Fast Network Simulation Through Approximation or: How Blind Men Can Describe Elephants

Charles W. Kazer, João Sedoc, Kelvin K.W. Ng*, Vincent Liu, and Lyle H. Ungar
University of Pennsylvania and *CUHK
{ckazer,joao,liuv,ungar}@seas.upenn.edu and kwng6@cse.cuhk.edu.hk

ABSTRACT

Network researchers today are unable to test their new ideas at scale before deployment due to the prohibitive costs of custom testbeds and the slow speed of large-scale network simulators. Data center simulation is particularly slow because of the massive amount of bandwidth and high degree of redundant computation incurred in simulating the network stacks of thousands of commodity machines. By using approximation to replace redundant portions of the simulation, we improve computation time while retaining high accuracy.

1 INTRODUCTION

The past few years have seen a staggering proliferation of interesting and novel ideas toward improving the performance and reliability of data center networks [3, 4, 8, 11, 21]. A testament to the intellectual curiosity and creativity of the networking community, these proposals span all layers of the stack and frequently demonstrate promising results.

Despite this, innovation in data centers faces a difficult challenge. As some of the largest networks in existence, researchers (and most data center operators) typically do not have spares lying around for the sole purpose of testing new ideas. *How then do we evaluate a new data center proposal?*

A brief survey of recently published data center networking research reveals two general approaches. The first is to assume that behavior on a small test bed scales directly. While accurate at a per-node-level, cost concerns mean that these experiments rarely encompass more than a few racks; test beds with device counts that are several orders of magnitude smaller than production deployments fail to capture a wide range of behavior that can only be seen at scale. Incremental rollouts to a production network suffer from the same limitations with the added risk of catastrophic failures.

Slightly better are packet-level simulations that allow users to deploy arbitrary systems at arbitrary scale. In principle, these can provide an approximate sense for at-scale network behavior; however, these too are limited in size. Figure 1 shows the performance of OMNeT++ simulations of several leaf-spine topologies (methodology in Section 2.2). Even for these small cases, simulation is 3–4 orders of magnitude slower than real-time (it takes 15 minutes to obtain a single second of simulated behavior, and over a month to obtain just 1 hour of data). The natural approach of using multi-core execution can actually *reduce* performance compared to a single thread due to the highly interconnected nature of data center networks. The other approach of running multiple single-threaded instances increases throughput, but not the time to a useful result. In the end, the tools to evaluate these new protocols lag behind the ideas themselves.

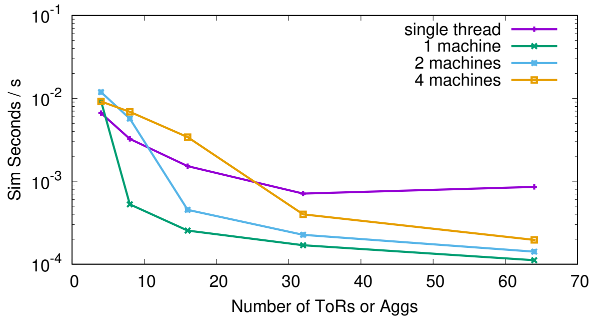

Figure 1: OMNeT++ performance on leaf-spine topologies of various size. Even for these small cases, 5 minutes of simulation time takes multiple days to process.

The approach we propose in this paper takes inspiration from another big (literally) problem: global climate modeling. In climate modeling, researchers predict the behavior of the Earth's climate using simulations of the atmosphere and oceans. As the Earth is too complex a system to model exactly, a common technique is to partition it into sub-regions such that much of the complexity can be approximated locally, with the inter-cell interface handled at a higher-level. In this paper, we argue that the data center can benefit from a similar partitioning and approximation-based approach.

We propose to divide the data center into several regions, then construct a fast, accurate machine-learning approximation of each region. One region can be left unapproximated so as to allow arbitrary packet-level analysis of behavior. Thus, when a user wishes to test a new protocol they would, at packet-level granularity, simulate (1) a small but full-fidelity network containing a single machine, rack, or cluster connected to (2) a large network of ML models that faithfully and efficiently approximate the rest of the data center.

