NetBOA: Self-Driving Network Benchmarking

Johannes Zerwas, Patrick Kalmbach, Laurenz Henkel
Technical University of Munich, Germany

Gábor Rétvári
Budapest University of Technology and Economics, Hungary

Wolfgang Kellerer, Andreas Blenk
Technical University of Munich, Germany

Stefan Schmid
Faculty of Computer Science, University of Vienna, Austria

ABSTRACT

Communication networks have not only become a critical infrastructure of our digital society, but are also increasingly complex and hence error-prone. This has recently motivated the study of more automated and "self-driving" networks: networks which measure, analyze, and control themselves in an adaptive manner, reacting to changes in the environment. In particular, such networks hence require a mechanism to recognize potential performance issues.

This paper presents `NetBOA`, an adaptive and "data-driven" approach to measure network performance, allowing the network to identify bottlenecks and to perform automated what-if analysis, exploring improved network configurations. As a case study, we demonstrate how the `NetBOA` approach can be used to benchmark a popular software switch, Open vSwitch. We report on our implementation and evaluation, and show that `NetBOA` can find performance issues efficiently, compared to a non-data-driven approach. Our results hence indicate that `NetBOA` may also be useful to identify algorithmic complexity attacks.

CCS CONCEPTS

• **Networks** → **Network performance evaluation**; **Network experimentation**; Network security; Network reliability; • **Computing methodologies** → *Machine learning*.

KEYWORDS

self-driving networks, automated network measurements, automated performance analysis, bayesian optimization

ACM Reference Format:
Johannes Zerwas, Patrick Kalmbach, Laurenz Henkel, Gábor Rétvári, Wolfgang Kellerer, Andreas Blenk, and Stefan Schmid. 2019. NetBOA: Self-Driving Network Benchmarking. In *2019 Workshop on Network Meets AI & ML (NetAI'19), August 23, 2019, Beijing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3341216.3342207

1 INTRODUCTION

Motivated by the complex, manual, and error-prone operation of today's communication networks, as well as the increasing dependability requirements in terms of availability and performance, the network community is currently very much engaged in developing more automated approaches to manage and operate networks. A particularly interesting vision in this context are *self-driving networks* [10, 17]: rather than aiming for specific optimizations for certain protocols and objectives, networks should learn to drive themselves, maximizing *high-level* goals (such as end-to-end latency), in a "context-aware", *data-driven* manner. At the heart of such self-driving networks hence lies the ability to adaptively measure, analyze, and control themselves. While over the last years, many interesting first approaches have been proposed related to how self-driving networks can control themselves [4, 10, 16], less is known today about how self-driving networks can analyze and measure themselves efficiently.

This paper makes the case for a data-driven approach to measuring and evaluating the performance of a network in an adaptive, self-driving manner. Indeed, existing work on automated benchmarking and fuzzing systems either *(i)* targets *general computing systems* and hence may not lend itself readily to be adopted in a networked setting [19, 22, 24, 26, 30], *(ii) aims at verifying logical properties* in networked systems related to policy-compliance of configurations and implementations and ignore performance [3, 21], or *(iii) requires human assistance* and *software source code* [14, 25] to guide the performance evaluations and experiments, relying on hand-crafted, and often proprietary, benchmark tools, inputs, and system settings [5, 18, 20]. We argue that *in the context of self-driving networks the performance evaluation tool itself must also be self-driving*, taking into account the specialties of networked systems and the environment these systems are typically used in.

In this paper, we take the position that *machine learning* should become fundamental constituent in the overall network measurement process, in order to allow fast, robust, and unassisted performance evaluation of black-box networked systems. In the context of self-driving networks, there is a number of challenges that lead us to turn to a machine learning approach. First, a network system is typically stateful and runs on top of a network substrate, comprising black-box network devices, proprietary software, virtualization tools, etc., which is itself a complex and stateful system. Correspondingly, *the useful components of network benchmarks are often masked by disturbances* of unknown source, difficult-to-explain performance artifacts, and general white noise [18]. A machine learning search approach has the potential to automatically adapt to such disturbances. Second, *the vastness of the parameter space*, including static configuration parameters, software versions, hardware platforms, etc., that may critically affect the performance of a networked system *may prevent comprehensive manual performance benchmarking*. A machine-driven approach on the other hand alleviates the network operator from manually testing through the vast parameter and state space, eliminates the human from the loop and guarantees unbiased results, and, critically, *fosters reproducibility*. Third, a *networked system requires valid input traffic*, imposing stringent constraints on the type and characteristics of the input that can be posed to such a system, which is in contrast with conventional performance fuzzing that may use arbitrary inputs [19, 22, 24, 26, 30]. Fourth, in the context of self-driving networks, where the network continuously measures and adapts itself, a means is required that can thoroughly evaluate an updated component to ensure its correct functioning. Especially in the case where machine learning is used to make the updates, an automated process is required that can identify possible weak-spots and thus avoid potential performance degradation. In such a setting, weak-spots must be identified as fast as possible to minimize the duration of an update cycle of a self-driving network.

