Robustifying Network Protocols with Adversarial Examples

Tomer Gilad
Hebrew University of Jerusalem
tomer.gilad1@mail.huji.ac.il

Nathan H. Jay
University of Illinois at Urbana-Champaign
njay2@illinois.edu

Michael Shnaiderman
Open University of Israel
mickey946@gmail.com

Brighten Godfrey
University of Illinois at Urbana-Champaign
pbg@illinois.edu

Michael Schapira
Hebrew University of Jerusalem
schapiram@cs.huji.ac.il

ABSTRACT

Ideally, network protocols (e.g., for routing, congestion control, video streaming, etc.) will perform well across the entire range of environments in which they might operate. Unfortunately, this is typically not the case; a protocol might fail to achieve good performance when network conditions deviate from assumptions implicitly or explicitly underlying its design, or due to specific implementation choices. Identifying exact conditions in which a *specific* protocol fares badly (though good performance is feasible to attain) is, however, not always easy as the reasons for protocol suboptimality or misbehavior might be elusive.

We make two contributions: (1) We present a novel framework that leverages reinforcement learning (RL) to generate network conditions in which a given protocol fails to perform well. Our framework can be used to assess the robustness of a given protocol and to guide changes to the protocol for making it more robust. (2) We show how our framework for generating adversarial network conditions can be used to enhance the robustness of RL-driven network protocols, which have gained substantial popularity of late. We demonstrate the usefulness of our approach in two contexts: adaptive video streaming and Internet congestion control.

ACM Reference Format:
Tomer Gilad, Nathan H. Jay, Michael Shnaiderman, Brighten Godfrey, and Michael Schapira. 2019. Robustifying Network Protocols with Adversarial Examples. In *The 18th ACM Workshop on Hot Topics in Networks (HotNets '19), November 13–15, 2019, Princeton, NJ, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3365609.3365862

HotNets '19, November 13–15, 2019, Princeton, NJ, USA

ACM ISBN 978-1-4503-7020-2/19/11…$15.00
https://doi.org/10.1145/3365609.3365862

1 INTRODUCTION

Designing protocols for Internet environments is an incredibly difficult task due the vast range of possible network conditions a protocol might potentially encounter. Consider, for instance, the perennial topic of Internet congestion control. Despite almost roughly three decades of research and deployment, the right paradigm for designing protocols that robustly provide high performance remains the subject of contention [1, 3, 6, 22]. Other examples include the design of intradomain and interdomain routing protocols, video streaming protocols, and more.

The huge diversity of possible network conditions implies that even a protocol that works well across a wide variety of network conditions may suffer from bad performance on other networks [29]. Thus, enhancing the robustness of protocols is clearly desirable. This typically involves identifying scenarios that result in poor performance by the protocol and using these scenarios for debugging and for guiding changes to the protocol. Unfortunately, even finding such scenarios can be challenging, for the following reasons: (1) Merely identifying conditions under which a protocol fares badly is typically easy (e.g., no congestion control protocol is expected to achieve high performance when almost every packet is dropped). However, such trivial examples are not interesting because even an optimal algorithm would perform poorly. Instead, meaningful examples should involve network conditions in which a protocol performs far from optimally. (2) The search for challenging network conditions must be customized for the specific protocol under consideration. Conditions under which one protocol fails miserably might be quite good for other protocols. (3) Bad protocol behavior might be triggered by complex sequences of changes in network conditions, making identifying such examples challenging. (4) Ideally, network conditions that have been shown to induce bad performance for a protocol

will also contain hints regarding where the problem lies, i.e., the demonstrated problem should be *explainable*.

Our first main contribution is a novel framework for finding challenging network scenarios for a given protocol. Our framework leverages Reinforcement Learning (RL) to generate adversarial network traces for an input protocol by observing protocol behavior and adaptively changing its network conditions to harm its performance relative to the optimal. We apply our framework to adaptive video streaming protocols as a way to assess their robustness. We also apply our framework to the relatively new BBR congestion control protocol [3] and show that the generated adversarial network traces demonstrate a specific weakness in BBR.