HotNets-XVII, November 15–16, 2018, Redmond, WA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6120-0/18/11...$15.00
https://doi.org/10.1145/3286062.3286083

We note that deep learning models to approximate complex systems have been successfully applied to similar tasks [7, 12, 33]. Among deep learning methods, recurrent neural networks, particularly long-short term memory models (LSTMs) [14], are promising candidates for this problem. LSTMs provide a flexible, non-linear mechanism for forecasting. One of the main advantages of deep learning models is that due to their highly non-linear nature, the hidden state of the LSTM can capture complex underlying relationships without explicit feature engineering. While the upfront cost of training these models is high, once trained they are cheap to run, reusable, and beneficial to asymptotic behavior.

The point of this paper, however, is not to present a complete design for such a system. Instead, our goal is simply to argue for the *promise* of an ML-based approximation approach to simulation of data center networks. Major challenges remain, such as how to model network behavior accurately, how to balance accuracy and speed of simulation, the correct interface between simulated regions of the network, and how to generalize from smaller networks to larger ones. Our results, though preliminary, are encouraging—they demonstrate a different approach to using machine learning to assist in the evaluation of large systems.

2 MOTIVATION

Modern data center networks are composed of up to hundreds of thousands of devices 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, Clos networks like the one in Figure 2 [2, 28]. In a canonical 3-layer deployment, the layers from bottom to top, consist of servers, Top-of-Rack (ToR) switches, Cluster switches, and Core switches. We refer to the components under a single ToR as a rack, and the subtree of components under and including a group of Cluster switches as a cluster.

2.1 Background on Network Simulation

Recent work has touched on every part of the above architecture, from forwarding table implementation in the switches to congestion control on the hosts. Simulations are a popular tool with which to evaluate these ideas in a controlled way.

Many approaches for simulation exist, including flow-level systems, closed-form solutions, and a vast array of optimized custom simulators. These solutions assume an idealized and approximate view of the network, but are limited in their generality and the types of phenomena they can detect. For that reason, packet-level simulators are the preferred approach for most use cases. Within that category, most use Discrete Event Simulation (DES), e.g., ns-2, ns-3, and OMNeT++. In DES, network behavior is represented as a series of events (packets, timeouts, etc.) in a temporally ordered event queue. DES is a useful technique in packet-level simulation because it naturally lends itself to the processing of packets as they traverse network hops and are encapsulated/decapsulated at different network layers.

Users can change any piece of the system by simply changing the implementation of event handlers. The eventual output of the simulation is also configurable; users can compute arbitrary statistics (e.g., flow completion time, throughput, latency, drop rate, etc.) or can print raw packet/event traces.

2.2 Today's Simulators Do Not Scale

Modern simulators are notoriously slow. Even for relatively small networks, they can require weeks for just a few simulated minutes. Full-size data centers over similar timescales could take years to simulate. Because of this, it is typical to limit their execution to severely truncated network sizes and/or timescales. Both introduce the potential to miss important patterns present in larger experiments.

One example of such a pattern is pathological minimum window issues in TCP. Given enough simultaneous connections, it is possible that the fair share of each connection is *less than* their minimum window size. When this occurs, TCP will never back off enough to prevent high packet loss. These circumstances, non-existent in moderately sized networks, contributed to the adoption of rate-based congestion control in Google's data center networks [5, 21].

Multi-threading. One potential solution to the problem of simulator speed is multi-threading, for instance using Parallel Discrete Event Simulation (PDES). PDES partitions the network and has each partition, in parallel, simulate events that are not causally dependent [10]. Many popular simulation frameworks include support for this technique [31]. However, the improvement offered by PDES is limited because causality must be maintained in order to ensure accurate simulation.

As mentioned in Section 1, for highly interconnected networks like those found in data centers, synchronization can actually cause PDES to perform *worse* than a single-threaded implementation. To demonstrate this effect, we run an OMNeT++ simulation on servers with two Intel Xeon E5-2660 processors (10 cores each). The simulation is of a leaf-spine topology with 10 GbE links and racks of four severs. We vary the size of the network by increasing the number of ToRs and Cluster switches from 4 to 64, while maintaining oversubscription and average load. For parallel executions, we leverage OMNeT++'s built-in MPI-based PDES framework.

