

Proof. For the ACKSystem, as shown in Figure 8, when a batch starts, the order of events to be processed by ACKSystem is already determined by TransmitSystem. No new events will be inserted when executing this batch, so ACKSystem conforms to LCC. This is the same for the ForwardSystem and the SendSystem: When a batch starts, the order of events to be processed is already determined by the preceding system, so both systems conform to LCC. For the TransmitSystem, because multiple IngressPorts can forward packets to an EgressPort, and the ForwardSystem processes them in parallel, so when the batch starts, the packets in the buffer components of the EgressPorts are not in the correct order. Because TransmitSytem first sorts the packets in chronological order and then perform the egress scheduling, it also conforms to LCC. In conclusion, all systems in DONS process event at simulated time t without worrying about other events with $t_e<t$. By Theorem 1, DONS's execution model on a single machine produces exactly the same results as execution on a sequential computer. □

Thus, DONS achieves correctness while not requiring any synchronization algorithms within a single machine.

4 DONS DISTRIBUTED EXECUTION

The computing and memory capacity of a single machine limits the scale of simulation that DONS can run on it. For example, in our evaluations (§6), FatTree48 is the largest FatTree topology that we can run on a single machine, but recent data centers exceed this scale significantly. To simulate larger networks, we must enable distributed execution for DONS.

Design Overview. It is not feasible to apply the same batch-based threading model design for a cluster of servers, as the inter-server communication latency is much larger than that between cores, the bandwidth is smaller, and the clocks of the machines are not synchronized. To conduct parallel DES simulation among servers, similar to existing DES simulators, DONS partitions the simulation topology into sub-graphs, and each server executes one of them. Unfortunately, we find that the current partitioning methods are coarse-grained and result in unbalanced workload distribution.

To enable balanced distributed simulation, we make three design decisions. First, we design a precise time-cost model which improves the state-of-the-art. Second, we develop an automatic heuristic partitioning algorithm that utilizes the new time-cost model to find a partition that minimizes the overall simulation completion time. Third, we implement a simple conservative synchronization algorithm [16] to guarantee the correctness of distributed parallel DONS.

4.1 Automatic Partitioning

Problem formalization. Finding the best partition across multiple machines can be approached as an optimization problem, where the input is the setup of the simulation task and the computing and communication capabilities of the cluster; the output is the partition result; the objective is minimizing the completion time of

each machine, which can be estimated by a time-cost model. In the following, we will try to solve this optimization problem through a novel time-cost model and a heuristic algorithm.

Time-cost Model. OMNeT++ uses a Coupling factor [52] to roughly evaluate whether a simulation scenario can be accelerated using distributed execution. However, the only constraint considered by the Coupling factor is that the communication latency between servers should be less than the wall-clock time for the server to execute one lookahead. This condition does not directly relate to the completion time of the whole simulation. For example, if the load is extremely unbalanced among these servers, it may have a similar low coupling factor, but its completion time may be large.

In order to accurately predict the simulation completion time for a given network and partition, the cost model must take into account the following three factors:

- *The traffic pattern in the simulation network.* This represents the load of each node and link in the network (*i.e.*, the number of events to be simulated), which helps balance the computation load of each sub-graph and the communication load between the servers. For example, if we cut a link with high utilization, its two end nodes are simulated by different servers, then these servers need to communicate frequently.
- *The computation capacity of the physical servers.* This represents the computation efficiency of the servers, which may have heterogeneous computing power. Based on the computation capacity of a server and the load it is assigned, we are able to infer the time that the server spends in computation.
- *The communication capacity of the physical cluster.* This represents the communication efficiency between servers, and the efficiency and the total amount of communication traffic can calculate the network communication time required by the simulation.

We define a time-cost model capable of estimating the simulation completion time for a given network and partition as follows:

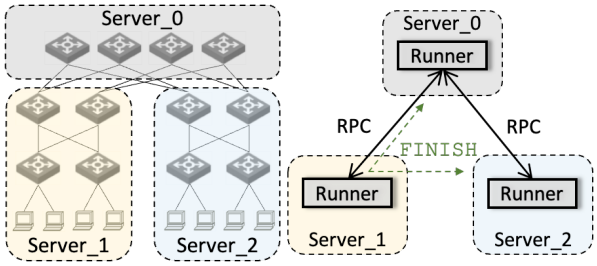
$$T_a = \frac{E_a}{P_a} + \frac{\tau_a}{B_a} \tag{1}$$

$$T = \max_{a \in Cluster} \{T_a\} \tag{2}$$

where E_a represents the computation load in server a, P_a represents the computation efficiency of server a, τ_a represents out-going traffic from server a, and B_a represents the bandwidth of server a. T_a is an estimate of the execution time of server a, and T is an estimate of the overall simulation completion time.

Load Estimator. Given the simulation setting, the Load Estimator obtains the parameters in Eq. (1) using a simple flow-level network model. The network is a topology of connected devices and links.

First, flows are added into the network in the same order and time as in the simulation. They are also routed using the same approach as in the simulation. When a new flow is added to the model, we add the bandwidth of that flow to the load value of the devices and links along its path. The load of each device and link in the network is represented by the total bandwidth passing through it. In the end, we can obtain quick estimates of loads on all devices and links in the network from this model. We ignore fairness or interaction between flows in this model, and the bandwidth on a link can exceed the link capacity. We do this to control the time complexity ($O(n)$) on large networks.



(a) An example of partition **(b)** Synchronization

Figure 9: Implementation details of distributed execution.

The actual load of both nodes and links is a function over time. Note that DONS supports dynamic partitioning to better balance the load among servers, and we describe this feature in Appendix A. In cases where the flow demand is unknown in advance, we adopt an alternative partitioning scheme that directly cuts the static topology.