Our second main contribution is showing that adversarial network traces generated by our adversarial framework can be leveraged to train more robust RL-based protocols. RL-based network protocols have recently been applied to improve decision making in adaptive video streaming [17], congestion control [15], routing [26], and more. While RL may be valuable for developing new protocols, the resulting protocols, which often leverage deep learning, can be more difficult to understand, making the scenarios in which they fail potentially harder to characterize and to identify in the real world. We show that the adversarial network traces generated by our framework for RL-based protocols not only expose scenarios in which these protocols might fail to provide the desired performance, but can also be incorporated into the training of such protocols, a promising direction for improving robustness.

2 HIGH-LEVEL APPROACH

2.1 On Generating Adversarial Traces

RL-based adversaries. We formulate the problem of creating challenging network conditions as an RL problem. In RL, an agent repeatedly observes the environment's state, performs an action, and then observes the reward for the performed action [25]. Our goal is to use RL to train an adversary to output network traces that trigger and exploit flaws in existing protocols, ideally in a way that aids in fixing those flaws. Thus, we not only need to induce poor performance in the target protocols, we need to do so in a repeatable fashion, and hopefully in a manner that provides some insight into what protocol flaw is being exploited. We consider two possible approaches for formulating our RL-driven adversary: trace-based and online.

Trace-based *vs.* online adversaries. A trace-based adversary generates an entire trace (a time-ordered list of network conditions like bandwidth, latency and loss rate) as a single output, and is evaluated by running the target protocol on that trace and quantifying its performance. This type of adversary's output would also make results easy to reproduce in testing; simply re-run a trace produced by the adversary and if the adversary has chosen good traces, the protocol will likely perform poorly. However, this approach has two significant drawbacks: (1) depending on the length of trace required to observe the flawed behavior of a protocol, this might result in a very long training process since each trace constitutes only a single data point; and (2) as the adversary cannot react to the protocol at a very fine granularity, it may be difficult to understand what actions by the protocol precipitated certain aspects of the adversary's output.

In contrast, an online adversary takes actions against a protocol by observing the protocol's behavior during operation (in our examples, adversaries make observations every video chunk in video streaming and every 30ms in congestion control), and producing only the next network conditions (e.g., a single tuple of bandwidth, latency and loss rate). As a result, an online adversary can collect training data much faster (because inputs and outputs take less time to evaluate). However, replicating the results of an online adversary might be more difficult; whereas a trace-based adversary should produce traces on which a protocol's average performance is poor, an online adversary might generate a sequence of network conditions that resulted in poor performance only because of a specific sequence of actions that it observed during the run. This may mean that replicating an online adversary's results requires re-running the adversary every time.

In this exploratory study, we use online adversaries for both adaptive video streaming and congestion control. We discuss the specifics of the adversaries at the beginning of each evaluation. We show that traces from these adversaries are sufficient to reproduce flawed performance in a variety of target protocols without having to re-run the adversary.

Quantifying proximity to the optimum. When training a RL agent, the definition of the reward function has a significant impact on the end result. In our case, we would like to find an example where the performance of a protocol drops significantly. This scenario can be easily achieved by a network which drops every packet, which is a trivial and uninteresting example. Thus, we would like the adversary find a network where the protocol could have achieved better performance than it actually got. This means the reward should reflect the difference between the reward the protocol received, and the optimal reward it could have gotten.

Seeking explainable examples. Consider a congestion control protocol that performs badly on a network trace in which network bandwidth, end-to-end latency, and packet loss rate vary within specific ranges of values. Suppose, however, that the same effect could have been achieved by only varying one of these network parameters (e.g., loss rate) and leaving the other network parameters fixed. While both network traces might induce the same effect in terms of degraded performance, the second is more useful for exposing the underlying weakness. Indeed, some protocols, like TCP Cubic [11], are vulnerable to packet losses but not to jitter in latency, whereas other protocols, like BBR [3], exhibit the

opposite behavior. Thus, intuitively, the adversary should only introduce changes to the environment if these trigger bad behavior and avoid injecting unnecessary noise. This is captured in our framework by penalizing the adversary for "non-smoothness" of the network trace.

Remark: Relation to Generative Adversarial Networks (GANs). When the input protocol being tested is itself based on an RL agent, our approach bears similarities to GANs in that two ML schemes train to defeat each other. But our goal is substantially different; GANs are typically useful for generating new data that is indistinguishable from an existing dataset. In our context, we do not already possess traces of challenging network conditions to which GANs might be applied, but aim to create such challenging traces.

