iBox: Internet in a Box

Sachin Ashok
Microsoft Research India

Sai Surya Duvvuri
Microsoft Research India

Nagarajan Natarajan
Microsoft Research India

Venkata N. Padmanabhan
Microsoft Research India

Sundararajan Sellamanickam
Microsoft Research India

Johannes Gehrke
Microsoft Research Redmond

ABSTRACT

We present a vision of data-informed network simulation to address significant shortcomings in the state of the art. We substantiate our position with proof points based on iBox, which leverages networking domain knowledge and machine learning (ML) models, coupled with plentiful data, to provide a pathway to perpetual renewal of network simulators.

CCS CONCEPTS

• **Networks** → **Network simulations**; **Network measurement**; *Network performance modeling*; *Network experimentation*; *Network performance analysis.*

KEYWORDS

network simulation, cross-traffic estimation, data-driven simulation

ACM Reference Format:
Sachin Ashok, Sai Surya Duvvuri, Nagarajan Natarajan, Venkata N. Padmanabhan, Sundararajan Sellamanickam, and Johannes Gehrke. 2020. iBox: Internet in a Box. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets '20), November 4–6, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3422604.3425935

1 INTRODUCTION

There have traditionally been two broad approaches to network simulation [1]: (1) *mimicking* the operation of *individual network components*, e.g., links, queues, protocols (e.g. TCP), to then recreate overall network behaviour (e.g. ns [4]), and (2) *replaying* the network behavior from collected *offline* data *as is* (e.g. [33, 34]). The former typically lacks realism in terms of configuration (e.g., network topology and cross-traffic patterns) and the latter, though informed by real measurements, does not capture the impact on the network of the application or protocol under test (e.g., it might congest the network, invalidating the delay measurements). The end result is that network simulation can yield misleading, often optimistic, conclusions [16, 43, 44]. Further, the need for fast, inexpensive, and realistic network simulation is only growing, e.g., for leveraging machine learning (ML) in networking (e.g., [25, 32, 40, 44]).

[1]Unless stated otherwise, "simulation" encompasses emulation.

HotNets '20, November 4–6, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8145-1/20/11…$15.00
https://doi.org/10.1145/3422604.3425935

Here, we make the case for simulation based on *learning the network, informed by data.* Our proposal, dubbed iBox, or "Internet in a Box", is to turn traces of end-to-end network behaviour into simulation models that faithfully recreate network behaviour. iBox is not a single or specific design; rather, it is an articulation of our vision for how the networking research community could build on both conventional work and advances in ML, and transform network simulation, setting it on a path of perpetual renewal, fueled by data.

We focus on recreating the *end-to-end behaviour of a network path* rather than recreating the innards of the network. Even so, we argue that it is critical to *learn the network* rather than just characterize or mimic, even if with the benefit of data, the end-to-end behavior of the network; the difference is akin to modeling a disease versus merely characterizing its symptoms. The former would allow us to accurately predict how the network would treat a *new* application or protocol that behaves differently from the ones seen before. Our quest to learn the network goes beyond prior work [45] (see §7).

We start by considering two extremes for iBox. One centers on a network model, such as a traditional ns-like simulator, but with the configuration of the network and importantly also a model of the cross-traffic learnt based on data. The advantage is the interpretability of the model learnt in terms of familiar constructs (e.g., the bottleneck bandwidth) and the efficiency of execution for simulation. However, tractability of learning the configuration demands that the network model be kept simple (e.g., with a single bottleneck link), which means that more complex network behaviours (e.g., packet reordering) are not captured (§5.1).

The other extreme uses ML to learn a state-space model, which encodes the state of the network, using a deep LSTM neural network operating on input time series (e.g., sending rate), and then recreates end-to-end network behaviour such as packet delay or loss. The advantage here is that learning the ML model in its continuous parameter space is more tractable than the combinatorial search over the discrete-event network model above. The main challenge, however, is that unlike with the seq2seq models for NLP [15, 17, 35], in our setting, the sender adapts to the network, which introduces difficulties such as a *control loop bias* in the learnt model (§4.2).

After establishing the above extremes, we explore designs that meld networking domain knowledge with ML. For instance, we use ML to "discover" new behaviours and "correct" the network model's output accordingly (e.g., introduce appropriate reordering) or use domain knowledge to craft features (e.g., cross-traffic estimate) to feed to an ML model. We argue that such melding enables learning the network better, thereby catering to the diverse end-to-end metrics of interest to applications. We report preliminary results using Internet traces, mostly from a TCP testbed [45] and some from a real-time conferencing (RTC) service.

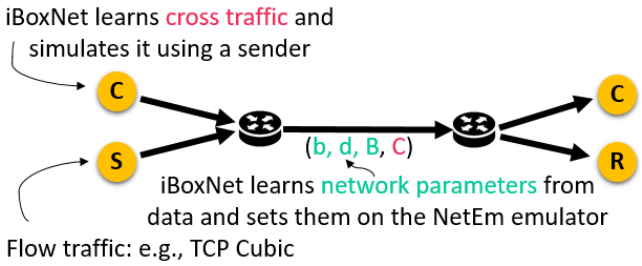


Figure 1: Topology of iBoxNet model.