Figure 1 shows the results, where higher simulation seconds per second indicates better performance. While multi-threading helps for smaller network sizes, as connections increase, so does the overhead of synchronization. Thus, for large networks, single-threaded instances beat the parallel deployments significantly. Performance of PDES increases with machine count, so a large cluster could, in principle, outperform a single thread, but at disproportionate cost.

Other methods of parallelization. An oft-recommended alternative to multi-threading is simply running many instances. This trivially provides a proportional speedup to aggregate simulation throughput, but does not improve the time to results, which may be important for iteration, etc.

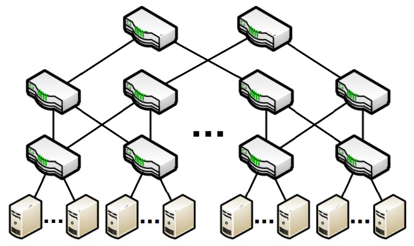

Figure 2: A 3-layer Clos network with servers connected with a tree of ToR, Cluster, and Core switches.

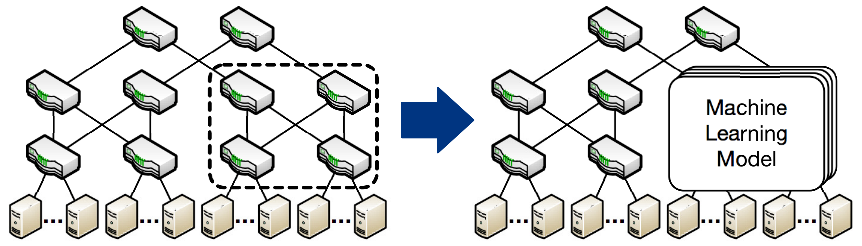

Figure 3: A sketch of our proposed approach. We first run a simulation with just two clusters and train a model to replace one of them. We can then reuse the trained cluster model in large-scale simulations to replace the majority of the network.

More generally, scaling through parallelization is typically limited to linear speedups while the speed of current simulators is orders of magnitude away from practical.

3 OVERVIEW

In this paper, we argue that the right way to model the behavior of large networks is to leverage machine-learning (ML) aided approximation. While there has been much recent interest in applying ML to predict how systems will function given different inputs (e.g., configurations, failure conditions, etc.), the target of our work is in how we might predict the behavior of *large systems* using observations of *smaller systems*.

Compared to the lower-granularity and customized approximation approaches mentioned in the previous section, approximation using ML (and specifically, using neural networks) lends itself to generality and flexible accuracy.

Said differently, the Hindu parable we alluded to in this paper's title is a lesson in how the subjective reality of different viewers can be hard to synthesize. In this paper we show that synthesis is possible as long as we leverage careful planning, coordination, and deep learning techniques.

The design goals of our framework are as follows:

- *Orders of magnitude faster simulation:* Our simulator must be able to run a data center traffic matrix several orders of magnitude faster than existing approaches. Parallelization, on its own, is not enough—we seek to decrease the total work done by the system.
- *Adjustable accuracy:* We seek to give users the option to control an accuracy versus speed trade-off. Full scale simulation is needed in some scenarios, where the fidelity of each packet is crucial. However, approximating portions of the network allows users to simulate longer spans of time for the same cost of computation, while taking an accuracy hit. Our work aims to develop this trade-off for data center simulation.
- *Modularity:* The method we choose must be able to model different protocols and traffic patterns. These can be at any layer of the networking stack.

The workflow we propose is one in which a user partitions the data center network into identical regions containing both servers and switches. We first briefly simulate a small network in full packet-level fidelity to generate training and testing sets for a machine learning model that can take incoming packets as inputs and generate properly timed outgoing packets.

