m3: Accurate Flow-Level Performance Estimation using Machine Learning

Chenning Li*△, Arash Nasr-Esfahany*△, Kevin Zhao⊗, Kimia Noorbakhsh△
Prateesh Goyal□, Mohammad Alizadeh△, Thomas Anderson⊗
△MIT CSAIL, ⊗University of Washington, □Microsoft Research

ABSTRACT

Data center network operators often need accurate estimates of aggregate network performance. Unfortunately, existing methods for estimating aggregate network statistics are either inaccurate or too slow to be practical at the data center scale.

In this paper, we develop and evaluate a scale-free, fast, and accurate model for estimating data center network tail latency performance for a given workload, topology, and network configuration. First, we show that path-level simulations — simulations of traffic that intersects a given path — produce almost the same aggregate statistics as full network-wide packet-level simulations. We use a simple and fast flow-level fluid simulation in a novel way to capture and summarize essential elements of the path workload, including the effect of cross-traffic on flows on that path. We use this coarse simulation as input to a machine-learning model to predict path-level behavior, and run it on a sample of paths to produce accurate network-wide estimates. Our model generalizes over the choice of congestion control (CC) protocol, CC protocol parameters, and routing. Relative to Parsimon, a state-of-the-art system for rapidly estimating aggregate network tail latency, our approach is significantly faster (5.7×), more accurate (45.9% less error), and more robust.

CCS CONCEPTS

• **Networks** → **Network simulations**; **Network performance modeling**.

KEYWORDS

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

ACM Reference Format:
Chenning Li*△, Arash Nasr-Esfahany*△, Kevin Zhao⊗, Kimia Noorbakhsh△, Prateesh Goyal□, Mohammad Alizadeh△, Thomas Anderson⊗, △MIT CSAIL, ⊗University of Washington, □Microsoft Research. 2024. m3: Accurate Flow-Level Performance Estimation using Machine Learning. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3651890.3672243

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672243

1 INTRODUCTION

Network simulation is widely used in the design, planning, and operation of networks. Prominent simulators, e.g., ns-3 [46], OPNET [30], OMNET++ [51], and htsim [22], are packet-level discrete-event simulators. They take every event at each network component (e.g., packet arrival, timer expiration, etc.), serialize them in a single event queue, and process them one by one. As a result, they are inherently slow and cannot keep up with the size and speed of modern networks. Recent work proposes new machine learning techniques (e.g., MimicNet [54], DeepQueueNet [53]) and parallelization strategies (e.g., Parsimon [55], DONS [19]) to accelerate and improve the scalability of traditional simulators. However, these proposals still operate at the packet level. As network speeds continue to increase, packet-level models inevitably become too slow. For example, a single data center switch chip can forward 25 billion packets per second [52], making even the most efficient packet-level simulator much slower than real-time for even a single switch.

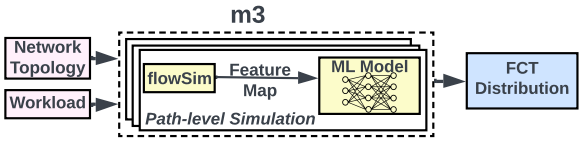
Our goal is to design a performance model that overcomes the limitations of packet-level simulation without sacrificing fidelity. Most network simulations are not used to inspect the behavior of individual packets or even individual flows. In many use cases, a network designer is interested in certain performance metrics (e.g., network throughput, tail latency, flow completion time) and how they are affected by changes in network conditions (e.g., workload characteristics) and various design choices (e.g., congestion control parameters, routing policies, job placement). Rather than simulate every packet interaction, can we *learn* a model that predicts these performance metrics using a higher level of abstraction?

We propose m3, a system that uses machine learning to predict the flow-level performance of a data center network. m3 is trained using ground-truth data from a packet-level simulator such as ns-3.[1] Given a network topology, a workload — specified as a sequence of flows and their network paths — and optionally a set of design parameters (e.g., congestion control knobs), m3 can predict the flow completion time (FCT) *distribution* for a class of traffic, such as the flows in a certain size range, flows sent from certain endpoints, flows traversing certain paths, and so forth.

To understand m3's design, let us consider a packet-level simulator like ns-3 as implementing a function that maps an input workload and a network topology to some performance statistics (Figure 1(a)). Our goal is to learn a fast and accurate approximation of this function from training examples derived from packet-level simulations. Conceptually, this is a supervised learning problem. However, two key challenges make it difficult.

*Equal contribution
[1]The techniques we develop can in theory be used to learn a performance model based on a real network, but we leave this to future work.

(a) Packet-level simulator is a complex function.



(b) m3's proposal

Figure 1: m3's high-level architecture

First, the space of possible workloads and network topologies is vast, and we cannot collect training data for every scenario. We could perhaps consider only certain workloads or topologies during training, but ideally, we want the model to generalize. Retraining for each new scenario may end up being slower than using a packet-level simulator. Moreover, there is a limit to the network scale we can simulate to collect training data. Beyond a few hundred nodes, packet-level simulators take hours to days for each second of simulation time [55]. Training a complex model can easily require hundreds of thousands to millions of examples, so it is impractical for large-scale networks.

Second, it is not clear how to represent the inputs to the model efficiently. The network topology is an arbitrary graph and the workload is an arbitrary sequence of flows (with their arrival times, sizes, and paths). Existing approaches such as using graph neural networks to process network topology information [15, 16, 49] face significant scalability and generalization challenges [12, 20] (a data center network can have hundreds of thousands of nodes and links). Similarly, processing millions of flows using standard sequence models such as Transformers [50] is prohibitively expensive. Simple features such as the traffic load, flow size, and inter-arrival time distributions, cannot capture complex workloads such as non-stationary or correlated traffic patterns (e.g., small flows occurr in bursts, large flows are spread out).

m3 addresses these challenges using two key ideas. First, it decomposes a large-scale network simulation into a set of path-level simulations. Each path-level simulation consists of only those flows that traverse at least one link on a specific path. The flows traversing the entire path are referred to as the foreground traffic, and the other flows sharing a link with the foreground flows are referred to as background traffic. Any flows that interact with background traffic at other network links (not along the path) are ignored. m3's machine learning model is trained to predict the FCT distribution of the foreground traffic in an arbitrary path-level simulation. To estimate network-wide behavior, m3 samples several paths and combines their predictions to derive the network-wide FCT distribution.

Our use of path-level decomposition is inspired by Parsimon [55], which proposed to approximate a large-scale network simulation via independent link-level simulations that can be executed in parallel. Path-level decomposition is more accurate than link-level decomposition (since it captures interactions between links along a path), and our experiments show that it provides an accurate approximation of network-wide performance for real-world data center workloads and topologies. Using path-level scenarios as the building block for network-wide performance estimation also greatly simplifies m3's learning task. We only need to collect training data for path scenarios, which is scalable since even large data center networks have a modest maximum path length. Providing topological information to the model is also straightforward using a sequence of features associated with each link along the path.

m3's second key idea is to use a fast *flow-level* simulator to extract rich workload-related features suited to FCT performance prediction. Given a path-level scenario consisting of sequences of foreground and background flows, m3 first runs *flowSim*, a simple simulator that assigns flows their max-min fair rate allocations at each point in time and computes the flow completion times. It then extracts a *feature map* of FCT statistics for flows of different sizes, which serve as the primary input to the machine learning model (Figure 1(b)). flowSim is extremely fast, e.g., it simulates 800K flows on a path in around 1 second (687× faster than ns-3). However, bandwidth sharing models [33] such as max-min fairness only provide a coarse approximation of the behavior of congestion-controlled flows. Such models are particularly inaccurate for short flows since they do not capture queuing dynamics and latency. Nevertheless, we show that flowSim's FCT statistics are excellent features for predicting the network's true behavior. The feature map derived from flowSim is sensitive to many important aspects of the workload, such as the volume, burstiness, and size characteristics of the flows.

We train m3 using a diverse mix of synthetically generated path scenarios. These synthetic scenarios capture the complex dynamics of network workloads including flow size variations, burstiness levels, congestion control protocols, and maximum link load conditions, all within "parking-lot" topologies of 2 to 6 hops. In the evaluation, we validate m3 against production workloads and actual network topologies. A primary metric we use is the estimation error of the p99 FCT slowdown — the ratio of the flow completion time for different flow sizes, normalized to the ideal flow completion time for that flow size on an unloaded network, at the 99th-percentile. We also present FCT distributions for different flow sizes, which can be used to derive alternate metrics such as packet latency and average long flow throughput. We summarize our evaluation results below.

- Given a diverse mix of production workloads on a 32-rack, 256-host fat tree topology, m3 delivers an average speedup of 5.7× in simulation time over Parsimon [55], alongside better accuracy in p99 FCT slowdown estimation. m3 demonstrates mean estimation errors of 9.89%, compared to Parsimon's 18.29%.
- On a larger scale 384-rack, 6144-host fat-tree topology, m3 completes the simulation in 40 seconds, compared to 1 minute and 24 seconds for Parsimon and 11.9 hours for ns-3, with a notable reduction in estimation error from 11.9% (with Parsimon) to 5.74%.
- m3 can adapt to a variety of workloads, topologies, and network conditions. Even when trained on scenarios with varied congestion control settings, m3 accurately forecasts tail FCT slowdown for new, unseen parameters, highlighting its capacity for effective counterfactual analysis.

m3's code is available at https://github.com/netiken/m3. This work does not raise any ethical issues.

2 INSIGHTS

In this section, we use data from ns-3 to motivate our model's use of path-level decomposition and workload featurization.

