MimicNet: Fast Performance Estimates for Data Center Networks with Machine Learning

Qizhen Zhang, Kelvin K.W. Ng, Charles W. Kazer[§], Shen Yan[†], João Sedoc[◇], and Vincent Liu
University of Pennsylvania, [§]Swarthmore College, [†]Peking University, [◇]New York University
{qizhen,kelvinng,liuv}@seas.upenn.edu, [§]ckazer1@swarthmore.edu, [†]yanshen@pku.edu.cn, [◇]js11531@stern.nyu.edu

ABSTRACT

At-scale evaluation of new data center network innovations is becoming increasingly intractable. This is true for testbeds, where few, if any, can afford a dedicated, full-scale replica of a data center. It is also true for simulations, which while originally designed for precisely this purpose, have struggled to cope with the size of today's networks.

This paper presents an approach for quickly obtaining accurate performance estimates for large data center networks. Our system, MimicNet, provides users with the familiar abstraction of a packet-level simulation for a portion of the network while leveraging redundancy and recent advances in machine learning to quickly and accurately approximate portions of the network that are not directly visible. MimicNet can provide over two orders of magnitude speedup compared to regular simulation for a data center with thousands of servers. Even at this scale, MimicNet estimates of the tail FCT, throughput, and RTT are within 5% of the true results.

CCS CONCEPTS

• **Networks** → **Network simulations**; *Network performance modeling*; *Network experimentation*; • **Computing methodologies** → Massively parallel and high-performance simulations;

KEYWORDS

Network simulation, Data center networks, Approximation, Machine learning, Network modeling

ACM Reference Format:
Qizhen Zhang, Kelvin K.W. Ng, Charles W. Kazer[§], Shen Yan[†], João Sedoc[◇], and Vincent Liu. 2021. MimicNet: Fast Performance Estimates for Data Center Networks with Machine Learning. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–27, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3452296.3472926

1 INTRODUCTION

Over the years, many novel protocols and systems have been proposed to improve the performance of data center networks [5–7, 12, 19, 33, 39]. Though innovative in their approaches and promising in their results, these proposals suffer from a consistent challenge: the difficulty of evaluating systems at scale. Networks, highly interconnected and filled with dependencies, are particularly challenging in that regard—small changes in one part of the network can result in large performance effects in others.

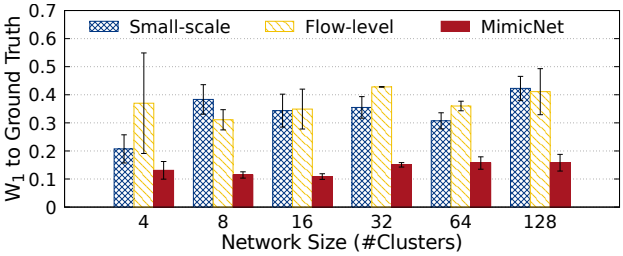


Figure 1: Accuracy for MimicNet's predictions of the FCT distribution for a range of data center sizes. Accuracy is quantified via the Wasserstein distance (W_1) to the distribution observed in the original simulation. Lower is better. Also shown are the accuracy of a flow-level simulator (SimGrid) and the accuracy of assuming a small (2-cluster) simulation's results are representative.

Unfortunately, full-sized testbeds that could capture these effects are prohibitively expensive to build and maintain. Instead, most pre-production performance evaluation comprises orders of magnitude fewer devices and fundamentally different network structures. This is true for (1) hardware testbeds [47], which provide total control of the system, but at a very high cost; (2) emulated testbeds [43, 54, 56], which model the network but at the cost of scale or network effects; and (3) small regions of the production network, which provide *'in vivo'* accuracy but force operators to make a trade-off between scale and safety [48, 59]. The end result is that, often, the only way to ascertain the true performance of the system at scale is to deploy it to the production network.

We note that simulation was originally intended to fill this gap. In principle, simulations provide an approximation of network behavior for arbitrary architectures at an arbitrary scale. In practice, however, modern simulators struggle to provide both simultaneously. As we show in this paper, even for relatively small networks, packet-level simulation is 3–4 orders of magnitude slower than real-time (5 min of simulated time every ~3.2 days); larger networks can easily take months or longer to simulate. Instead, researchers often either settle for modestly sized simulations and assume that performance translates to larger deployments, or they fall back to approaches that ignore packet-level effects like flow approximation techniques. Both sacrifice substantial accuracy.

In this paper, we describe MimicNet, a tool for fast *performance estimation* of at-scale data center networks. MimicNet presents to users the abstraction of a packet-level simulator; however, unlike

SIGCOMM '21, August 23–27, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8383-7/21/08…$15.00
https://doi.org/10.1145/3452296.3472926

existing simulators, MimicNet only simulates—at a packet level—the traffic to and from a single 'observable' cluster, regardless of the actual size of the data center. Users can then instrument the host and network of the designated cluster to collect arbitrary statistics. For the remaining clusters and traffic that are *not* directly observable, MimicNet approximates their effects with the help of deep learning models and flow approximation techniques.

As a preview of MimicNet's evaluation results, Figure 1 shows the accuracy of its Flow-Completion Time (FCT) predictions for various data center sizes and compares it against two common alternatives: (1) flow-level simulation and (2) running a smaller simulation and assuming that the results are identical for larger deployments. For each approach, we collected the simulated FCTs of all flows with at least one endpoint in the observable cluster. We compared the distribution of each approach's FCTs to that of a full-fidelity packet-level simulation using a W_1 metric. The topology and traffic pattern were kept consistent, except in the case of small-scale simulation where that was not possible (instead, we fixed the average load and packet/flow size). While MimicNet is not and will never be a perfect portrayal of the original simulation, it is 4.1× more accurate than the other methods across network sizes, all while improving the time to results by up to two orders of magnitude.

To achieve these results, MimicNet imposes a few carefully chosen restrictions on the system being modeled: that the data center is built on a classic FatTree topology, that per-host network demand is predictable a priori, that congestion occurs primarily on fan-in, and that a given host's connections are independently managed. These assumptions provide outsized benefits to simulator performance and the scalability of its estimation accuracy, while still permitting application to a broad class of data center networking proposals, both at the end host and in the network.

Concretely, MimicNet operates as follows. First, it runs a simulation of a small subset of the larger data center network. Using the generated data, it trains a Mimic—an approximation of clusters' 'non-observable' internal and cross-cluster behavior. Then, to predict the performance of an N cluster simulation, it carefully composes a single observable cluster with $N-1$ Mimic'ed clusters to form a packet-level generative model of a full-scale data center. Assisting with the automation of this training process is a hyperparameter tuning stage that utilizes arbitrary user-defined metrics (e.g., FCT, RTT, or average throughput) and MimicNet-defined metrics (e.g., scale generalizability) rather than traditional metrics like L1/2 loss, which are a poor fit for a purely generative model.

This entire process—small-scale simulation, model training/tuning, and full-scale approximation—can be orders of magnitude faster than running the full-scale simulation directly, with only a modest loss of accuracy. For example, in a network of a thousand hosts, MimicNet's steps take 1h3m, 7h10m, and 25m, respectively, while full simulation takes over a week for the same network/workload. These results hold across a wide range of network configurations and conditions extracted from the literature. This paper contributes:

- Techniques for the modeling of cluster behavior using deep-learning techniques and flow-level approximation. Critical to the design of the Mimic models are techniques to ensure the scalability of their accuracy, i.e., their ability to generalize to larger networks in a zero-shot fashion.

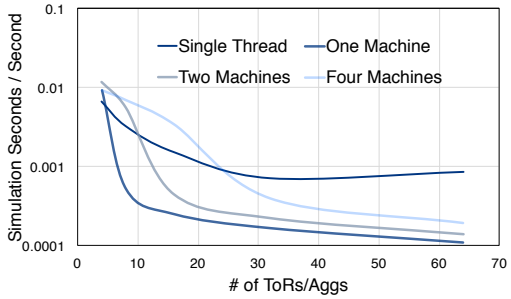


Figure 2: OMNeT++ performance on leaf-spine topologies of various size. Even for these small cases, 5 mins of simulation time can take multiple days to process. Results were similar for ns-3 and other simulation frameworks.

- An architecture for composing Mimics into a generative model of a full-scale data center network. For a set of complex protocols and real-world traffic patterns, MimicNet can match ground-truth results orders of magnitude more quickly than otherwise possible. For large networks, MimicNet even outperforms flow-level simulation in terms of speed (in addition to producing much more accurate results).
- A customizable hyperparameter tuning procedure and loss function design that ensure optimality in both generalization and a set of arbitrary user-defined objectives.
- Implementations and case studies of a wide variety of network protocols that stress MimicNet in different ways.

The framework is available at: https://github.com/eniac/MimicNet.

2 MOTIVATION

Modern data center networks connect up to hundreds of thousands of machines that, in aggregate, are capable of processing hundreds of billions of packets per second. They achieve this via scale-out network architectures, and in particular, FatTree networks like the one in Figure 3 [4, 18, 50]. In the canonical version, the network consists of Top-of-Rack (ToR), Cluster, and Core switches. We refer to the components under a single ToR as a *rack* and the components under and including a group of Cluster switches as a *cluster*. A large data center might have over 100 such clusters.

The size and complexity of these networks make testing and evaluating new ideas and architectures challenging. Researchers have explored many potential directions including verification [15, 26, 27, 35, 57], emulation [52, 54, 56], phased rollouts [48, 59], and runtime monitoring [20, 58]. In reality, all of these approaches have their place in a deployment workflow; however, in this paper, we focus on a critical early step: pre-deployment performance estimation using simulation.

2.1 Background on Network Simulation

The most popular simulation frameworks include OMNeT++ [34], ns-3 [42], and OPNET [1]. Each of these operates at a packet-level and are built around an event-driven model [53] in which the operations of every component of the network are distilled into a sequence of events that each fire at a designated 'simulated time.'

Compared to evaluation techniques such as testbeds and emulation, these simulators provide a number of important advantages:

- *Arbitrary scale:* Decoupling the system model from both hardware and timing constraints means that, in principle, simulations can encompass any number of devices.
- *Arbitrary extensions:* Similarly, with full control over the simulated behavior, users can model any protocol, topology, design, or configuration.
- *Arbitrary instrumentation:* Finally, simulation allows the collection of arbitrary information at arbitrary granularity without impacting system behavior.

In return for the above benefits, simulators trade-off varying levels of accuracy compared to a bare-metal deployment. Even so, prior work has demonstrated their value in approximating real behavior [5, 6, 33, 46, 55].

2.2 Scalability of Today's Simulators

While packet-level simulation is easy to reason about and extend, simulating large and complex networks is often prohibitively slow. One reason for this is that discrete-event simulators, in essence, take a massive distributed system and serialize it into a single event queue. Thus, the larger the network, the worse the simulation performs in comparison.

Parallelization. A natural approach to improving simulation speed is parallelization, for instance, with the parallel DES (PDES) technique [17]. In PDES, the simulated network is partitioned into multiple *logical processes* (LPs), where each process has its own event queue that is executed in parallel. Eventually, of course, the LPs must communicate. In particular, consistency demands that a process cannot finish executing events at simulated time t unless it can be sure that no other process will send it additional events with $t_e < t$. In these cases, synchronization may be necessary.

Parallel execution is therefore only efficient when the LPs can run many events before synchronization is required, which is typically not the case for highly interconnected data center networks. In fact, simulation performance often *decreases* in response to parallelization (see Figure 2). Many frameworks instead recommend running several instances with different configurations [14]. This trivially provides a proportional speedup to aggregate simulation throughput but does not improve the time to results.