After training a robust model that accomplishes the above, we can assemble a full-size simulation in which the vast majority of the network is replaced with fast and accurate models as shown in Figure 3. The benefit of this approach is that, during at-scale simulation, the internals of each approximated region can be safely skipped, thereby significantly reducing the total number of events and therefore the time it takes to run the simulation. Instead, their complexity is reduced to matrix multiplications—a highly optimized operation on modern computer architectures. For example, if we were to approximate 63 of 64 clusters of a data center, we could potentially reduce the number of events scheduled by orders of magnitude.

Further, a portion of the network can be left un-approximated so that we can continue to draw full-fidelity statistics from the simulation. The symmetric structure of data centers means that these subregions will exhibit representative behavior.

We leave a thorough design and implementation of such a simulation framework to future work. Instead, this paper lays out the key challenges of this approach, discusses our initial thoughts on the topic, and presents a preliminary prototype that demonstrates several aspects of the system.

4 MODELING A NETWORK REGION

A key decision is how to choose the size of a region in order to balance fidelity, accuracy, and training speed. The larger the target region, the more events that can be removed by approximation, but the harder it is to get high accuracy in training quickly. Our preliminary prototype uses clusters as the unit of approximation. For example, Figure 2 and Figure 3 show how our system would replace the four switches of each approximated cluster with a single black box approximation.

Given that split, we model the network region's behavior as time series prediction problem: given a incoming trace of packets, can we predict if and when packets will egress the region? Our preliminary prototype for addressing that question is driven by both data center specific domain knowledge as well as statistical analysis of commonly used data center packet traces [3]. In particular, our analysis of latency and drop data from sample TCP web traffic revealed meaningful patterns in the data both at the timescale of seconds as well as microseconds. At the seconds scale, the average latency of packets perceptibly shifts up and down as queues fill and drain. At the microsecond scale, small jitter occurs in queues as different flows join and leave the network, which appears as small troughs and peaks in latency. Because of this multi-scale structure, we developed a hierarchical set of models: a single "macro" model for longer-term regimes, and a "micro" model for regression over individual packets.

4.1 Macro Model

We identify four macro states in the data:

(1) *Minimal congestion*, where queues are mostly empty and packets experience minimal queuing latency.
(2) *Increasing congestion*, where paths are becoming congested, but latency has not peaked.
(3) *High congestion*, where a significant number of packets are being dropped due to full queues.
(4) *Decreasing congestion*, where congestion is subsiding, allowing queues to drain.

Currently, our simulation platform identifies macro states using a simple and fast auto-regressive model. Based on previously observed latency and drop rates, if latency is relatively low, it classifies the network as (1). If drops are relatively high, it classifies the network as (4). (2) and (3) are distinguished based on prior state by observing whether latency and drops are rising or falling.

4.2 Deep-learning Micro Models

Our micro models use LSTMs [14], a type of recurrent neural network (RNN). Like other neural networks, RNNs are composed of a series of neural nodes that take input features, and perform a series of matrix multiplications on them based on each node's activation function. In RNNs, hidden layers carry forward state information, which allows the network to "remember" previous inputs when processing future inputs. This gives them the ability to recognize patterns. RNNs are able to approximate any dynamic system with arbitrary complexity [27].

LSTMs are a special kind of RNN that include "memory gates." These gates are sigmoid activation functions coupled with a multiplication step that serve to choose which inputs are remembered by the LSTM and which are forgotten [23]. During training, samples are run through the LSTM, and gradient back propagation based on a loss function is used to determine the parameters of the network.

Our LSTM models are trained at the packet level—the model takes packets as inputs and predicts whether they will be dropped *or* after what delay they will be delivered. Other network-level effects such as route and packet-level modifications can be handled by either computing the effect directly (if it is deterministic like TTLs, MAC addresses, or ECMP path choice), or including it as part of the ML training and prediction (for instance, when setting an ECN bit).