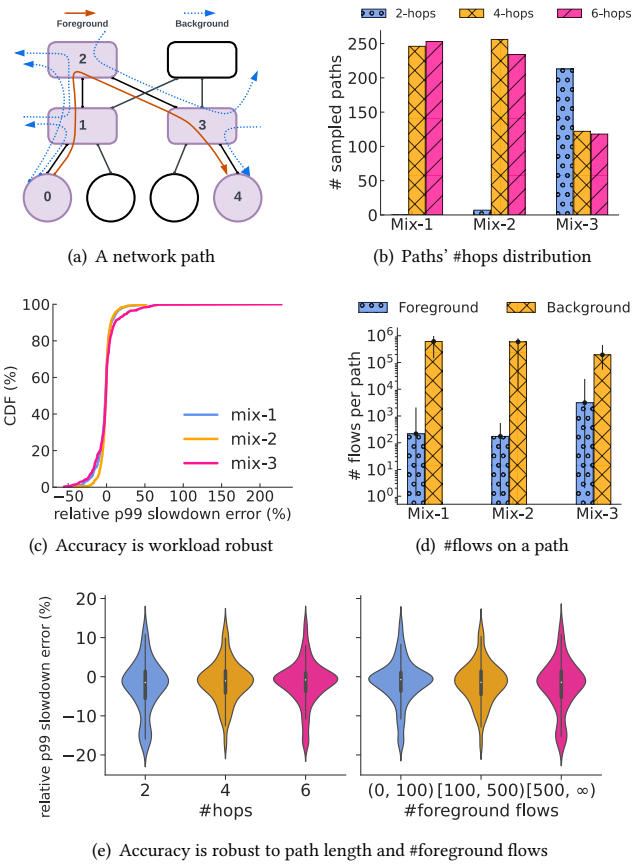
2.1 Path-level Decomposition

Modern hyperscalar data center networks can be enormously large, with hundreds of thousands of servers and network links and thousands of network switches. With network core and server link speeds continuing to increase exponentially, accurately simulating network behavior at scale with a packet-switched simulator is a daunting task. In recent work on Parsimon [55], Zhao et al. suggest decomposing the network into a set of independent queues representing each link, and then simulating the traffic traversing each queue in parallel. If each queue experiences congestion independently and transiently, the per-queue results can be combined to approximate aggregate network behavior. However, this approximation breaks down with higher utilization, higher levels of oversubscription, and workloads with correlated endpoint behavior. In the recent work on Mimicnet [54], Zhang et al. use machine learning to train a generative model of the impact of clusters, or subsets, of the network on other clusters. This allows fast, small scale cluster-level simulations to be generalized to larger scale systems. However, this work is limited to FatTree topology [3] with uniform traffic among equal-sized clusters of machines.

Our work is inspired by these earlier efforts but aims to work at scale for general workloads and topologies, without implicit assumptions about traffic independence or topological regularity. While Mimicnet showed that it is possible to train a model on a specific topology, it is hard to envision how to train a model of an entire network in a way that is topology independent, so that it produces accurate aggregate performance even when we remove or add a link or switch, or upgrade a portion of the network [38], or use optical switching to dynamically change core link capacities [45].

Instead, we set ourselves a simpler problem. We decompose the network into a set of paths; each path is a sequence of links and switches connecting a source node with some destination, as illustrated in Figure 2(a). A large scale data center network may have billions of such paths; there may be hundreds of paths even between the same source and destination node. Paths can be of varying length (in a data center setting, they typically have an even number of hops) with varying link capacities and traffic. We call the traffic from the path's source to its destination foreground traffic; background traffic intersects the foreground traffic over at least one hop. Importantly, the number of possible configurations and workloads for individual paths is vastly smaller than that for arbitrary networks, making the challenge of building an accurate model tractable.

We make a simplifying assumption, that the performance of foreground traffic is primarily determined by the latency, capacity, and scheduling policies of the links along the path, along with the characteristics of the foreground and background traffic. In other words, we assume that flows that do not intersect a path do not significantly affect the behavior of foreground traffic. This is of course an approximation. For example, the presence of upstream bottlenecks can smooth cross-traffic, affecting its interaction with the foreground flows. However, it is a weaker assumption than some prior work, such as Parsimon which assumes independence of individual queues, rather than individual paths [55].



(a) A network path (b) Paths' #hops distribution

(c) Accuracy is workload robust (d) #flows on a path

(e) Accuracy is robust to path length and #foreground flows

Figure 2: (a) Illustration of foreground and background flows on a path in a fat-tree network topology. (b) Distribution of hop counts on sampled paths for different workloads. (c) Accuracy of path-level ns-3 relative to full-network simulation for tail (99^{th} percentile) slowdown. (d) Number of foreground and background flows on sampled paths for different workloads. (e) Path-level ns3's error distribution as a function of path length and number of foreground flows. Violin plots depict the distribution of errors, with wider areas signifying a higher density of errors at that value. The center box within the violin captures the middle half of the data (25th to 75th percentile).

To validate this approximation, we use ns-3 to simulate three scenarios, where each scenario is a simulation on a 32-rack, 256-host topology with different traffic matrices and flow sizes drawn from production workloads, along with different maximum link load and oversubscription levels (Table 1) (setup details in §5.1). For each scenario, we simulate 10 million flows with Equal-Cost Multi-Path (ECMP) routing using ns-3. We randomly sample 500 paths with the probability proportional to the number of foreground flows they carry, with replacement. This selection is further explained in §3.2. The distribution of hop counts of these paths is shown

Scenario	#Flows	Traffic	Max load	Workload	Oversub	ns-3		Parsimon		ns-3-path	
						p99 sldn	time	p99 sldn	time	p99 sldn	time
Mix 1	10M	Mat A	42.46%	CacheFollower	4-to-1	4.565	41.70h	5.023	345s	4.527	11.50h
Mix 2		Mat B	28.46%	WebServer	1-to-1	4.602	9.648h	4.893	65s	4.504	1.781h
Mix 3		Mat C	73.83%	WebServer	2-to-1	13.891	8.064h	15.24	40s	13.07	0.566h

Table 1: Comparison of the 99th-percentile flow completion time (FCT) slowdown (sldn) and computation times for 10 million flows for different simulation methods, workload, and oversubscription scenarios. Configuration is the same as Section 5.2.

in Figure 2(b). For each selected path, we simulate its foreground and background flows, again using ns-3, but excluding the flows that do not intersect that path. We call this approach *ns-3-path*. When we compare the per-path results from ns-3 with ns-3-path in Figures 2(c) and 2(e), we find that this approach has high accuracy and is robust to different scenarios, hop counts, and the ratio of foreground to background flows. We then aggregate the flow completion time slowdown across the 500 sampled paths from ns-3-path and compare that against the network-wide aggregate statistics from ns-3. Table 1 shows the p99 tail latency slowdown of both methods across the three sample scenarios. ns-3-path has an average p99 slowdown estimation error of only 2%. However, like ns-3, ns-3-path is slow. Since ns-3-path must simulate all flows intersecting the foreground traffic at the packet level, its runtime is nearly the same as the full ns-3 simulation. Parsimon is much faster but less accurate than ns-3-path.

2.2 Workload Featurization

Another key aspect of our approach is to use flow-level simulation to quickly characterize and summarize path-level workload information as input to a machine learning model. Even when we narrow our focus to an individual path, there are hundreds of thousands of flows and millions of packets intersecting and affecting the performance of foreground traffic on the path. The path level workload is a long sequence of foreground and background flow arrival times and sizes. Even if we were to try to use that data to train a model, it is not clear how to featurize the workload [5] and represent it as input to a model in a way that generalizes to a sufficiently large space of workloads. Simple features such as flow size and inter-arrival time distributions are plausible choices but insufficient. For example, marginal distributions of the flow size and inter-arrival time cannot represent the joint distribution of flow size and inter-arrival times, e.g., whether we have bursts of large or small flows. This approach also cannot model non-stationary or diurnal arrival patterns, something that is trivial in ns-3.

We observe that a max-min flow-level simulation [37] can capture much of what we are interested in with respect to workload characterization. Rather than use hand-crafted features based on statistical properties of the workload, we simulate the workload in a flow-level simulator and summarize the attained performance characteristics in a compact feature map. Our hypothesis is that the performance observed in flow-level simulation provides good features for characterizing the relevant properties of the workload. To test this hypothesis, we built a fast max-min flow-level simulator called flowSim (Algorithm 1 in Appendix A); flowSim assumes "fluid" flows that proceed at a uniform rate defined by the max-min fair-share rate given the other flows along the path. A flow's rate is recalculated after the arrival or completion of any competing flow. The flow completes when its rate consumes the flow size, plus a topology-specific end-to-end latency factor.

For characterizing traffic along a path, flowSim offers a number of benefits:

- It operates at the flow-level abstraction, and its computational complexity increases based on the combined number of foreground and background flows along a given path.
- Unlike ns-3 which must model switch queueing, packet marking/dropping, and endpoint congestion control, flowSim involves only basic calculations that are fast and easy to use.
- Although it does not model queuing effects, latency interactions, or the impact of congestion control protocols, and as we show later it is not accurate for small flows (Figure 6), flow-level simulation creates a rich representation capturing the bandwidth interaction of flows.

To illustrate this workload representation, Figure 3 shows the flow completion time (FCT) slowdown computed by flowSim for a single link. The heatmap shows the FCT slowdown for flows of each bucket size (y-axis), using percentile buckets (x-axis) to capture the FCT slowdown distribution. Thus, the right hand side of each heatmap shows the 99th-percentile tail latency for each flow size; the left its 1-percentile latency. All heatmaps in the middle column use the CacheFollower size distribution, a burstiness level of $\sigma = 1.5$, and a maximum link load of 50% (these parameters are further explained in §5.1). In the first row, traffic burstiness increases from left to right. As evident by the figure, increasing the burstiness increases the tail slowdown for small flows and almost all slowdown percentiles for large flows. The second row shows the impact of increasing load. This has an effect similar to burstiness, but if we look closer, the effect of increasing burstiness is more skewed across different size buckets. The third row shows the impact of workload; despite running at the same max link load and burstiness level, different workloads induce different FCT slowdown distributions. This simple example illustrates the effectiveness of using flow slowdown statistics under max-min flow-level simulation for featurizing the workload. FCT slowdown distributions (across different flow sizes) are a compact representation sensitive to many aspects of the workload, allowing a machine learning model to pick up on the differences between workloads with distinct behavior and produce accurate slowdown estimates.