Partitioner. The objective is to minimize T. The simulated network, as profiled by the Load Estimator, can be viewed as a graph with node weights and link weights, which represent the device load and link load, respectively. For each server a, T_a consists of two parts: E_a/P_a and τ_a/B_a. We observe that: (1) minimizing E_a/P_a is to solve the balanced cut problem in the graph, *i.e.*, minimizing the maximum sum of node weights among all sub-graphs; (2) minimizing τ_a/B_a is to solve the minimum cut problem in the graph, *i.e.*, minimizing the sum of weights of the cut links. Thus, minimizing T is an instance of the well-known Minimum Balanced Cut (MBC) problem [11], which is NP-hard [7, 11].

Thus, we propose a heuristic algorithm. Given that there are many approximation algorithms that can solve the MBC (k=2) problem in near-linear time, we propose an iterative algorithm, its pseudo-code is described in Appendix B. It solves an MBC (k=2) problem at each iteration and recursively partitions the entire network. The iteration terminates when either one of the following two conditions is met: (1) all machines in the cluster are fully utilized, or (2) the current partitioning does not bring performance benefits as estimated by the time-cost model. Figure 9a shows an example of a partitioning result. Because we only cut the links, communication between machines is restricted to the TransmitSystem. We explain the execution of distributed DONS below.

4.2 Distributed Execution

In order to guarantee correctness, DONS requires that each Runner executes the same batch at the same time. In other words, the entire cluster should work on the same lookahead window. If one TransmitSystem's next hop is located on a remote server, it sends one RPC to carry the information of a batch of packets in order to overlap communication and computation. To determine the completion of the TransmitSystem and the communication in all Runner, we design a simple conservative synchronization mechanism, as shown in Figure 9b. When a machine finishes the TransmitSystem and remote communications, it sends a `FINISH` signal to the

other machines in the cluster. Thus, when a machine receives $N-1$ `FINISH` signals, it indicates that it will not receive any further RPC requests and can execute the subsequent system. In the evaluation in §6, we find that thanks to the improved load-balancing achieved by our partitioning algorithm, this synchronization mechanism is sufficient for DONS to obtain up to 65× speedup against OMNeT++, which uses the more sophisticated Null-Message algorithm [16].

5 IMPLEMENTATION

Inspired by Mark Handley, who used Unity at SIGCOMM '17 [21] and SIGCOMM '19 [20] to show how NDP [21] and the Internet work, we saw the potential to develop a DES engine based on Unity. We mainly use Unity's Data-Oriented Technology Stack (DOTS) [48], which provides an implementation of ECS framework. DONS's current code base has ~3000 lines of C# code, which we release for the networking community[6]. We follow the design in §3, and implement functions in their respective systems. Currently, DONS supports the following features: UDP [36], congestion control (DCTCP [4]), Random Early Detection [15] (packet tagging), ECMP, and four popular packet schedulers (First-In-First-Out, Round Robin, Deficit Round Robin, and Strict Priority). We devise many optimization techniques to solve the problems encountered in the implementation of DONS, such as write-conflicts, sorting packets, and active queue management. Due to space limits, we describe them in Appendix C. With the prototype, we proceed to evaluate DONS against existing simulators.

6 EVALUATION

We evaluate DONS's fidelity, simulation speed, and scalability. We summarize our results as follows:

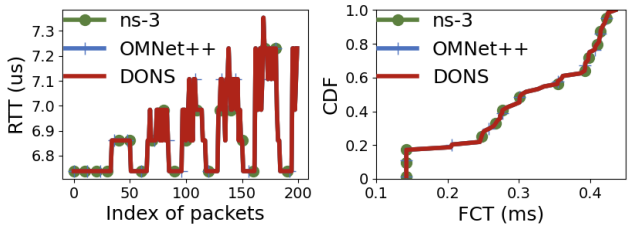
Key Results.

- **Fidelity:** The simulation accuracy of DONS is equivalent to that of existing DESs (*e.g.*, ns-3 and OMNeT++), even down to the timestamp of all events.
- **Simulation speed:** DONS is capable of simulating large-scale networks (FatTree32) on a commodity 32-core server with only CPUs, and the simulation time is 22× shorter than ns-3.
- **Scalability:** With the help of automatic distributed parallelization, DONS can simulate larger networks on a cluster easily. Using 8 commodity servers, DONS can finish a simulation of FatTree64 (65,536 servers) in under 3 hours (a 65× speedup compared to OMNeT++).

Baselines. We select ns-3 [39], OMNeT++ [51], and DeepQueueNet [55] (DQN) as the comparison alternatives, where ns-3 and OMNeT++ are widely used DES simulators, and DQN is the state-of-the-art APA scheme recently proposed. We also release the source code for the experiments[7].