Contributions. This paper proposes NetBOA, an automated network traffic generator for creating "adversarial" workloads challenging implementations of black-box network entities like middleboxes, software and hardware switches, or other network functions. NetBOA is designed to explore configurations of traffic workloads[1] that affect critical metrics in the networking context, such as network latency or CPU usage[5, 18, 20].

In order to account for complex (and as we will show: non-proportional) performance aspects, NetBOA does not assume any functional relation between workload and performance, unlike most machine learning applications. In particular, this rules out gradient-based optimization schemes. Rather, NetBOA performs black-box optimization, using a Bayesian Optimization algorithm, which has already been successfully applied in other contexts, e.g., the configuration of cloud-based workloads [1]. In other words, NetBOA is a data-driven approach that guides the search for weak-spots through the overall configuration space by taking a system feedback, e.g., the current CPU load, into account.

We demonstrate the feasibility of our approach, using a case study with Open vSwitch (OvS), one of the most widely used software switches in data centers. By applying NetBOA to benchmarking OvS, we find that weaknesses in the default configuration of OvS can be found efficiently, providing new insights compared to prior literature [2, 7, 13, 15, 20, 27, 28].

Succinctly, we make the following contributions:

- We propose a machine-driven traffic generation framework, NetBOA, for analyzing implementations of network entities (switches, network functions, etc.). NetBOA relies on black-box (Bayesian) optimization, accounts for noise, and provides confidence intervals.
- We implement and evaluate NetBOA for a specific case study: benchmarking Open vSwitch. Our proof-of-concept implementation shows the benefits of our approach over non-data driven approaches.

[1]I.e., NetBOA "wriggles", like a snake, through traffic configurations.

In order to ensure reproducibility and facilitate follow-up work, we will make our measurement data available to the research community, together with this paper [2].

Organization. The remainder of this paper is organized as follows. Sec. 2 introduces our framework, and with it the challenges behind designing machine-driven measurement frameworks. Sec. 3 outlines the measurement results taken from a real testbed. Sec. 4 reports on related work. Finally, we conclude and propose future work in Sec. 5.

2 THE NETBOA FRAMEWORK

NetBOA addresses the following problem: for a given network entity (network function like a firewall or software switch), the goal is to find a weak-spot, i.e., an optimal or near-optimal network traffic configuration that maximizes (or minimizes) a network measure (e.g., latency or CPU). Given the huge amount of settings for potential configuration parameters, as well as the possible inter-arrival times between packets, this is a non-trivial task. Hence, NetBOA uses Bayesian optimization to intelligently explore the configuration space. Although alternatives to Bayesian optimization such as reinforcement learning, deep neural networks, linear regression might exist to automatically learn performance models, they might show one or more of the following shortcomings: not generalizing due to non-linear performance behavior, not targeting at minimizing number of samples, requiring a lot of data [1].

2.1 Black-box Optimization

A key challenge addressed by NetBOA is that the dependency between the input parameters (including workload) and performance may be highly non-trivial. In fact, as we will see, the performance may not even improve monotonically with lower workload. Furthermore, the objective (or performance) function can be multimodal or noisy. Also perceiving exact performance values, e.g., from CPU usage, is hard. Additional challenges are introduced due to (noisy) timings.

Accordingly, simple gradient-based approaches are not applicable, which is why NetBOA uses black-box optimization. That is, NetBOA does not assume any information about how the objective (performance) function $f(x)$, e.g., the average CPU value of a benchmarked entity, depends on the input x, e.g., the network traffic configuration parameters, and how the x is structured. On a high level, it proceeds as follows: by querying the system, NetBOA aims to better estimate the model describing the objective function $f(x)$. Given a predefined budget (e.g., a time limit or limited amount of iterations) for triggering the system queries, a data-driven algorithm may be able to find near-optimal or optimal values x.

While NetBOA queries the network function for the objective function $f(x)$, it needs to choose the next point of evaluation. The goal is generally to find a mode of a function with as little function evaluations as possible. Whereas different alternatives (algorithms) exist to solve this problem, NetBOA uses Bayesian Optimization (BO) for its guided search. The strength of BO, in contrast to approximate gradient-based or population based approaches, is its ability to calculate confidence intervals of the objective function given the samples. Generally, BO needs a prior function to represent its

[2]https://github.com/tum-lkn/netboa-data

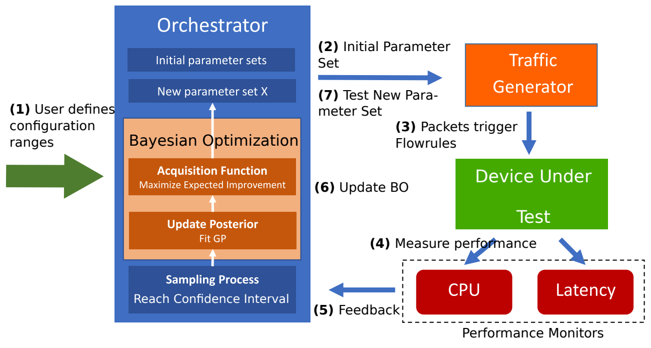


Figure 1: NetBOA' overview. A user defines the overall configuration ranges. After initial sampling, the orchestrate triggers a traffic generation process to send traffic to a device under test. CPU or latency can currently be used as performance measures.