The iBox vision is rich in research challenges, some of which we articulate here but only scratch the surface on.

2 PROBLEM FORMULATION

In iBox, the end-to-end behaviour of a network path is expressed in terms of packet delay. Each packet in an input (sender) stream is delayed by the appropriate amount and then delivered at the output (receiver). While being conceptually simple, this formulation can nevertheless accommodate a range of behaviours, including queue buildup (increasing delay), packet loss (infinite delay), and reordering (sufficient drop in delay between successive packets). With this framework, we consider two tests for evaluating iBox.

The *instance test* focuses on the counterfactual, i.e., what the end-to-end behaviour of the network would be if, in a particular instance (i.e., on a particular path at a particular time), sender type A (e.g., a particular flavour of TCP) were replaced with sender type B (e.g., another flavour of TCP), while keeping the rest of the network, including its configuration and the competing cross-traffic workload, unchanged. In other words, the network model is learnt using end-to-end traces of A and then used to predict behaviour if B were run instead. The dynamic nature of networks and the competing workloads means that it is impossible to perform an instance test in a real network, so we have to rely on controlled settings. However, once it has been shown to fare well in the instance test, iBox can be employed in real settings too, providing a new and powerful ability for counterfactual analysis.

The *ensemble test*, on the other hand, focuses on learning a model based on a collection of flows of type A, each from a possibly different time and over a different path, and then uses the model to predict the performance *distribution* of B. So, the ensemble test allows us to recreate flighting-based A/B tests but within the confines of the simulator itself.

3 NETWORK MODEL-BASED APPROACH

We begin with the network model based approach (iBoxNet), seeking to learn a parameterized, albeit simplified, network model to mimic a target network path (Fig. 1). The goal is to learn not only the mostly static parameters of the network path, e.g., the bottleneck bandwidth (b), propagation delay (d), and buffer size (B), but importantly also the dynamic, competing cross-traffic (C) time series.

We estimate the parameters of the iBoxNet model using input-output packet traces. The estimation of the various static parameters is relatively straightforward. Specifically, we calculate (i) the bottleneck bandwidth as the peak receiving rate, over 1s sliding

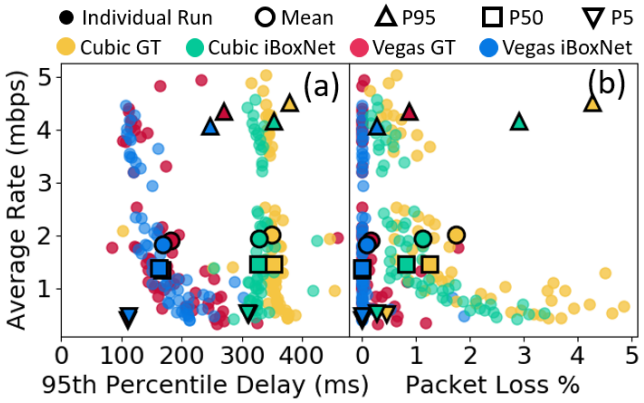


Figure 2: iBoxNet on Pantheon (India Cellular).

windows, seen in the training data (even if the sender does not fill the bottleneck link on a sustained basis, short bursts would still enable accurate estimation); (ii) the propagation delay as the minimum delay seen in the traces (the assumption being that in a long-enough trace, at least some packets will likely encounter an empty bottleneck queue); and (iii) the buffer size as the estimated bandwidth times the difference between the maximum and minimum delays (the assumption being that at least some packets would encounter an almost full buffer). The implicit assumption of a byte-based buffer is a simplification but nevertheless reasonable, given multi-size buffer pools [9].

In addition, we estimate the cross traffic that was experienced by a flow by analyzing its input-output trace and using the static parameters estimated above. In the interest of space, we just sketch out the approach — we model the three "forces" acting on the bottleneck queue: (1) packets enqueued from sender S (at a known rate), (2) packets enqueued from cross-traffic flows (at unknown rate, which we seek to estimate), and (3) packets dequeued at the bottleneck link (estimated). Care is needed since the dequeuing in (3) only happens while the queue is non-empty. We make a conservative estimate (i.e., lower bound) of cross-traffic, focusing just on periods when we are sure that the queue was non-empty. The cross-traffic so estimated is non-adaptive, i.e., it does not react to competition from the sender under test; see §6 for a brief discussion on adaptive models.

3.1 Evaluation

We evaluate iBoxNet using the Pantheon testbed [45] data. This testbed includes packet traces from 15 network paths between the AWS cloud and clients in 8 countries. Data corresponding to several congestion control protocols (such as TCP Cubic, Vegas, BBR, etc.) has been gathered over 4 years. The full Pantheon dataset includes tens of thousands of 30-second traces, with a typical cellular trace containing 5-10k packets and a typical Ethernet trace containing 100-200k packets. We focus here on a cellular network that is likely to stress-test the simple network model assumed in iBoxNet (Fig. 1), though we have evaluated iBoxNet on other paths too.