2.2 Our Adversarial RL Framework

Given the above considerations, we next describe our RL-based adversarial framework.

Each time step t represents a certain time period in which network conditions remain fixed. At the beginning of time step t, the RL agent (the adversary) observes a state s_t that reflects its past interaction with the target protocol. The adversary then chooses an action a_t that specifies the fixed network conditions at that time step, and receives a reward for the protocol's resulting performance. We elaborate on the states, actions, and rewards, below.

Actions. Actions are tuples of network parameters (e.g., bandwidth, latency loss rate) that influence the target protocol's inputs. The ranges of values from which these parameters are drawn might be constrained so as to reflect real-world conditions, or to investigate the behavior of a protocol under specific network conditions. We refer to a sequence of such tuples generated by the adversary as a *trace*.

States. States can depend on previously produced tuples of network parameters, past inputs to the target protocol, and the protocol's produced actions. The collection of state values observed by the adversary are often referred to as the *input features*.

Rewards. The adversary's reward is as follows:

$$r_{adversary} = r_{opt} - r_{protocol} - p_{smoothing} \quad (1)$$

Equation 1 captures the adversary's goal of outputting network conditions for which the performance of the target protocol $r_{protocol}$ is far from the optimal performance r_{opt}. The $p_{smoothing}$ term penalizes the adversary for producing noisy or high-variance traces, which may be less explainable and thus less useful for protocol development. Each of the three reward terms will be defined later as appropriate for the application domain (adaptive video streaming or congestion control).

2.3 Learning from Adversarial Traces

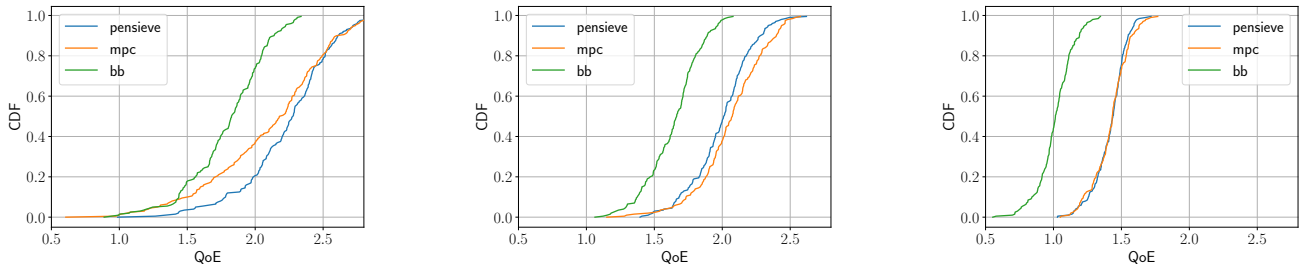
Training a new network protocol using machine learning (ML) typically requires a large dataset of network traces to train on that have a similar properties to environments the protocol will encounter in the real world. Our goal is to utilize our adversarial framework to augment these traces with adversarial traces in the training of RL-based protocols, thereby improving the protocols' robustness to unseen conditions. Conceivably, this approach may apply to many ML-based control systems, like Internet congestion control [15], routing [26], data center network optimization [4, 21, 28], compute resource allocation [7], and more. In this paper, we demonstrate our by enhancing the robustness of Pensieve [17], a recent RL-based protocol for adaptive video streaming.

To avoid over-fitting to adversarial examples, which might be edge cases, we suggest incorporating the generated traces late into the training of a protocol. Our overall training process is as follows: (1) train the protocol of interest, (2) train an adversary against it, (3) use the trained adversary to generate traces, and (4) continue the protocol's training with the new adversarial traces in its training dataset.

3 ADAPTIVE VIDEO STREAMING

Video streaming is one of the most significant uses of today's Internet [5]. Adaptive Bit Rate (ABR) protocols repeatedly download chunks of the video at the resolution (bitrate) expected to maximize the user's Quality of Experience (QoE). QoE must balance considerations such as sending at high bitrates (which take longer to download per chunk), avoiding rebuffering time, and avoiding frequent changes of bitrates [17].