belief in $f(x)$ given the current samples. Here, we use Gaussian Process (GP) as most BO approaches in literature. After querying the system, i.e., perceiving a new sample, BO updates this function and obtains the posterior function. The posterior function always depends on the perceived data, i.e., is purely data-driven. In order to proceed, BO uses an acquisition function, which simply determines the potentially next best point given the updated posterior function. In a nutshell, acquisition functions trade-off exploitation (where the posterior shows high values) and exploration (where the uncertainty about the current model is still high). Whereas many options in literature exist, we use the Expected Improvement (EI) due to its generally good performance.

2.2 NetBOA Algorithm

Alg. 1 summarizes the overall procedure of NetBOA. The main loop (Line 1-9) consists of the use of the BO procedure to find the configurations and the triggering of the measurement procedure (Line 10-18). NetBOA takes ranges of network traffic configurations as input and produces an optimal or near-optimal configuration as output:
Input: NetBOA can take as input all the possible configuration ranges of IP network packets: e.g., the IP address ranges for source and destination IPs, the ranges of application ports (TCP and UDP), the ranges of VLAN tags, etc. Moreover, NetBOA takes as input all parameters that specify the traffic pattern over time: the inter-arrival time between packets, the burstiness of the network traffic, the overall sending time interval, and the total amount of network packets to be sent. Given such configurations and a budget (as described in the previous section), NetBOA tries to find the network traffic configuration that maximizes (or minimizes) a target measure, e.g., CPU utilization of a network function or end-to-end latency of network packets. In general, the configuration possibilities are only limited by the underlying physical network infrastructure. However, a human operator might have to specify whether, e.g., VLAN tags can be used or not for the targeted infrastructure. Hence, the human is still needed in this pre-configuration process of NetBOA.

Despite the network configuration parameters, NetBOA has three algorithm configuration parameters: the numInitialSamples, the

Algorithm 1: NetBOA Measurement Procedure

```
Input: Network Traffic Configuration Ranges: $x_{iat}$, $x_p$, $x_b$, $x_{VLAN}$,
       Measurement Procedure: numInitialSamples, maxNumIterations,
       confidenceLevel
Output: Optimal or Near-optimal Traffic Configuration: $\mathbf{x}_{opt}$
1  NetBOA main()
2      $\mathcal{D} \cup \{createInitialSamples(\text{numInitialSamples})\}$;
3      GaussianProcess.fit($\mathcal{D}$);
4      for maxNumIterations do
5          nextConfig $\leftarrow arg\,max_x EI(\mathbf{x}|\mathcal{D})$ ;
6          objVal $\leftarrow$ RunMeasurement(nextConfig) ;
7          $\mathcal{D} \leftarrow \mathcal{D} \cup$ (nextConfig, objVal) ;
8          GaussianProcess.fit($\mathcal{D}$);
9      return $\mathbf{x}_{opt}$;
10 RunMeasurement proc(nextConfig)
11     currentConfidence $\leftarrow \infty$ ;
12     for count = 0; count < 12; count++ do
13         objValues $\leftarrow$ objValues $\cup$ takeObjAfterTime(1 s) ;
14     currentConfidence $\leftarrow$ calculateConfidence(objValues) ;
15     while currentConfidence $\geq$ confidenceLevel do
16         objValues $\leftarrow$ objValues $\cup$ takeObjAfterTime(1 s) ;
17         currentConfidence $\leftarrow$ calculateConfidence(objValues) ;
18     return objVal;
```

maxNumIterations, and the confidenceLevel. For our proof-of-concept, we evaluated different settings and report on the ones that worked best; these parameters, however could be tuned further or even adapted dynamically, which we study in future work.
Procedure and Outputs: NetBOA starts with creating initial samples in order to make a first fit of the GP (Line 2). For an initial fitting, at least two points are necessary. After that, NetBOA starts with Bayesian optimization. In every iteration (and also after the final iteration), the output of NetBOA is a sequence of N concrete network packets over a time-interval T; both N and T are configurable parameters. This sequence of packets is then sent to a network entity, e.g., a software switch, that processes the network packets. As already stated, in our scenario, we keep track of all performance metrics, e.g., CPU. Those metrics are then used to evaluate the generated workload. Given the measurements of M intervals, NetBOA uses BO to find the next better network traffic configuration, $M+1$. Based on the BO algorithm, NetBOA prioritizes configurations leading to (adversarial) workloads resulting in high operator costs, e.g., high CPU or high latency. Note that NetBOA can also be used to minimize networking costs.
Proof-of-concept. For the sake of demonstrating feasibility, in our initial study, we consider only a limited number of configuration parameters, however, with interval ranges, i.e., unlimited opportunities of concrete values. Concretely, NetBOA allows to configure the network traffic configuration vector $\mathbf{x}$. Here, vector $\mathbf{x}$ may consist of the Inter-Arrival Time (IAT) between packets x_{iat}, the number of total packets with unique IP source/dst and unique src/dst ports to be sent within one measurement round x_p, the burst size of packets sent at one point of time x_b, and the number of different VLANs within one sending round x_{VLAN}. Note that for different network functions, other settings can be chosen.