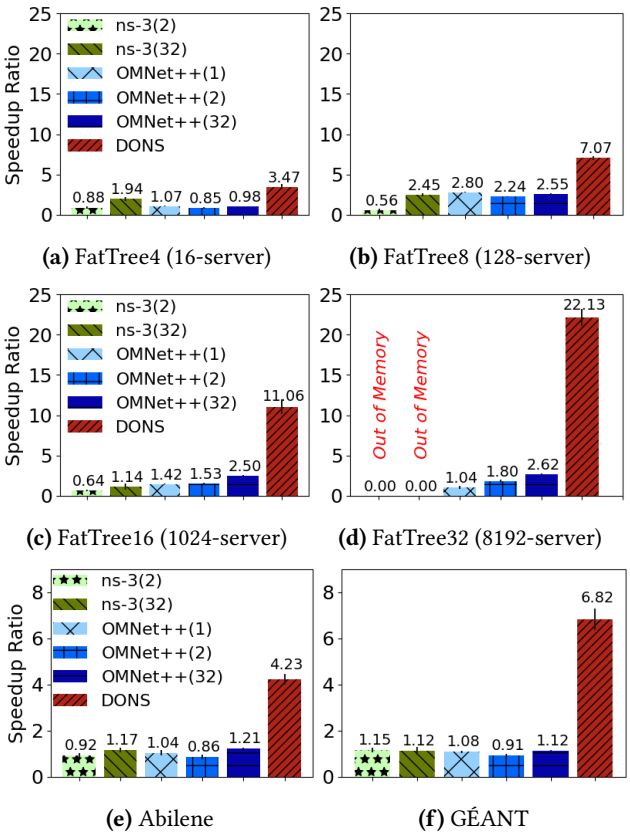
Setup. Our cluster used for simulation consists of 8 Linux servers, each including one Nvidia Tesla V100-16GB GPU, two Intel Xeon CPUs (totaling 32 cores), and 128 GB memory. All servers are connected to a switch through a 40 Gbps link. To showcase versatility, we use a MacBook Air (M1, 2020) with 8 cores and 8 GB memory to test the performance of DONS on a consumer laptop, which is an affordable and available computing hardware for networking practitioners. The simulation topologies encompass data center networks (DCN) and wide area networks (WAN) of various scales. Flow sizes and intervals are obtained from real-world traces [4, 42]. The source and destination of a flow are selected uniformly at random from the servers in the topology. In all simulations, we use DCTCP [4] as the congestion control algorithm.

[6] https://github.com/dons2023/Data-Oriented-Network-Simulator
[7] https://github.com/dons2023/



(a) RTT evolution **(b)** FCT distribution

Figure 10: DONS has the same fidelity with existing DES.



(a) FatTree4 (16-server) **(b)** FatTree8 (128-server)

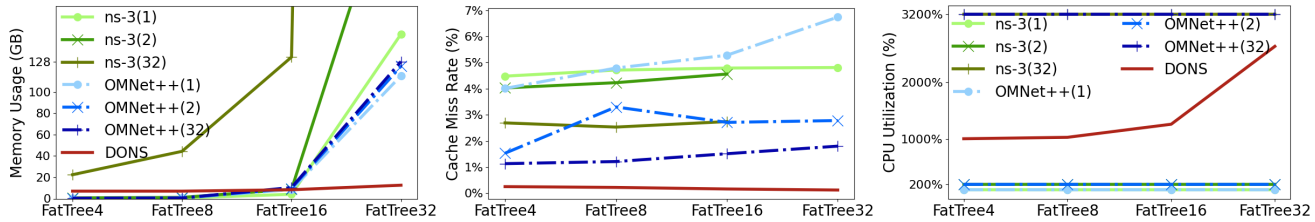
(c) FatTree16 (1024-server) **(d)** FatTree32 (8192-server)

(e) Abilene **(f)** GÉANT

Figure 11: Simulation running time speedup brought by different simulators under FatTree and WAN topology.

6.1 Simulation on a Single Machine

All the following experiments were run on a single machine. ns-3 and OMNeT++ employ one process or multiple processes, and



(a) Memory usage **(b)** Cache miss rate **(c)** CPU utilization

Figure 12: The performance of different simulators.

DONS's multi-threaded parallelization automatically utilizes all CPU cores.

Fidelity. We use these simulators to simulate a FatTree [3] topology with k=8 (FatTree8) and run 64 flows, each transmitting 1.50 MB of data. We collect RTT and FCT to compare the fidelity. Figure 10a depicts the RTT experienced by the first 200 packets, and Figure 10b shows the distribution of the FCTs. DONS has the same RTT evolution and FCT distribution as ns-3 and OMNeT++, and according to our measurement, even the timestamps of all events in these DES-based simulators are identical.

Simulation speed. We use different scales of DCN and WAN to compare the speed of these DES simulators. The metric is the speedup ratio ($t_{ns-3(1)}/t_x$), where t_x is the simulator's simulation completion time, and $t_{ns-3(1)}$ is taht of the single-process ns-3. Figure 11 shows the speedup ratios of these simulators under FatTree topology. Using 2 processes actually reduces the performance of ns-3 due to high synchronization overhead, and using 32 processes barely speeds up performance. Furthermore, ns-3 cannot use multi-process to run FatTree32 due to an Out-of-Memory (OOM) error, the reason will be introduced in the following. The simulation speed of OMNeT++ is comparable to ns-3, with a speedup ratio less than 3×, but it does not encounter OOM errors. In these experiments, the speedup ratio of DONS increases from 3× to 22×, which indicates that even large-scale networks can be simulated quickly using only a multi-core CPU on a single machine.

In addition, we compare the simulation speed of different simulators for WANs (*e.g.*, Abilene [1] and GÉANT [46]). Abilene has 12 routers and 15 links, and GÉANT has 23 routers and 36 links. A server was connected to each router to send and receive traffic. In this simulation task, we construct full-mesh dynamic flows among all servers. As shown in Figure 11e and Figure 11f, DONS achieved speedups of at most ~4× and ~7×, respectively.

Memory efficiency. During the above experiments, we recorded various performance metrics of the simulators. Figure 12a shows the memory usage of each simulator, which represents the maximum value during execution. As ns-3 stores a complete simulation setting in each LP (process), its memory usage increases with the number of LPs. Calculations show that executing FatTree32 with 32 processes using ns-3 requires ~5,000 GB of memory. In contrast, the memory usage of OMNeT++ remains roughly the same regardless of the number of LPs. The consumption of both ns-3 and OMNeT++ for FatTree32 is close to the memory capacity (128 GB) of a single machine. Since DONS adopts DOD and focuses on data layout, it is memory-efficient. For FatTree32, DONS only uses 12.6 GB of memory. At maximum, DONS can run FatTree48 (27648 servers) on a single server.