Our adversary is realized by a neural network with two fully connected hidden layers, the first with 32 neurons and the second with 16 neurons. We chose this simple architecture as it fared well in our experiments. Experiments with even simpler architectures (with only one layer or less neurons per layer) yielded lower rewards. Each action of the adversary is a choice of bandwidth in the range of 0.8-4.8 Mbps, which, together with the bitrate, determines the time it takes the targeted protocol to download a chunk. The adversary observes the bitrate chosen by the protocol for the previous chunk, the client buffer occupancy, the possible sizes of the next chunk, the number of remaining chunks, and the throughput and download time for the last downloaded video chunk. The adversary's state is the history of the last 10 observations. The reward for the adversary is as in Equation 1, where r_{opt} is the highest possible QoE over the last 4 network changes, $r_{protocol}$ is the QoE of the protocol over the last 4 network changes, and $p_{smoothing}$ is the absolute difference between the last two chosen bandwidths. The QoE metric chosen is the linear QoE used in MPC [30]: for a video with n chunks, $QoE_{lin} = \sum_{i=1}^{n} R_i - 4.3 \sum_{i=1}^{n} T_i - \sum_{i=1}^{n-1} |R_i - R_{i+1}|$



(a) Traces targeting MPC. (b) Traces targeting Pensieve. (c) QoE CDF on random traces.

Figure 1: Performance of different ABR algorithms for traces by adversary trained against MPC (a), against Pensieve (b), and on randomly generated traces (c).

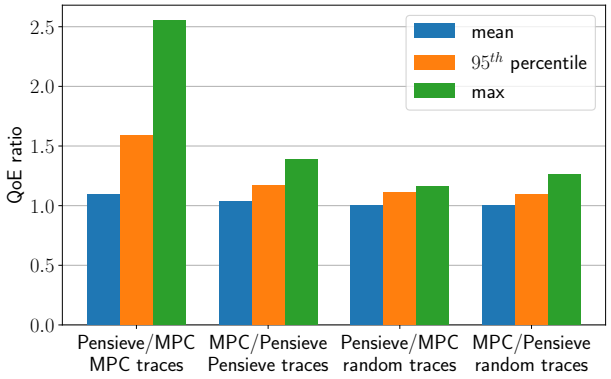


Figure 2: Our adversarial framework generates bad examples for different protocols where a better QoE is attainable.

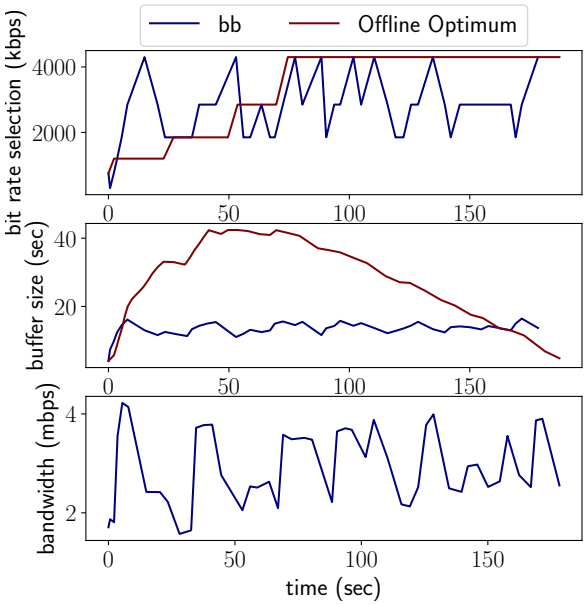


Figure 3: BB running on an adversarial trace.

where R_i is the bitrate of video chunk i, and T_i is the rebuffering time caused by chunk i. The training algorithm used was PPO [23], with the default arguments of the stable-baselines [12] implementation except for the learning rate, which is a constant. We trained the adversary for 600k steps and used the simulator of [17] for training and testing.

3.1 Generating Adversarial Traces

We trained our adversarial framework against a pre-trained model of Pensieve, provided by its authors, and against a re-implementation of the MPC ABR protocol [30], and produced 200 traces for each of the two protocols. We also tested against a re-implementation of a buffer-based (BB) approach, as described in [13], but in all of our experiments it performed worse than all other protocols. As a baseline, we used 200 random traces generated using the same action space as the adversary. Figure 1 shows the per-video QoE CDF using traces targeting MPC and Pensieve, and on the random traces. As expected, the adversary, which is trained to sabotage a specific protocol, does not do that by making the network hostile for all protocols. Figure 1 also shows that while some randomly generated network traces can cause a protocol to misbehave, these traces do not necessarily allow another protocol to succeed.