Our evaluation is in terms of A/B testing. We configure iBoxNet (Fig. 1) with the parameters, both static and dynamic, estimated using a "control" protocol (A) and then use the simulator to evaluate

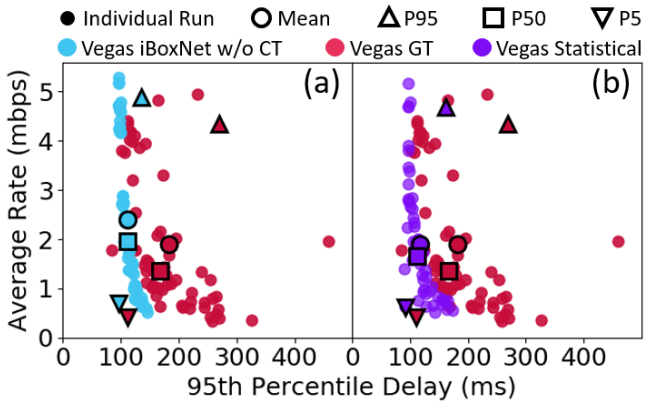


Figure 3: Comparison with ground truth (GT) of an iBoxNet model (a) without cross-traffic (CT) input, and (b) with statistical packet loss to recreate effect of CT.

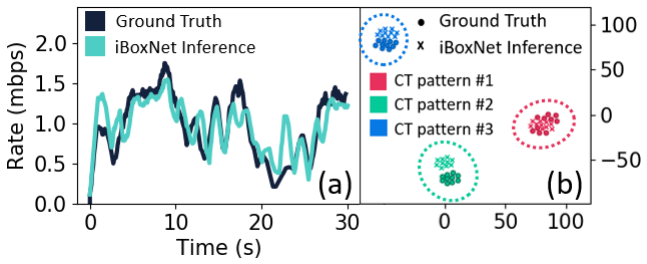


Figure 4: Instance tests with iBoxNet: (a) an example iBoxNet trace aligning well with a real-world (Pantheon) trace, (b) clustering (t-SNE plot) showing the traces corresponding to different instances clustering well together.

a "treatment" protocol (B). In the instance test, the parameters are estimated based on a particular trace of A. In the ensemble test, the parameters should ideally be drawn from the joint distribution learnt over the training data set comprising a potentially large number of traces, thereby ensuring that the appropriate combinations of bottleneck bandwidth, buffer size, cross-traffic, etc. are picked. For simplicity, however, we just use the parameters combinations derived from individual training traces for testing; so, N training traces from various network paths would enable ensemble testing with N parameter combinations.

We pick TCP Cubic as the control, or A, (since it is the most prevalent TCP flavour in the Internet) and TCP Vegas as the treatment, or B, (since its delay sensitivity makes it quite different from Cubic and hence challenging for iBoxNet).

3.1.1 Ensemble test with iBoxNet. Fig. 2 shows the distribution of the (a) 95th percentile delay and (b) packet loss rate, both versus rate (as in [45]). We see that despite the complexity of cellular networks (e.g., proportional fair scheduling [27]), the simple iBoxNet model trained using Cubic data is quite accurate. It yields a good match with the ground truth (GT), not only for Cubic but also for Vegas, which was never seen during model training (match verified through a two-sample KS test [2]).

We also see that either excluding cross-traffic as a parameter (Fig. 3(a)) or using a simple statistical packet loss model, as in [45], to recreate the effect of cross-traffic (Fig. 3(b)), yields a worse match with the ground truth than iBoxNet Fig. 2(a). These results underscore the importance of incorporating cross-traffic in the model and doing so with care.

3.1.2 Instance test with iBoxNet. As indicated in §2, we use a controlled emulator setup, with a known and fixed network configuration, a single main TCP Cubic flow, and 3 different cross-traffic (CT) patterns. The level and duration of the cross-traffic is kept the same (one Cubic cross-traffic flow of 10s duration) but with a different timing in the 3 "instances" (0-10s, 20-30s, and 40-50s during the 60s duration of the main Cubic flow). We learn an iBoxNet model for each instance, based on a single run with the corresponding cross-traffic pattern. (The cross-traffic pattern and other network configuration is treated as unknown and is estimated, just as it would be on an actual network.)

We then run a different protocol (Vegas) 10 times on the emulator for each cross-traffic configuration. We also run it 10 times using the iBoxNet model learnt for each of the 3 instances. k-means clustering (with $k = 3$) of these runs (using, as features, the cross-correlation between the iBoxNet rate and delay time series and their respective ground truth time series) is perfect, i.e., with no mistakes. Fig. 4 (b) is a t-SNE [31] plot showing the clusters. The circles (ground truth runs) corresponding to each cross-traffic pattern are well-clustered but do not exactly coincide because slight timing variations in the emulator execution can lead to somewhat different evolutions of the Vegas flow in each run. The important point, however, is that the crosses (obtained by running Vegas on the Cubic-derived iBoxNet instance models) are also well-clustered with the ground truth. Furthermore, Fig. 4 (a) illustrates that the rate time series for Cubic from an iBoxNet instance test matches the real-world ground truth (only known for the control — Cubic) well.