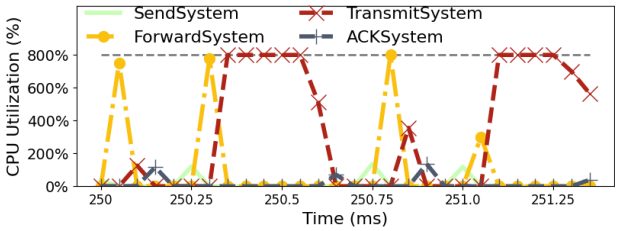


Figure 13: The breakdown of CPU utilization of DONS.

Cache efficiency. Figure 12b depicts each simulator's L3 cache miss rate (CMR) during the execution period. CMR refers to the proportion of the number of cache misses on all cache query times, collected by `perf`. Both ns-3 and OMNeT++ have a CMR greater than 1% in all scenarios and increase with the topology scale. In contrast, DONS has the lowest CMR of 0.12% for FatTree32. Compared to both ns-3 and OMNeT++, the CMR of DONS has been reduced by 56× at the highest and 4.5× at the lowest. Such low cache miss accelerated the simulation completion time of DONS.

Multi-core utilization. Figure 12c shows the CPU utilization of each simulator. The number of CPUs used by ns-3 and OMNeT++ is equal to the number of processes enabled. DONS adaptively uses CPU cores using a multi-threaded approach. For different topologies, DONS's CPU utilization rises from 1,003% to 2,634%, consistently using all cores. Furthermore, the utilization rate remains stable throughout the execution period. DONS's CPU utilization is 3,200% under FatTree48, which indicates that, when running DONS on the Linux server, the maximum topology size that will still experience parallelism speedups is FatTree48, and for larger topologies (such as FatTree64), DONS requires distributed parallelization to improve efficiency.

Then, we delve into analyzing the CPU overhead of each system in DONS. To this end, we simulate FatTree16 on a MacBook Air (8 cores) using DONS and collect fine-grained CPU information using Unity Profiler [50]. Figure 13 displays the CPU utilization of each system within 1 ms. Most of the time, all 8 cores are fully utilized, with TransmitSystem taking the lion's share. It also shows that each system is executed in the order that guarantees correctness.

Scale of simulation. We analyze the maximum topology that a single machine can simulate using different simulators. We set the link capacity to 100 Gbps in all experiments. On our Linux server,

# Machines	Simulator	# GPUs	Time	Speedup	w_1
4	OMNeT++	0	9d 14h 24m	baseline	-
	DQN	4	2h 56m	78.5×	0.43
	DONS	0	5h 27m	42.2×	0
8	OMNeT++	0	7d 19h 8m	baseline	-
	DQN	8	1h 48m	104.1×	0.46
	DONS	0	2h 53m	65.0×	0

Table 1: Simulation time under FatTree64 (65,536 servers).

both ns-3 and OMNeT++ are limited at FatTree32. Simulations with larger FatTree topology are terminated due to an Out-of-Memory error. In comparison, the maximum size of DONS can simulate is FatTree48 with 27,648 servers.

DONS can also support large-scale simulation on affordable consumer laptops. The maximum size of DONS can simulate on a MacBook Air with an M1 chip is FatTree16 with 1,024 servers, and 1,000 ms of simulated time takes 22 minutes, while OMNeT++ takes ~7.8 hours.

6.2 Distributed Simulation on a Cluster

We use multiple machines for distributed simulation of larger DCN and WAN topologies. Ns-3 is not compared as it cannot run such large topologies, instead, we use OMNeT++ as the baseline, which is set to utilize all cores in all machines. It should be noted that to enable distributed parallelism in OMNeT++, users are required to add a substantial amount of configuration code, with the number of lines equivalent to the count of network nodes. Conversely, DONS only requires users to program the simulation as if it is on a single machine. DONS uses Load Estimator, Time-cost Model, and Partitioner to divide the simulation network and generate a distributed execution plan, then leverage the resources of all machines for simulation. DQN makes use of all GPUs in these machines. We compare the speed and scalability of different simulators and evaluate the overhead of DONS's Partitioner.

Simulation speed. First, Table 1 shows the simulation time of different simulators under FatTree64. DONS achieved a maximum acceleration of 65×, only 1.6× slower than DQN, but possesses full fidelity. The speedup of DONS using 8 machines is ~2× that of using 4 machines, which indicates that DONS has near-linear scalability. DQN achieves higher acceleration using 4 and 8 GPUs, but it sacrifices accuracy. We compute the normalized Wasserstein distance (w_1) of the RTT distribution between the simulators and OMNeT++, and find that DQN cannot produce accurate results. For all settings, its w_1s are all greater than 0.4.