3 SYSTEM ARCHITECTURE OF M3

m3 uses machine learning to predict flow performance distributions in data center networks. Its efficiency and generality are supported

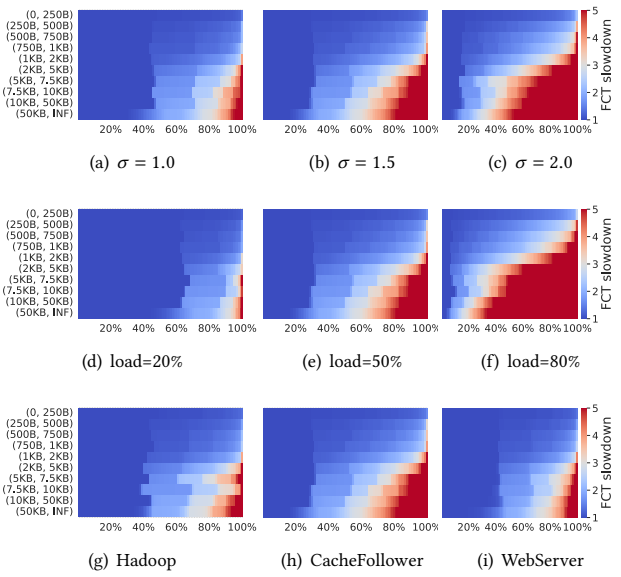


Figure 3: Distribution of flow completion time (FCT) slowdown (x-axis) computed by flowSim for a single link simulation for different flow size buckets (y-axis). The baseline workload is given by the middle column: CacheFollower size distribution, burstiness level of $\sigma = 1.5$, and maximum link load of 50%. Each row varies a single dimension of the workload.

by two key ideas: 1) decomposing large networks into independent paths and 2) extracting rich workload features with flow-level simulation. This section describes how m3 implements these ideas.

3.1 High-Level Overview

Figure 4 illustrates m3 's architecture. Given the traffic workload and the network topology, m3 first decomposes ❶ the network topology into independent paths and, for each path, identifies all foreground and background flows. To reduce the number of paths that must be simulated, m3 uses weighted sampling to select a representative sample (§3.2). The sampled paths are used for path-level simulations ❷, which, owing to their independence, can be executed in parallel. Each path-level simulation uses an efficient max-min fair sharing algorithm [33, 37] called flowSim ❸ to compute initial FCT slowdown estimates, separately for foreground and background traffic. These estimates are then translated into a feature map ❹ used as input to a machine learning model (§3.3). To account for different network configurations such as the choice of congestion control protocol and bandwidth-delay product, the feature map is combined with network specifications ❺. Then, m3 uses machine learning to refine its predictions of FCT slowdowns for foreground traffic to match the ground truth ❻ from ns-3-path (§2.1), factoring in the dynamics of the foreground and background traffic, queueing delays, and the congestion control protocol (§3.4). The above process is carried out once for each sampled path (in parallel). Once all results are obtained, m3 aggregates them ❼ into

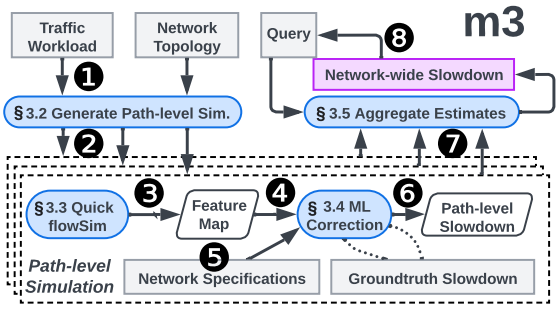


Figure 4: m3's workflow: Inputs (grey boxes), outputs (purple boxes), intermediate artifacts (parallelograms), and core components (rounded boxes).

network-wide performance metrics (§3.5). Lastly, m3 offers an interactive user interface ❽, supporting targeted queries that can enhance network management decisions.

3.2 Generating Path-Level Simulations

We begin by specifying the path-level simulation, which consists of a workload and a topology.

Path-Level Specification. Given a full network topology and a set of flows, m3 uses the flows' routes to associate each link with the flows traversing it, assuming static routes known in advance.[2] A path is a sequence of links, and its *path-level workload* consists of all flows that traverse any link in the path. The flows' arrival times and sizes are unmodified. We distinguish between *foreground* flows, which traverse the entire path, and *background* flows, which only intersect the path at one or more (but not all) hops (Figure 2(a)). More precisely, suppose $P = (l_1, l_2, \ldots, l_n)$ is a path that consists of n links, let $\mathcal{F}$ be the set of all flows, and let $\texttt{traverses}(f, l)$ be a predicate which is true when a flow $f \in \mathcal{F}$ traverses a link $l \in P$. The set of foreground flows F for path P is

$$F \triangleq \{f \in \mathcal{F} \mid \forall l \in P : \texttt{traverses}(f, l)\}, \tag{1}$$

and the set of background flows B is

$$B \triangleq \{f \in \mathcal{F} \mid f \notin F \ \wedge \ \exists l \in P : \texttt{traverses}(f, l)\}. \tag{2}$$

The goal of path-level simulations is to predict the performance of foreground flows in F given background flows in B (context), for later downstream processing. §3.4 describes how these outputs are formatted and used.

Each path also has a *path-level topology* which contains only the nodes and links on the path, as well as whatever other nodes and links are needed to support the background traffic. For brevity, we refer to the links on the path as *original* links, and all other links as *synthetic* links. Conceptually, a path-level topology is a parking lot topology like the one shown in Figure 7(a). In this figure, the original links are the ones connecting purple nodes, and all others are synthetic. To avoid introducing artificial contention among background flows, each background flow connects to the point where it joins/exits the foreground path with a bandwidth equal to its source/destination capacity.

[2]This assumption does not hold in case of packet-spraying [9] or flowlets [1].

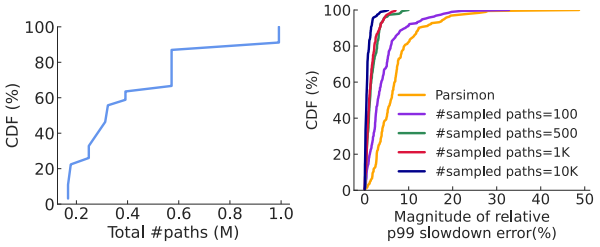


Figure 5: (Left) Distribution of the number of active paths across 192 workloads on a 32-rack, 256-host fat-tree topology; (Right) sampling error distribution shrinks quickly when increasing the number of sampled paths.

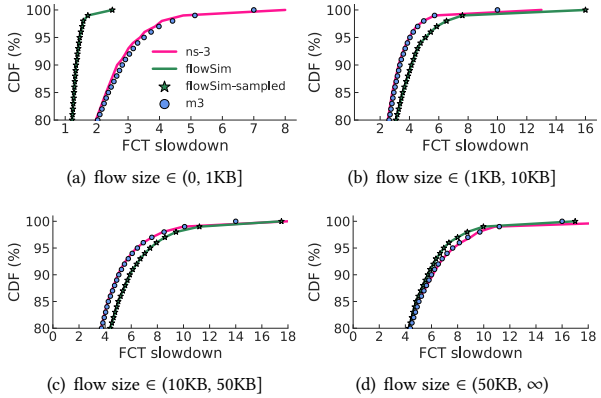


Figure 6: Distribution of FCT slowdown for different flow size buckets from ns-3, flowSim, and m3 on a 4-hop parking-lot topology

Weighted Path Sampling. m3's path-level decomposition presents an additional challenge: the number of paths grows rapidly with network size. Figure 5(a) shows a CDF of the number of populated paths when simulating 192 different workloads on a 32-rack, 256-host topology (see §5.1). Even on small topologies, the number of populated paths can be in the hundreds of thousands, and it is prohibitively expensive to simulate each one. To reduce the number of simulated paths, we use a weighted sampling strategy wherein the probability of sampling a path P is proportional to the number of foreground flows on P, with replacement (a popular path may appear in the sample more than once).

To investigate the sensitivity of aggregate slowdown to the number of sampled paths, we first run 192 different scenarios in ns-3. Then, for each scenario, we sample different numbers of paths using the strategy described above. For each set of sampled paths, we aggregate the foreground flows and compute the p99 FCT slowdown. We then compare the p99 slowdown of the sampled paths to the p99 slowdown of the entire network to derive a relative error. Figure 5(b) shows the cumulative distribution function (CDF) of the relative p99 slowdown error for different path sample sizes. We observe that sampling 100 paths is enough to exceed Parsimon's [55] accuracy; sampling 500 paths bounds the relative p99 slowdown error to within 10%.

3.3 Quick Estimation via flowSim

To produce initial FCT estimates for the path-level topologies, m3 uses a simple simulator, which we call flowSim, that assigns flows their max-min fair rate allocation [33, 37] at each point in time. Appendix A has the implementation details.

Figure 6 shows that flowSim provides good estimates of FCT slowdown for large flows exceeding 10KB since the performance of DCTCP (the congestion control protocol used in these experiments) is reasonably modeled as bandwidth sharing for large flows. However, flowSim underestimates the FCTs of short flows, especially in the tail of the distribution, because it does not model queueing dynamics. The next section describes how we use machine learning to reduce this error.

3.4 Improving Estimates with Machine Learning

m3 uses flowSim's initial estimates to create feature maps as input to a machine learning model. The foreground estimates are refined by machine learning, incorporating the dynamics of queueing and congestion control, while the background estimates are used as context to help the model produce accurate predictions for the performance of foreground traffic.