Approximation. The other common approach is to leverage various forms of approximation. For example, flow-level approaches [38] average the behavior of many packets to reduce computation. Closed-form solutions [37] and a vast array of optimized custom simulators [33, 45, 46] also fall in this category. While these approaches often produce good performance; they require deep expertise to craft and limit the metrics that one can draw from the analysis.

3 DESIGN GOALS

MimicNet is based around the following design goals:

- *Arbitrary scale, extensions, and instrumentation:* Acknowledging the utility of packet-level simulation in enabling flexible and rich evaluations of arbitrary network designs, we seek to provide users with similar flexibility with MimicNet.
- *Orders of magnitude faster results:* Equally important, MimicNet must be able to provide meaningful performance estimates several orders of magnitude faster than existing approaches. Parallelism, on its own, is not enough—we seek to decrease the total amount of work.
- *Tunable and high accuracy:* Despite the focus on speed, MimicNet should produce observations that resemble those of a full packet-level simulation. Further, users should be able to define their own accuracy metrics and to trade this accuracy off with improved time to results.

Explicitly *not* a goal of our framework is full generality to arbitrary data center topologies, routing strategies, and traffic patterns. Instead, MimicNet makes several carefully chosen and domain-specific assumptions (described in Section 4.2) that enable it to scale to larger network sizes than feasible in traditional packet-level simulation. We argue that, in spite of these restrictions, MimicNet can provide useful insights into the performance of large data centers.

4 OVERVIEW

MimicNet's approach is as follows. Every MimicNet simulation contains a single 'observable' cluster, regardless of the total number of clusters in the data center. All of the hosts, switches, links, and applications in this cluster as well as all of the remote applications with which it communicates are simulated in full fidelity. All other behavior—the traffic between un-observed clusters, their internals, and anything else not directly observed by the user—is approximated by trained models.

While prior work has also attempted to model systems and networks (e.g., [54, 56]), these prior systems tend to follow a more traditional script by (1) observing the *entire* system/network and (2) fitting a model to the observations. MimicNet is differentiated by the insight that, by carefully composing models of small pieces of a data center, *we can accurately approximate the full data center network using only observations of small subsets of the network.*

4.1 MimicNet Design

MimicNet constructs and composes models at the granularity of individual data center clusters: Mimics. From the outside, Mimics resemble regular clusters. Their hosts initiate connections and exchange data with the outside world, and their networks drop, delay, and modify that traffic according to the internal queues and logic of the cluster's switches. However, Mimics differ in that they are able to predict the effects of that queuing and protocol manipulation without simulating or interacting with other Mimics—only with the observable cluster.

We note that the goal of MimicNet is not to replicate the effects of any particular large-scale simulation, just to generate results that exhibit their characteristics. It accomplishes the above with the help of two types of models contained within each Mimic: (1) deep-learning-based *internal models* that learn the behavior of switches, links, queues, and intra-cluster cross-traffic; and (2) flow-based *feeder models* that approximate the behavior of inter-cluster cross-traffic. The latter is parameterized by the size of the data center. Together, these models take a sequence of observable packets and their arrival times and output the cluster's predicted effects:

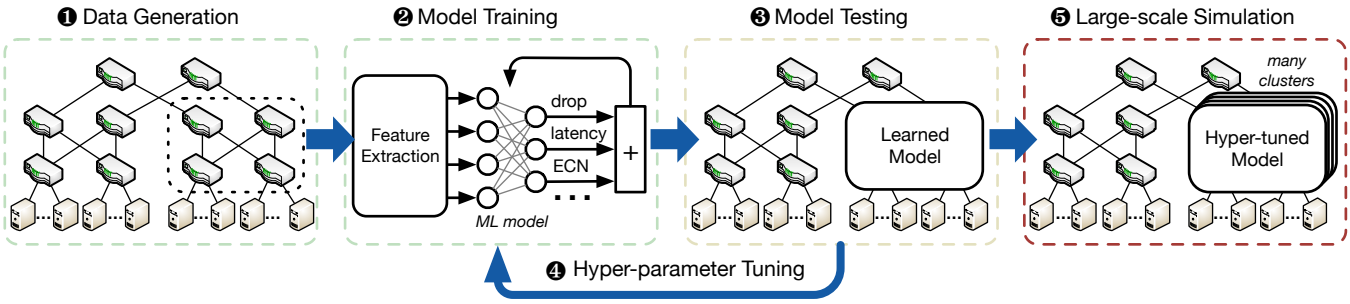


Figure 3: The end-to-end, fully automated workflow of MimicNet. (❶) Small-scale observations, (❷) model training, (❸) model testing, and (❹) optional hyper-parameter tuning produce tuned machine learning models for use in Mimics, which speed up large-scale simulations (❺) by replacing the majority of the network. A key feature of MimicNet is that the traditionally slow steps of ❶, ❷, ❸, and ❹ are all done at small scale and are, therefore, fast as well.

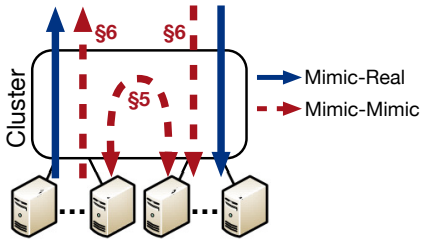


Figure 4: Breakdown of traffic in a to-be-approximated cluster. MimicNet approximates all traffic that does not interact with the observable cluster (dotted-red lines) using the models in the referenced sections.

(1) Whether the packets are dropped as a result of the queue management policy.
(2) When the packets egress the Mimic, given no drop.
(3) Where the packets egress, based on the routing table.
(4) The contents of the packets after traversing the Mimic, including modifications such as TTL and ECN.

Workflow. The usage of MimicNet (depicted in Figure 3) begins with a small subset of the full simulation: just two user-defined clusters communicating with one another. This full-fidelity, small-scale simulation is used to generate training and testing sets for supervised learning of the models described above. Augmenting this training phase is a configurable hyper-parameter tuning stage in which MimicNet explores various options for modeling with the goal of maximizing both (a) user-defined, end-to-end accuracy metrics like throughput and FCT, and (b) generalizability to larger configurations and different traffic matrices.

Using the trained models, MimicNet assembles a full-scale simulation in which all of the clusters in the network (save one) are replaced with Mimics. For both data generation and large-scale simulation, MimicNet uses OMNeT++ as a simulation substrate.

Performance analysis. To understand MimicNet's performance gains, consider the Mimic in Figure 4 and the types of packets that flow through it. At a high level, there are two such types: (1) traffic that interacts with the observable cluster (Mimic-Real), and (2) traffic that does not (Mimic-Mimic).

As a back-of-the-envelope computation, assume that we simulate N clusters, $N \gg 2$. Also assume that T is the total number of packets sent in the full simulation of the data center and that p is the ratio of traffic that leaves a cluster vs. that stays within it (inter-cluster-to-intra-cluster), $0 \leq p \leq 1$. The number of packets that leave a single cluster in the full simulation is then approximately $\frac{Tp}{N}$.

Because Mimics only communicate with the single observable cluster and not each other, the number of packets that leave a Mimic in an approximate simulation is instead:

$$\frac{Tp}{N(N-1)}$$

Thus, the total number of packets generated in a MimicNet simulation (the combination of all traffic generated at the observable cluster and $N-1$ Mimics) is:

$$\frac{T}{N} + \frac{(N-1)Tp}{N(N-1)} = \frac{T+Tp}{N}$$

The total decrease in packets generated is, therefore, a factor between $\frac{N}{2}$ and N with a bias toward N when traffic exhibits cluster-level locality. Fewer packets and connections generated mean less processing time and a smaller memory footprint. It also means a decrease in inter-cluster communication, which makes the composed simulation more amenable to parallelism than the full version.

4.2 Restrictions

MimicNet makes several domain-specific assumptions that aid in the scalability and accuracy of the MimicNet approach.

- *Failure-free FatTrees:* MimicNet assumes a FatTree topology, where the structure of the network is recursively defined and packets follow a strict up-down routing. This allows it to assume symmetric bisection bandwidth and to break cluster-level modeling into simpler subtasks.
- *Traffic patterns that scale proportionally:* To ensure that models trained from two clusters scale up, MimicNet requires a per-host synthetic model of flow arrival, flow size, packet size, and cluster-level locality that is independent of the size of the network. In other words (at least at the host level), users should ensure that the size and frequency of packets in the first step resemble those of the last step. We note that popular datasets used in recent literature already adhere to this [6, 8, 33, 40].
- *Fan-in bottlenecks:* Following prior work, MimicNet assumes that the majority of congestion occurs on fan-in toward the

destination [24, 50]. This allows us to focus accuracy efforts on only the most likely bottlenecks.

- *Intra-host isolation:* To enable the complete removal of Mimic-Mimic connections at end hosts, MimicNet requires that connections be logically isolated from one another inside the host—MimicNet models network effects but does not model CPU interactions or out-of-band cooperation between connections.

MimicNet, as a first step toward large-scale network prediction is, thus, not suited for evaluating every data center architecture or configuration. Still, we argue that MimicNet can provide useful performance estimates of a broad class of proposals. We also discuss potential relaxations to the above restrictions in Appendix A, but leave those for future work.

5 INTERNAL MODELS

As mentioned, Mimics are composed of two types of models. The first type models internal cluster behavior. Its goal is twofold:

(1) For *external traffic* (both Mimic-Real and Mimic-Mimic), to be able to predict how the network of the cluster will affect the packet: whether it drops, its latency, its next hop, and any packet modifications.
(2) For *internal traffic* (between hosts in the same Mimic), to remove it and bake its effects into the above predictions. In other words, during inference, the model should account for the observable effects of internal traffic without explicitly seeing it.

Note that not all observable effects need to be learned, especially if the result can be computed using a simple, deterministic function, e.g., TTLs or ECMP. However, for others—drops, latency, ECN marking, NDP truncation, and so on—the need for the models to scale to unobserved configurations presents a unique challenge for generalizable learning. To address the challenge, MimicNet carefully curates training data, feature sets, and models with an explicit emphasis on ensuring that generated models are *scale-independent*.

5.1 Small-scale Observations

MimicNet begins by running a full-fidelity, but small-scale simulation to gather training/testing data.

Simulation and instrumentation. Data generation mirrors the depiction in Figure 3. Users first provide their host and switch implementations in a format that can be plugged into the C++-based OMNeT++ simulation framework.

Using these implementations, MimicNet runs a full-fidelity simulation of two clusters connected via a set of Core switches. Among these two clusters, we designate one as the cluster to be modeled and dump a trace of all packets entering and leaving the cluster. In a FatTree network, this amounts to instrumenting the interfaces facing the Core switches and the Hosts. Between these two junctures are the mechanics of the queues and routers—these are what is learned and approximated by the Mimic internal model.

Pre-processing. MimicNet takes the packet dumps and matches the packets entering and leaving the network using identifiers from the packets (e.g., sequence numbers). Examining the matches helps to determine the length of time it spent in the cluster and any changes to the packet. There are two instances where a 1-to-1 matching may not be possible: loss and multicast. Loss can be detected as a packet entering the cluster but never leaving. Multicast must be tracked by the framework. Both can be modeled.

5.2 Modeling Objectives

MimicNet models the clusters' effects as machine learning tasks. More formally, for each packet of external traffic, i:

Latency regression. We model the time that i spends in the cluster's network as a bounded continuous random variable and set the objective to minimize the Mean Absolute Error (MAE) between the real latency and the prediction:

$$\min \sum |y_i^l - \hat{y}_i^l|,$$

where y_i^l is $(L_{\max}+\epsilon)$ if the packet is dropped and $(lat \in [L_{\min}, L_{\max}])$ otherwise. $\hat{y}_i^l$ is the predicted latency. To improve the accuracy of this task, MimicNet uses *discretization* in training latency models. Specifically, MimicNet quantizes the values using a linear strategy:

$$f(y^l) = \left\lfloor \frac{y^l - L_{\min}}{L_{\max} - L_{\min}} \times D \right\rfloor$$

where D is the hyperparameter that controls the degree of discretization. By varying D, we can trade off the ease of modeling and the recovery precision from discretization.