3 CASE STUDY: OPEN VSWITCH

We implemented a proof-of-concept of NetBOA for a specific case study: benchmarking OvS. We compared our experimental results using NetBOA to a Random Search (RS) procedure. In detail, for a

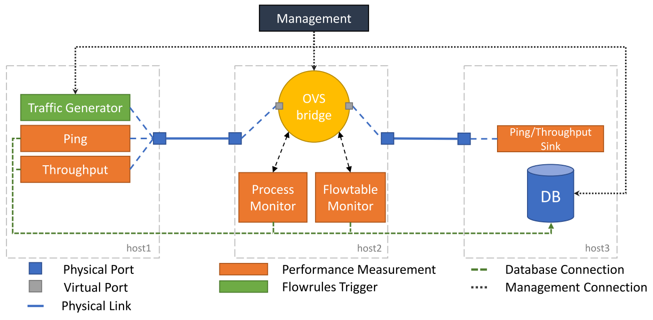


Figure 2: Experimental setup consisting of three hosts and the management running NetBOA.

pre-determined parameter setting, we run NetBOA and RS in order to find the most challenging workload configuration, given a limited number of iterations. Note that comparing BO-based approaches to procedures such as random search or grid search is common in the BO research area in general [11], and also when applying BO to networking problems [1].

This section first presents the experimental setup, provides more details on the settings of NetBOA, and describes the grid measurement data (cf. Fig. 4) and the evaluation setting when comparing NetBOA to Random Search. The grid search is an extensive measurement of the OvS; it shows the performance behavior of OvS over the whole configuration space. The performance evaluation of the comparison reports on details for settings with 1-4 parameters, the impact of the number of iterations, and the likelihood of finding close to optimal configurations with NetBOA.

3.1 Performance Evaluation Methods

3.1.1 Experiment Setup. Fig. 2 shows the measurement setup. The setup consists of three hosts with an Intel i7-4790 CPU at 3.60 GHz with 16 GB of RAM. The OS is Ubuntu 18.10 with kernel 4.15.0-43. The left machine runs the traffic generator. A simple ping client is used to measure the latency on the data path. The machine in the middle runs the Open vSwitch instance (version 2.9.0). OvS is used in its default configuration: the megaflow caches are enabled. OvS is connected with two bridges to the left and the right host. Two rules are manually inserted: one for forwarding ICMP traffic and one for dropping all other traffic. This rule setup is similar to [6]. A CPU monitoring tool is directly fetching the CPU time of the OvS process. The third host provides the sink for the ping client. Additionally, it runs the database to store all measurement results. A fourth machine hosts the NetBOA framework.

3.1.2 NetBOA Settings. This section describes the settings of the main parts of NetBOA, including the three main settings of the Bayesian Optimization: prior function, acquisition function, and the stopping conditions.

Prior function. GP is chosen as the prior function similar to many other BO approaches [1]: the final model (CPU vs network traffic configurations) can be described by a sample from a trained GP instance (i.e., after having fitted the GP to the acquainted measurement data). The Kernel of the GP is a Matern with smoothness parameter between 0.5 and 1.5.

Acquisition function. Generally, different possibilities for the acquisition function exist [1]: Probability of Improvement (PI), Expected Improvement (EI), Gaussian Process Upper Confidence Bound (GP-UCP). Similar to [1], NetBOA uses the EI acquisition function, as it performs mostly better than PI and does not need tuning in contrast to GP-UCP [1].

Stopping conditions. Different alternatives exist to stop the measurement procedure. For instance, the process can stop when the EI value reaches a pre-determined threshold, like 10 %, or when a maximum number of iterations (e.g., 50 or 100) is reached. For NetBOA, the process stops either when 100 iterations are reached or when the current maximum lies within 10 % of the true maximum value (which we know from the grid measurement), to speed up the measurement procedure.

3.1.3 Grid Measurement Data Collection. Beside our evaluation of NetBOA, we collected data from a grid measurement over the whole configuration space for two parameters. This data leads to a complete performance model describing OvS. We provide access to the data to enable other researchers to compare their implementations of data-driven optimization algorithms to NetBOA using the data.

The data was collected in the testbed from Fig. 2 with a grid search iterating over two parameters of the traffic generator: IAT from 1 ms to 14 ms with step-size 0.1 ms and number of packets in one sending round from 1000 to 5000 with step-size 100.