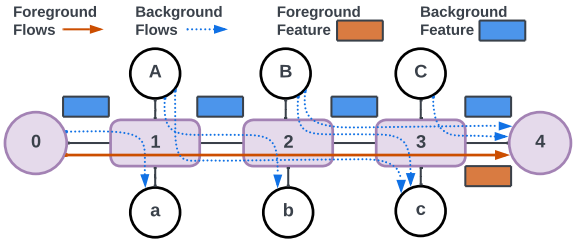
Deriving Feature Maps from flowSim's FCT Slowdown. flowSim estimates FCT slowdowns for all flows in the path-level workload, both foreground and background. The number of background flows can be very large, as shown in Figure 2(d). We wish to refine these estimates with machine learning, but what should the features be? Processing large numbers of flows directly using standard sequence models such as Transformers [50] is prohibitively expensive. On the other hand, statistical features like traffic volume, mean flow size, and inter-arrival times may not capture enough workload dynamics, as discussed in §2.2.

To balance efficiency against fidelity, m3 converts flowSim's estimates into concise feature maps, as shown in Figure 7(a). Given a path $P = (l_1, l_2, \ldots, l_n)$ with n links and a set of foreground flows F (the red solid line) the feature map M is:

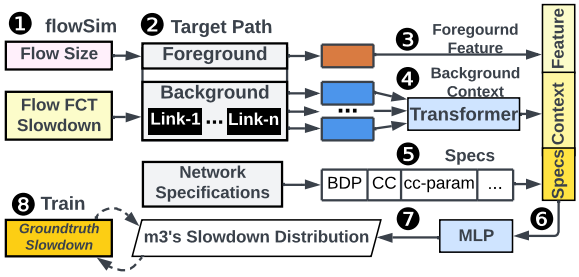
$$M^F_{s,p} = \{\text{Sldn}(f) \mid f \in F,\ \text{size}(f) \in \text{bucket}_s,\ \text{percentile}(f) = p\} \quad (3)$$

where m3 first categorizes foreground flows into s buckets based on the size of each flow. Within each bucket s, m3 records the slowdown (Sldn) predicted by flowSim across p fixed percentiles. The final feature map has dimension $s \times p$, represented by the orange rectangle. By default, the feature map has 10 flow size buckets, ranging from flows with a single packet under 250B to flows exceeding 50KB, as shown in Figure 3. Additionally, it includes 100 fixed percentiles, ranging from 1% to 100% in 1% increments.

The performance of the foreground flows is also affected by the amount and character of the background traffic (shown as blue dotted lines). For each link along the foreground path, m3 creates a similar feature map (flowSim computed FCT slowdown of dimension $s \times p$) for the background flows traversing that link.



(a) m3 uses path-level simulations with flowSim to generate compact foreground and background features for its ML model.



(b) m3's ML model predicts FCT slowdown distribution for the network configurations of interest using flowSim generated features.

Figure 7: Design of path-level m3

This yields n contextual feature maps (represented by blue rectangles) $\{M_{s,p}^{B^{l_1}}, \ldots, M_{s,p}^{B^{l_n}}\}$, one for each hop in an n-hop path $P = (l_1, l_2, \ldots, l_n)$.

Refining flowSim's FCT Slowdown Estimations. Figure 7(b) shows how m3 refines flowSim's FCT slowdown estimates: ❶ Starting with flow sizes and their associated slowdown estimates, ❷ m3 transforms the FCT slowdowns into $n+1$ structured feature maps $(M_{s,p}^{F};\ \{M_{s,p}^{B^{l_1}}, \ldots, M_{s,p}^{B^{l_n}}\})$, corresponding to both foreground and background traffic along the n-hop path. ❸ The feature map $M_{s,p}^{F}$ is then flattened to serve as a feature for foreground flows F. ❹ m3 feeds the sequence of n background feature maps $\{M_{s,p}^{B^{l_1}}, \ldots, M_{s,p}^{B^{l_n}}\}$ into a generic sequence model (small Llama-2 [50]) to generate a fixed-length vector that we call *background context*. The only reason we use a transformer is its ability to process a variable number of inputs (one feature map per hop, representing the competing background traffic). We did not try other sequence model architectures or tune hyper-parameters, although they could improve our results. ❺ An additional input to the model is the foreground path specification, such as the bandwidth-delay product (BDP) , congestion control protocol used (e.g., DCTCP [2], TIMELY [36], DCQCN [56]), and parameters for those protocols. We show that m3 generalizes across the space of those parameters.

❻ The combined foreground feature, background context, and network specifications are then fed into a two-layer multilayer perceptron (MLP) model to predict the final slowdown distribution of foreground flows for this path. ❼ Responding to user-defined queries, m3 generates the foreground FCT slowdown at specific percentiles for designated flow size buckets. For example, the default

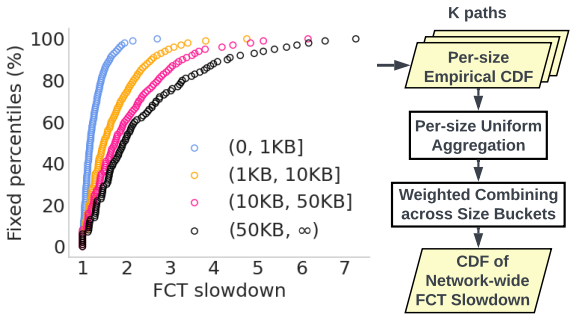


Figure 8: Aggregating FCT slowdown at different size buckets from k path-level simulations into an empirical CDF for network-wide FCT slowdown analysis.

output has four size buckets for (0, 1KB], (1KB, 10KB], (10KB, 50KB], (50KB, ∞). Each bucket has the corrected FCT slowdown at 100 fixed percentiles, spanning from 1% to 100% in 1% increments. ❽ In training, m3 optimizes its transformer and DNN using L1 loss for all 100 fixed percentiles to align it with the user-provided ground truth, such as FCT slowdown data from ns-3. In future work, we hope to test the model's ability in learning the slowdown distribution of real networks with different configurations and live application demand.

Our results suggest these features sufficiently capture network's dynamics for effective prediction of foreground FCT slowdown distribution for various flow sizes. Figure 6 compares the corrected FCT slowdowns at specific percentiles (represented by blue dots) against the original estimates of flowSim (represented by green stars) for a 4-hop path topology and Meta's workloads (details in §5.1). m3 is able to accurately adjust flowSim's FCT slowdowns across various flow size buckets, even for tail slowdowns of short flows.

3.5 Estimating Network-Wide Slowdown

Carrying out the above for k sampled paths results in k size-bucketed FCT slowdown distributions as shown in Figure 8. What remains is to combine the k path-level results into a network-wide set of size-bucketed distributions, and then, optionally, to further combine the distributions in each bucket into a single FCT slowdown distribution.

Figure 8 illustrates how this is done. First, recall from §3.2 that the k paths already constitute a flow-count-weighted random sample of the entire network. Therefore, to combine them into a single set of buckets in a manner that respects workload volume, we only need to aggregate them uniformly. Second, m3 combines the distributions from each bucket into a single distribution via probabilistic sampling, where the probability of sampling a particular bucket is proportional to the number of flows in that bucket. Because performance on each path is different, some paths may contribute more (or less) than their share to the aggregate tail latency at a given percentile. Thus, averaging the buckets at a given percentile across all paths will not produce accurate statistics for the network-wide performance at that same percentile.

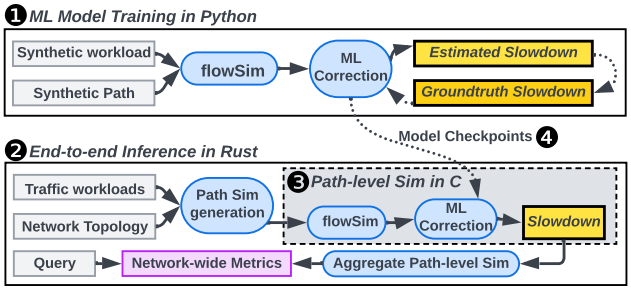


Figure 9: m3's implementation

3.6 What m3 does and what it does not

m3 predicts FCT slowdown distributions for individual paths, making its predictions topology-agnostic. This also enables sampling from specific paths of interest. However, m3 assumes static routes known in advance, associating each link with the traversing flows. This assumption breaks in the case of dynamic routing strategies like packet-spraying [9] or flowlets [1]. Furthermore, m3 abstracts detailed per-flow information by converting complex workloads into a compact feature map through flowSim and predicts the *distribution* of FCT slowdown, not the performance of specific flows. Moreover, m3 encodes the CC algorithms as a one-hot vector in its features fed into the MLP model. Hence, it cannot predict the performance of new CC protocols not seen during training. m3's current implementation also does not model the effects of priority classes; we leave this for future work.

In addition to the FCT slowdown distribution, m3's output captures the network's throughput and latency performance. For example, the distribution of FCT slowdown for short flows indicates packet latency and queueing delay, and the FCT of medium to long flows captures throughput effects. However, we note that our evaluation is based on FCT slowdown distribution; we leave for future work on how to adapt m3 to predict other metrics such as packet loss rates.

4 IMPLEMENTATION

Figure 9 depicts m3's main components:

- **ML Model Training** (1): m3 uses the PyTorch Lightning framework for distributed training to speed up the training processing. We train the transformer and the MLP from scratch. Our training runs for 400 epochs on four A100 GPUs with four workers launched on each. Each worker processes batches of 20 data samples. The entire training process is completed in two days, with each epoch taking roughly 7 minutes. 4 m3's model checkpoints include a 66.5MB transformer and a 4.9MB MLP.
- **End-to-End Inference** (2): m3's inference pipeline is written in 4300 lines of Rust and C. The Rust component exposes the top-level interface and implements path decomposition, parallel execution, and aggregation. 3 Path level computations including flowSim, feature map extraction, and ML inference are written in C. Inference code runs on CPU and is optimized for speed, facilitating interactive network performance querying and design exploration. We used a single machine with dual AMD EPYC 7763 64-core processors (256 CPUs and 512GB RAM in total) for inference in all our experiments.