Drops and packet modification classification. For most other tasks, classification is a better fit. For example, the prediction of a packet drop has two possible outcomes, and the objective is to minimize Binary Cross Entropy (BCE):

$$\min \sum -y_i^d \log \hat{y}_i^d - (1 - y_i^d) \log(1 - \hat{y}_i^d)$$

where y_i^d is 1 if i is dropped and 0 otherwise, and $\hat{y}_i^d \in [0, 1]$ is the predicted probability that i is dropped. Packet modifications like ECN-bit prediction share a similar objective.

Both regression and classification tasks are modeled together with a unified loss function, which we describe in Section 5.4.

5.3 Scalable Feature Selection

With the above formulations, MimicNet must next select features that map well to the target predictions. While this is a critical step in any ML problem, MimicNet introduces an additional constraint—that the features be *scalable*.

A scalable feature is one that remains meaningful regardless of the number of clusters in the simulation. Consider a packet that enters the Mimic cluster from a Core switch and is destined for a host within the cluster. The local index of the destination rack ([0, R) for a cluster of R racks) would be a scalable feature as adding more clusters does not affect the value, range, or semantics of the feature. In contrast, the IP of the source server would NOT be a scalable feature. This is because, with just two clusters, it uniquely identifies the origin of the packet, but as clusters are added to the simulation, never-before-seen IPs are added to the data.

Table 1 lists the scalable features in a typical data center network with ECMP and TCP, applicable to both ingress and egress packets. Other scalable features that are not listed include priority bits, packet types, and ECN markings.

Feature	Count
Local rack	*# Racks per cluster*
Local server	*# Servers per rack*
Local cluster switch	*# Cluster switches per cluster*
Core switch traversed	*# Core switches*
Packet size	*single integer value*
Time since last packet	*single real value (discretized)*
EWMA of the above feature	*single real value (discretized)*

Table 1: Basic set of scalable features.

MimicNet performs two transformations on the captured features: one-hot encoding the first four features to remove any implicit ordering of devices and discretizing the two time-related features as in Section 5.2. Crucially, all of these features can quickly be determined using only packets' headers, switch routing tables, and the simulator itself.

5.4 DCN-friendly Loss Functions

The next task is to select an appropriate training loss function. Several characteristics of this domain make it difficult to apply the objective functions of Section 5.2 directly.

Class imbalances. Even in heavily loaded networks, adverse events like packet drops and ECN tagging are relatively rare occurrences. For example, Figure 5a shows an example trace of drops over a one-second period in a simulation of two clusters. 99.7% of training examples in the trace are delivered successfully, implying that a model of loss could achieve high accuracy even if it always predicts 'no drop.' Figure 5b exemplifies this effect using an LSTM trained using BCE loss on the same trace as above. It predicts a drop rate of almost an order of magnitude lower than the true rate.

To address this instance of class imbalance, MimicNet takes a cost-sensitive learning approach [13] by adopting a Weighted-BCE (WBCE) loss:

$$\ell^d = -(1-w)\sum y_i^d \log \hat{y}_i^d - w\sum (1-y_i^d)\log(1-\hat{y}_i^d)$$

where w is the hyperparameter that controls the weight on the drop class. Figure 5c and 5d show that weighting drops can significantly improve the prediction accuracy. We note, however, that setting w too high can also produce false positives. From our experience, 0.6~0.8 is a reasonable range, and we rely on tuning techniques in Section 7.2 to find the best w for a given network configuration and target metric.

Outliers in latencies. In latency, an equivalent challenge is accurately learning tail behavior. For example, consider the latencies from the previous trace, shown in Figure 6a. While most values are low, a few packets incur very large latencies during periods of congestion; these outliers are important for accurately modeling the network.

Unfortunately, MAE as a loss function fails to capture the importance of these values, as shown in the latency predictions of an MAE-based model (Figure 6b), which avoids predicting high latencies. We note that the other common regression loss function, Mean Squared Error (MSE), has the opposite problem—it squares the loss for each sample and produces models that tend to overvalue outliers (Figure 6c).

MimicNet strikes a balance with the Huber loss [23]:

$$\ell^l = \sum H_\delta(y_i^l, \hat{y}_i^l)$$

$$H_\delta(y^l, \hat{y}^l) = \begin{cases} \frac{1}{2}(y^l - \hat{y}^l)^2, & \text{if } |y^l - \hat{y}^l| \le \delta, \\ \delta|y^l - \hat{y}^l| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases}$$

where $\delta \in \mathbb{R}^+$ is a hyperparameter. Essentially, the Huber loss assumes a heavy-tailed error distribution and uses the squared loss and the absolute loss under different situations. Figure 6d shows results for a model trained with the Huber loss ($\delta = 1$). In this particular case, it reduces inaccuracy (measured in MAE) of the 99-pct latency from 13.2% to only 2.6%.

Combining loss functions. To combine the above loss functions during model training, MimicNet normalizes all values and weights them using hyperparameters. Generally speaking, a weight that favors latency over other metrics is preferable as regression is a harder task than classification.

5.5 Generalizable Model Selection

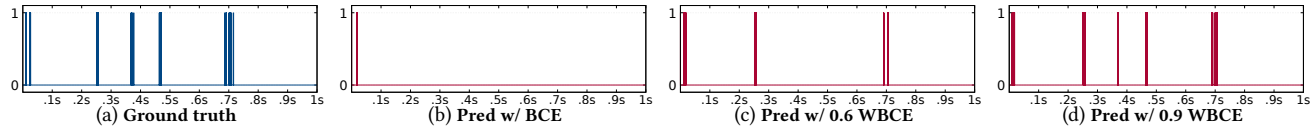
Finally, with both features and loss functions, MimicNet can begin to model users' clusters. The model should be able to learn to approximate the mechanics of the queues and interfaces as well as cluster-local traffic and its reactions to network conditions (e.g., as a result of congestion control).

Many models exist and the optimal choice for both speed and accuracy will depend heavily on the target network. To that end, MimicNet can support any ML model. Given our desire for generality, however, it currently leverages one particularly promising class of models: LSTMs. LSTMs have gained recent attention for their ability to learn complex underlying relationships in sequences of data without explicit feature engineering [22].

Ingress/egress decomposition. To simplify the required models and improve training efficiency, MimicNet models ingress and egress traffic separately. This approach is partially enabled by MimicNet's requirement of strict up-down routing, the intrinsic modeling of cluster-local traffic, and the assumption of fan-in congestion. While there are still some inaccuracies that arise from this decision (e.g., the effect of shared buffers), we found that this choice was another good speed/accuracy tradeoff for all architectures we tested. For each direction of traffic, the LSTMs consist of an input layer and a stack of flattened, one-dimensional hidden layers. The hidden size is $\#features \times \#packets$ where $\#packets$ is the number of packets in a sample, and $\#features$ is post one-hotting.

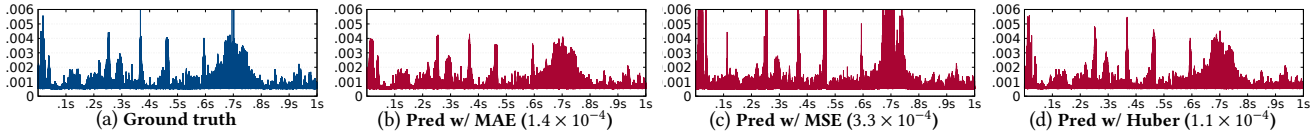
Congestion state augmentation. While in principle, LSTMs can retain 'memory' between predictions to learn long-term patterns, in practice, they are typically limited to memory on the order of 10s or 100s of samples. In contrast, the traffic seen by a Mimic may exhibit self-similarity on the order of hundreds of thousands of packets. Our problem, thus, exhibits properties of multiscale models [11].

Because of this, we augment the LSTM model with a piece of network domain knowledge: an estimation of the presence of congestion in each cluster's network. Specifically, four distinct states are considered: (1) little to no congestion, (2) increasing congestion as queues fill, (3) high congestion, and (4) decreasing congestion as queues drain. These states are estimated by looking at the latency



(a) Ground truth (b) Pred w/ BCE (c) Pred w/ 0.6 WBCE (d) Pred w/ 0.9 WBCE

Figure 5: Ground truth and LSTM-predicted drops for a one-second test set using different loss functions. The y-axis is 1 for dropped, 0 for not. Ground truth has 0.3% drop rate and BCE loss has 0.01%. WBCE results in more realistic drop rates depending on the weight ($w = 0.6$: 0.14%; $w = 0.9$: 0.49%).



(a) Ground truth (b) Pred w/ MAE (1.4×10^{-4}) (c) Pred w/ MSE (3.3×10^{-4}) (d) Pred w/ Huber (1.1×10^{-4})

Figure 6: Ground truth and LSTM-predicted latency (in seconds) for a one-second test set using different loss functions. With each, we report the output of the objective, MAE (listed in parentheses). Unfortunately, using MAE directly as the loss function fails to capture outliers. Instead, Huber produces more realistic results and a better eventual MAE score.

and drop rate of recently processed packets in the cluster. By breaking the network up into these four coarse states, the LSTM is able to efficiently learn patterns over these regimes, each with distinct behaviors. This feature is added to the others in Table 1.

6 FEEDER MODELS

While the above (internal) models can model the behavior of the queues, routers, and internal traffic of a cluster, the complete trace of external traffic is still required to generate accurate results. In the terminology of Figure 4, internal models bake in the effects of the intra-cluster traffic, but the LSTMs are trained on *all* external traffic, not just Mimic-Real.

To replace the remaining non-observable traffic, the internal models are augmented with a *feeder* whose role is to estimate the arrival rate of inter-Mimic traffic and inject them into the internal model. Creating a feeder model is challenging compared to internal cluster models as inter-Mimic traffic is not present in the small-scale simulation and varies as the simulation scales. MimicNet addresses this by creating a parameterized and fully generative model that uses flow-level approximation techniques to predict the packet arrival rate of Mimic-Mimic traffic in different network sizes.

The feeder model is trained in parallel to the internal models. MimicNet first derives from the small-scale simulation characteristic packet interarrival distributions for all external flows, separated by their direction (ingress/egress). In our tests, we observed, as others have in the past [8, 31] that simple log-normal or Pareto distributions produced reasonable approximations of these interarrival times. Nevertheless, more sophisticated feeders can be trained and parameterized in MimicNet. During the full simulation, the feeders will take the hosts' inter-cluster demand as a parameter, compute a time-series of active flow-level demand, and draw packets randomly from that demand using the derived distributions.

Crucially, when feeding packets, the feeders generate 'packets' independently, pass their raw feature vectors to the internal models, and immediately discard any output. This means that internal models' hidden state is updated as if the packets were routed without actually incurring the costs of creating, sending, or routing them. While this approach shares the weaknesses of other flow-level approximations, like the removal of intra-cluster traffic, these packets are never directly measured and, thus, an approximation of their effect is sufficient. Further, while the traffic is never placed in the surrounding queues, i.e., queues of the Core switch or the egress queues on the Hosts; as prior work has noted, the majority of drops and congestion are found elsewhere in the network [50].

7 TUNING AND FINAL SIMULATION

MimicNet composes Mimics into a parallelized large-scale data center simulation. In addition to designing the internal and feeder models with scale-independence in mind, it ensures the models survive scaling with a hyper-parameter tuning phase.