Figure 2 plots the QoE values of different protocols on these traces. Our adversary found traces where MPC achieves 1.38× the QoE provided by Pensieve, and traces where Pensieve achieves 2.55× the QoE provided by MPC. For both protocols, in over 75% of the adversary's traces, the targeted protocol (MPC or Pensieve) performed worse than the other protocol (Pensieve or MPC), showing that the adversary is effective at finding targeted suboptimal performance. The results over the randomized traces indicate that finding bad examples by using randomly generated traces is harder and the resulting examples are not as bad as those generated by our framework.

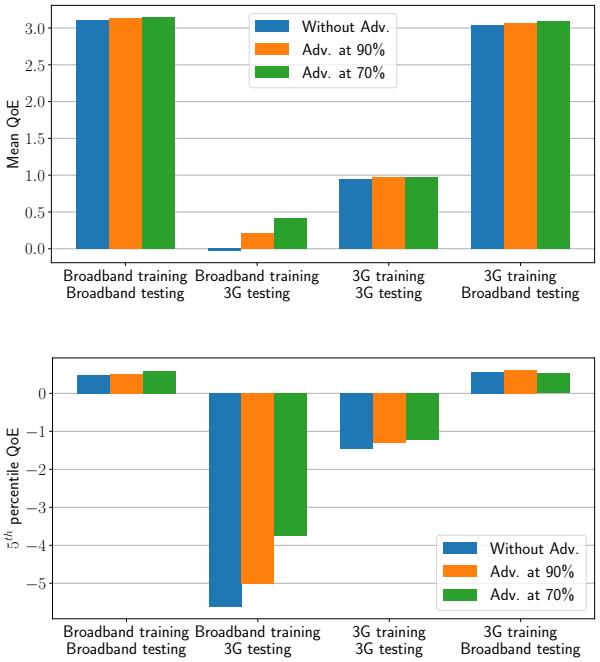


Figure 4: Improvements in QoE with adversarial training in the mean (top) and in the 5^{th} percentile (bottom).

3.2 Finding Weaknesses in Buffer-Based

To illustrate how our adversarial framework can be utilized for pointing out weaknesses in protocols, we trained the adversary against BB. Figure 3 shows the outcome of running BB on the adversarial network trace. This can be interpreted as follows: The adversary identified that BB tries to maintain a playback buffer at the client of size at least 10 seconds, and that BB changes its rate when the buffer size is in the range of 10-15 seconds. The trace outputted by the adversary keeps the buffer in this range, causing BB to constantly change its choice of bitrate and pay a price in smoothness and quality. Note that the best strategy for this network environment is using a low bitrate at the start and gradually increase it.

3.3 A Better Pensieve

We use the training process described in 2.3 to show how Pensieve can be made more robust to unseen network traces. We train Pensieve once on the FCC broadband traces [8] and once on the 3G/HSDPA mobile dataset of traces collected in Norway [19]. We pause the training after 90% of the training iterations, add the traces generated adversarially to the partially-trained model into the training set of Pensieve, and continue its training. We also show results when the adversarial traces are introduced after 70% of the training iterations instead of 90%.

Figure 4 shows the improvement in QoE when training Pensieve with adversarial traces, comparing to Pensieve trained without these traces on the same dataset. Training using the adversarial traces improved the QoE across all test sets. The most notable improvement is when training on the FCC broadband traces and testing on the 3G Norway traces, which reflects how the FCC dataset lacks the challenges found in 3G networks. Incorporating the adversary earlier in the training helped to generalize better as Pensieve had more iterations to adjust itself. The main improvement in the QoE was at the bottom 5^{th} percentile, e.g., 1.22× improvement when training and testing on the broadband dataset. This is consistent with the adversary having produced particularly challenging examples for Pensieve.

4 INTERNET CONGESTION CONTROL

Congestion control determines how much data each communication endpoint injects into the network and how quickly. Despite decades of effort, robustly providing high performance in congestion control remains unsolved [1, 3, 6, 29].