To compare the effects of different topology partitioning methods, we simulate a large-scale irregular WAN in this experiment. We obtain a large topology from a major ISP, which includes a backbone network, provincial network, and metropolitan area network, the WAN provides both home broadband and Internet private line services. The WAN has 13k core routers and 32k links, and its connections are very irregular. Figure 14 shows a part of the WAN topology. We compare the impact of different topology partitioning methods on simulation speed. The first method is the static balanced cut algorithm, which aims to distribute the number of nodes across multiple machines evenly. The second is the Coupling-factor-based

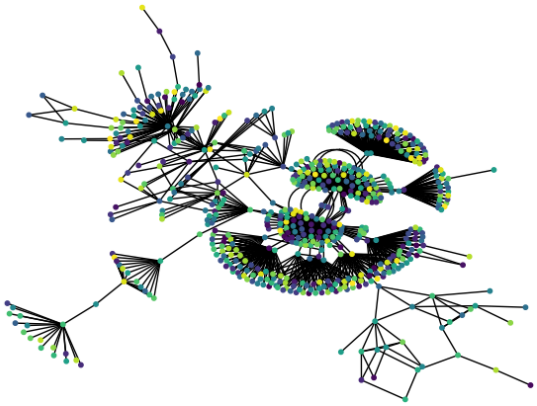


Figure 14: A part of the large-scale WAN from a major ISP.

Method	Simulator	# GPUs	Time	Speedup	w_1
Balanced cut	OMNeT++	0	6d 7h 6m	baseline	-
	DQN	8	4h 46m	31.7×	0.53
	DONS	0	11h 59m	12.6×	0
CFP	OMNeT++	0	4d 1h 13m	1.6×	0
	DQN	8	4h 10m	36.2×	0.52
	DONS	0	9h 6m	16.6×	0
DONS Partitioner	OMNeT++	0	1d 15h 58m	3.8×	0
	DQN	8	2h 26m	61.8×	0.57
	DONS	0	4h 17m	35.2×	0

Table 2: Simulation time using 8 servers for a large-scale WAN under different partitioning methods.

Method	Planning time	Execution time
Balanced cut	15s	11h 59m 17s
CFP	42s	9h 6m 03s
DONS Partitioner	1m 46s	4h 17m 24s

Table 3: Planning and execution time of DONS.

partitioning (CFP) method recommended by OMNeT++ [52], which only considers the relationship between communication delay and the lookahead time. The third is DONS's Partitioner, which uses the time-cost model for automatic partitioning.

We integrate these three methods into distributed OMNeT++, DQN, and DONS, and the simulation speed is shown in Table 2. Compared to the baseline (OMNeT++ with balanced cut), CFP has a slight performance improvement. OMNeT++, using DONS's Partitioner, precisely plans the distributed simulation, resulting in a 3.8× improvement in simulation speed. Partitioner also benefits DQN. In summary, the static CFP and static balanced cut have similar effects, as they do not consider dynamic traffic patterns. Partitioner can improve the simulation speed by ~2× compared to CFP.

Partitioner overhead. We examine the completion time of different partitioning methods in the above experiment. As shown in Table 3, the entire simulation time is divided into two parts: 1) Planning time, which represents the time spent running partitioning algorithm to generate the distributed execution plan; 2) Execution

time, which represents the actual time spent running the simulation task. Using *Load Estimator* and *Partitioner* with the time-cost model for planning takes ~2 minutes, which is 2.5× that of CFP, but reduces the execution time from 9 hours to 4 hours. Hence, the time spent on planning in DONS is negligible but brings significant performance improvement.

7 RELATED WORK

Discrete-event simulators (DES). As critical tools for network practitioners, DES simulators have existed for decades [25]. Notable examples include ns-2/3 [22, 39], OMNeT++ [51], and OPNET [29]. They all face the scalability issue for simulating modern networks. Parallel and distributed DES [16, 23] has also been extensively studied, but parallelization of simulation often leads to performance degradation due to inefficient data layout and high synchronization overhead [55, 57].

Continuous-time simulators (CTS). Control-theoretic simulators [5, 30, 31, 35], network calculus [12, 26], and queueing theoretic simulators [40] have several limitations: (1) they require users to have extensive knowledge and expertise in defining state evolution equations; (2) they only provide steady state results; and (3) accurately estimating performance using CTS simulators can be computationally demanding. DONS, on the other hand, provides a simpler and more robust packet-level simulation abstraction.

AI-powered performance approximators (APA). APAs all have an inherent loss of accuracy, a time-consuming training process, and often come with limitations on their flexibility and generality, stemming from assumptions about what can be approximated and what cannot. DeepQueueNet [55] assumes static sender demands, MimicNet [24, 57] is unable to accommodate topologies other than FatTrees [3], and RouteNet [43] cannot generalize to different traffic generation models. While approximation may eventually be necessary, DONS proves that there is still significant headroom for full-fidelity packet-level simulation. It can also provide complementary benefits, *e.g.*, to training data generation.

Data-oriented design (DOD). DOD [14, 54] is a software design philosophy that emphasizes the importance of data organization and manipulation in the development of software systems. DOD can be applied to various areas of computer science, including game development [6, 13, 48, 49], programming languages [2], and transaction processing [34]. To the best of our knowledge DONS is the first to apply it to network simulation.

8 DISCUSSION

Extending DONS: Once the users are familiar with the DOD style of programming, implementing a new functionality in DONS can be as straightforward as in ns-3 and OMNeT++. We also provide basic templates and building blocks for users. Taking the congestion control algorithm (CCA) as an example, DONS offers a foundational TCP-based state machine for transport layer protocols, making the integration of a novel CCA a relatively simple task. By implementing the necessary logic and components, a new CCA can be seamlessly added without the need for any additional systems. The coding process closely resembles the development of a new protocol in ns-3. In the future, we plan to improve DONS to support more network protocols, such as IPv6, routing, wireless, etc.