7.1 Composing Mimics

An N-cluster MimicNet simulation consists of a single real cluster, $N - 1$ Mimic clusters, and a proportional number of Core switches. The real cluster continues to use the user implementation of Section 5.1, but users can add arbitrary instrumentation, e.g., by dumping `pcaps` or queue depths.

The Mimic clusters are constructed by taking the ingress/egress internal models and feeders developed in the previous sections and wrapping them with a thin shim layer. The layer intercepts packets arriving at the borders of the cluster, periodically takes packets from the feeders, and queries the internal models with both to predict the network's effects. The output of the shim is, thus, either a packet, its egress time, and its egress location; or its absence. Adjacent hosts and Core switches are wired directly to the Mimic, but are otherwise unaware of any change.

Aside from the number of clusters, all other parameters are kept constant from the small-scale to the final simulation. That includes the feeder models and traffic patterns, which take a size parameter but fix other parameters (e.g., network load and flow size).

7.2 Optional Hyper-parameter Tuning

Mimic models contain at least a few hyper-parameters that users can optionally choose to tune: WBCE weight, Huber loss δ, LSTM layers, hidden size, epochs, and learning rate among others. MimicNet provides a principled method of setting these by allowing users to define their own optimization function. This optimization function is distinct from the model objectives or the loss functions. Instead, they can evaluate end-to-end accuracy over arbitrary behavior in the simulation (for instance, tuning for accuracy of FCTs). Users can add hyper-parameters or end-to-end optimization functions depending on their use cases.

For every tested parameter set, MimicNet trains a set of models and runs validation tests to evaluate the resulting accuracy and its scale-independence. Specifically, MimicNet runs an approximated and full-fidelity simulation on a held-out validation workload in three configurations: 2, 4, and 8 clusters. It then compares the two versions using the user's target metric.

The full-fidelity comparison results are only gathered once, and the MimicNet results are evaluated for every parameter set, but the sizes are small enough that the additional time is nominal. Based on the user-defined metric, MimicNet uses Bayesian Optimization (BO) to pick the next parameter set that has the highest 'prediction uncertainty' via an *acquisition function* of EI (expected improvement). In this way, BO quickly converges on the optimal configuration.

MimicNet supports two classes of metrics natively.

MSE-based metrics. For 1-to-1 metrics, MimicNet provides a framework for computing MSE. For example, when comparing the FCT of the same flow in both simulations:

$$\text{MSE} = \frac{1}{|Flows|} \sum_{f \in Flows} (\text{realFCT}_f - \text{mimicFCT}_f)^2$$

A challenge in using this class of metrics is that the set of completed flows in the full-fidelity network and MimicNet are not necessarily identical—over a finite running timespan, flow completions that are slightly early/late can change the set of observed FCTs. To account for this, we only compute MSE over the intersection, i.e.,

$$Flows = \{f \mid (\exists\ \text{realFCT}_f \wedge (\exists\ \text{mimicFCT}_f)\}$$

By default, MimicNet ignores models with overlap < 80%.

Wasserstein-based metrics. Unfortunately, not all metrics can be framed as above. Consider per-packet latencies. While in training we assume that we can calculate a per-packet loss and backpropagate, in reality when a drop is mistakenly predicted, the next prediction should reflect the fact that there is one fewer packet in the network, rather than adhering to the original packet trace. In some protocols like TCP, the loss may even cause packets to appear in the original but not in any MimicNet version or vice versa.

MimicNet's hyper-parameter tuning phase, therefore, allows users to test distributions, e.g., of RTTs, FCTs, or throughput, via the Wasserstein metric. Also known as the Earth Mover's Distance, the metric quantifies the minimum cost of transforming one distribution to the other [16]. Specifically, for a one-dimensional CDF, the metric (W_1) is:

$$W_1 = \int_{-\infty}^{+\infty} |CDF_{\text{real}}(x) - CDF_{\text{mimic}}(x)|$$

W_1 values are scale-dependent, with lower numbers indicating greater similarity.

8 PROTOTYPE IMPLEMENTATION

We have implemented a prototype of the full MimicNet workflow in C++ and Python on top of PyTorch/ATen and the OMNeT++ [34] simulation suite. Given an OMNeT++ router and host implementation, our prototype will generate training data, train/hypertune a set of MimicNet models, and compose the resulting models into an optimized, full-scale simulation. This functionality totals to an additional 25,000 lines of code.

Simulation framework. MimicNet is built on OMNeT++ v4.5 and INET v2.4 with custom C++ modules to incorporate our machine learning models into the framework. To ensure that the experiments are repeatable, all randomness, including the seeds for generating the traffic are configurable. They were kept consistent between variants and changed across training, testing, and cross validation.

Parallel execution. A side benefit MimicNet is that it significantly reduces the need for synchronization in a parallel execution. In order to take advantage of this property, we parallelize each cluster of the final simulation using an open-source PDES implementation of INET [51].

Machine learning framework. Our LSTM models are trained using PyTorch 0.4.1 and CUDA 9.2 [41, 44]. Hyperparameter tuning was done with the assistance of `hyperopt` [2]. At runtime, Mimic cluster modules accept OMNeT++ packets, extract their features, perform a forward step of the LSTMs, and forward the packet via ECMP based on the result. For speed, our embedded LSTMs were custom-built inference engines that leverage low-level C++ and CUDA functions from the Torch, cuDNN, and ATen libraries.

9 EVALUATION

Our evaluation focuses on several important properties of MimicNet including: (1) its accuracy of approximating the performance of data center networks, (2) the scalability of its accuracy to large networks, (3) the speed of its approximated simulations, and (4) its utility for comparing configurations.

Methodology. Our simulations all assume a FatTree topology, as described in Section 2. We configured the link speed to be 100 Mbps with a latency of 500 μs. To scale up and down the data center, we adjusted the number of racks/switches in each cluster as well as the number of clusters in the data center. We note that higher speeds and larger networks were not feasible due to the limitation of needing to evaluate MimicNet against a full-fidelity simulation, which would have taken multiple years to produce even a single equivalent execution.

The base case uses TCP New Reno, Drop Tail queues, and ECMP. To test MimicNet's robustness to different network architectures, we use a set of protocols: DCTCP [6], Homa [40], TCP Vegas [9], and TCP Westwood [36] that stress different aspects of MimicNet. Our workload uses traces from a well-known distribution also used by many recent data center proposals [6, 40]. By default, the traffic utilizes 70% of the bisection bandwidth and the mean flow size is 1.6 MB. All experiments were run on CloudLab [47] using machines with two Intel Xeon Silver 4114 CPUs and an NVIDIA P100 GPU. When evaluating flow-level simulation, we use the SimGrid [10] v3.25 and its built-in `FatTreeZone` configured with the same topology and traffic demands as full/MimicNet simulation.

Evaluation metrics. As mentioned in Section 7.2, traditional per-prediction metrics like training loss are not useful in our context. Instead, we leverage three end-to-end metrics: (1) FCT, (2) per-server Throughput binned into 100 ms intervals, (3) and RTT. In the flow-level simulation, FCT is computed using flow start/end times, Throughput is computed with a custom load-tracking plugin, and RTT is not possible to compute. In MimicNet and full simulation, all three are computed by instrumenting the hosts in the observable

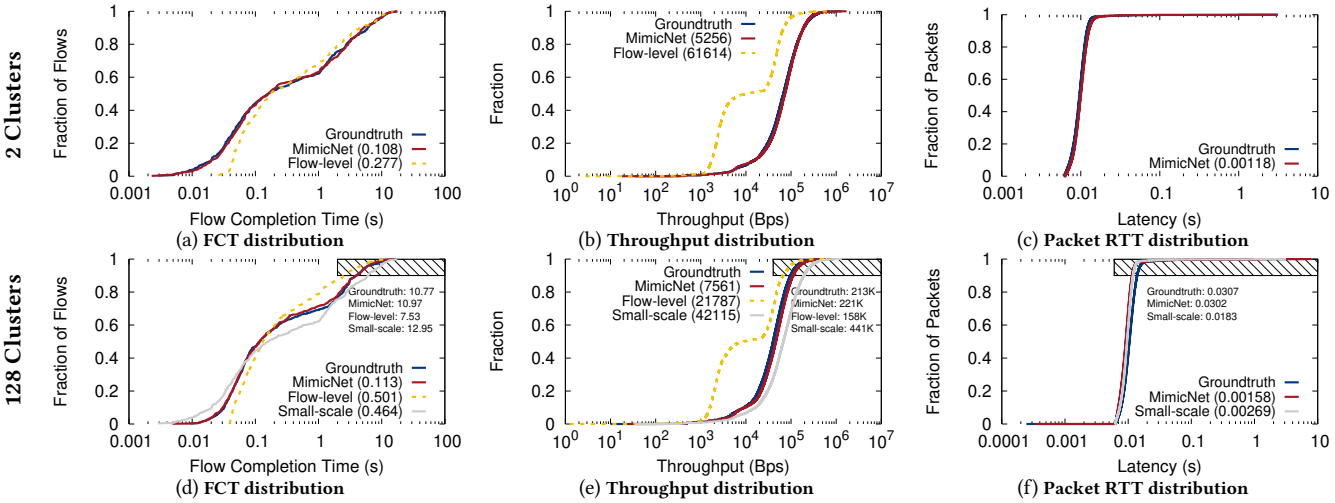


Figure 7: The accuracy of MimicNet in the baseline configuration for 2 clusters and 128 clusters. Also shown are results from SimGrid and the assumption that small-scale results are representative. W_1 to ground truth is shown in parentheses. We annotate the 99-pct value of each metric for every approach at the tail in 128 clusters.

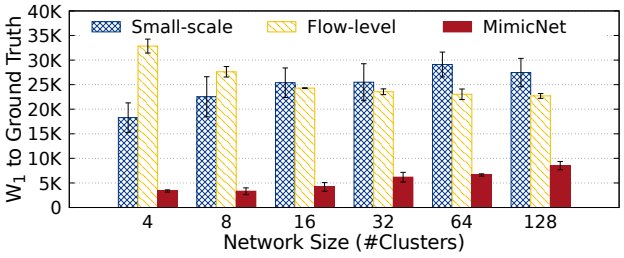


Figure 8: Throughput Scalability.

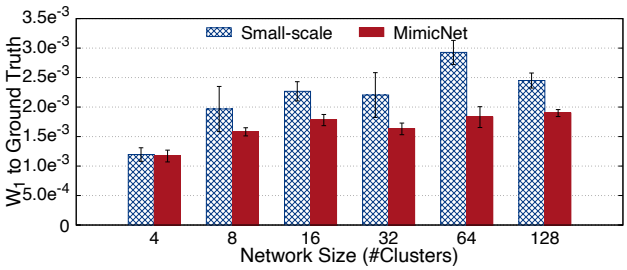


Figure 9: RTT Scalability. Flow-level simulation is too coarse-grained to provide this metric.

cluster to track packets sends and ACK receipts. Where applicable, we compare CDFs using a W_1 metric.

9.1 MimicNet Models Clusters Accurately

We begin by evaluating MimicNet's accuracy when replacing a single cluster with a Mimic before examining larger configurations in the next section. Note that in this configuration, there is no need for feeder models. Rather, this experiment directly evaluates the effect of replacing a cluster's queues, routers, and cluster-local traffic with an LSTM. For this test, we use the baseline set of protocols described above. The final results use traffic patterns that are not found in the training or hyper-parameter validation sets.