TCP congestion control variants like Cubic, Reno and HTCP all share a trivial weakness to packet loss even as low as 1%. However, recently proposed protocols such as BBR [3], PCC Vivace [6], and Copa [1] do not not have as clear weaknesses. We take BBR as a case study. We devise an adversary whose action space is entirely within BBR's intended design range and show that such an adversary can still create patterns of network conditions on which BBR performs poorly.

After testing several neural network architectures for the adversary (including up to three hidden layers and up to 32 neurons wide), we chose a simple neural network with only one hidden layer of 4 neurons, on the grounds that the the simplest neural network that works well is best to illustrate our methods. We feed the adversary two inputs: current link utilization and current queuing delay. The adversary is given control over link bandwidth, latency and random loss rate at a granularity of 30 milliseconds. The available choices of each parameter are shown in Table 1. The adversary's reward is $1 - U - L - 0.01 \cdot S$, where U is the link utilization, L is the loss rate chosen by the adversary, and S is a smoothing factor computed based on the difference between the current bandwidth and latency, and an exponentially-weighted moving average of both bandwidth and latency. We implement our adversary in a modified version of Mahimahi [18], which relies on an event-based approach to packet delivery. Because this environment is not designed for precise timing, and the events that drive Mahimahi are packet deliveries, our traces are not usually identical when played multiple times. Still, we show that our adversary learns to significantly reduce BBR's throughput. We train our adversary using PPO [23] for around 600k action/observation pairs of 30 ms each, split into 200 training iterations.

Bandwidth	Latency	Loss rate
6-24 Mbps	15-60 ms	0-10%

Table 1: Range of link parameters produced by adversary.

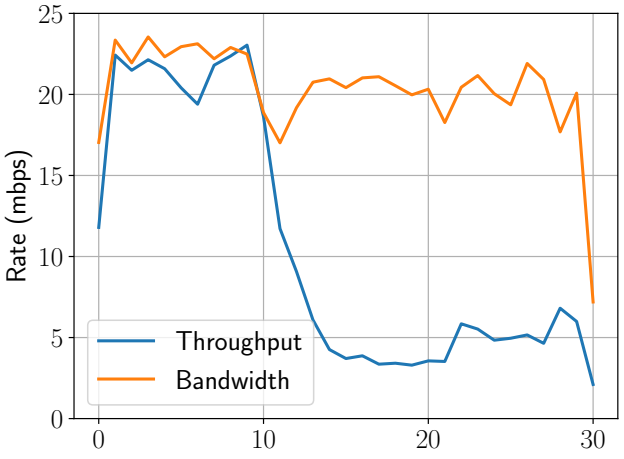


Figure 5: The BBR congestion control protocol running on a 30-second adversarial trace.

Despite the constraints on our adversary in Table 1, which are clearly within BBR's expected design range, our adversary can reduce BBR's average throughput to just 45 – 65% of link capacity by exploiting a critical weakness in BBR: the infrequent, but performance-critical probing.

Figure 5 shows a trace of BBR's performance against a 30-second adversarial trace. BBR starts with high link utilization and suddenly begins sending much more slowly. We can use an action trace of the adversary running online (instead of as a trace) to help us understand why. When running online, our adversary's actions are derived from direct observation of the protocol, which is useful for identifying weaknesses. Figure 6 shows the adversary's deterministic actions (i.e., before exploration noise from training is added) over a 30 second trace, split into 1000 intervals of 30ms. The rapid fluctuations in bandwidth and latency correspond exactly to the probing phases of BBR, and cause BBR to choose a very low sending rate. Note that the raw actions of the adversary may appear to be outside of the parameter range, but exploration and clipping done by PPO will return the actions to the acceptable range.