These results show the promise of iBoxNet in effectively learning models corresponding to specific instances and its ability to reproduce time series behaviour (and not merely distributions) on rate and delay. If this promise carries over to actual networks, it would enable powerful counterfactual analysis using iBoxNet (§2).

3.2 Summary

The simplicity of iBoxNet and the use of network domain knowledge to directly estimate the parameters makes both learning the model and running it very efficient. The simplicity, and in particular, the small number of parameters, also mitigates the risk of over-fitting. [2]

The downside of the simple network model assumed in iBoxNet, with a single bottleneck link and FIFO queue, is that it is unable to accommodate network behaviours such as packet reordering, variable bandwidth (e.g., in wireless networks or with a token bucket regulator [38]), etc., which motivates our ML-based approach, iBoxML (§4). Nevertheless, as seen above, iBoxNet produces promising results even in the context of not-so-simple cellular networks.

4 ML-BASED APPROACH

We now turn to a machine learning (ML) based approach to network path simulation, which does not rely on any knowledge or assumptions regarding the inner workings of the network but rather seeks

[2] We plan to release iBoxNet profiles for the community at https://aka.ms/ibox/.

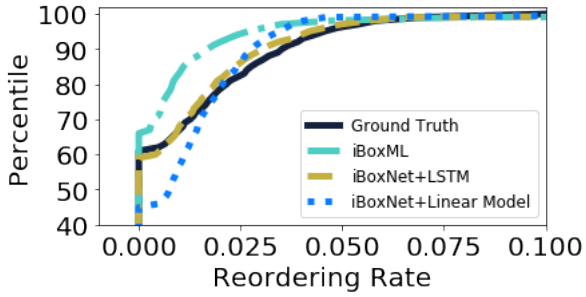


Figure 5: CDF of reordering rate, over 1-sec windows, with the Pantheon test set (Vegas) (see §4.1 and §5.1).

to learn and reproduce the input-output behaviour of the network from data. A paradigm shift is already underway in some domains, e.g. natural language processing, where ML-based systems are beginning to replace traditional modeling approaches [15, 17, 35]. In this vein, we can model the network simulation problem (recall from §2) as that of learning $\mathbb{P}(\text{output}|\text{input})$ where input and output are continuous-valued time series with temporal structure. While this appears like a standard time-series modeling problem, there are certain unique aspects in the networking domain that render naïve application of existing techniques ineffective (§4.2).

An ML-based approach offers some significant potential advantages over a network model-based approach. First, it avoids the limitations and simplifications of an assumed model of the network, such as single-bottleneck in iBoxNet or random packet loss [45] ; instead, the end-to-end learning is data-driven and agnostic to innards of the network. Second, a pure-ML based system derives its power and its ability to generalize (e.g., [35]) to new behaviors from access to rich and diverse training data, high-capacity models, computational infrastructure, and the availability of general-purpose, efficient and scalable optimization techniques, each on a path of continual growth and progress, which ML-based network simulation can leverage "for free". This allows the ML model training to scale well (in terms of training time per parameter), even though the learning problem can be very high-dimensional. In contrast, extending network-based modeling to diverse scenarios (e.g., multi-path routing to recreate packet reordering) and network topologies can quickly become very challenging and computationally prohibitive because of the implicit combinatorial optimization — the search space is over several discrete parameters (e.g. cross-traffic type, buffer size). Using gradient-free optimization to learn the optimal network configuration (in the spirit of [45]) involves deploying updated parameters and running multiple simulations during each iteration of optimization, so it remains computationally expensive and challenging.

However, the ML-based approach calls for designing the "right" loss (i.e., objective) functions and very fast inference (§4.2), and lacks interpretability of the resulting models.

4.1 Sketch of Deep LSTM Model

We consider a state-space model for the delay prediction problem, where we first estimate a sequence of "network states" (albeit domain-agnostic) conditioned on past states, past and present inputs (i.e., packet sending rate), and then predict output (delay/loss)

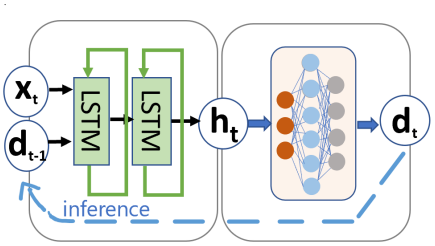


Figure 6: Proposed deep state-space model (iBoxML) for predicting delay d_t given packet stream features x_t.

from a certain delay distribution conditioned on the estimated current state. The goal is to mimic a typical network that, for instance, tends to suffer higher delay/loss when it is in a congested state. A growing area of research in the time-series domain leverages LSTMs [20, 26, 36] to carefully encode autoregressive forecasting models. More sophisticated but often expensive models, e.g. transformers [17], meant for symbolic computation tasks in the text domain (and also beginning to be applied in the time-series domain [42]) could be explored for delay/loss prediction in the future.