We train one model for packets entering the approximated cluster and one for packets leaving because the distribution of flows in either direction can differ significantly at a given point of time. The features used for training are crucial to the success of both models. For each packet, these include: the origin and destination servers; the ToR, Cluster, and Core switches that the packet *would* pass through in the cluster replaced by approximation; the time since the last packet arrived at the model; a moving average of these times; and finally, the current macro state of the cluster. The model then outputs both a binary decision whether to drop the packet and the predicted latency that the packet would experience if it is not dropped. For both ingress and egress, all of the input features can be calculated directly from the packet header information, simulation time, and knowledge of routing strategy.

The multi-dimensional hidden state output from the LSTM is given to one fully connected layer to predict the latency and another fully connected layer to predict packet drop. This is superior to training two separate models as the neural network representation can learn the joint distribution of drops and latency. As a result the loss function for training has two components: binary cross entropy loss for the drop decision per packet and mean squared error for the latency values. A hyper-parameter α balances the relative contribution of error prediction, $\mathcal{L} = \mathcal{L}_{drop} + \alpha\mathcal{L}_{latency}$. However, if there is a packet drop then no latency error can be back-propagated. In practice, we set α to a value $0 < \alpha \leq 1$ because the contribution of drops in determining future behavior is more significant than latency.

The micro models are trained on >50,000 batches of size 64. We used the stochastic gradient descent optimizer with a learning rate of 0.0001 and momentum of 0.9. The model prediction is relatively fast since prediction only involves a few matrix multiplications and non-linear transformations.

We note that predicted latency can sometimes result in impossible schedules if two packets are scheduled for the same time. In this case, the one processed first is given priority, with conflicting packet sent at the next possible time. We also note that in some cases, the behavior of packets can logically depend on messages that have not yet arrived. Delays and lookahead mechanisms are thus important open challenges.

5 SIMULATING AN ENTIRE NETWORK

We can use approximations for each cluster fabric as building blocks for a faster network simulator as shown in Figure 3. Our prototype extends the popular OMNeT++ simulator through a custom library that implements both a macro state classifier and micro latency/drop predictors. In our prototype, a single cluster and all core switches are implemented in full fidelity. Approximated clusters run full TCP stacks because it is more efficient to implement them than try to learn the TCP state machine and protocol specification.

The aspects of the simulation this is able to elide are (1) the ToR and Cluster switches of approximated clusters including their queuing, routing, and packet processing procedures,

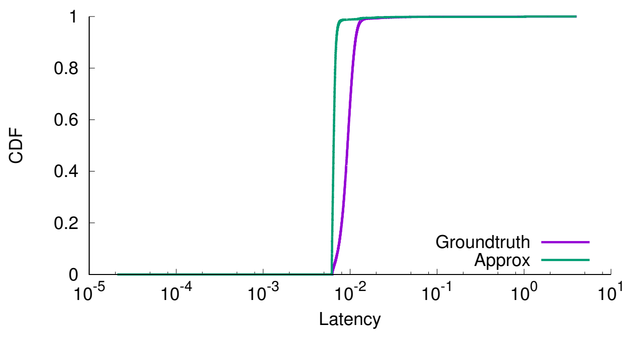

Figure 4: CDF of real and approximate packet RTTs.

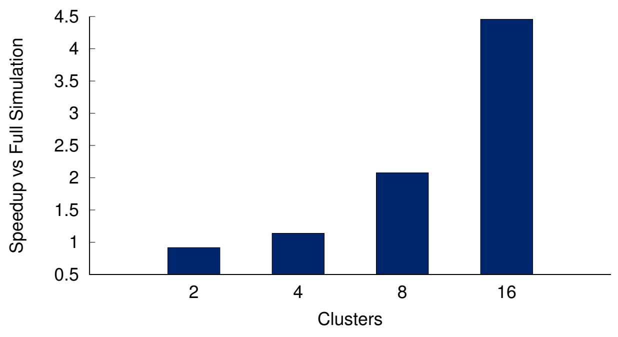

Figure 5: Speedup our approximate simulation compared to full-fidelity, across a range of data center sizes.

and (2) the traffic within and between approximated clusters, which is not needed by the fully approximated cluster for analysis. These aspects significantly reduce the total amount of work done by the simulation, and thus have the potential to improve performance substantially.