The trace of BBR's performance combined with the action trace of the adversary can already tell us a great deal about a weakness in BBR, even without looking at the code. This tool is clearly preliminary, and additional information taken from the source code of BBR would obviously be necessary to engineer a more robust version, but even rudimentary

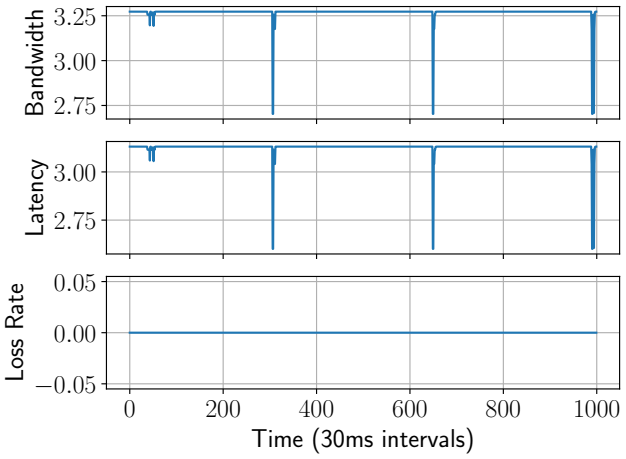


Figure 6: The actions of the BBR adversary over 30 seconds (1000 intervals of 30ms) without training noise. Every 10 seconds, when BBR runs its probing phase, the adversary suddenly varies bandwidth and latency.

knowledge of how BBR operates and our automatically generated trace could point a developer in the right direction.

5 DISCUSSION

Our work presents an early effort to use ML as a way to generate test inputs for network protocols and even train more robust ML-driven protocols. However, this early effort leaves open many questions, as discussed below.

Constraining Adversaries: To generate useful adversarial traces, our adversaries had action spaces constrained to realistic values, and we included the smoothing factor in the adversary reward to avoid unnecessary fluctuations in network conditions. Having a more sophisticated way of constraining an adversary to actions similar to those recorded in the real world might help us create more realistic, but still challenging conditions. Developers might also be interested in constraining adversaries relative to a particular set of traces, e.g., to making only small changes to an existing test case.

Different adversarial goals: Our work uses QoE and link utilization as the basis for adversary reward in adaptive bit rate video streaming and congestion control, respectively, but the adversary's goals can be related to any behavior of the protocol. For example, the congestion control adversary could be given a goal of finding conditions in which the protocol *causes* the highest amount of congestion. Likewise, an ABR adversary could be created with the specific goal of causing rebuffering or low bit-rate playback. Specific goals like these might yield better insights about protocol behavior than general goals, like minimizing QoE. Adversaries trained in other contexts to cause route flapping, BGP leaks, or incast might be useful since such problems generally occur rarely, but represent a significant problem when they do occur.

Guiding protocol development: We think that there are rich possibilities for using RL adversaries to assist humans in protocol development. Consider the case of continuous integration, where the protocol is changed over time, but it is desirable that all previously-fixed problems remain fixed. In such a case, using an adversary to create inputs that cause the exact problem in question, instead of running a fixed set of traces that caused problems in an earlier version of the code, would help developers create a more robust fix.

6 RELATED WORK

Testing the performance of network protocols has long been an important part of networking, because applications using the network rely on high performance to meet key quality metrics like latency, rebuffering and download time. As a result, a vast collection of simulators [14, 20] and emulators [18, 27] provide a means to test many network protocols. Pantheon [29] runs a variety of congestion control algorithms on real-world paths and emulated paths intended to reflect real world scenarios. However, Pantheon does not identify conditions that cause undesirable behavior in as-yet-unobserved circumstances. In addition, when undesirable behavior is observed in Pantheon, it may not be as easy to understand what part of a trace was important to the bad behavior. Running algorithms in the real world as Pantheon does may also significantly slow down the development of RL-based algorithms like Pensieve [17], which can be trained much faster in a simulator or against an adversary. In general, our work should not be seen as a replacement for running real-world tests or trace-based tests in simulation or emulation. Instead, our work should be regarded as a way to generate a broader set of test cases to improve robustness and understand a protocol's flaws.

Using ML to generate complex test input (such as a network trace, in our case) is not a new idea. For example, recent work has integrated ML and test input generation in the domain of fuzz testing for PDF parsers [9]. Using ML to generate tests for software dates back more than a decade [24]. [2] uses *supervised* learning to aid in testing network protocols for security flaws. While such an approach may work when researchers have a reasonably universal way to model all possible implementations of a protocol, as in an extremely constrained security context, it does not extend well to cases where protocols may be implemented in a wide variety of ways. Additionally, this method can be used to test correctness, but does not appear to have a straightforward extension for testing performance.