For every point, we generated a sequence of packets according to the configuration and repeatedly sent this sequence to the OvS instance while collecting values of CPU load and latency until a stopping criteria is met. In the following, we describe the collection process of the two metrics with respect to their stopping criteria:

CPU load. We assess the CPU usage of OvS as the time spent in the datapath (kernel) module using the *proc* plugin of *Telegraf* ("*cpu_time_system*" of the OvS process). The value is fetched every second. We repeatedly collect samples until the 95%-confidence interval (CI) is smaller than 0.02 s. The resulting mean of the samples is the value of the grid point.

Latency. Latency values are obtained as RTT (pings from *host*1 to *host*3 in Fig. 2). Every 0.5 s, we use the "ping" plugin of *Telegraf* to collect and average the RTT of 20 pings, which are sent sequentially with a 1 ms pause. This process is repeated until the 95%-CI of all collected latency samples is smaller than 0.01 ms. The mean of the resulting samples is used as the value for the grid point.

For both metrics, we collected at least 10 samples to calculate the CI. If the CI is still larger than the threshold after 100 collected samples, the collection is canceled and the mean values are calculated over 100 samples. Between two traffic configurations, the data collection pauses for at least 15 s to cool down the system, i.e., to clear the megaflow cache. NetBOA uses the same procedure for collecting the data of the evaluation configuration points (cf. Alg. 1-"RunMeasurements").

3.1.4 Evaluation Settings and Random Search Description. The evaluation of NetBOA and RS was run on the same testbed as the grid search measurements. The four considered parameters in Fig. 5 and Fig. 6 are IAT (x_{iat}), number of packets per sending round (x_p), the burst size (x_b) and the number of different VLAN tags (x_{VLAN}) within one sending round. Table 1 lists the used parameter ranges.

Table 1: Ranges for parameter searches with 1, 2, 3 or 4 variable parameters.

#	x_{iat}	x_p	x_b	x_{VLAN}	# Init. Smpls
1	[1.0, 14.0]	4000	1	1	4
2	[1.0, 14.0]	[1000, 5000]	1	1	8
3	[1.0, 14.0]	[1000, 5000]	[1, 5]	1	12
4	[1.0, 14.0]	[1000, 5000]	[1, 5]	[1, 5]	16

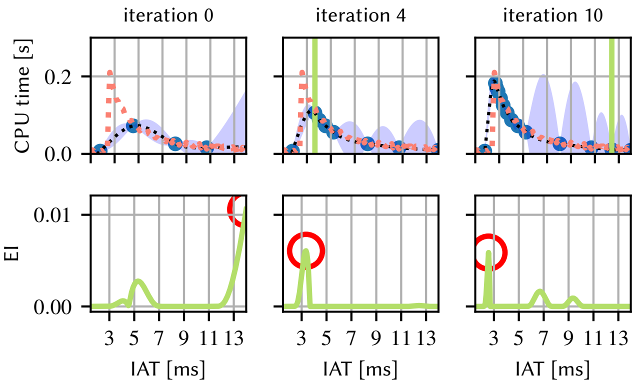


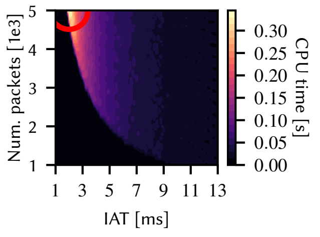
Figure 3: Example of steps for one parameter (x_{iat}). In the upper row, the orange-dashed line shows data from the baseline measurements. The black-dashed lines illustrates the current mean values, whereas the blue shadows represent the belief ot the BOA. The blue circles illustrate the current searching points of the iteration (Style is chosen from [8]).

Note that x_p, x_b and x_{VLAN} have discrete values, while x_{IAT} is continuous. RS samples independently from the available parameter space in every iteration and selects the configuration with highest objective value.

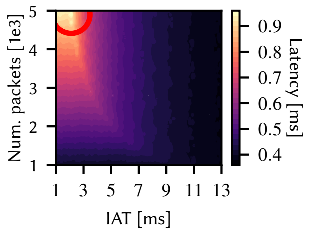
3.2 Experimental Results

How does NetBOA work? Fig. 3 shows how NetBOA searches for the most challenging traffic configuration in case of using only the inter-arrival time (IAT) as our configuration parameter. All figures show the pre-measured performance profile of the CPU over the IAT via a red dashed line. In the first iteration, NetBOA fits the GP to four initial samples. Note the shaded area around the blue curve: this area illustrates the confidence area of the true values around the current blue curve. Based on this value and the EI function, the BO algorithm searches for the next configuration settings. The lower plot for the first iteration shows the acquisition function. Here, the largest EI improvement lies at IAT= 14 ms. After 10 iterations, the next samples lead the configuration towards the maximum that lies at IAT=2.1 ms.