6 EVALUATION

We evaluate our system for two primary metrics: speed compared to full simulation and accuracy in end-to-end latency. For these preliminary results, we tested a Clos topology with TCP New Reno and ECMP implemented on OMNeT++/INET. Machine learning models were trained using PyTorch 0.4.0 and accessed within OMNeT's C++ environment via ATEN. All experiments were run on servers with two Intel Xeon Silver 4114 CPUs, 192 GB of memory, and an NVIDIA Tesla P100 GPU. Traffic patterns are drawn from a well-known trace of datacenter web traffic [3].

6.1 Accuracy for ECMP TCP

To understand the accuracy of our system, we compare the CDFs of observed RTTs by hosts in both the full and approximate simulations. The results are shown in Figure 4. We chose to do a CDF comparison rather than report a per packet metric because TCP interaction with the model makes these measurements unreliable. The interaction of TCP congestion control and the imperfect model predictions during run time will cause latencies to diverge and some packets to be dropped that were not dropped in the full simulation and vice versa. Thus, a packet-to-packet comparison is not as meaningful. Instead, we use a CDF to ask whether the overall distributions of the two simulations are similar.

This analysis gives a sense of how well the model is able to follow the changes in congestion of a real cluster due to packets entering the system. The approximated version has a steeper slope, but still turns upward at a similar value to the ground truth. This means that the approximation is consistently underestimating congestion, however the values are still within the realm of possibility. This shows that the approximation is predicting in the right ballpark, and methods of improving the accuracy are discussed in the future work.

6.2 Speed of Approximation

To test the speedup of our approach, we simulated varying numbers of clusters both fully, and using our approximation clusters. In each test, the full simulation runs clusters containing four switches and eight servers, while the approximate simulation replaces all but one of these clusters with approximation models. We note that our accuracy and thus performance results, particularly for larger cluster counts, are preliminary. Thus, these numbers represent a upper bound on the performance benefits of our current design. Figure 5 shows the results of these tests.

There are significant speedups that increase in magnitude as the number of clusters increases. The approximate version is faster than the full version for three reasons. The first is that the events scheduled in the approximated network fabrics are completely removed and replaced with LSTM classifications, a fast operation. The second is that traffic between servers in approximated clusters is entirely omitted from the flow schedule. This traffic is not needed because it does not directly affect the measurements of the fully simulated cluster, so it can be safely omitted so long as accuracy is retained. Both of these savings compound with approximating larger networks, meaning these techniques scales well. Third, the approximate version was run in parallel. Because the interdependencies between cluster fabric switches are removed, parallel execution provides better speedups here than it does for full simulation. These results indicate that our method has the potential to scale to hundreds of clusters and thousands of machines while still keeping the runtime to a useful result low.

7 DISCUSSION AND FUTURE WORK

In the previous section, we demonstrated an initial prototype of our system; however, this paper represents a first cut at a large and challenging problem—how to predict the behavior of massive systems without massive infrastructure. There are a number of future directions we wish to explore.

Improving accuracy. There is a great deal of room for improvement in accuracy. Our prototype currently uses a two-layer LSTM with 128 hidden nodes. Accuracy can be improved by stacking more layers, using more nodes per layer, and testing new LSTM variants. Each of these come with tradeoffs that must be carefully balanced—adding more complexity may increase the cost of training and prediction, but allows us to learn more complex behaviors.

Other potential directions include more advanced loss estimation that takes the nature of the traffic into account, better feature engineering, and hyperparameter tuning.

Further scalability improvements. An open question is how much more complexity we can remove while retaining accuracy. In the limit, the rest of the network could be modeled as a single black box, but training that black box to approximate such a large collection of machines is not trivial.

We also plan to explore other limitations to scalability. Though our evaluation demonstrates promising performance trends, one reason we could not evaluate full-sized data centers is that our servers simply did not have enough memory to hold state for millions of TCP connections. Further replacement of these connections with ML models can help to address this issue, but additional systems-level challenges may become important.