Our approach bears similarities to both generative adversarial networks (GANs) [10], and to competitive multi-agent environments [16]. GANs are similar to our approach for enhancing the robustness of ML-based protocols in that two ML schemes train to defeat each other with the ultimate goal of producing just one successful scheme. GANs, however, represent a *supervised learning* approach, and it is not clear how this can be extended to our context (problems to be solved include proper choice of loss and contending with delayed rewards). Our approach can involve two interacting RL-based agents (the adversary and the target RL-based protocol). In this respect, our work is more similar to ideas like self-play in competitive multi-agent learning, in which two agents sequentially act. However, we train the adversary only as a means to improve the RL-based protocol.

7 CONCLUSION

Our work presents a novel framework for testing network protocols on adversarial traces produced via RL. We showed that these traces can be used in several interesting ways, including identifying conditions in which a protocol performs poorly, gaining insights into algorithmic weaknesses of existing protocols, and improving RL-based algorithms.

We thank the Israel Science Foundation and Huawei for support of our work.

REFERENCES

[1] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical delay-based congestion control for the Internet. In *15th USENIX Symposium on Networked Systems Design and Implementation*. 329–342.

[2] L. C. Briand. 2008. Novel Applications of Machine Learning in Software Testing. In *2008 The Eighth International Conference on Quality Software*. 3–10. https://doi.org/10.1109/QSIC.2008.29

[3] 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

[4] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. 2018. AuTO: Scaling Deep Reinforcement Learning for Datacenter-scale Automatic Traffic Optimization. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. ACM, New York, NY, USA, 191–205. https://doi.org/10.1145/3230543.3230551

[5] Cisco. 2016. Cisco Visual Networking Index: Forecast and methodology, 2016-2021. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html. (2016).

[6] 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*. 343–356.

[7] F. Farahnakian, P. Liljeberg, and J. Plosila. 2014. Energy-Efficient Virtual Machines Consolidation in Cloud Data Centers Using Reinforcement Learning. In *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 500–507. https://doi.org/10.1109/PDP.2014.109

[8] Federal Communications Commission. 2016. Raw Data - Measuring Broadband America 2016. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016. (2016).

[9] Patrice Godefroid, Hila Peleg, and Rishabh Singh. 2017. Learn&Fuzz: Machine Learning for Input Fuzzing. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*. IEEE Press, Piscataway, NJ, USA, 50–59. http://dl.acm.org/citation.cfm?id=3155562.3155573

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014.

Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[11] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review* 42, 5 (2008), 64–74.

[12] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2018. Stable Baselines. https://github.com/hill-a/stable-baselines. (2018).

[13] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[14] Teerawat Issariyakul and Ekram Hossain. 2010. *An Introduction to Network Simulator NS2*. Springer Publishing Company, Incorporated.

[15] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In *International Conference on Machine Learning*. 3050–3059.

[16] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*. 6379–6390.

[17] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 197–210.

[18] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *USENIX Annual Technical Conference 2015*. Santa Clara, CA.

[19] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. 2013. Commute path bandwidth traces from 3G networks: analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, 114–118.

[20] G.F. Riley and T.R. Henderson. 2010. The ns-3 Network Simulator. (2010).

[21] Saim Salman, Christopher Streiffer, Huan Chen, Theophilus Benson, and Asim Kadav. 2018. DeepConf: Automating Data Center Network Topologies Management with Machine Learning. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*. ACM, 8–14.

[22] Michael Schapira and Keith Winstein. 2017. Congestion-control throwdown. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 122–128.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[24] G. Shu and D. Lee. 2007. Testing Security Properties of Protocol Implementations - a Machine Learning Based Approach. In *27th International Conference on Distributed Computing Systems (ICDCS '07)*. 25–25. https://doi.org/10.1109/ICDCS.2007.147

[25] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[26] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. 2017. A Machine Learning Approach to Routing. *CoRR* abs/1708.03074 (2017). arXiv:1708.03074 http://arxiv.org/abs/1708.03074

[27] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. 2002. An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 255–270.

[28] Xiaojie Zhou, Kun Wang, Weijia Jia, and Minyi Guo. 2017. Reinforcement learning-based adaptive resource management of differentiated services in geo-distributed data centers. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. 1–6. https://doi.org/10.1109/IWQoS.2017.7969161

[29] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 731–743. https://www.usenix.org/conference/atc18/presentation/yan-francis

[30] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 325–338.