At time t, let d_t denote the delay suffered at R by a packet sent from S (d_t is unobserved if the packet is lost). We model the conditional distribution $\mathbb{P}\left(d_t|\{x_i\}_{i=0}^{t}, \{d_i\}_{i=0}^{t-1}\right)$ using a multi-layer LSTM network architecture shown in Fig. 6. The input x_t to the model consists of simple features readily available from the sender packet stream at time t including instantaneous sending rate (the number of packet bytes sent during the second preceding the current packet timestamp t), inter-packet spacing, packet size, and previous delay d_{t-1}, and the output is a real-valued delay (or packet loss indicator) d_t. The LSTM network encodes the input x_t and d_{t-1} into an embedding h_t (representing the "network state" noted above), which in turn parameterizes $\mathbb{P}$.

We model $\mathbb{P}$ as a Gaussian $\mathcal{N}(w_\mu^T h_t, w_\sigma^T h_t)$; the weights w_μ, w_σ are learnt using a fully-connected neural network with a suitable loss function between the predicted delay distribution $\hat{\mathbb{P}}$ and ground truth $\mathbb{P}$. During inference, we feed the *predicted* delays as we unroll the LSTM network over time (blue dashed lines in Fig. 6).

We trained this model on 100 and tested on 60 Pantheon (India Cellular) Vegas flows ("Pantheon dataset" hereafter). Due to speed constraints with emulation §4.2, we tested by replaying the sending rate time series from the test set.

The reordering rates (computed over 1s windows) on the test data is shown in Fig. 5 (iBoxML curve). We find a reasonable match with the ground truth (much better than iBoxNet, which produces no reordering), though the model was trained only to match delays and no explicit knowledge of reordering was provided during training. This underscores the power of an ML approach, informed by data.

4.2 Key challenges

We discuss a couple of key challenges and mitigations.

Control loop bias. Conditioning the delays on the input packet stream alone is fraught with danger, as the data in the networking domain (e.g., from readily-available traces) often comes from protocols, whose sending is governed by a control loop (e.g., congestion control). This results in biases that traditional time-series forecasting formulations do not cope with. For instance, TCP senders carefully regulate their sending rate based on ongoing measurements of

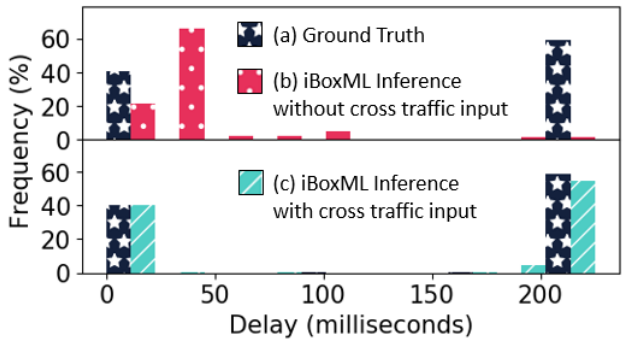
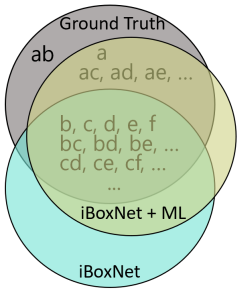


Figure 7: Control loop bias: iBoxML can be blind-sided by partial observations (top, §4.2); using cross-traffic estimates explicitly helps mitigate the bias (bottom).



(a)

Pattern	Ground Truth	iBoxNet+ML
a	2%	1.67%
ca	0.57%	1.26%
ae	0.54%	1.22%
af	0.20%	0.23%
ea	0.59%	0.21%
ad	0.58%	0.11%
fa	0.26%	0.08%

(b)

Figure 8: Behavior discovery on Pantheon traces (§5.1): (a) shows the patterns discovered; pattern 'a' denoting negative inter-packet arrival times, i.e. reordering event, is missing in iBoxNet; (b) shows iBoxNet + ML reasonably preserves the frequency of patterns involving 'a'.

delay and packet loss. So, a sustained low delay, say due to a high bandwidth, uncongested network, would likely lead to a sustained high sending rate. However, this might cause false correlations to creep in, leading us to conclude that a high sending rate *results in* a low network delay, which is clearly incorrect in general.

To illustrate the problem, we train iBoxML with traces of the delay-sensitive control loop of an RTC application on a simple ns-like topology. We then use this iBoxML model to predict delays for a high-rate CBR sender, in the presence of varying amounts of cross-traffic. The ground truth, as expected, exhibits high delay frequently, but iBoxML rarely outputs high delay (Fig. 7, top) due to the control loop bias. Augmenting iBoxML with cross-traffic estimates (from §3) as additional input, helps mitigate the bias (Fig. 7, bottom).