Generality. While our LSTM-based approach is agnostic to many details of the target architecture, it is an open question as to the extent of this generality. In particular, more exotic protocols with stochastic behavior, long-term dependencies, and sparse data may be difficult for traditional LSTMs to handle. We note that the ML community is actively pursuing solutions to these challenges.

An insight from our data is that there are multiple long term regimes in latency and packet drop. Multi-scale and hierarchical recurrent neural network models [6] are interesting future directions as these models can simultaneously capture macro and micro effects. Recent work in focused hierarchical RNNs [15] and State Frequency Memory RNNs [33] will also be considered for the micro model to incorporate longer term information. Another area of further research is to understand if subsets of the model can be trained independently or with interactions only at hidden layers of the RNN.

Generalizing even further, we plan to explore applicability to other network protocols and other network structures like those found in the Internet or experimental topologies. Though our approach targets data center networks specifically, neural network models are able to adapt to wide variety of behaviors, making them potentially well-suited for a general purpose simulator.

Applications to systems. Finally, we note that the proposed ideas have broader applicability than just simulation. As mentioned in Section 3, the target of our work is in how we might predict the behavior of *large systems* using observations of *smaller systems*. This ability may be useful in other systems as well. At the very least, it could be used as an enhancement to recent work on predicting the behavior of systems to find optimal configurations, etc. For instance, it may enable systems to *sample* the behavior of pieces of the network with the aim of adapting to network-wide effects.

8 RELATED WORK

As critical tools for both researchers and operators, network simulators have been around for decades [17]. Popular choices include ns-3 [13, 22] and OMNeT++ [18] (upon which our system is built). To our knowledge, our system is the first to leverage an approximate learning model.

When simulating large networks, the predominant approach is to sacrifice granularity by eschewing packet-level analysis entirely. Flow-level simulation is one example of this approach [20, 25]. As mentioned in Section 2, these simulators can provide insight into the general behavior of the system, but miss out on many import network effects, particularly in the presence of bursty traffic. Other solutions rely more on analysis rather than simulation. BigHouse [19], for instance, models data center behavior using traffic drawn from empirically generated distributions, and then again modeling data center metrics based off of the generated traffic. Our system, in contrast, begins with a faithful reproduction of the target system, providing a more realistic simulation.

A related topic is emulation. Emulation seeks approximately real-time results, which makes iteration fast and introduces the possibility of attaching real systems to the framework [1, 16, 24, 26, 29, 30]. These are nice properties, but in terms of scale, emulators are generally not built to extend beyond the number of available servers. In contrast, our system runs offline and can potentially scale to arbitrary sizes.

Finally, recent work has explored the use of machine learning and statistical approximation to tailor system configurations. In particular, several systems have used reinforcement learning to train congestion control based on assumptions about the underlying network [9, 32]. These works focus on optimizing systems rather than using statistical methods to approximate computation, as our work does.

9 CONCLUSION

We have presented a system to speed up network simulation by replacing redundant network fabrics with machine learning approximations. Our system uses a machine learning model which considers the observed history of packet headers passing through the network to incur drops and latency on new packets that pass through it. Preliminary results using our model show speed ups for larger clusters while still maintaining decent accuracy in terms of latency.

ACKNOWLEDGEMENTS

We gratefully acknowledge our shepherd Jia Wang and the anonymous HotNets reviewers for their thoughtful comments.

REFERENCES

[1] Mohammad Al-Fares, Rishi Kapoor, George Porter, Sambit Das, Hakim Weatherspoon, Balaji Prabhakar, and Amin Vahdat. 2012. NetBump: User-extensible Active Queue Management with Bumps on the Wire. In *Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '12)*. ACM, New York, NY, USA, 61–72.

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM '08)*. ACM, New York, NY, USA, 63–74.

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

[4] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. 2012. 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)*. USENIX, San Jose, CA, 253–266.

[5] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *Queue* 14, 5, Article 50 (Oct. 2016), 34 pages. https://doi.org/10.1145/3012426.3022184

[6] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2016. Hierarchical Multiscale Recurrent Neural Networks. *CoRR* abs/1609.01704 (2016). arXiv:1609.01704 http://arxiv.org/abs/1609.01704

[7] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12)*. Curran Associates Inc., USA, 1223–1231.