Fault tolerance: Since network simulations typically run for extended periods, DONS may encounter failures. In single-machine multi-threading scenarios, DONS employs automatic recovery techniques across multiple threads to ensure robustness and fault tolerance. In multi-machine environments, DONS utilizes checkpointing to periodically preserve the run-time state of the simulation. This approach facilitates the retention of progress and enables the simulator to resume from a previous checkpoint in the event of failures or interruptions. To achieve this, DONS periodically saves the current state of the simulation to persistent storage. These checkpoints encompass the internal state of the simulator, including the current simulation time, object positions and attributes, and other necessary variables and data structures. By creating checkpoints at regular intervals, DONS ensures the ability to quickly recover to the most recent checkpoint in the event of system failures or interruptions, allowing the simulation to proceed from there. Furthermore, to enhance fault tolerance, DONS can replicate checkpoints across multiple locations to mitigate the risks of single-point failures. This ensures that even if one storage location experiences a failure, the simulation state can still be recovered from other locations.

Visualization: Visualization of network simulations is a crucial tool for both network research and education. However, the front-end of current network simulators [32, 41, 51] often fails to provide satisfactory performance when dealing with contemporary network scales and interface speeds. We chose the Unity framework because of its ability to support DOD and network simulation visualization. Based on Unity, we have developed an efficient visualization front-end [18] for DONS. The front-end offers a flow-level visualization of network behavior and key performance metrics. In addition, through parallel optimization, it supports real-time visualization for large-scale, high-speed networks.

9 CONCLUSION

This paper demonstrates the ability to simulate large-scale networks with full fidelity using only affordable hardware, such as CPUs. We first identify that the fundamental limitation of existing DESs is their Object-Oriented-Design-based software architecture. Then we utilize DOD principles (Data-Oriented Design) to design a novel DES system, DONS, which can be automatically parallelized for intra-server and inter-server settings. We prove the correctness of DONS. Extensive experiments show that the simulation results of DONS are exactly the same as ns-3/OMNeT++, and the simulation speed is up to 65× faster. The scalability of DONS is also demonstrated in distributed simulation experiments on a cluster.

ACKNOWLEDGMENT

We thank our shepherd Dr. Ying Zhang and the anonymous SIGCOMM reviewers for their constructive comments. Prof. Dan Li is the corresponding author. This work was supported by the National Key R&D Program of China under Grant 2019YFB1802600, the National Natural Science Foundation of China under Grant U21B2022, NSF grant CNS-1845749, and Tsinghua University-China Mobile Communications Group Co.,Ltd. Joint Institute.