How does actually the performance model look like for more than 1 parameter? Before evaluating the behavior of NetBOA when actively searching for the most challenging configuration, Fig. 4a shows the performance model for two configuration parameters (number of unique packets and inter-arrival time). This performance model shows the data of an intensive grid-based measurement. Here, the step size of the number of packets is 100 and the step size of the IAT is 0.1 ms. As we can see, the highest CPU load (the measured time that the OvS kernel process spends here on 1 CPU core), lies around 5000 packets and an IAT of roughly 2.1 ms. The reason is that here the most flow entries are created by the OvS process (5000) and the algorithm always removes and reads the megaflowcache entries. This is due to the setting of 10 seconds as default rule timeout in the OvS flow caches.

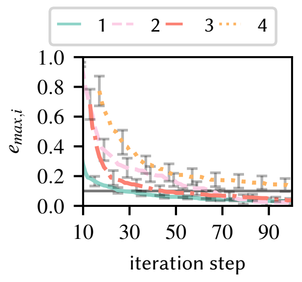


(a) CPU Heatmap.

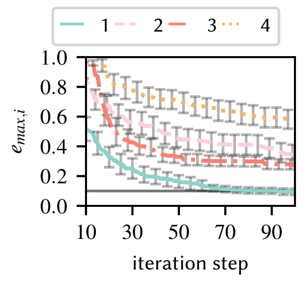


(b) Latency Heatmap.

Figure 4: CPU and latency heatmaps. Used parameters: x_p and x_{iat}. Note that the x-axis is limited to 13 ms for better appearance.



(a) Bayesian Optimization.

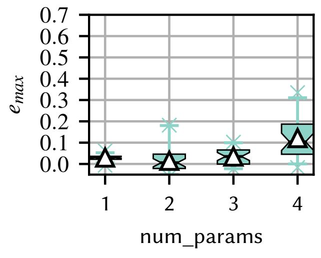


(b) Random Search.

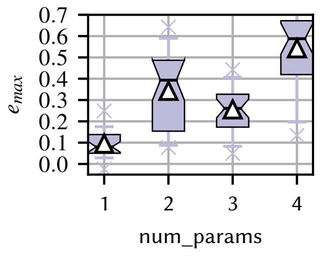
Figure 5: RS vs. BO, 95 % confidence intervals of mean error over the runs, i.e., the deviation of the found minimum value from the known optimal value. For each run, the minimum value is taken into account. Note that the performance during initial sampling is removed from the BO plot.

Fig. 4b shows NetBOA when measuring the latency. As expected from the OvS implementation, there are also high latency values for lower IAT intervals. However, interestingly, the setting with the highest latency is close to the one for the CPU. The reason is that of course the number of table entries mainly determines the latency, but that there is also a slight impact on the latency based on the overall CPU load. Hence, when having precise mechanisms for determining the latency, an attacker could also use the latency to infer the highest CPU load. Generally, not only the amount of rules determines the highest latency and load, but also the sending pattern of the network traffic.

Is NetBOA better than a Random Search (RS)? In order to illustrate the speedup in finding the hardest configuration, Fig. 5 compares the error in finding the worst configuration between NetBOA and RS. Vertical error bars indicate the 95%-CIs computed over 30 runs with different seeds for every configuration. Fig. 5 shows that for all parameter settings (1-4 parameters), NetBOA always finds more challenging configurations faster than RS. For



(a) Bayesian Optimization.



(b) Random Search.

Figure 6: RS vs. BO. Relative deviation of CPU load from the known optimal value after 100 iterations.

instance, NetBOA finds the most challenging configuration for 1-3 parameters within 70 iterations (after the initial sampling). In contrast, the RS procedure does not find a configuration that is within 10 % of the optimum for any parameter setting within 70 iterations. Generally, this shows that NetBOA is working well and that it is indeed possible to find among network configurations the ones that lead to the highest CPU load of a given network entity.

In total, how good is NetBOA? Fig. 6 shows a comprehensive summary of the comparison between NetBOA and RS. As the boxplots of both subfigures show, NetBOA has a high probability in finding the near-optimal configurations for 1-3 parameters. In contrast, RS finds only a few challenging settings for 1 parameter. Note further that NetBOA always finds a setting that has a performance of only 30 % less than the maximum. That means although NetBOA does not find the most challenging setting, it still finds configurations that increase the CPU load of one core by more than 20 % in contrast to settings that send more network traffic in general over time.

4 RELATED WORK

We see multiple research fields as related to our research.

Algorithmic Complexity Attacks. Algorithmic complexity attacks, and related mechanisms for mitigation, are concerned with finding inputs that trigger the worst-case behavior of an algorithm, like regular expression matching or sorting, in an application. SlowFuzz [26] uses evolutionary techniques to mutate the input of applications-under-test, e.g., the PCRE library for regular expressions used by firewalls, the bzip2 compression utility, and the hash table implementation of PHP. PerfFuzz [19] additionally uses the program code as input and looks for maximizing the execution paths of the components of the program code. Rampart [24] targets the opposite use-case of algorithm attacks: it protects applications from exhaustive CPU exhaustion DoS attacks. Instead of finding concrete inputs that exhaust programs, Singularity [30] targets at a higher abstraction level: analyzing given program code it tries to find input patterns that exhaust the program application. The work of [25] synthesizes program code of network functions in order to find challenging network traffic configurations. NetBOA is a parallel work that looks for challenging network traffic configurations for network functions, however, without looking at the source code of network functions.