Parameter	Sample space
#Foreground flows	20000
Flow size distribution	Pareto, Exp, Gaussian, Log-normal
Size parameter (θ)	5k (small) to 50k (large), continuous
Burstiness parameter (σ)	1 (low) to 2 (high), continuous
Max load	20% to 80%, continuous
Path length	2 hops, 4 hops, 6 hops
Network configuration	See Table 4

Table 2: Training Set Parameters

Parameter	Sample space
#Flows	10M
Oversubscription	1-to-1, 2-to-1, 4-to-1
Traffic matrix	A, B, C (See Figure 18(a))
Flow size distribution	CacheFollower, WebServer, Hadoop
Burstiness	Low ($\sigma = 1$), High ($\sigma = 2$)
Max load	26% to 83% (continuous range)
Fat-tree topology	Small (256-host), Large (6144-host)
Network configuration	See Table 4

Table 3: Test Set Parameters

Parameter	Sample space
Init window	5 to 30KB, continuous
Buffer size	200 to 500KB, continuous
PFC Flag	0 (disabled), 1 (enabled)
CC protocol	DCTCP, TIMELY, DCQCN, HPCC
DCTCP (K)	5 to 20KB, continuous
DCQCN (K_{min}, K_{max})	(20 to 50KB, 50 to 100KB)
HPCC (η, $Rate_{AI}$)	(0.70 to 0.95, 500 to 1000 Mbps)
TIMELY (T_{low}, T_{high})	(40 to 60μs, 100 to 150μs)

Table 4: Network Configuration Parameters

5 EVALUATION

We evaluate m3 using three criteria:

- Generalization across workloads and topologies (§5.2)
- Scalability for large-scale network topologies (§5.3)
- Counterfactual search for network parameter exploration (§5.4)

Further experiments (§5.5) demonstrate sources of error, and ablate the impact of design choices.

5.1 Setup

Training Dataset. We train m3 on a synthetic dataset of 120,000 parking lot topology (single path) ns-3 simulations. To generate this dataset, we select 2000 workload parameters randomly from Table 2. For each workload, we pick 20 random network configurations from Table 4, and use all the 3 path lengths in Table 2. We leave out 10% of the data points randomly for validation. Generated flows are divided uniformly at random among all source-destination pairs. We train m3 once and show its performance in §5.2, §5.3, and §5.4.

ML model. m3 uses a tiny [14] version of Llama-2 [50] to process flowSim feature maps for background flows and generate context features. This sequence model has 4 layers and 4 attention heads with an embedding size of 576 and a block size of 16, resulting in approximately 16.8 million parameters. Its output is a vector with 576 elements. m3 also uses a two-layer MLP with a hidden size of 512 to predict the slowdown distribution given foreground features, background context, and the network configuration of interest.

To generate flowSim feature maps, we partition flow sizes into 10 consecutive size buckets, ranging from less than 250 bytes to over 50KB. For flows in every size bucket, we extract slowdowns from flowSim and convert it to a 100-dimensional vector of percentiles from 1% to 100%, in 1% steps. We further stack vectors for all size buckets. This creates a feature map with a dimensionality of 10×100, offering a detailed and extensive view of flowSim's slowdown profile. m3 outputs the same percentile range for four flow size buckets, from less than 1KB to over 50KB. The output is a vector with 400 elements.

Real-world Test Set. We use Meta's traffic matrices [48], covering diverse clusters like databases (CacheFollower), web servers (WebServer), and Hadoop. Traffic within these matrices is rack-to-rack, with random intra-rack host selection. Flow size distributions come from the same study (Figure 18(b)). For inter-arrival times, we use log-normal distribution with two burstiness levels. For low burstiness, we select log-normal shape parameter $\sigma = 1$, and for high burstiness, we choose $\sigma = 2$. Load level is picked randomly such that no link exceeds its capacity. Tables 3 and 4 summarize the test set.

Network Topology. We evaluate m3's performance using two different fat-tree network topologies. We use a large-scale 384-rack, 6144-host fat-tree topology to evaluate m3's scalability in §5.3. This topology is based on Meta's data center fabric design [48], featuring layers of switches with hosts linked via 10 Gbps connections to top-of-rack (ToR) switches and higher-tier connections at 40 Gbps. Due to the high computational complexity of running ns-3 for gathering ground-truth data in this large setup, we scale down the topology and workload to fit a 32-rack, 256-host topology for extensive experiments in §5.2 and §5.4.

Baseline and Performance Metrics. We compare m3's performance with Parsimon [55], a state-of-the-art fast simulator, using the ns-3 simulator as ground truth. The primary performance metric is relative p99 slowdown estimation error defined as follows:

$$\frac{\text{estimated slowdown} - \text{ground-truth slowdown}}{\text{ground-truth slowdown}} \quad (4)$$

We drop the sign and use the magnitude when reporting median or average. We also record the wall clock running time of each scheme for a speed comparison.

5.2 Sensitivity Analysis

Setting. To assess m3's adaptability to workloads and topologies, we use the small-scale topology described in §5.1. It consists of two pods with 16 racks each and eight hosts per rack, with variable spine counts to reflect different oversubscription levels. We randomly sample 192 scenarios that use DCTCP[3] from Table 3 to create our test set. We show the impact of different protocols and their parameters in §5.4.

[3]Parsimon's fast implementation in Rust only supports DCTCP.

(a) p99 slowdown error distribution (b) m3 is robust to load variation

(c) Running time distribution (d) Speed vs. workload

Figure 10: m3 is faster, more accurate, and robust than Parsimon. Shaded areas represent confidence intervals for the median.

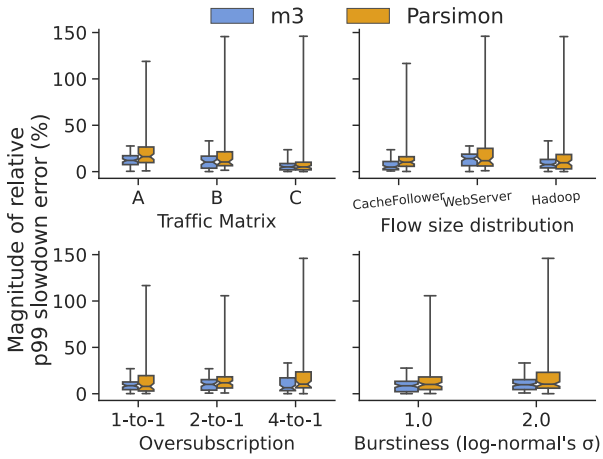


Figure 11: Sensitivity of p99 slowdown error distribution to workload parameters. Each boxplot depicts the distribution of relative p99 slowdown error for a configuration, with the center box capturing the middle 50% (between the 25th and 75th percentiles) and a center line marking the median. Whiskers extend outwards to encompass the remaining data.

Accuracy and Workload Robustness. Figure 10(a) shows the distribution of p99 FCT slowdown estimation errors across the test set for m3 and Parsimon. m3 achieves average relative p99 slowdown error of 9.9%, outperforming Parsimon's 18.3%. Notably, m3 maintained superior performance at the tail with a maximum p99 error of 33.2%, compared to Parsimon's 146%. Figure 10(b) illustrates the median of error in estimating p99 slowdown for different maximum link load buckets. While Parsimon's error and its variance increase at loads above 50%, m3's accuracy remains stable, exhibiting a consistent median error of about 8% throughout the

Init. Window	Methods	p99	Error	Time	Speedup
10KB	ns-3	2.05	-	13.5h	-
	Parsimon	4.29	+109%	1m29s	546×
	m3	2.10	+2.44%	37s	1314×
18KB	ns-3	2.44	-	11.9h	-
	Parsimon	2.73	+11.9%	1m24s	510×
	m3	2.30	-5.74%	40s	1071×

Table 5: Comparison of m3, Parsimon, and ns-3 in terms of p99 FCT slowdown and runtime in large-scale simulations.

load spectrum. Error variance for m3 increases modestly for loads above 50%, but less than Parsimon. Further analysis in Figure 11 depicts m3's robustness against variations in traffic matrix, flow size distribution, oversubscription, and burstiness. m3 suffers slightly for traffic matrix C since it has the most skewed traffic, resulting in many paths with less than 10 flows deviating from our training distribution. In contrast, Parsimon exhibits a more pronounced and skewed estimation error pattern when dealing with traffic matrix A, the flow size distribution of WebServer, an oversubscription ratio of 4-to-1, and burstier workloads ($\sigma = 2.0$).

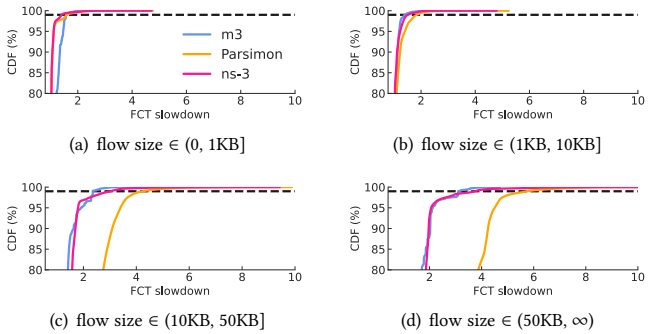
Runtime. The wall clock time for running simulations is demonstrated in Figure 10(c). Despite its better accuracy, m3 is 4-8× faster in end-to-end runtime compared to Parsimon on the same topology and workload. m3 has an average runtime of 36.4 seconds, while Parsimon and ns-3 take 3 minutes 27 seconds and nearly 40.5 hours on average, respectively. Figure 10(d) further indicates that flow size distribution does not affect m3's runtime, as its execution time depends only on the number of flows. However, the runtime of a discrete-event packet-level simulator like Parsimon depends on the number of packet-level events and therefore is affected by the flow size distribution. In other words, Parsimon is relatively slower for workloads with more packets per flow.