Simulation Speed. Deep learning-based simulation is slow. A 4-layer LSTM in iBoxML, with nearly 2M parameters, requires 2.2 ms per packet inference on a V100 GPU, implying an average data rate of just 5.5 Mbps, with 1500-byte packets; even short bursts of higher rates would be problematic. So, we are unable to use iBoxML for emulation at present. For high-speed links, the inference budget would be in μs, much more demanding than in other domains [15]. iBoxML could be sped up significantly using hybrid models (e.g., combining an accurate but expensive model with a less expensive, even if less accurate, model) and a hierarchical approach (e.g., making a decision for a group of packets instead of each individually).

5 MELDING NETWORK & ML MODELS

Melding network- and ML-based modeling could yield the benefits of each. While the possibilities are many, we discuss two examples, with promising early results.

5.1 iBoxNet + behaviour discovery & learning

While keeping the iBoxNet core simple, we can use ML to augment its ability to simulate complex behaviours (e.g., reordering [11], multi-rate links [24], etc.). The first step is discovering behaviours, both in real traces and in the iBoxNet simulator. A "diff" would surface behaviours present in the former but absent in the latter. A domain expert can then decide which of these is "interesting" and important to recreate in the simulator. This process can be iterated.

We employ a popular tool, SAX [29], which takes a given set of transformed traces (e.g., delay differences), and discretizes the transformed traces into symbolic representations; then, a motif finding algorithm [30] is applied to find frequently occurring segments (i.e., sequences of symbols). As an example, Figure 8 shows the behaviours discovered in the inter-packet arrival times (Δ_t) of the Pantheon test traces, compared to simulated traces of iBoxNet. Here, Δ_t is discretized into symbols 'a' through 'f', with 'a' denoting negative values (i.e., reordering), 'b' denoting small positive values all the way through high positive values 'f'. In Figure 8 (a), we see that the only length-1 pattern in the "diff" between the patterns in ground truth and iBoxNet traces is 'a', arising because packet reordering is (totally) absent in iBoxNet. Consequently, we also observe that higher-order patterns (length-2 sub-sequences) involving 'a' are also totally absent in the iBoxNet traces; but all other length-2 patterns are preserved (seen in the intersection region).

iBoxNet can be augmented with suitable ML models to incorporate such discovered missing behaviours. Indeed, we can easily induce any given packet reordering rate by simply choosing the appropriate number of packets at random and modifying their delays. However, such a naïve method cannot render realistic higher-order patterns. So, we train an LSTM model (similar to that in Fig. 6) to predict whether a packet should be reordered, using Pantheon training traces. We use this prediction to suitably modify the delay output by iBoxNet.

Figure 8 (b) shows that ML-augmented iBoxNet model traces have nearly 2% length-1 patterns of type 'a', pertaining to reordering, matching the ground truth; the augmented model also preserves the frequency of length-2 patterns involving reordering reasonably well. Furthermore, Fig. 5 shows that the reordering rates of the augmented model ("iBoxNet + LSTM" curve) on the Pantheon test traces computed over 1s windows matches the ground truth well. Observing that predicting reordering events is simpler than predicting actual delays and losses, we train a lightweight and much faster linear logistic regression model that also achieves a good match ("iBoxNet + Linear" curve); this model takes instantaneous sending rate (as defined in §4.1), inter-packet spacing and cross-traffic estimate (from §3) as input features and outputs the likelihood of the packet being reordered.

Cross traffic	Error in distribution of 95th percentile delay			
	p25	p50	p75	mean
No	20 (32%)	34 (36%)	63 (45%)	51 (44%)
Yes	3 (5%)	19 (19%)	35 (25%)	30 (26%)

Table 1: Feeding in cross-traffic improves iBoxML accuracy in real-time conferencing data: e.g., p50 column is the difference (in ms) between median of 95th percentiles of inferences and GT delays (on test set).

5.2 iBoxML + cross-traffic input

The importance of cross-traffic has been addressed in the context of iBoxNet (§3) and in mitigating control loop bias (§4.2). But, *does explicitly providing cross-traffic as input help improve the overall quality of* iBoxML *predictions?* Using about 540 traces from a real-time conferencing service, we evaluate iBoxML (Fig. 6), with and without cross-traffic estimates (obtained using domain knowledge, as in §3) as additional input. From Table 1, we note that providing cross-traffic as input reduces the deviation between the distribution of 95th percentile per-call delay values in the ground-truth and in the iBoxML predictions.

5.3 Summary

The above proof points suggest a "recipe" for perpetual renewal of network simulators, involving (i) a continual inflow of new data, (ii) leveraging the latest advances in ML (often from other problem domains) and also advancing the state of the art in ML itself (e.g., time series forecasting in the presence of a control loop), and (iii) leveraging networking domain knowledge to identify behaviors that the simulator should capture, in turn guiding the ML formulation and modeling. Such domain knowledge could also yield constructs (e.g., features) to help boost accuracy and/or speed.

6 DISCUSSION

We touch on a few open research challenges.