Applied Bayesian Optimization. BO has successfully been used to tune parameters in other domains, like configuration parameters of deep neural networks [29]. For clouds, CherryPick [1] uses BO to find the best configuration for cloud customers. Scout [12] builds up on Cherrypick: it uses historical optimization data to advance the search for better cloud configurations. Despite applying BO, there is work on improving BO itself: for instance, [23] implements BLOSSOM, an approach that selects between multiple acquisition functions and traditional optimization on each iteration step. NetBOA applies BO for network traffic configurations.

Open vSwitch Network Measurements. A recent number of research papers exist that measure OvS: [15] measures the forwarding delay of OvS; [7] determines the throughput and resource footprint of OvS. [2] and [20] propose benchmark tools to measure average latency and throughput of OvS. Moreover, [2] also looks into the impact of different matching and actions, table size, and queue size settings. Jive prohibitively probes switches with "Jive Patterns" in order to see the performance variation given different workloads for changing rule strategies: e.g., is it better to add rules in descending or ascending order. Most interesting for us is the recent work of *Fang et al.* [9]. They want to answer the question of how to find the best software switch among different systems and workloads. We believe that NetBOA can provide them with an automated benchmarking tool to address this question by self-driving benchmarks for different switches.

5 CONCLUSION

In order to provide an optimal performance and guide control, any self-driving network needs to rely on frequent and accurate network measurements, Our main position in this paper was that such performance measurements should be automated, accounting for non-trivial performance effects and noise. Accordingly, we proposed a blackbox, i.e., Bayesian, optimization approach and demonstrated its feasibility and effectiveness in a specific case study: benchmarking OvS.

We see our work as a first step and believe that our paper opens several interesting avenues for future research. In particular, our framework is still simple, and several optimizations can be made to further improve performance. For instance, the initial random sampling or the kernel selection process can be enhanced. Moreover, although recent studies argued that alternatives to Bayesian optimization have some shortcomings, it might still be worth to analyze most recent techniques such as deep reinforcement learning. Furthermore, it will be very interesting to consider alternative case studies using NetBOA, in particular, alternative network functions. More generally, our paper also opens the general question of what can and cannot be achieved by self-driving networks. For example, can a self-driving network notice its limitations?

ACKNOWLEDGMENT

This work has been performed in part in the framework of the EU project FlexNets funded by the European Research Council under the European Union's Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets), and in part in the DFG ModaNet Project (grant No KE 1863/8-1). This work reflects only the authors' view and the funding agency is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. *Proc. USENIX NSDI* (2017), 469–482.

[2] Simon Bauer, Daniel Raumer, Paul Emmerich, and Georg Carle. 2018. Behind the scenes: what device benchmarks can tell us. (2018).

[3] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *Proc. ACM SIGCOMM*. Los Angeles, CA, USA, 155–168.

[4] Andreas Blenk, Patrick Kalmbach, Stefan Schmid, and Wolfgang Kellerer. 2017. o'zapft is: Tap Your Network Algorithm's Big Data!. In *Proc. ACM SIGCOMM Big-DAMA*. Los Angeles, CA, USA.

[5] S. Bradner and J. McQuaid. 1999. Benchmarking Methodology for Network Interconnect Devices. RFC 2544. (March 1999).

[6] Levente Csikor, Christian Rothenberg, Dimitrios P. Pezaros, Stefan Schmid, László Toka, and Gábor Rétvári. 2018. Policy Injection: A Cloud Dataplane DoS Attack. In *Proc. ACM SIGCOMM Conference on Posters and Demos*. Budapest, Hungary, 147–149.

[7] Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2014. Performance characteristics of virtual switching. In *Proc. IEEE CloudNet*. IEEE, Luxembourg, Luxembourg, 120–125.

[8] Vlad M. Cora Eric Brochu and Nando de Freitas. 2010. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". *arXiv preprint* (2010). https://arxiv.org/pdf/1012.2599.pdf

[9] Vivian Fang, Tamás Lévai, Sangjin Han, Sylvia Ratnasamy, Barath Raghavan, and Justine Sherry. 2018. *Evaluating Software Switches: Hard or Hopeless?* Technical Report. 8 pages.

[10] Nick Feamster and Jennifer Rexford. 2017. Why (and how) networks should run themselves. *arXiv preprint arXiv:1710.11583* (2017).

[11] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. 2015. Initializing Bayesian Hyperparameter Optimization via Meta-learning. In *Proc. AAAI (AAAI'15)*. AAAI Press, Austin, Texas, USA, 1128–1135.