5.3 Scalability to Large Topologies and High Loads

Setup. To evaluate m3's scalability, we use the large-scale topology with 384 racks and 6,144 hosts [48] described in §5.1. We use traffic matrix B and a 2-to-1 oversubscription ratio in the core network. The network manages 11.4 million flows, achieving a maximum link load of 50%. We use the WebServer workload and set the traffic burstiness to a high level ($\sigma = 2$). The maximum Bandwidth-Delay Product (BDP) is 15KB. We use two different initial congestion window sizes, 10KB (smaller than maximum BDP) and 18KB (larger than maximum BDP).

Quantitative Results: Table 5 highlights the performance of m3, Parsimon, and ns-3 in terms of p99 FCT slowdown and simulation running time. Notably, m3 significantly accelerates simulation, achieves up to 1314× speedup over ns-3, and reduces simulation time from tens of hours to as low as 37 seconds. Regarding accuracy, m3 achieves a relative p99 FCT slowdown error of 2.44% for the 10KB initial congestion window size, compared to Parsimon's 109% error. In the case of 18KB initial congestion window size, m3 has an estimation error of -5.74%, while Parsimon's error is +11.9%.

Comparative Insight: Figure 12 shows the FCT slowdown distributions from m3, Parsimon, and ns-3 under 10KB initial window



(a) flow size ∈ (0, 1KB] (b) flow size ∈ (1KB, 10KB]

(c) flow size ∈ (10KB, 50KB] (d) flow size ∈ (50KB, ∞)

Figure 12: FCT slowdown estimated by m3, Parsimon, and ns-3 in a large-scale network simulation with 50% max link load and 10KB initial window. The horizontal dashed line shows the 99th percentile.

size. m3's estimation is close to ns-3 across different flow size buckets, especially for the tail. In contrast, Parsimon overestimates the FCT slowdown for large flows, resulting in a p99 FCT slowdown of 4.29, twice as large as ns-3's 2.05 (Table 5). Parsimon decomposes the network simulation into independent link-level simulations and aggregates the link-level results along a path. For the smaller 10KB initial window size, the initial window size becomes a bottleneck for flows larger than 10KB in each link-level simulation. Parsimon adds the slowdowns incurred in the link-level simulations, and thus it over-counts the effect of the window size on the delay. Essentially, Parsimon assumes that the slowdown in every link-level simulation is due to congestion at that link. But when the bottleneck is the transport itself (e.g., a small initial window), summing the slowdowns for links along a path is incorrect. In contrast, m3 learns the effect of the initial window size correctly from the ground-truth path-level simulation data.

5.4 Counterfactual Search for Design Exploration

As a case study to demonstrate m3's utility for quickly exploring the space of network design parameters via counterfactual search, we evaluate m3's ability in predicting the impact of changing HPCC [28]'s initial congestion window size and η (parameter controlling the tradeoff between utilization and transient queue length) on p99 FCT slowdown for different flow classes. We use the 32-rack, 256-host small network topology for this experiment. Flow size distribution is WebServer, traffic matrix is C, max link load is 50%, PFC is enabled, and buffer size is 400KB.

First, we fix η to 90% and sweep the range of initial congestion window sizes in Figure 13. As the figure shows, m3's p99 slowdown predictions are close to ns-3, and capture the trends. For example, it correctly predicts that increasing the congestion initial window size hurts the performance of small flows. Notably, m3 takes only 25.2 seconds to explore the effect of window size, whereas the same experiment takes 8 hours with ns-3. As a result, m3 enables live configuration exploration, opening new avenues for tuning data center network parameters in response to changes in workloads (a topic we leave to future work). Next, we fix the initial congestion window size to 20KB, and sweep η in Figure 14. Again, m3 is able

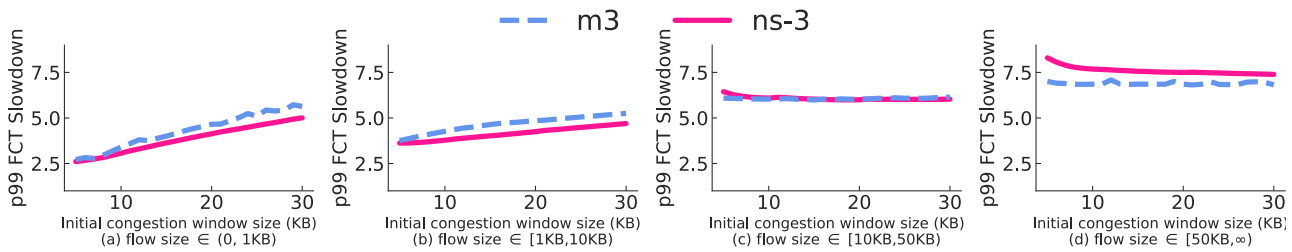


Figure 13: m3 accurately predicts the effect of changing the initial congestion window size on p99 FCT slowdown for different classes of flows, much faster (1139×) than ns-3.

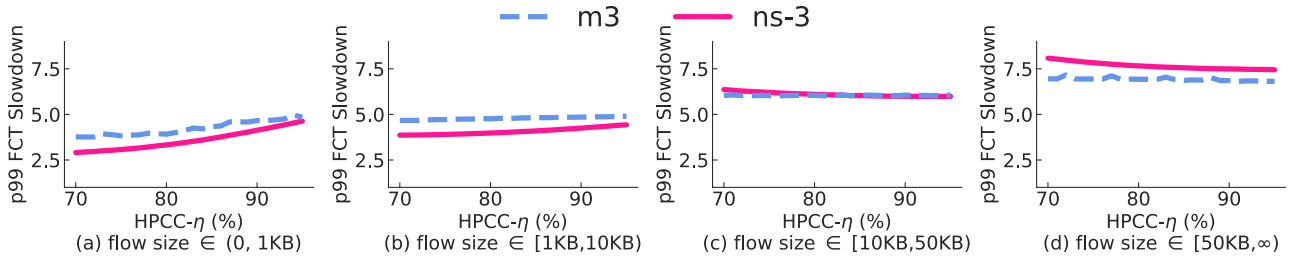


Figure 14: m3 accurately predicts the effect of changing CC parameters (HPCC's η) on p99 FCT slowdown for different classes of flows, much faster (763×) than ns-3.

to correctly capture the effect of η on p99 FCT slowdown, while having an average speedup of 763× compared to ns-3.

5.5 Ablation Study

Here, we ablate m3's design choices. m3's sources of estimation errors are twofold: First, it decomposes full networks into independent path-level simulations, ignoring the effect of any traffic not intersecting the path. Second, it approximates the path-level simulation with flowSim and machine learning. To measure the effect of ignoring traffic that does not intersect a path, we use ns-3-path defined in §2. It shows what the error is if the simulator is perfect (ns-3), but we ignore the effect of traffic not intersecting paths on its foreground flows. We estimate the slowdown of paths' foreground flows in the small-scale 32-rack, 256-host fat-tree topology with ns-3-path, m3, and Parsimon. Figure 15 shows that the assumption we made (ignoring traffic that does not intersect a path) accounts for less than half of m3's error, and more than half of the error is coming from approximation with machine learning. Furthermore, Parsimon's assumption of link independence is strictly worse than m3's assumption across all flow size buckets and path lengths.

We further evaluate the necessity of m3's components (including background contexts as input to the model, and using an ML model) for estimating the FCT slowdown of our building block, a parking-lot topology (a single path). If we don't use an ML model, we are left with the outputs of flowSim. If we do use the ML model but do not include context features from background flows in its input, we have a crippled version of the model that we call m3 without context. Figure 16 displays the distribution of the p99 FCT slowdown for flowSim, m3 w/o context, and the full implementation of m3 for synthetic workloads described in Table 2. flowSim underestimates slowdowns in general, particularly for smaller flows

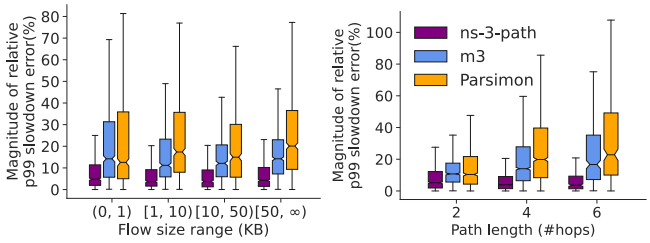


Figure 15: Error breakdown for paths' foreground flows in the small-scale 32-rack 256-host fat-tree topology.

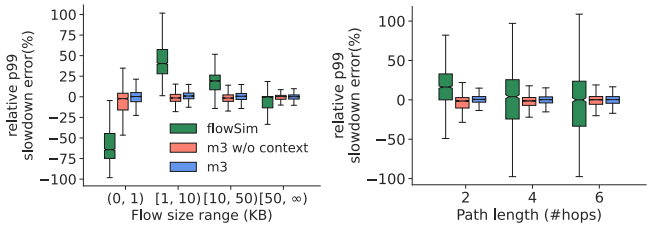


Figure 16: m3 components (machine learning model and background features) are both necessary for its path-level accuracy.

and on longer paths, resulting in errors as large as -80%. m3 corrects flowSim's estimation. Using context features improves m3's accuracy by about 33% on average, and significantly decreases variance. The observation is consistent across varying path lengths and flow sizes.

6 RELATED WORK

We organize the large literature on performance modeling for computer networks into three groups: (*i*) queueing theory (§6.1), (*ii*) flow-level methods (§6.2), and (*iii*) packet-level methods (§6.3). Like m3, various researchers have leveraged machine learning methods to compensate for some of the limitations of each approach; we discuss those in each section.

6.1 Queueing Theory

Queueing theory models networks as system of queues with arrival and service time processes [8, 47]. Closed-form results are possible under certain assumptions, such as Poisson processes with single packet flows. These assumptions are generally too simplistic for networks with endpoint congestion control, bursty arrivals, and arbitrary flow sizes. MQL [39, 40] uses per queue-type regression trees to learn to correct for systematic bias in latency estimates from queueing theory, for all queues that a packet traverses. Although correcting for systematic bias has similarities to our workload featurization technique, MQL is inherently less expressive since it assumes single packet flows and uses as inputs the average flow arrival rate and coefficient of variation. These are sufficient in the case of generalized exponential processes, but not for more general networks. More expressive models like Markovian arrival process [4, 10] can produce accurate estimates; however, this results in a huge state space that is computationally complex and scales poorly. Nevertheless, they have use-cases in performance modeling [21, 25, 26, 31, 34, 42, 44].