Test for Realism. There is no one definition of realism. We could define it in terms of the inability of a powerful discriminator (e.g., of the kind used to train Generative Adversarial Networks (GANs) [19]) to tell between the input-output behaviour of the simulator and that of the real network. Building such a discriminator for time series forecasting remains an open research challenge. Alternatively, we could define realism in terms of the application performance; e.g., whether the performance of an application that has been tuned using the simulator holds up in the actual network.

Establishing the Limits of Model Validity. Training data limits the ability of iBoxML to learn about the network. For instance, if the sending rate in the training data never exceeded a certain level R, even over short periods, it would not be possible for iBoxML to accurately predict the output when the rate does exceed R. Therefore, given the training data, say obtained as a by-product of an existing application (e.g., YouTube streams), establishing the limits of validity of the learnt model is important. Doing so would also help selectively gather new data that would expand the region of validity of the model, thereby optimizing effort.

iBoxNet is also limited by the assumptions it makes about the traces, although it is worth noting that violation of these assumptions will likely only result in a graceful degradation, rather than full invalidation, of the simulator. First, it assumes that the sender tries to saturate the bottleneck, in order to be able to estimate the peak bandwidth. Secondly, it assumes that at some point a packet traverses an empty queue (and similarly an almost full queue) which enables good estimation of propagation delay (and similarly buffer size). Currently, we aggregate data from multiple flows from around the same time between two nodes, which increases the likelihood of these assumptions being satisfied. But we need to verify these assumptions on traces from large-scale applications running on hundreds of thousands of nodes.

Learning adaptive cross traffic. Merely replaying the estimated cross-traffic is not ideal, since it would not account for the cross-traffic adapting to the sender. Learning an adaptive cross-traffic model, say by expressing it in terms of a certain number of flows of TCP Cubic (the dominant transport protocol in the Internet), is an interesting research challenge.

7 RELATED WORK

Packet-level network simulators, both research [4, 5] and commercial [1, 3], and also emulators [13, 14, 22, 39]), take great care to replicate the functionality of network elements and protocols, but often lack realism in configuration (e.g., link speeds, cross-traffic).

Trace-driven replay, both in the Internet at large (e.g., [33]) and in specialized settings (e.g., wireless networks [34]), either replays the raw delay or loss traces during simulation or builds statistical models to drive the replay. However, a significant limitation is that the impact of the protocol under test itself on the network would not be reflected.

Lack of realism can result in misleading and even overly optimistic conclusions. For example, Sprout [41], a bandwidth estimator for cellular networks, was evaluated using Cellsim, a trace replay tool, and was shown to outperform competing approaches. However, Sprout under performs significantly on real networks [43, 44]. Similar conclusions [16] have been made for Pensieve [32], which was originally evaluated through replay of public traces [8, 37].

Distributed network testbeds [6, 7, 28, 45] promise realism, but are challenging to scale up and extend to diverse networks (e.g., mobile) and hence tend to not be representative. It is likely more onerous to recruit or install a diverse set of testbed nodes than it is to gather traces from such nodes (e.g., to train iBox), say by riding on popular applications.

Against these challenges, the need for fast, inexpensive, and realistic evaluation (using ML or statistical models [10, 12]) of network protocols is only growing; more so with the emerging interest in using ML, including reinforcement learning, to learn or tune protocols [18, 21, 25, 32, 40].

Work on calibrated emulators [45] has sought to address this need. While it is related to iBoxNet, it does not model cross-traffic and so is unsuitable for testing protocols different from the ones for which training data is gathered. Also, as discussed, behaviours such as packet reordering require going beyond a simple network model.

Finally, [23] uses deep learning and hardware acceleration to speed up packet-level simulation, but assuming full knowledge of network configuration and traffic, unlike iBox.

8 CONCLUSION

We have presented iBox, a data-informed network simulator, which turns input-output network traces into network models. We have shown that both the network and ML-based approaches to modeling hold promise, and advocate melding the two approaches to enable effective network simulation.

ACKNOWLEDGEMENTS

We thank our shepherd, Justine Sherry, and the anonymous reviewers for their valuable feedback. We also thank Martin Ellis, Vasiliy Novikov, and Sriram Srinivasan for their help and inputs on this project.