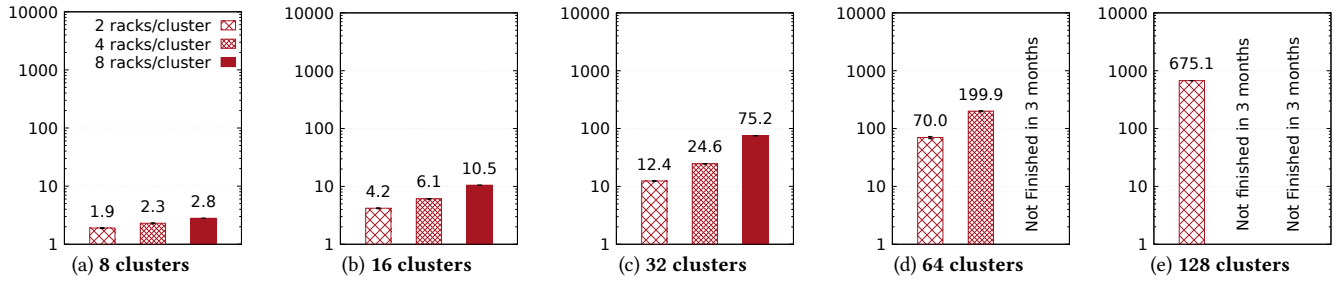
Figure 7a–c show CDFs of our three metrics for this test. As the graphs show, MimicNet achieves very high accuracy on all metrics. The LSTM is able to learn the requisite long-term patterns (FCT and throughput) as well as packet RTTs. Across the entire range, MimicNet's CDFs adhere closely to the ground truth, i.e., the full-fidelity, packet-level simulation; just as crucial, the shape of the curve is maintained. Flow-level simulation behaves much worse.

9.2 MimicNet's Accuracy Scales

A key question is whether the accuracy translates to larger compositions where traffic interactions become more complex and feeders are added. We answer that question using a simulation composed of 128 clusters (full-fidelity simulation did not complete for larger sizes). In MimicNet, 127 clusters are replaced with the same Mimics as the previous subsection. Figure 7d–f show the resulting accuracy. There are a couple of interesting observations.

First, while the accuracy of MimicNet estimation does decrease, the decrease is nominal. More concretely, for FCT, throughput, and RTT, we find W_1 metrics of 0.113, 7561, and 0.00158 compared to the ground truth, respectively. For reference, we also plot SimGrid and the original 2-cluster simulation's results. The W_1 error between 2-cluster simulation and 128-cluster groundtruth are 311%, 457%, and 70% higher than MimicNet's values; the W_1s of FCT and throughput between SimGrid and the groundtruth are similarly high. The results indicate that our composition methods are successfully approximating the scaling effects. Critically, MimicNet also predicts tails well: the p99 of MimicNet's FCT, throughput, and RTT distributions are within 1.8%, 3.3%, and 2% of the true result.

We evaluate MimicNet's scalability of accuracy more explicitly in Figures 1, 8, and 9. Here, we plot the W_1 metric of all three approaches for several data center sizes ranging from 4 to 128. Recall that the 2-cluster results essentially hypothesize that FCT, throughput, and RTT do not change as the network scales. An upward trend on their W_1 metric in all three graphs suggests that the opposite is true. Compared to that baseline, MimicNet on average achieves a 43% lower RTT W_1 error, 78% lower throughput error, and 63% lower FCT error. In all cases, MimicNet also shows much lower



(a) **8 clusters** (b) **16 clusters** (c) **32 clusters** (d) **64 clusters** (e) **128 clusters**

Figure 10: Simulation running time speedup brought by MimicNet in different sizes of data centers. In a network of 128 clusters (256 racks), MimicNet reduces the simulation time from 12 days to under 30 minutes, achieving more than two orders of magnitude speedup. The speedups are consistent and stable across different workloads.

variance across workloads, demonstrating better predictability at approximating large-scale networks.

9.3 MimicNet Simulates Large DCs Quickly

Equally important, MimicNet can estimate performance very quickly. The multiple phases of MimicNet—small-scale simulation, model training, hyper-parameter tuning, and large-scale composition—each require time, but combined, they are still faster than running the full-fidelity simulation directly. By paying the fixed costs of the first two phases, the actual simulation can be run while omitting the majority of the traffic and network connections.

Execution time breakdown. Table 2 shows a breakdown of the running time of both the full simulation and MimicNet, factored out into its three phases for the 128 cluster, 1024 host simulation in Figure 1. For 20 seconds of simulated time, the full-fidelity simulator required almost 1w 5d. In contrast, MimicNet, in aggregate, only required 8h 38m, where just 25m was used for final simulation—a 34× speedup. Longer simulation periods or multiple runs for different workload seeds would have led to much larger speedups.

Simulation time speedup. We focus on the non-fixed-cost component of the execution time in order to better understand the benefits of MimicNet. Figure 10 shows the speedup of MimicNet after the initial, fixed cost of training a cluster model. For each network configuration, we run both MimicNet and a full simulation over the exact same sets of generated workloads. We then report the average speedup and the standard error across those workloads.

In both systems, simulation time consists of both setup time (constructing the network, allocating resources, and scheduling the traffic) as well as packet processing time. MimicNet substantially speeds up both phases.

MimicNet can provide consistent speedups up to 675× for the largest network that full-fidelity simulation was able to handle. Above that size, full-fidelity could not finish within three months, while MimicNet can finish in under an hour. Somewhat surprisingly, *MimicNet is also 7× faster than flow-level approximation at this scale* as SimGrid must still track all of the Mimic-Mimic connections.

Groups of simulations. We also acknowledge that simulations are frequently run in groups, for instance, to test different configuration or workload parameters. To evaluate this, we compare several different approaches to running groups of simulations and evaluate them using two metrics: (1) *simulation latency*, i.e., the total time it takes to obtain the full set of results, and (2) *simulation throughput*, i.e., the average number of aggregate simulation seconds that can be processed per second. In this section, we focus on the effect of network size on these metrics, but we also evaluated the effect of simulation length in Appendix F and the effect on compute consumption in Appendix G.

	Factor	Time
MimicNet	Small-scale simulation	1h 3m
	Training and hyper-param tuning	7h 10m
	Large-scale simulation	**25m**
Full	Simulation	**1w 4d 22h 25m**

Table 2: Running time comparison for 20 s of simulated time of a 128 cluster, 1024 host data center. Benefits of MimicNet increase with simulated time as the first two values for MimicNet are constant.

Simulation latency: For latency, N cores in a machine, and S simulation seconds, we consider five different approaches: (1) single simulation, i.e., one full simulation that runs on a single core and simulates S seconds; (2) single MimicNet w/ training, i.e., one end-to-end MimicNet instance, running from scratch; (3) single MimicNet, i.e., one MimicNet instance that reuses an existing model; (4) partitioned simulation, i.e., N full simulations, each simulating S/N seconds; and (5) partitioned MimicNet, i.e., N MimicNet instances, each simulating S/N seconds. N=20 as our machines have 20 cores.

Figure 11 shows the results for network sizes ranging from 8 to 128 clusters. We make the following observations. First, when the network is relatively small, the model training overhead in MimicNet is significant, so 'single MimicNet w/ training' takes longer than 'single simulation' to finish. When the network size reaches 64 clusters, even when training time is included, MimicNet runs faster than any full simulation approach. When the network is as large as 128 clusters, MimicNet is 2-3 orders of magnitude faster than full simulations. The results hold when partitioning, with MimicNet gaining an additional advantage in larger simulations where the removal of the majority of packets/connections introduces substantial gains to the memory footprint of the simulation group.

Simulation throughput: For throughput, we consider a similar set of five approaches. Specifically, the first three are identical to (1)–(3) above, while the last two run for S seconds to maximize throughput: (4) parallel simulation, i.e., N full simulations, each simulating S seconds and (5) parallel MimicNet, i.e., N MimicNet instances, each simulating S seconds.

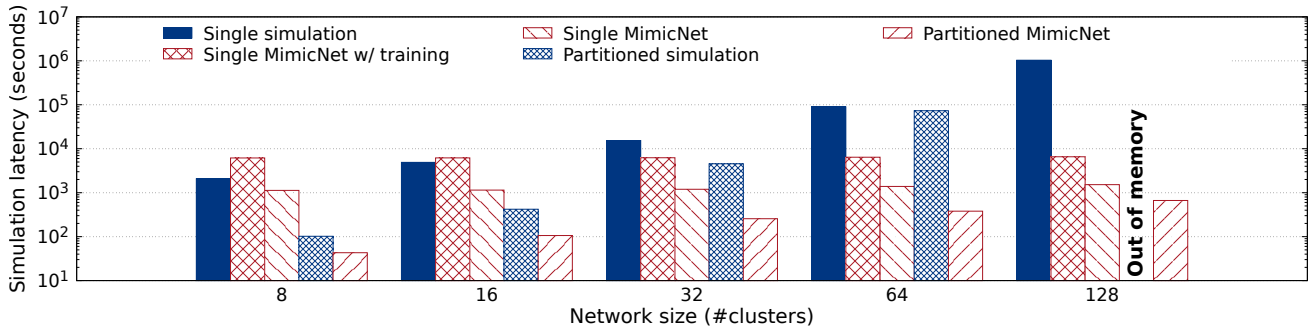


Figure 11: Simulation latency with different network sizes (lower is better).

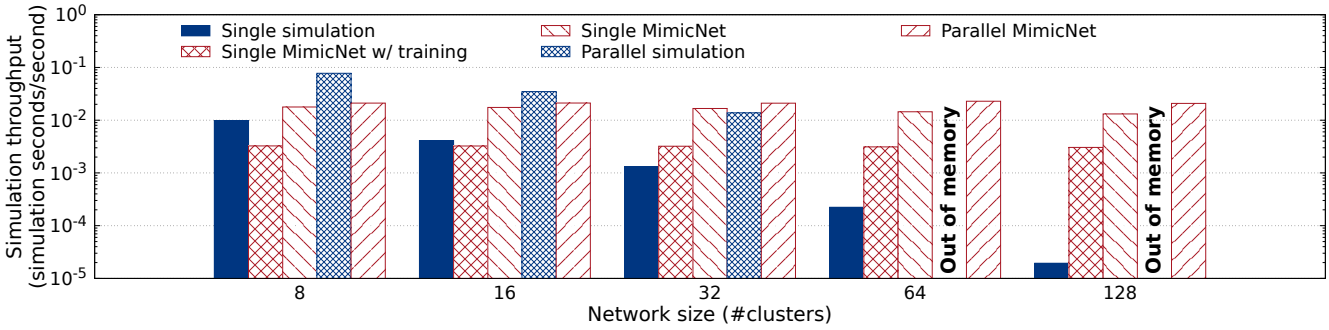


Figure 12: Simulation throughput with different network sizes (higher is better).

Figure 12 shows the throughput results for the range of network sizes. Overall, MimicNet maintains high throughput regardless of the network size because the amount of observable traffic is roughly constant. Single simulation, on the other hand, slows down substantially as the size of the network grows, and at 128 clusters, full simulation is almost *five orders of magnitude slower than the real-time*. As mentioned in Section 2.2, a remedy prescribed by many simulation frameworks is to run multiple instances of the simulation. Our results indeed show that the throughput of parallel simulation compared to single simulation improves by up to a factor of N. When contrasted to the scale-independent throughput of MimicNet, however, a single instance of MimicNet overtakes even parallelized simulation at 32 clusters. Larger parallelized instances begin to suffer from the memory issues described above, but even with unlimited memory, MimicNet would still likely outperform parallelized simulation by 2–3 orders of magnitude at 128 clusters.

9.4 Use Cases

MimicNet can approximate a wide range of protocols and provide actionable insights for each. This section presents two potential use cases: (1) a method of tuning configurations of DCTCP and (2) a performance comparison of several data center network protocols.