6.2 Flow-level granularity

Network Calculus [11, 27] models worst-case metric bounds using min-plus and max-plus algebras. However, it cannot estimate the mean or any percentile. As we have seen, max-min flow approximations like flowSim can accurately model the performance of long flows, but fall short when asked to estimate the performance of short and medium-size flows where queue dynamics dominate. Fluid-based approaches [2, 6, 32, 35, 43] can correct for this by modeling the evolution of flows using partial differential equations (PDEs). However, they require a high level of expertise to define PDEs describing system dynamics for every new system, and can only model the average behavior of a stochastic system [13]. The Routenet line of work [15, 16, 49] uses graph neural networks with flow-level inputs to predict performance metrics. However, their flow-level features, e.g., mean rate or pre-defined parameters for simple processes, are not expressive enough to capture complex workloads. Furthermore, they have challenges in generalizing across topologies, link capacities, and path lengths [12, 20]. As with MQL, QT-Routenet [12] uses predictions of queuing theory techniques assuming Poisson arrivals as inputs to the graph neural network, and has many of the same limitations.

6.3 Packet-level granularity

The most popular tools for estimating network performance model the network behavior down to the granularity of individual packet arrivals and departures from every switch. Examples include ns-3 [46], OPNET [30], and OMNET++ [51]. Although widely used by practitioners and researchers, their main issue is performance for networks of data center scale.

It has been difficult to get significant speedup [23, 29, 41] using standard parallelization techniques [17, 18], leading to slower performance than single-threaded simulation in some cases [46, 54]. Recently, DONS [19] and Parsimon [55] showed significant speedups for packet-level simulation. DONS uses a data-oriented design to improve multi-core, cache, and memory efficiency of precise packet-level simulation, achieving a speedup of 65× on a cluster of CPU-based servers. m3, on the other hand, does approximate flow-level performance prediction and achieves a speedup of about 1300×. Parsimon assumes that simultaneous congestion events at multiple bottleneck links are second-order effects. This assumption enables reasoning about links independently, leading to speedup gains, as we have seen at some cost in accuracy.

Inspired by a workshop paper [24], a line of research uses machine learning to speed up packet-level simulation. Mimicnet [54] uses a traditional packet-level simulation of a cluster in a data center to learn the behavior of a cluster of machines; exploiting symmetries in FatTree topology [3] with uniform traffic among equal-sized clusters of machines, it composes "mimics" to model the behavior of the network. DeepQueueNet [53] uses packet-level simulation to train a model of the packet-level behavior of every network component, that is, every link and switch, using RNNsearch [7]. It achieves a speedup of about 70× using four V100 GPUs, compared to traditional packet-level simulation. However, m3 trains a model of path behavior that runs on CPUs for inference and achieves a speedup of about 1300×.

7 CONCLUSION

We presented m3, a fast and accurate model for estimating aggregate flow-level statistics for data center networks under different workloads and configuration choices. The model is novel in several aspects. First, it is path-based, in that it approximates aggregate network-wide performance by considering only the traffic that intersects with a given path. Second, it uses a max-min flow-level simulator to quickly summarize and featurize the broad space of possible workload characteristics that can affect path-level performance. Feature maps for the foreground and background traffic are combined with topology and configuration options such as the choice of TCP congestion control protocol, protocol parameters such as initial window size, and link capacity and latency. These inputs are then used to train the model on synthetically generated input workloads, and tested against more realistic workloads taken from industry standard benchmarks. Our experiments show that m3 outperforms prior estimation approaches in execution speed, prediction accuracy, and generalization capabilities.

ACKNOWLEDGEMENT

We thank our shepherd Kai Chen and anonymous reviewers whose comments helped improve the paper. We also thank Myungjin Lee and Ratul Mahajan for their comments on an early draft of this paper. This work was supported by NSF Career Award #1751009, DARPA FastNICs program under contract #HR0011-20-C-0089, and grants from Cisco Research and the UW Center for the Future of Cloud Infrastructure.

REFERENCES

[1] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: distributed congestion-aware load balancing for datacenters. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 503–514.

[2] Mohammad Alizadeh, Adel Javanmard, and Balaji Prabhakar. 2011. Analysis of DCTCP: stability, convergence, and fairness. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*. 73–84.

[3] Marina Alonso, Salvador Coll, Juan-Miguel Martínez, Vicente Santonja, and Pedro López. 2015. Power consumption management in fat-tree interconnection networks. *Parallel computing* 48 (2015), 59–80.

[4] Søren Asmussen and Ger Koole. 1993. Marked point processes as limits of Markovian arrival streams. *Journal of Applied Probability* 30, 2 (1993), 365–372.

[5] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. 2020. On the complexity of traffic traces and implications. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* (2020), 47–48.

[6] François Baccelli and Dohy Hong. 2003. Flow level simulation of large IP networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*. 1911–1921.

[7] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[8] Gunter Bolch, Stefan Greiner, Hermann De Meer, and Kishor S Trivedi. 2006. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, Ltd. 821–868 pages. https://doi.org/10.1002/0471791571.biblio

[9] Jiaxin Cao, Rui Xia, Pengkun Yang, Chuanxiong Guo, Guohan Lu, Lihua Yuan, Yixin Zheng, Haitao Wu, Yongqiang Xiong, and Dave Maltz. 2013. Per-packet load-balanced, low-latency routing for clos-based data center networks. In *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies*. 49–60.

[10] Srinivas R. Chakravarthy. 2011. *Markovian Arrival Processes*. John Wiley & Sons, Ltd. https://doi.org/10.1002/9780470400531.eorms0499

[11] Florin Ciucu and Jens Schmitt. 2012. Perspectives on network calculus: no free lunch, but still good value. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 311–322.

[12] Bruno Klaus de Aquino Afonso and Lilian Berton. 2022. QT-Routenet: Improved GNN generalization to larger 5G networks by fine-tuning predictions from queueing theory. *ITU Journal on Future and Evolving Technologies* 3, 2 (2022), 134–141. https://doi.org/10.52953/fbrb3688

[13] Do Young Eun. 2005. On the limitation of fluid-based approach for Internet congestion control. In *Proceedings of the International Conference On Computer Communications and Networks, ICCCN*. 463–468.

[14] Facebookresearch. Retrieved by Feb 1st 2024. Inference code for LLaMA models. In *https://github.com/facebookresearch/llama/blob/main/llama/model.py*.

[15] Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Krzysztof Rusek, Shihan Xiao, Xiang Shi, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2023. RouteNet-Fermi: Network Modeling With Graph Neural Networks. *IEEE/ACM Transactions on Networking* 31, 6 (2023), 3080–3095.

[16] Miquel Ferriol-Galmés, Krzysztof Rusek, José Suárez-Varela, Shihan Xiao, Xiang Shi, Xiangle Cheng, Bo Wu, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2022. RouteNet-Erlang: A Graph Neural Network for Network Performance Evaluation. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*. 2018–2027.

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

[18] Richard M Fujimoto. 2001. Parallel and distributed simulation systems. In *Proceeding of the Winter Simulation Conference*. 147–157 vol.1.

[19] Kaihui Gao, Li Chen, Dan Li, Vincent Liu, Xizheng Wang, Ran Zhang, and Lu Lu. 2023. DONS: Fast and Affordable Discrete Event Network Simulation with Automatic Parallelization. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 167–181.

[20] Martin Happ, Jia Lei Du, Matthias Herlich, Christian Maier, Peter Dorfinger, and José Suárez-Varela. 2022. Exploring the Limitations of Current Graph Neural Networks for Network Modeling. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium*. 1–8.

[21] Gábor Horváth, B Van Houdt, and M Telek. 2014. Commuting matrices in the queue length and sojourn time analysis of MAP/MAP/1 queues. *Stochastic Models* 30, 4 (2014), 554–575.

[22] Broadcom Inc. Retrieved by Feb 1st 2024. htsim Network Simulator. In *https://github.com/Broadcom/csg-htsim*.

[23] Shafagh Jafer, Qi Liu, and Gabriel Wainer. 2013. Synchronization methods in parallel and distributed discrete-event simulation. *Simulation Modelling Practice and Theory* 30 (2013), 54–73. https://doi.org/10.1016/j.simpat.2012.08.003

[24] Charles W. Kazer, Jo ao Sedoc, Kelvin K.W. Ng, Vincent Liu, and Lyle H. Ungar. 2018. Fast Network Simulation Through Approximation or: How Blind Men Can Describe Elephants. In *Proceedings of the ACM Workshop on Hot Topics in Networks*. 141–147.

[25] Abbas Eslami Kiasari, Zhonghai Lu, and Axel Jantsch. 2013. An Analytical Latency Model for Networks-on-Chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21, 1 (2013), 113–123. https://doi.org/10.1109/TVLSI.2011.2178620

[26] Alexander Klemm, Christoph Lindemann, and Marco Lohmann. 2003. Modeling IP traffic using the batch Markovian arrival process. *Performance Evaluation* 54, 2 (2003), 149–173. https://doi.org/10.1016/S0166-5316(03)00067-1

[27] Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag, Berlin, Heidelberg.

[28] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 44–58.