[12] Chin-Jung Hsu, Vivek Nair, Tim Menzies, and Vincent W. Freeh. 2018. Scout: An Experienced Guide to Find the Best Cloud Configuration. *arXiv:1803.01296 [cs]* (March 2018). http://arxiv.org/abs/1803.01296

[13] Danny Yuxing Huang, Kenneth Yocum, and Alex C Snoeren. 2013. High-fidelity switch models for software-defined network emulation. In *Proc. of the second ACM SIGCOMM workshop on HotSDN*. 43–48.

[14] Rishabh Iyer, Luis Pedrosa, Arseniy Zaostrovnykh, Solal Pirelli, Katerina Argyraki, and George Candea. 2019. Performance Contracts for Software Network Functions. In *Proc. USENIX NSDI*. Boston, MA, USA, 517–530.

[15] Michael Jarschel, Simon Oechsner, Daniel Schlosser, Rastin Pries, Sebastian Goll, and Phuoc Tran-Gia. 2011. Modeling and performance evaluation of an OpenFlow architecture. In *Proc. ITC*. San Francisco, CA, USA, 1–7.

[16] Patrick Kalmbach, Johannes Zerwas, Peter Babarczi, Andreas Blenk, Wolfgang Kellerer, and Stefan Schmid. 2018. Empowering Self-Driving Networks. In *Proc. ACM SIGCOMM 2018 Workshop on Self-Driving Networks (SDN)*. Budapest, Hungary.

[17] Wolfgang Kellerer, Patrick Kalmbach, Andreas Blenk, Arsany Basta, Martin Reisslein, and Stefan Schmid. 2019. Adaptable and Data-Driven Softwarized Networks: Review, Opportunities, and Challenges. *Proc. IEEE* 107, 4 (April 2019), 711–731.

[18] Maciej Kuźniar, Peter Perešíni, and Dejan Kostić. 2015. What you need to know about SDN flow tables. In *Passive and Active Measurement*. Springer, 347–359.

[19] Caroline Lemieux, Rohan Padhye, Koushik Sen, and Dawn Song. 2018. PerfFuzz: automatically generating pathological inputs. In *Proc. ACM SIGSOFT*. ACM, Amsterdam, Netherlands, 254–265.

[20] T. Lévai, G. Pongrácz, P. Megyesi, P. Vörös, S. Laki, F. Németh, and G. Rétvári. 2018. The Price for Programmability in the Software Data Plane: The Vendor Perspective. *IEEE Jrnl. on Sel. Ar. in Com.* 36, 12 (Dec. 2018), 2621–2630.

[21] Jed Liu, William Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Călin Caşcaval, Nick McKeown, and Nate Foster. 2018. P4V: Practical Verification for Programmable Data Planes. In *Proc. ACM SIGCOMM*. Budapest, Hungary, 490–503.

[22] Chenyang Lv, Shouling Ji, Yuwei Li, Junfeng Zhou, Jianhai Chen, Pan Zhou, and Jing Chen. 2018. SmartSeed: Smart Seed Generation for Efficient Fuzzing. *arXiv:1807.02606 [cs]* (July 2018).

[23] Mark McLeod, Michael A. Osborne, and Stephen J. Roberts. 2018. Optimization, fast and slow: optimally switching between local and Bayesian optimization. *arXiv:1805.08610 [cs, stat]* (May 2018).

[24] Wei Meng, Chenxiong Qian, Shuang Hao, Kevin Borgolte, Giovanni Vigna, Christopher Kruegel, and Wenke Lee. 2018. Rampart: Protecting Web Applications from CPU-Exhaustion Denial-of-Service Attacks. In *USENIX Security*. Baltimore, MD, USA, 393–410.

[25] Luis Pedrosa, Rishabh Iyer, Arseniy Zaostrovnykh, Jonas Fietz, and Katerina Argyraki. 2018. Automated Synthesis of Adversarial Workloads for Network Functions. In *Proc. ACM SIGCOMM*. Budapest, Hungary, 372–385.

[26] Theofilos Petsios, Jason Zhao, Angelos D. Keromytis, and Suman Jana. 2017. SlowFuzz: Automated Domain-Independent Detection of Algorithmic Complexity Vulnerabilities. In *Proc. ACM SIGSAC CCS*. Dallas, Texas, USA, 2155–2168.

[27] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. 2015. The Design and Implementation of Open vSwitch. In *Proc. USENIX NSDI*. Oakland, CA, USA, 117–130.

[28] Daniel Raumer, Sebastian Gallenmüller, Florian Wohlfart, Paul Emmerich, Patrick Werneck, and Georg Carle. 2016. Revisiting benchmarking methodology for interconnect devices. In *Proc. ACM ANRW*. Berlin, Germany, 55–61.

[29] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. *arXiv:1206.2944 [cs, stat]* (June 2012). http://arxiv.org/abs/1206.2944 arXiv: 1206.2944.

[30] Jiayi Wei, Jia Chen, Yu Feng, Kostas Ferles, and Isil Dillig. 2018. Singularity: Pattern Fuzzing for Worst Case Complexity. In *Proc. ACM ESEC/FSE*. ACM, Lake Buena Vista, FL, USA, 213–223.