REFERENCES

[1] 2004. Abilene. http://www.cs.utexas.edu/~yzhang/research/AbileneTM/. (2004).
[2] Mike Acton and Insomniac Games. 2014. Data-oriented design and c++. *Luento. CppCon* (2014).
[3] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM CCR* (2008).
[4] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center tcp (dctcp). In *ACM SIGCOMM*.
[5] Mohammad Alizadeh, Adel Javanmard, and Balaji Prabhakar. 2011. Analysis of DCTCP: stability, convergence, and fairness. *ACM SIGMETRICS Performance Evaluation Review* 39, 1 (2011), 73–84.
[6] Blizzard. 2023. Overwatch2. https://overwatch.blizzard.com/. (2023).
[7] Paul Bonsma. 2010. Most balanced minimum cuts. *Discrete Applied Mathematics* 158, 4 (2010), 261–276.
[8] Randal Everitt Bryant. 1977. *Simulation of packet communication architecture computer systems.* Massachusetts Institute of Technology.
[9] Qizhe Cai, Mina Tahmasbi Arashloo, and Rachit Agarwal. 2022. DcPIM: Near-Optimal Proactive Datacenter Transport *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 53–65. https://doi.org/10.1145/3544216.3544235
[10] K. Mani Chandy and Jayadev Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on software engineering* 5 (1979), 440–452.
[11] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. 2020. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1158–1167.
[12] Florin Ciucu and Jens Schmitt. 2012. Perspectives on network calculus: no free lunch, but still good value. In *ACM SIGCOMM*. 311–322.
[13] Unreal Engine. 2023. Mass Entity. https://docs.unrealengine.com/5.0/en-US/mass-entity-in-unreal-engine/. (2023).
[14] Richard Fabian. 2018. Data-oriented design. *framework* 21 (2018), 1–7.
[15] Sally Floyd and Van Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking* 1, 4 (1993), 397–413.
[16] Richard M Fujimoto. 2000. *Parallel and distributed simulation systems*. Vol. 300. Citeseer.
[17] Erich Gamma, Ralph Johnson, Richard Helm, Ralph E Johnson, and John Vlissides. 1995. *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
[18] Kaihui Gao, Li Chen, Dan Li, Vincent Liu, Xizheng Wang, Ran Zhang, and Lu Lu. 2023. NetVision: Efficient Visualization Front-End for Packet-level Discrete-Event Network Simulation. In *Proceedings of the SIGCOMM'23 Poster and Demo Sessions*.
[19] Prateesh Goyal, Akshay Narayan, Frank Cangialosi, Srinivas Narayana, Mohammad Alizadeh, and Hari Balakrishnan. 2022. Elasticity Detection: A Building Block for Internet Congestion Control. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 158–176. https://doi.org/10.1145/3544216.3544221
[20] Mark Handley. 2019. Keynote. In *ACM SIGCOMM '19*. https://doi.org/10.1145/3341302.3359768
[21] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-Architecting Datacenter Networks and Stacks for Low Latency and High Performance. In *ACM SIGCOMM*.
[22] Teerawat Issariyakul, Ekram Hossain, Teerawat Issariyakul, and Ekram Hossain. 2009. *Introduction to network simulator 2 (NS2)*. Springer.
[23] Shafagh Jafer, Qi Liu, and Gabriel Wainer. 2013. Synchronization methods in parallel and distributed discrete-event simulation. *Simulation Modelling Practice and Theory* 30 (2013), 54–73.
[24] Charles W. Kazer, Jo ao Sedoc, Kelvin K.W. Ng, Vincent Liu, and Lyle H. Ungar. 2018. Fast Network Simulation Through Approximation or: How Blind Men Can Describe Elephants. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks (HotNets '18)*. Association for Computing Machinery, New York, NY, USA, 141–147.
[25] LBNL. 2023. network simulator man page. https://ee.lbl.gov/ns/man.html. (2023).
[26] Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer.
[27] Hejing Li, Jialin Li, and Antoine Kaufmann. 2022. SimBricks: End-to-End Network System Evaluation with Modular Simulation. In *ACM SIGCOMM*. 380–396.
[28] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *ACM SIGCOMM*.
[29] Zheng Lu and Hongji Yang. 2012. *Unlocking the power of OPNET modeler*. Cambridge University Press.
[30] Marco Ajmone Marsan, Michele Garetto, Paolo Giaccone, Emilio Leonardi, Enrico Schiattarella, and Alessandro Tarello. 2005. Using partial differential equations to model TCP mice and elephants in large IP networks. *IEEE/ACM Transactions on Networking* 13, 6 (2005), 1289–1301.
[31] Vishal Misra, Wei-Bo Gong, and Don Towsley. 2000. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 151–160.
[32] NetAnim. 2023. https://www.nsnam.org/wiki/NetAnim. (2023).
[33] Sergei Nikolaev, Peter D Barnes, James M Brase, Thomas W Canales, David R Jefferson, Steve Smith, Ron A Soltz, and Peter J Scheibel. 2012. *Performance of distributed ns-3 network simulator*. Technical Report. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).
[34] Ippokratis Pandis, Ryan Johnson, Nikos Hardavellas, and Anastasia Ailamaki. 2010. Data-oriented transaction execution. *Proceedings of the VLDB Endowment* 3, ARTICLE (2010), 928–939.
[35] Qiuyu Peng, Anwar Walid, Jaehyun Hwang, and Steven H Low. 2014. Multipath TCP: Analysis, design, and implementation. *IEEE/ACM Transactions on networking* 24, 1 (2014), 596–609.
[36] Jon Postel. 1980. *User datagram protocol*. Technical Report.
[37] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, et al. 2022. Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking. In *ACM SIGCOMM*. 66–85.
[38] Shixiong Qi, Leslie Monis, Ziteng Zeng, Ian-chin Wang, and K. K. Ramakrishnan. 2022. SPRIGHT: Extracting the Server from Serverless Computing! High-Performance EBPF-Based Event-Driven, Shared-Memory Processing. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 780–794. https://doi.org/10.1145/3544216.3544259
[39] George F Riley and Thomas R Henderson. 2010. The ns-3 network simulator. *Modeling and tools for network simulation* (2010), 15–34.
[40] Thomas G Robertazzi. 2000. *Computer networks and systems: queueing theory and performance evaluation*. Springer Science & Business Media.
[41] Caitlin J Ross, Noah Wolfe, Mark Plagge, Christopher D Carothers, Misbah Mubarak, and Robert B Ross. 2019. Using scientific visualization techniques to visualize parallel network simulations. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 197–200.
[42] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. [n. d.]. Inside the Social Network's (Datacenter) Network. In *ACM SIGCOMM'15*.
[43] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2020. RouteNet: Leveraging graph neural networks for network modeling and optimization in SDN. *IEEE Journal on Selected Areas in Communications* 38, 10 (2020), 2260–2270.
[44] Network Simulator. 2023. ns-3.38. https://www.nsnam.org/releases/ns-3-38/. (2023).
[45] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. 2015. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM CCR* 45, 4 (2015), 183–197.
[46] Steve Uhlig, Bruno Quoitin, Jean Lepropre, and Simon Balon. 2006. Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM Computer Communication Review* 36, 1 (2006), 83–86.
[47] Unity. 2023. https://unity.com. (2023).
[48] Unity. 2023. Data-Oriented Technology Stack (DOTS). https://unity.com/dots. (2023).
[49] Unity. 2023. Entities package. https://docs.unity3d.com/Packages/com.unity.entities@1.0/manual/index.html. (2023).
[50] Unity. 2023. Profiler package. https://docs.unity3d.com/2023.1/Documentation/Manual/Profiler.html. (2023).
[51] András Varga. 2019. A practical introduction to the OMNeT++ simulation framework. *Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem* (2019), 3–51.
[52] András Varga, Yasar Ahmet Sekercioglu, and Gregory K Egan. 2003. A practical efficiency criterion for the null message algorithm. In *European Simulation Symposium 2003*. SCS Europe Publishing House, 81–92.
[53] Shuai Wang, Kaihui Gao, Kun Qian, Dan Li, Rui Miao, Bo Li, Yu Zhou, Ennan Zhai, Chen Sun, Jiaqi Gao, et al. 2022. Predictable vFabric on informative data plane. In *ACM SIGCOMM*. 615–632.
[54] wikipedia. 2023. Data-oriented design. https://en.wikipedia.org/wiki/Data-oriented_design. (2023).
[55] Qingqing Yang, Xi Peng, Li Chen, Libin Liu, Jingze Zhang, Hong Xu, Baochun Li, and Gong Zhang. 2022. DeepQueueNet: towards scalable and generalized network performance estimation with packet-level visibility. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 441–457.
[56] Hyunho Yeo, Hwijoon Lim, Jaehong Kim, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2022. NeuroScaler: Neural Video Enhancement at Scale. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 795–811. https://doi.org/10.1145/3544216.3544218

[57] Qizhen Zhang, Kelvin KW Ng, Charles Kazer, Shen Yan, João Sedoc, and Vincent Liu. 2021. Mimicnet: fast performance estimates for data center networks with machine learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 287–304.