[29] Yu Liu, Boleslaw K. Szymanski, and Adnan Saifee. 2006. Genesis: A scalable distributed system for large-scale parallel network simulation. *Computer Networks* 50, 12 (2006), 2028–2053. https://doi.org/10.1016/j.comnet.2005.10.002

[30] Zheng Lu and Hongji Yang. 2012. *Unlocking the power of OPNET modeler*. Cambridge University Press. 1–238 pages. https://doi.org/10.1017/CBO9780511667572

[31] Sumit K. Mandal, Raid Ayoub, Micahel Kishinevsky, Mohammad M. Islam, and Umit Y. Ogras. 2021. Analytical Performance Modeling of NoCs under Priority Arbitration and Bursty Traffic. *IEEE Embedded Systems Letters* 13, 3 (2021), 98–101. https://doi.org/10.1109/LES.2020.3013003

[32] Marco Ajmone Marsan, Michele Garetto, Paolo Giaccone, Emilio Leonardi, Enrico Schiattarella, and Alessandro Tarello. 2004. Using partial differential equations to model TCP mice and elephants in large IP networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*. 2821–2832 vol.4.

[33] Laurent Massoulié and James Roberts. 1999. Bandwidth sharing: objectives and algorithms. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*. 1395–1403 vol.3.

[34] Hiroyuki Masuyama and Tetsuya Takine. 2003. Sojourn time distribution in a MAP/M/1 processor-sharing queue. *Operations Research Letters* 31, 5 (2003), 406–412. https://doi.org/10.1016/S0167-6377(03)00028-2

[35] Vishal Misra, Wei-Bo Gong, and Don Towsley. 2000. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 151–160.

[36] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based congestion control for the datacenter. *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 537–550.

[37] Pooria Namyar, Behnaz Arzani, Srikanth Kandula, Santiago Segarra, Daniel Crankshaw, Umesh Krishnaswamy, Ramesh Govindan, and Himanshu Raj. 2024. Solving Max-Min Fair Resource Allocations Quickly on Large Graphs. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 1937–1958.

[38] Shruti Narayana, Emily Shriver, Kenneth O'Neal, Nuriye Yildirim, Khamida Begaliyeva, and Umit Ogras. 2023. Similarity-Based Fast Analysis of Data Center Networks. *IEEE Design & Test* PP, 1–1. https://doi.org/10.1109/MDAT.2023.3310450

[39] Shruti Yadav Narayana, Emily Shriver, Kenneth O'Neal, Nuriye Yildirim, Khamida Begaliyeva, and Umit Y Ogras. 2023. Similarity-Based Fast Analysis of Data Center Networks. *IEEE Design & Test* (2023).

[40] Shruti Yadav Narayana, Jie Tong, Anish Krishnakumar, Nuriye Yildirim, Emily Shriver, Mahesh Ketkar, and Umit Y. Ogras. 2023. MQL: ML-Assisted Queuing Latency Analysis for Data Center Networks. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 50–60.

[41] David Nicol and Richard Fujimoto. 1994. Parallel simulation today. *Annals of Operations Research* 53 (1994), 249–285.

[42] Umit Y. Ogras, Paul Bogdan, and Radu Marculescu. 2010. An Analytical Approach for Network-on-Chip Performance Analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29, 12 (2010), 2001–2013. https://doi.org/10.1109/TCAD.2010.2061613

[43] Qiuyu Peng, Anwar Walid, Jaehyun Hwang, and Steven H. Low. 2016. Multipath TCP: Analysis, Design, and Implementation. 24, 1 (2016), 596–609. https://doi.org/10.1109/TNET.2014.2379698

[44] Xi Peng, Fan Zhang, Li Chen, and Gong Zhang. 2021. A MAP-based Performance Analysis on 5G-powered Cloud VR Streaming. In *Proceedings of the IEEE International Conference on Communications (ICC)*. 1–6.

[45] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. 2022. Jupiter Evolving: Transforming Google's Datacenter Network via Optical Circuit Switches and Software-Defined

Networking. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 66–85.

[46] George F. Riley and Thomas R. Henderson. 2010. *The ns-3 Network Simulator*. Springer Berlin Heidelberg. 15–34 pages. https://doi.org/10.1007/978-3-642-12331-3_2

[47] Thomas G. Robertazzi. 2000. *Computer Networks and Systems: Queueing Theory and Performance Evaluation*. Springer-Verlag.

[48] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 123–137.

[49] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2020. RouteNet: Leveraging graph neural networks for network modeling and optimization in SDN. *IEEE Journal on Selected Areas in Communications* 38, 10 (2020), 2260–2270.

[50] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023). https://arxiv.org/abs/2307.09288

[51] Andras Varga. 2019. *A Practical Introduction to the OMNeT++ Simulation Framework*. Springer International Publishing. 3–51 pages. https://doi.org/10.1007/978-3-030-12842-5_1

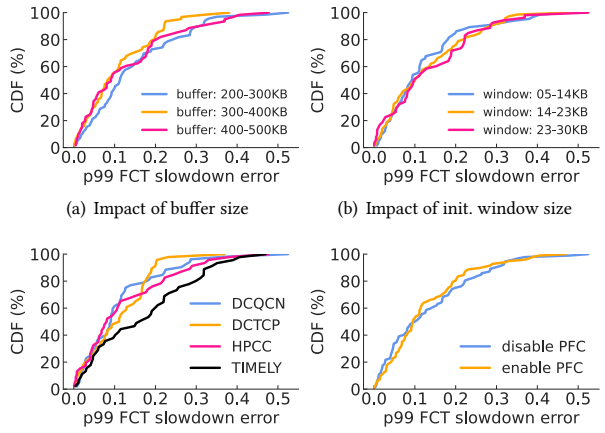
[52] Bob Wheeler. 2019. Tomahawk 4 switch first to 25.6 Tbps. *Microprocessor Report* (2019). https://docs.broadcom.com/doc/12398014

[53] Qingqing Yang, Xi Peng, Li Chen, Libin Liu, Jingze Zhang, Hong Xu, Baochun Li, and Gong Zhang. 2022. DeepQueueNet: towards scalable and generalized network performance estimation with packet-level visibility. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 441–457.

[54] Qizhen Zhang, Kelvin K. W. Ng, Charles Kazer, Shen Yan, João Sedoc, and Vincent Liu. 2021. MimicNet: fast performance estimates for data center networks with machine learning. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 287–304.

[55] Kevin Zhao, Prateesh Goyal, Mohammad Alizadeh, and Thomas E Anderson. 2023. Scalable Tail Latency Estimation for Data Center Networks. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 685–702.

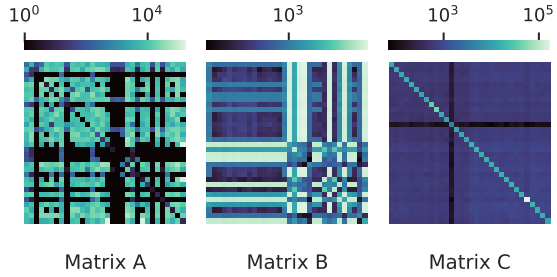
[56] Yibo Zhu, Haggai Eran, Daniel Firestone, ChaunXiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 523–536.



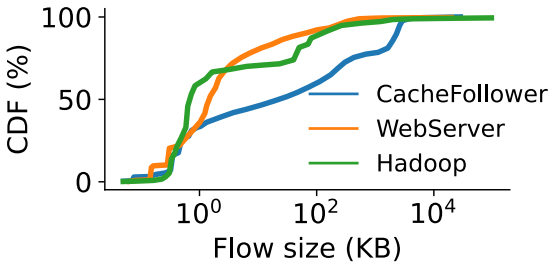
(a) Impact of buffer size (b) Impact of init. window size

(c) Impact of CC algorithms (d) Impact of triggering PFC

Figure 17: m3's estimation errors of p99 slowdown across different sample spaces in Table 4.



(a) Traffic matrices (32-rack sample)



(b) Flow size distributions

Figure 18: We use data from Meta's data center network [48], including (a) the traffic matrices extracted from the accompanying dataset, and (b) the flow size distributions estimated from the published data for evaluation.

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

A DETAILS ON FLOWSIM

Algorithm 1 depicts the path-level simulation with a path-level workload on a parking-lot topology. flowSim begins by initializing a priority queue Q with all flows F, including their sizes and arrival times (lines 2). Next, flowSim schedules the flow arrival and completion events (lines 4-10). For each event, flowSim iteratively identifies the bottleneck link along with its associated flows (line 14) and assigns the max-min rate to each associated flow by considering the capacity of its bottleneck link (lines 11-17). flowSim ends when all flows are completed. flowSim processes even a 6-hop parking-lot topology with 1 million flows in just a few seconds.

Algorithm 1: flowSim's FCT estimation based on flow event scheduling and max-min fair sharing.

Input: Set of n flows F, Set of k links L with initial capacities C
Output: Flow Completion Times (FCT) for flows in F

1 **Function** getFctFlowsim(F, L, C)
  ▷ *Dynamic flow event scheduling*
2 $Q \leftarrow$ PriorityQueue(F) // Initialize an event queue with flow sizes and arrival times
3 $F^* \leftarrow \emptyset$ // Set of active flows
4 **while** !Q.isEmpty() **do**
5 $(f, t,$ EventType$) \leftarrow Q$.pop()
6 **if** *EventType is Arrival* **then**
7 F^*.add(f) // Add a new flow
8 **else**
9 F^*.remove(f) // Remove a completed flow
10 addFCT(f, t)
  ▷ *Iterative max-min fair rate allocation*
11 $R^* \leftarrow \emptyset$ // Flow rates of active flows in F^*
12 $C^* \leftarrow C$ // Initialize link capacities
13 **while** $len(R^*) \neq len(F^*)$ **do**
14 $(r, l) \leftarrow$ getBottleneckLinkRate(F^*, L, C^*)
  foreach $f \in$ getFlowsOnLink(F^*, l) **do**
15 **if** $f \notin R^*$ **then**
16 $R^*[f] \leftarrow r$ // get its max-min fair rate
17 $C^* \leftarrow$ UpdateCapacities(C^*, F^*, R^*)
18 $Q \leftarrow$ UpdatePriorityQueue(Q, F^*, R^*)
19 **return** getFCT()

B M3'S ESTIMATION ERROR FOR COUNTERFACTUAL SEARCH

Figure 17 demonstrates m3's p99 slowdown estimation error across different sample spaces in Table 4.