9.4.1 *Configuration Tuning*

DCTCP leverages ECN feedback from the network to adjust congestion windows. An important configuration parameter mentioned in the original paper is the ECN marking threshold, K, which influences both the latency and throughput of the protocol.

Essentially, a lower K signals congestion more aggressively ensuring lower latency; however, a K that is too low may underutilize network bandwidth, thus limiting throughput. FCTs are affected by both: short flows benefit from lower latency while long flows favor higher throughput. The optimal K, thus, depends on both the network and workload. Further, a simulation's prescription for K has implications for its feasibility, its latency/throughput comparisons to other protocols, and the range of parameters that an operator might try when deploying to production.

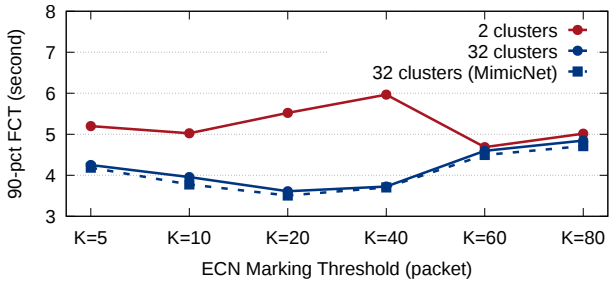
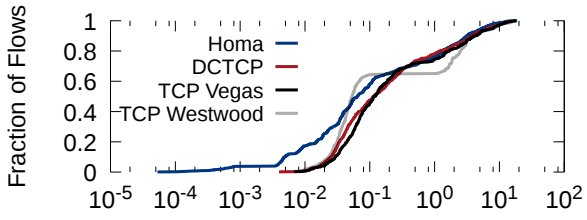


Figure 13: Tuning the marking threshold K in DCTCP: the configuration that achieves the lowest 90-pct FCT is different between 2 clusters ($K = 60$) and 32 clusters ($K = 20$). MimicNet provides the same answer as the full simulation for 32 clusters, but it is 12× faster.

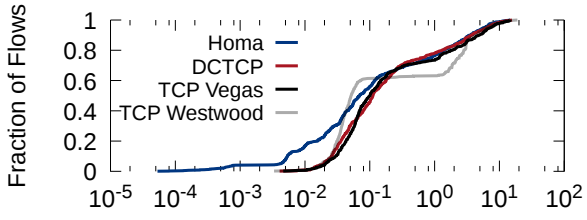
Figure 13 compares the 90-pct FCT for different Ks. Looking only at the small-scale simulation, one may be led to believe that the optimal setting for our workload is $K = 60$. Looking at the larger 32-cluster simulation tells a very different story—one where $K = 60$ is among the worst of configurations tested and $K = 20$ is instead optimal. MimicNet successfully arrives at the correct conclusion.

9.4.2 *Comparing Protocols*

Finally, MimicNet is accurate enough to be used to compare different transport protocols. We implement an additional four such



(a) **Ground truth of the comparison**



(b) **The approximation of MimicNet.**

Figure 14: FCT distributions of Homa, DCTCP, TCP Vegas, and TCP Westwood for a 32-cluster data center.

protocols that each stress MimicNet's modeling in different ways. Homa is a low-latency data center networking protocol that utilizes priority queues—a challenging extra feature for MimicNet as packets can be reordered. TCP Vegas is a delay-based transport protocol that serves as a stand-in for the recent trend of protocols that are very sensitive to small changes in latency [28, 39]. TCP Westwood is a sender-optimized TCP that measures the end-to-end connection rate to maximize throughput and avoid congestion. DCTCP ($K = 20$) uses ECN bits, which add an extra feature and prediction output compared to the other protocols. We run the full MimicNet pipeline for each of the protocols, training separate models. We then compare their performance over the same workload, and we evaluate the accuracy and speed of MimicNet for this comparison. The FCT results are in Figure 14 (other metrics are in Appendix D).

As in the base configuration, for all protocols, MimicNet can match the FCT of the full-fidelity simulation closely. In fact, on average, the approximated 90-pct and 99-pct tails by MimicNet are within 5% of the ground truth. Because of this accuracy, MimicNet performance estimates can be used to gauge the rough relative performance of these protocols. For example, the full simulation shows that the best and the worst protocol for 90-pct of FCT is Homa (3.1 s) and TCP Vegas (4.5 s); MimicNet predicts the correct order with similar values: Homa with 3.3 s and TCP Vegas with 4.6 s. While the exact values may not be identical, MimicNet can predict trends and ballpark comparisons much more accurately than the small-scale baseline. It can arrive at these estimates in a fraction of the time—12× faster.

10 RELATED WORK

Packet-level simulation. As critical tools for networking, simulators have existed for decades [30]. Popular choices include ns-3 [21, 42], OMNeT++ [34], and Mininet [29]. When simulating large networks, existing systems tend to sacrifice one of scalability or granularity. BigHouse, for instance, models data center behavior using traffic drawn from empirically generated distributions and a model of how traffic distributions translate to a set of performance metrics [37]. Our system, in contrast, begins with a faithful reproduction of the target system, providing a more realistic simulation.

Emulators. Another class of tools attempts to build around real components to maintain an additional level of realism [3, 32, 54]. Flexplane [43], for example, passes real, production traffic through models of resource management schemes. Pantheon [56] runs real congestion control algorithms on models of Internet paths. Unfortunately, emulation's dependency on real components often limits the achievable scale. Scalability limitations even impact systems like DIABLO [52], which leverages FPGAs to emulate devices with low cost, but may still require ~$1 million to replicate a large-scale deployment.

Phased deployment. Also related are proposals such as [49, 59] reserve slices of a production network for A/B testing. While showing true at-scale performance, they are infeasible for most researchers.

Preliminary version. Finally, we note that a published preliminary version of this work explored the feasibility of approximating packet-level simulations using deep learning [25]. This paper represents a substantial evolution of that work. Critical advancements include the notion of scale-independent features, end-to-end hyperparameter tuning methods/metrics that promote scalability of accuracy, the addition of feeder models, improved loss function design, and other machine learning optimizations such as discretization. These are in addition to significant improvements to the MimicNet implementation and a substantially deeper exploration of the design/evaluation of MimicNet.

11 CONCLUSION AND FUTURE WORK

This paper presents a system, MimicNet, that enables fast performance estimates of large data center networks. Through judicious use of machine learning and other modeling techniques, MimicNet exhibits super-linear scaling compared to full simulation while retaining high accuracy in replicating observable traffic. While we acknowledge that there is still work to be done in making the process simpler and even more accurate, the design presented here provides a proof of concept for the use of machine learning and problem decomposition for the approximation of large networks.

As part of the future work, we would like to further improve MimicNet's speed with the support of incremental model updates when models need retraining; and its accuracy with models that involve more network events at higher levels such as flow dependencies (details are in Appendix H). More generally, extending its accuracy and speed for the evaluation of more data center protocols and architectures is how MimicNet evolves in the future.

This work does not raise any ethical issues.

ACKNOWLEDGMENTS

We gratefully acknowledge Rishikesh Madabhushi, Chuen Hoa Koh, Lyle Ungar, our shepherd Brent Stephens, and the anonymous SIGCOMM reviewers for all of their help and thoughtful comments. This work was supported in part by Facebook, VMWare, NSF grant CNS-1845749, and DARPA Contract No. HR001117C0047. João Sedoc was partially funded by Microsoft Research Dissertation Grant.

REFERENCES

[1] Opnet network simulator, 2015. https://opnetprojects.com/opnet-network-simulator/.

[2] Hyperopt, 2018. http://hyperopt.github.io/hyperopt/.

[3] M. Al-Fares, R. Kapoor, G. Porter, S. Das, H. Weatherspoon, B. Prabhakar, and A. Vahdat. Netbump: User-extensible active queue management with bumps on the wire. In *2012 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 61–72, Oct 2012.

[4] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[5] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 503–514, New York, NY, USA, 2014. ACM.

[6] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[7] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 253–266, San Jose, CA, 2012. USENIX.

[8] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010.

[9] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. *SIGCOMM Comput. Commun. Rev.*, 24(4):24–35, October 1994.

[10] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, June 2014.

[11] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

[12] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[13] Charles Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 973–978, 2001.

[14] G. Ewing, Krzysztof Pawlikowski, and Donald Mcnickle. Akaroa-2: Exploiting network computing by distributing stochastic simulation. *Proceedings of 13th European Simulation Multiconference, ESM'99*, pages 175–181, 6 1999.

[15] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. A general approach to network configuration analysis. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 469–483, Berkeley, CA, USA, 2015. USENIX Association.

[16] Andrew Frohmader and Hans Volkmer. 1-wasserstein distance on the standard simplex. *CoRR*, abs/1912.04945, 2019.

[17] Richard M. Fujimoto. Parallel discrete event simulation. In *Proceedings of the 21st Winter Simulation Conference, Washington, DC, USA, December 4-6, 1989*, pages 19–28, 1989.

[18] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[19] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 202–215, New York, NY, USA, 2016. ACM.

[20] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 139–152, New York, NY, USA, 2015. ACM.

[21] Thomas R. Henderson, Mathieu Lacage, and George F. Riley. Network simulations with the ns-3 simulator. In *In Sigcomm (Demo*, 2008.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[23] Peter J. Huber. Robust estimation of a location parameter. In *The Annals of Mathematical Statistics*, pages 73–101, 1964.

[24] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, and Changhoon Kim. Eyeq: Practical network performance isolation for the multi-tenant cloud. In *4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 12)*, Boston, MA, June 2012. USENIX Association.

[25] Charles W. Kazer, João Sedoc, Kelvin K. W. Ng, Vincent Liu, and Lyle H. Ungar. 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 2018, Redmond, WA, USA, November 15-16, 2018*, pages 141–147. ACM, 2018.

[26] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: Verifying network-wide invariants in real time. *SIGCOMM Comput. Commun. Rev.*, 42(4):467–472, September 2012.

[27] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russ Clark. Kinetic: Verifiable dynamic network control. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 59–72, Berkeley, CA, USA, 2015. USENIX Association.

[28] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 514–528, New York, NY, USA, 2020. Association for Computing Machinery.

[29] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[30] LBNL. network simulator man page. https://ee.lbl.gov/ns/man.html.

[31] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, Feb 1994.

[32] Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno P. Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. Crystalnet: Faithfully emulating large production networks. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 599–613, New York, NY, USA, 2017. Association for Computing Machinery.

[33] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. F10: A fault-tolerant engineered network, 2013.

[34] OpenSim Ltd. Omnet++, 2018. http://omnetpp.org.

[35] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 290–301, New York, NY, USA, 2011. ACM.

[36] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In Christopher Rose, editor, *MOBICOM 2001, Proceedings of the seventh annual international conference on Mobile computing and networking, Rome, Italy, July 16-21, 2001*, pages 287–297. ACM, 2001.

[37] David Meisner, Junjie Wu, and Thomas F. Wenisch. Bighouse: A simulation infrastructure for data center systems. In *IEEE International Symposium on Performance Analysis of Systems & Software*, 2012.

[38] Vishal Misra, Wei-Bo Gong, and Don Towsley. 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*, SIGCOMM '00, pages 151–160, New York, NY, USA, 2000. ACM.

[39] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 537–550, New York, NY, USA, 2015. ACM.

[40] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 221–235, New York, NY, USA, 2018. Association for Computing Machinery.

[41] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, March 2008.

[42] nsnam. ns-3, 2017. http://nsnam.org.

[43] Amy Ousterhout, Jonathan Perry, Hari Balakrishnan, and Petr Lapukhov. Flexplane: An experimentation platform for resource management in datacenters. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 438–451, Boston, MA, 2017. USENIX Association.