REFERENCES

[1] OPNET Technologies. https://www.riverbed.com/in/products/steelcentral/opnet.html.
[2] Kolmogorov-Smirnov test for goodness of fit. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kstest.html.
[3] QualNet - Network Simulation. https://www.scalable-networks.com/qualnet-network-simulation.
[4] The Network Simulator - ns-2. https://www.isi.edu/nsnam/ns/.
[5] ns-3 Network Simulator. https://www.nsnam.org/.
[6] PlanetLab. https://www.planet-lab.org/.
[7] Measurement Lab (M-Lab). https://www.measurementlab.net/.
[8] FCC: Raw Data - Measuring Broadband America 2016. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016.
[9] Understanding Buffer Misses and Failures. https://www.cisco.com/c/en/us/support/docs/interfaces-modules/channel-interface-processors/14620-41.html.
[10] C. Avin, M. Ghobadi, C. Griner, and S. Schmid. On the complexity of traffic traces and implications. In *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '20, page 47–48, New York, NY, USA, 2020. Association for Computing Machinery.
[11] J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking*, 7(6):789–798, 1999.
[12] Y. Cao, J. Nejati, A. Balasubramanian, and A. Gandhi. Econ: Modeling the network to improve application performance. In *Proceedings of the Internet Measurement Conference*, IMC '19, page 365–378, New York, NY, USA, 2019. Association for Computing Machinery.
[13] M. Carbone and L. Rizzo. Dummynet Revisited. *SIGCOMM Computer Communication Review*, 40(2):12–20, 2010.
[14] M. Carson and D. Santay. NIST Net: A Linux-based Network Emulation Tool. *SIGCOMM Computer Commununication Review*, 33(3):111–126, 2003.
[15] M. X. Chen, B. N. Lee, G. Bansal, Y. Cao, S. Zhang, J. Lu, J. Tsay, Y. Wang, A. M. Dai, Z. Chen, et al. Gmail smart compose: Real-time assisted writing. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2287–2295, 2019.
[16] P. Crews and H. Ayers. CS244 '18: Recreating and Extending Pensieve. https://reproducingnetworkresearch.wordpress.com/2018/07/16/cs-244-18-recreating-and-extending-pensieve/, 2018.
[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.
[18] T. Gilad, N. H. Jay, M. Shnaiderman, B. Godfrey, and M. Schapira. Robustifying network protocols with adversarial examples. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, page 85–92, New York, NY, USA, 2019. Association for Computing Machinery.
[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. In *NeurIPS*, 2014.
[20] A. Graves. Long short-term memory. In *Supervised sequence labelling with recurrent neural networks*, pages 37–45. Springer, 2012.
[21] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar. A deep reinforcement learning perspective on internet congestion control. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3050–3059, Long Beach, California, USA, 09–15 Jun 2019. PMLR.
[22] A. Jurgelionis, J.-P. Laulajainen, M. Hirvonen, and A. I. Wang. An Empirical Study of NetEm Network Emulation Functionalities. In *ICCCN 2011: Proceedings of 20th International Conference on Computer Communications and Networks*, pages 1–6. IEEE, 2011.
[23] C. W. Kazer, J. Sedoc, K. K. Ng, V. Liu, and L. H. Ungar. Fast Network Simulation Through Approximation or: How Blind Men Can Describe Elephants. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks - HotNets '18*, pages 141–147, Redmond, WA, USA, 2018.
[24] M. O. Khan, L. Qiu, A. Bhartia, and K. C. Lin. Smart retransmission and rate adaptation in wifi. In *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, pages 54–65, 2015.
[25] Y. Kong, H. Zang, and X. Ma. Improving TCP Congestion Control with Machine Intelligence. In *NetAI*, 2018.
[26] R. G. Krishnan, U. Shalit, and D. Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2017.
[27] H. J. Kushner and P. A. Whiting. Convergence of proportional-fair sharing algorithms under general conditions. *IEEE Transactions on Wireless Communications*, 3(4):1250–1259, 2004.
[28] K. Lakshminarayanan and V. N. Padmanabhan. Some Findings on the Network Performance of Broadband Hosts. In *ACM IMC*, 2003.
[29] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, DMKD '03, page 2–11, New York, NY, USA, 2003. Association for Computing Machinery.
[30] J. Lin, E. Keogh, S. Lonardi, and P. Patel. Finding motifs in time series. In *Proc. of the 2nd Workshop on Temporal Data Mining*, pages 53–68, 2002.
[31] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.
[32] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210, 2017.
[33] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, Santa Clara, CA, 2015.
[34] B. D. Noble, M. Satyanarayanan, G. T. Nguyen, and R. H. Katz. Trace-based Mobile Network Emulation. In *ACM SIGCOMM*, 1997.
[35] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.
[36] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski. Deep State Space Models for Time Series Forecasting. In *Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, Montréal, Canada.*, pages 7796–7805, 2018.
[37] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In *ACM Multimedia Systems (MMSys)*, 2013.
[38] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband internet performance: A view from the gateway. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 134–145, New York, NY, USA, 2011. Association for Computing Machinery.
[39] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, Dec. 2003.
[40] K. Winstein and H. Balakrishnan. TCP ex Machina: Computer-Generated Congestion Control. In *ACM SIGCOMM*, 2013.
[41] K. Winstein, A. Sivaraman, and H. Balakrishnan. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *NSDI*, 2013.
[42] N. Wu, B. Green, X. Ben, and S. O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317*, 2020.
[43] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein. Towards continual learning for networking algorithms. https://platformlab.stanford.edu/Presentations/2019/retreat-2019/KeithWinstein.pdf, 2019.
[44] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein. Learning in situ: a randomized experiment in video streaming. In R. Bhagwan and G. Porter, editors, *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 495–511. USENIX Association, 2020.
[45] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, 2018.