[8] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation (OSDI'04)*. USENIX Association, Berkeley, CA, USA, 10–10.

[9] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 343–356.

[10] Richard M. Fujimoto. 1990. Parallel Discrete Event Simulation. *Commun. ACM* 33, 10 (Oct. 1990), 30–53.

[11] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. ACM, New York, NY, USA, 202–215.

[12] J. B. Heaton, Nicholas G. Polson, and J. H. Witte. 2016. Deep Learning in Finance. *CoRR* abs/1602.06561 (2016), 20. arXiv:1602.06561 http://arxiv.org/abs/1602.06561

[13] Thomas R. Henderson, Mathieu Lacage, and George F. Riley. 2008. Network simulations with the ns-3 simulator.

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

[15] Nan Rosemary Ke, Konrad Zolna, Alessandro Sordoni, Zhouhan Lin, Adam Trischler, Yoshua Bengio, Joelle Pineau, Laurent Charlin, and Christopher Joseph Pal. 2018. Focused Hierarchical RNNs for Conditional Sequence Processing. *CoRR* abs/1806.04342 (2018).

[16] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. 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)*. ACM, New York, NY, USA, Article 19, 6 pages.

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

[18] OpenSim Ltd. 2018. OMNeT++. http://omnetpp.org.

[19] David Meisner, Junjie Wu, and Thomas F. Wenisch. 2012. BigHouse: A Simulation Infrastructure for Data Center Systems. In *Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS '12)*. IEEE Computer Society, Washington, DC, USA, 35–45.

[20] Vishal Misra, Wei-Bo Gong, and Don Towsley. 2000. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '00)*. ACM, New York, NY, USA, 151–160.

[21] 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. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, New York, NY, USA, 537–550.

[22] nsnam. 2017. ns-3. http://www.nsnam.org.

[23] Cristopher Olah. 2015. Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs.

[24] Amy Ousterhout, Jonathan Perry, Hari Balakrishnan, and Petr Lapukhov. 2017. Flexplane: An Experimentation Platform for Resource Management in Datacenters. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17)*. USENIX Association, Berkeley, CA, USA, 437–451.

[25] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. 2011. Improving Datacenter Performance and Robustness with Multipath TCP. In *Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM '11)*. ACM, New York, NY, USA, 266–277. https://doi.org/10.1145/2018436.2018467

[26] Luigi Rizzo. 1997. Dummynet: A Simple Approach to the Evaluation of Network Protocols. *SIGCOMM Comput. Commun. Rev.* 27, 1 (Jan. 1997), 31–41.

[27] Anton Maximilian Schäfer and Hans Georg Zimmermann. 2006. Recurrent Neural Networks Are Universal Approximators. In *Proceedings of the 16th International Conference on Artificial Neural Networks - Volume Part I (ICANN'06)*. Springer-Verlag, Berlin, Heidelberg, 632–640.

[28] 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. 2015. 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)*. ACM, New York, NY, USA, 183–197.

[29] Zhangxi Tan, Zhenghao Qian, Xi Chen, Krste Asanovic, and David Patterson. 2015. DIABLO: A Warehouse-Scale Computer Network Simulator Using FPGAs. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)*. ACM, New York, NY, USA, 207–221.

[30] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. 2002. Scalability and Accuracy in a Large-scale Network Emulator. *SIGOPS Oper. Syst. Rev.* 36, SI (Dec. 2002), 271–284.

[31] András Varga. 2009. Parallel Simulation Made Easy With OMNeT++.

[32] Keith Winstein and Hari Balakrishnan. 2013. TCP Ex Machina: Computer-generated Congestion Control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. ACM, New York, NY, USA, 123–134.

[33] Qin Zhang, Hui Wang, Junyu Dong, Guoqiang Zhong, and Xin Sun. 2017. Prediction of sea surface temperature using long short-term memory. *IEEE Geoscience and Remote Sensing Letters* 14, 10 (2017), 1745–1749.