[44] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[45] Dan R. K. Ports, Jialin Li, Vincent Liu, Naveen Kr. Sharma, and Arvind Krishnamurthy. Designing distributed systems using approximate synchrony in data center networks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, 2015.

[46] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath tcp. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 266–277, New York, NY, USA, 2011. ACM.

[47] Robert Ricci, Eric Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX ;login:*, 39(6), December 2014.

[48] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Can the production network be the testbed? In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 365–378, Berkeley, CA, USA, 2010. USENIX Association.

[49] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Can the production network be the testbed? In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 365–378, Berkeley, CA, USA, 2010. USENIX Association.

[50] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 183–197, New York, NY, USA, 2015. ACM.

[51] Mirko Stoffers, Ralf Bettermann, James Gross, and Klaus Wehrle. Enabling distributed simulation of omnet++ inet models. In *Proceedings of the 1st OMNeT++ Community Summit*, 2014.

[52] Zhangxi Tan, Zhenghao Qian, Xi Chen, Krste Asanovic, and David Patterson. Diablo: A warehouse-scale computer network simulator using fpgas. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 207–221, New York, NY, USA, 2015. ACM.

[53] Andras Varga. The omnet++ discrete event simulation system. *Proc. ESM'2001*, 9, 01 2001.

[54] K. V. Vishwanath, D. Gupta, A. Vahdat, and K. Yocum. Modelnet: Towards a datacenter emulation environment. In *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, pages 81–82, Sep. 2009.

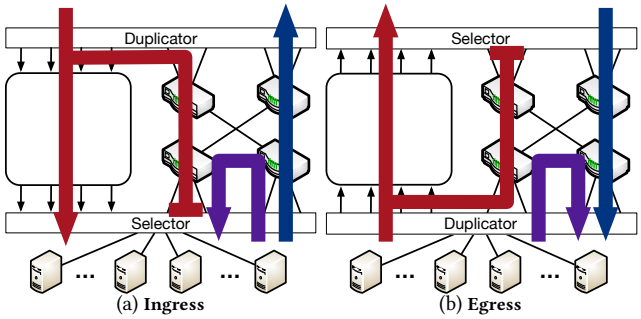
[55] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *NSDI*, 2011.

[56] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, Boston, MA, 2018. USENIX Association.

[57] Nofel Yaseen, Behnaz Arzani, Ryan Beckett, Selim Ciraci, and Vincent Liu. Aragog: Scalable runtime verification of shardable networked systems. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 701–718. USENIX Association, November 2020.

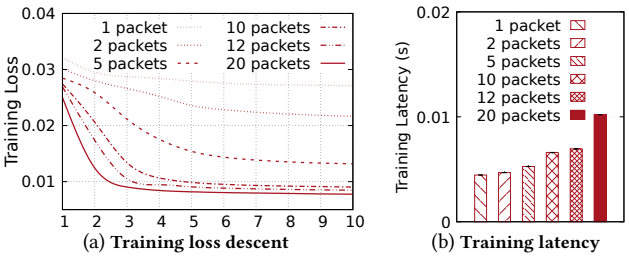
[58] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y. Zhao, and Haitao Zheng. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 479–491, New York, NY, USA, 2015. ACM.

[59] Danyang Zhuo, Qiao Zhang, Xin Yang, and Vincent Liu. Canaries in the network. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016.



(a) **Ingress** (b) **Egress**

Figure 15: Hybrid Mimic clusters for use in separate modeling tuning/debugging. These include both an ML model (white box) and a full-fidelity network. Depending on the model tested, ingress, egress, and internal traffic are routed through the parallel networks. Flat-headed arrows indicate that all traffic of that type is dropped.



(a) **Training loss descent** (b) **Training latency**

Figure 16: The impact of the window size on modeling accuracy and speed. The BDP of the network is around 12 packets. More packets in the window help loss descent (through epochs), but can make the training slower (training latency is per batch in Python).

APPENDIX

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

A RELAXATIONS TO RESTRICTIONS

In Section 4.2, we listed a series of restrictions that MimicNet uses to promote accuracy and speed, even as we scale the simulation by composing increasing numbers of Mimics. We note that not all of the restrictions are necessarily fundamental. In this section, we briefly speculate on possible techniques to relax the restrictions.

Topology and routing. In principle, deep learning models could learn the behavior of arbitrary network topologies, and even incorporate the effects of failures and more exotic routing policies, e.g., those used in optical circuit-switched networks. This would require a unified model instead of the ingress/egress/routing models that we currently use, which may slow down the training and execution of the system. The only piece that would be difficult to relax is the implicit requirement that the network be decomposed in a way that small-scale results are representative of a subset of the larger scale simulation. Random networks, would therefore be challenging for the MimicNet approach; however, heterogeneous but structured networks may be possible, as described below.

Traffic patterns. The expectations of compatible traffic generators in MimicNet are carefully selected, and thus, would be difficult to separate from the MimicNet approach. Certainly, MimicNet could be used on packet traces rather than the synthetic patterns used in this work (by characterizing the trace using a distribution). We also note that it may be possible to relax the symmetry assumption by training distinct models for different types of clusters, e.g., frontend clusters, Hadoop clusters, and storage clusters. More baked-in are the requirements that per-cluster traffic adhere to a consistent distribution regardless of the size of the simulation; however, given that clusters maintain the same capacity, it is reasonable to expect that they maintain similar demand.

Bottleneck locations. The assumption that the most common bottlenecks exist in the downward-facing direction of a packet's path allows MimicNet to elide the modeling of effects like oversubscription coming out of the hosts and core-level congestion from inter-Mimic traffic. These could easily be added back in via similar mechanisms to inter-Mimic modelling, but at additional performance costs.

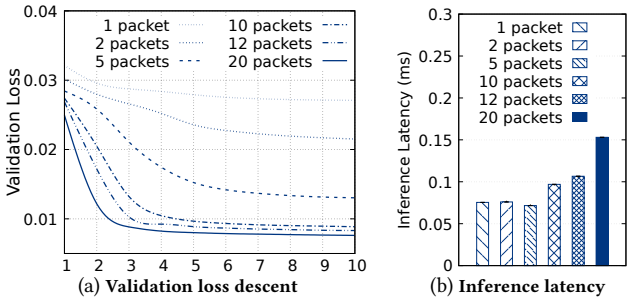
Host-internal isolation. MimicNet's removal of connections from the host is a large source of improved performance as those implementations tend to be more complicated and require more state than even switch queues. Hosts and connections also outnumber, significantly other components in the simulation. Their removal from the network is replaced by MimicNet's constituent models, but the hosts in Mimics actually have fewer connections. The effects of CPU contention could likely be modelled accurately. The effects of out-of-band cooperation between connections, e.g., an RCP-like mechanism running on each hosts, could also potentially be modelled with sufficient domain-expertise. Both would add to the execution time, though the training time could be parallelized.

B SEPARATE INGRESS/EGRESS TUNING

MimicNet, by default, tunes the ingress and egress models together, but in order to tune/debug the ingress model and the egress model separately, and also avoid a quadratic increase in the configuration space that we must explore, we create two separate testing frameworks: one for ingress traffic and one for egress traffic. These frameworks isolate the effect of each direction so that, when training an ingress model, egress traffic travels through a full-fidelity network, and vice versa for an egress model.

The testing frameworks resemble the structure of the small-scale simulation of Section 5.1. Like the original simulation, two clusters are set up to communicate with one another. One is kept at full-fidelity, while the other is converted to use a specialized testing cluster.

See Figure 15 for diagrams of the specialized testing clusters. Isolation of the two directions depends on the model being tested. Consider, for instance, the ingress testing cluster shown in Figure 15a. Traffic ingressing the cluster flows through the model before the hosts receive it, and traffic egressing the cluster flows through the full-fidelity network. Unfortunately, only feeding the egress traffic into the full-fidelity component would result in inaccurate results as egress traffic contends with local traffic, which contends in turn with ingress traffic. In other words, the congestion of the full-fidelity network depends on every packet in the full-fidelity trace. To account for this, we duplicate ingress packets and continue to feed them and local traffic into the full-fidelity cluster,



(a) **Validation loss descent** (b) **Inference latency**

Figure 17: The impact of the window size in the LSTM model on modeling accuracy and speed. Larger window sizes help improve the accuracy (lower validation loss), but can make the inference slower (inference latency is per packet in C++).

dropping them in favor of the model's results when applicable. A similar process occurs when testing an egress model.

A dedicated investigation and evaluation of this function of MimicNet is beyond the scope of this paper.

C THE IMPACT OF MODEL COMPLEXITY

We note that in MimicNet, a significant, domain-specific factor in model complexity is the size of the training window. The window is a number of packets (their features) that we input to the model. This size decides (1) the amount of data that the model learns from one sample, and (2) the hidden size of the LSTM model. Having a larger window helps learning and potentially improves the prediction accuracy, but at the cost of training and inference speed.

Figure 16 shows both of these effects on the training of an ingress model. From Figure 16a, we can see that a window size of only 1 packet performs very poorly, even after several epochs. The training accuracy is quickly improved with additional packets in the window, but this comes with diminishing returns after the window size reaches the BDP of the network (around 12 packets). Figure 16b shows a reverse trend for training time. This suggests that the BDP of the network strikes a good balance between accuracy and speed for the LSTM model.

We also evaluated the impact of the window size on the validation accuracy and the inference speed. Figure 17 shows the result. Specifically, Figure 17a shows that the validation loss resembles the trend of the training loss as shown in Figure 16b. When there is only one packet in the window, the model does not perform well—the validation loss decreases very slowly over ten epochs. Including more packets helps the accuracy: a 2-packet window works significantly better than a 1-packet window, and 5-packet window works better than both. However, when the window size reaches the BDP of the network (~12 packets), having more packets in the window does not improve the accuracy significantly. Figure 17b shows that the model complexity also affects the inference speed. When the window has only a few packets, e.g., 1 packet, 2 packets and 5 packets, the inference latency for a packet is as low as 70 μs. When the window size increases to 10 and 12, the inference latency rises to 100 μs, and with 20 packets, the inference time goes up further to more than 150 μs. This evaluation validates the conclusion in Appendix C: using BDP as the window size strikes a good balance between accuracy and speed for the LSTM model.

D THROUGHPUT AND RTT

Figure 18 shows the comparison for throughput between Homa, DCTCP (with $K = 2$), TCP Vegas, and TCP Westwood in a data center with 32 clusters. Figure 19 shows the results for packet RTTs.

Similar to FCT, MimicNet can closely match the throughput and RTT of a real simulation for all protocols. We can use the estimation of MimicNet to compare these protocols—not only their general trends of throughput and RTT distributions, but also their ranking at specific points. For example, TCP Westwood achieves the best 90 percentile throughput performance due to its optimizations on utilizing network bandwidth; in comparison, DCTCP has the lowest throughput at this particular point. MimicNet successfully predicts the order. The situation in RTT, however, is the opposite: TCP Westwood now has the highest 90 percentile latency, while DCTCP performs the best among these four protocols. This comparison is also correctly predicted by MimicNet.