[58] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. *ACM SIGCOMM CCR* (2015).

Algorithm 1: Heuristic partitioning algorithm

```
1  Function partitioner(network):
2      subnet_1, subnet_2 = MBC(network, k=2);
3      if num_subnet+1 >num_machines then
4          return;
5      if max(time_cost(subnet_1), time_cost(subnet_2))
         <time_cost(network) then
6          num_subnet+=1;
7          partitioner(subnet_1);
8          partitioner(subnet_2);
9      else
10         return;
```

APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A DYNAMIC PARTITIONING IN DISTRIBUTED EXECUTION

If the partition is set in stone and does not change during the distributed execution, then it will be very inefficient at some time because the traffic pattern may vary greatly. A good partitioning scheme should be able to change as the traffic pattern changes dramatically but should also avoid frequent changes.

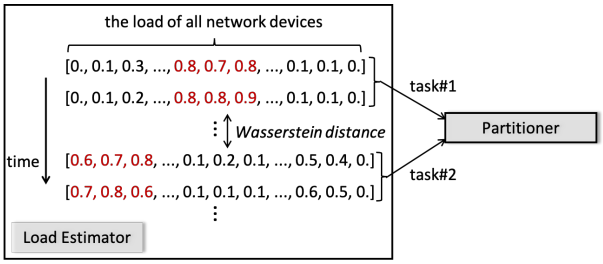


Figure 15: Load Estimator generates multiple simulation tasks when the traffic pattern changes drastically.

To address this challenge, we first use a vector to record the average load (normalized) of all network devices over a certain period of time, as shown in Figure 15. Then, multiple consecutive vectors record the changes in the network load over time. Next, we calculate the Wasserstein distance between two adjacent vectors, if it exceeds a certain threshold, it indicates that the traffic pattern of the network has changed dramatically, and DONS get a new simulation scenario. Finally, DONS will get several simulation scenarios (maybe one if the traffic pattern does not change much). DONS should treat them as separate simulation tasks, that is, perform the heuristic partitioning algorithm on each of them. All results form an overall simulation configuration, based on it, DONS Manager orchestrates the clusters.

B ITERATIVE PARTITIONING ALGORITHM

Algorithm 1 describes the heuristic partitioning algorithm used in DONS's Partitioner.

C OPTIMIZING DONS

Multi-threaded simulation builder. As shown in Figure 4, Simulation Builder constructs the simulation *world* based on the setting information. The Builder performs the following tasks: 1. creation and initialization of all entities, with a time complexity of O(#*node*); 2. calculation of routes for each destination through BFS, with a time complexity of O(#*host*×(#*node*+#*link*)); 3. conversion of routing tables to forwarding tables for all IngressPorts, with a time complexity of O(#*link*×#*host*). The latter two steps have a time complexity of O(N^2), and for large networks, executing them on one thread can be quite time-consuming. To enhance the efficiency of the build phase, the Builder employs multithreading, enabling threads to compute the BFS for different hosts and install the forwarding tables for different IngressPorts.

Avoiding write conflicts. The ForwardSystem has a many-to-one write-conflict problem because multiple IngressPorts can forward packets to a single EgressPort. To solve this problem, we adopt the *command pattern* [17]. It is a behavioral design pattern in which an object is used to encapsulate all information needed to perform an action or trigger an event at a later time. For each IngressPort, ForwardSystem records its write requests in its own *command buffer*, and in the end, the main thread consolidates the write operations in all command buffers.

Sorting and scheduling packets. The TransmitSystem must arrange all packets in chronological order prior to writing them to the next IngressPort or Receiver. Suppose there are m IngressPorts forwarding packets to a single EgressPort, as previously mentioned, the main thread executes these writes in a serial manner. Since the packets generated by an IngressPort are ordered, meaning that the EgressPort buffer will contain multiple sorted arrays of packets. Consequently, the TransmitSystem can use Merge Sort to combine these sorted arrays before scheduling, with a time complexity of O($Nlog_2m$), where m represents the number of arrays. After sorting, the TransmitSystem checks the buffer size, and drop packets when necessary. We currently use tail-drop in our prototype, and more sophisticated dropping strategies can be extended in the future.

Then the TransmitSystem uses a scheduling algorithm to determine the order of transmission. We implement four popularly used scheduling disciplines in the DONS prototype: First-In-First-Out (FIFO), Round Robin, Deficit Round Robin, and Strict Priority. To enable FIFO on an EgressPort, only one buffer component need to be attached to it. For all other three disciplines, multiple buffer components must be attached, representing multiple queues. For each batch, the TransmitSystem looks at all the packets in the buffer component(s) and determines the exit order based on the strategy of each EgressPort.

Queue length. To determine tail-drop or Random Early Drops, DONS must provide accurate queue lengths for each packet entering a port. However, in DONS's batch-based execution, obtaining the queueing length is not direct. This is because, in the lookahead window of a batch, for each EgressPort, the packets in its buffer components is processed by the TransmitSystem at the same time. After the batch-processing, some packets are moved to the linked Receiver or IngressPort, some remains in the buffer component(s), and the others dropped. Before the TransmitSystem finished processing, the exact order of the events (exit event, no-op event, drop event) cannot be determined. When a packet arrives at the EgressPort, its accurate queue length can only be calculated at the end of the processing of the TransmitSystem. To address this issue, the TransmitSystem maintains a TXhistory component for each EgressPort, which is a snapshot of the history of events on this port. When a packet is dequeued, the accurate queueing length is determined based on its enqueue and dequeue timestamps recorded in the TXhistory.