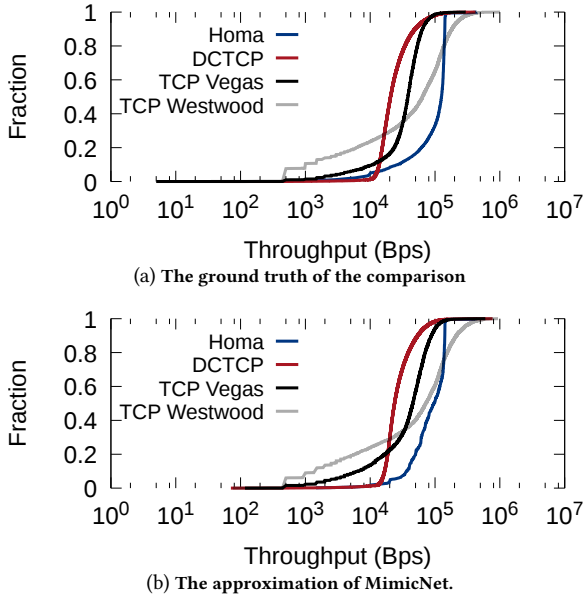
For all protocols, MimicNet estimates are much more accurate than the small-scale (2-cluster) baseline. Again, MimicNet can achieve an order of magnitude higher simulation speed at this scale.

E HEAVIER NETWORK LOADS

In addition to the default network load at 70% bisection bandwidth, we have evaluated the performance of MimicNet with heavier network loads. Figure 20 shows MimicNet's estimation of the FCTs in a network of 32 clusters where the aggregation network load is 90% bisection bandwidth. Similar to previous experiments, MimicNet provides high accuracy in approximating the ground truth: the overall W_1 score is low at 0.15, and the shape is maintained. MimicNet completes the execution 10.4× faster than the full simulation.

F MORE GROUPS OF SIMULATIONS

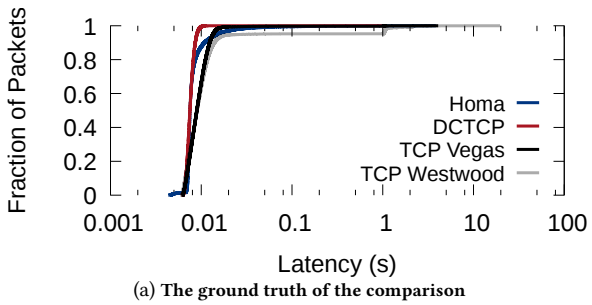
We also ran additional experiments on the latency/throughput of different methods to execute groups of simulations. For these experiments, we fix the network size as 32 clusters and vary the simulation



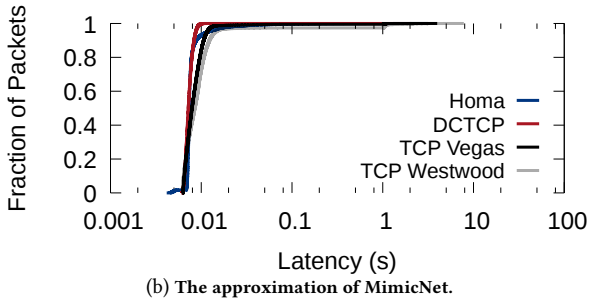
(a) **The ground truth of the comparison**

(b) **The approximation of MimicNet.**

Figure 18: Throughput distributions of Homa, DCTCP, TCP Vegas, and TCP Westwood in 32 clusters.



(a) **The ground truth of the comparison**



(b) **The approximation of MimicNet.**

Figure 19: Packet RTT distributions of Homa, DCTCP, TCP Vegas, and TCP Westwood in 32 clusters.

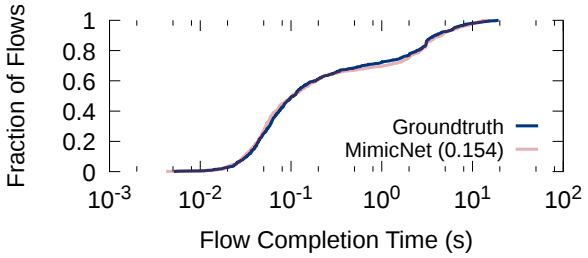


Figure 20: MimicNet approximation for high aggregation network load (90% of the bisection bandwidth).

length from 20 simulation seconds to 320 simulation seconds. Figures 21 and 22 show the simulation latency and throughput results, respectively, for different simulation approaches (we use the same approaches introduced in Section 9.3).

The results are somewhat expected: the relative simulation speeds of different approaches barely change with the simulation length. When simulation length increases, the latency of each approach increases correspondingly. The latency of full simulations increases slightly slower than that of MimicNet because the constant simulation setup overhead in full simulations is significantly higher than MimicNet. The relative latency eventually stabilizes—the latency of single MimicNet is lower than that of single simulation, even when the model training time is included in MimicNet, and partitioned MimicNet is better than partitioned simulation. For all approaches, the simulation throughput does not change at all with the simulation length. Similarly, single MimicNet outperforms single full simulations, and parallel MimicNet outperforms parallel full simulations. The speedup of MimicNet further grows when the simulation scales to larger networks.

G COMPUTE CONSUMPTION

A potential concern in using MimicNet is its compute resource consumption: it uses GPU resources for model training and runtime inference while the full simulations only use CPUs. This section evaluates this aspect.

Specifically, we calculate the total number of floating-point operations (FLOPs) in both CPUs (for both full simulations and MimicNet) and GPUs (for MimicNet only) of the simulation approaches in Section 9.3 as their compute resource consumption. Figure 23 shows the result for the evaluation of latency (similar findings in the evaluation of throughput). Indeed, MimicNet shows significant computational load, primarily because of the use of GPUs for training and inference. This makes its compute consumption higher than full simulations when the network to be simulated is small, especially when the training overhead is counted. However, in large networks, e.g., 128 clusters, the use of deep learning models in MimicNet pays off by much lower simulation latency, and *its total compute consumption is lower than full simulations even with the computational overhead in training models.* We leave the investigation of alternative training, tuning, and models for optimizing the compute consumption to future work.

H FUTURE DIRECTIONS

Finally, we note that the MimicNet framework offers plenty of opportunities for improvement beyond those mentioned in Appendix A. We introduce a small subset of such directions here.

Model reuse and retraining. An important goal in the design of MimicNet is arbitrary scale, which is achieved by its end-to-end workflow (Figure 3) and assumptions described in Section 4. In that spirit, the models that are trained and tuned in MimicNet can be safely reused to evaluate the network at any scale, i.e., no matter how the network scales up or down by adding or removing clusters. Generally, there is no need of retraining the models if the training data and steps in MimicNet workflow do not change. However, if any factor in the data and steps for generating the models changes, the models should be updated to reflect the change. This includes changes in the workload, routing/switching protocol, internal structure of a cluster, and accuracy in the step of hyper-parameter tuning.

Although we have shown that MimicNet runs faster than full simulations even when the model training time is counted (Section 9.3 and Appendix F), we would like to explore techniques that can minimize the overhead of model retraining. This requires considerations in both model design and MimicNet's workflow, for example, whether it is possible or how easily to transfer knowledge between models and how MimicNet supports such incremental model updates. We leave this exploration for future work.

Flow modeling. Recall from Section 6 that MimicNet uses feeder models that currently learn offline flow-level patterns to approximate and remove *non-observable inter-Mimic traffic*. The ordering and dependencies between observable flows are still simulated in full fidelity, i.e., not approximated. That said, we acknowledge that co-flow modeling is currently missing in MimicNet, which can help the accuracy in the evaluation of some real-world systems like MapReduce and BSP-style data processing. In order to support co-flows in MimicNet, they have to be identified when extracting

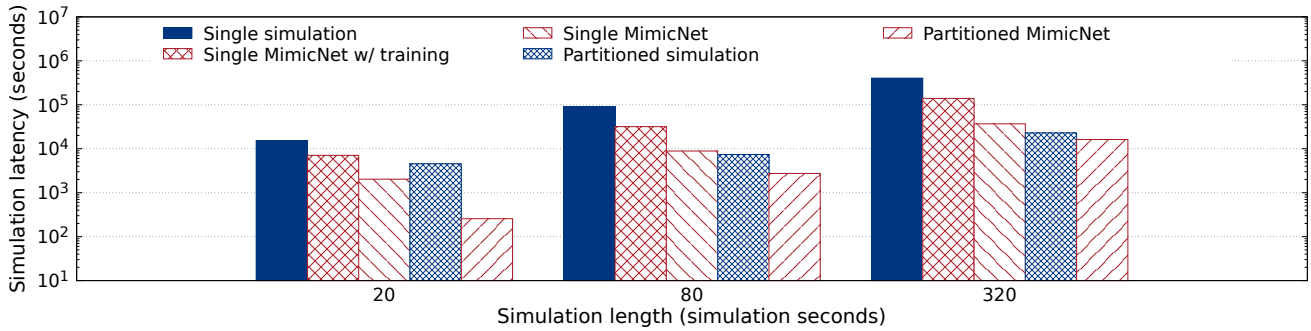


Figure 21: Simulation latency with different simulation lengths (lower is better).

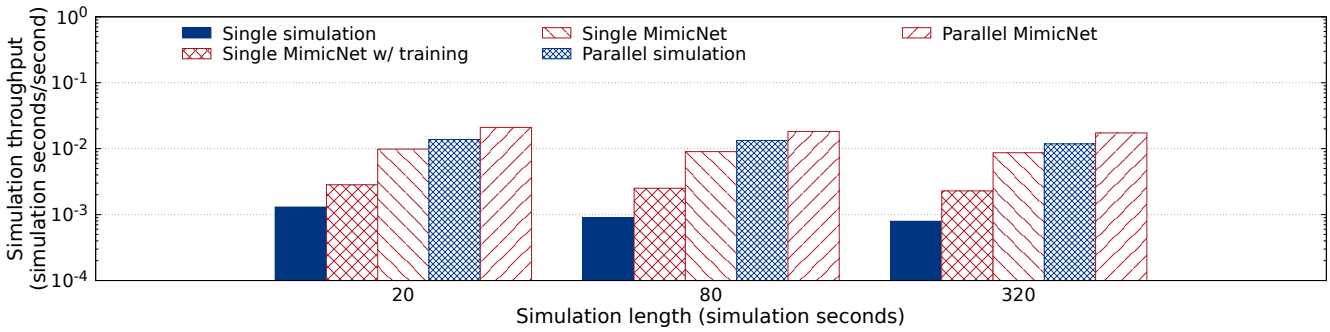


Figure 22: Simulation throughput with different simulation lengths (higher is better).

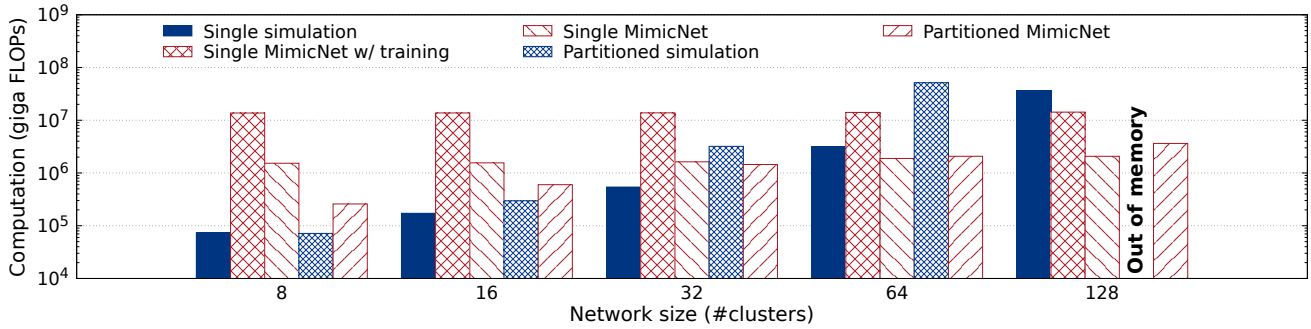


Figure 23: Compute resource consumption in different simulation approaches (lower is better).

features for training the internal models and using them for predictions. We leave enabling the ability of identifying co-flows in MimicNet and studying the benefit in the evaluation of applications where co-flows present for future work.