AdFAT: Adversarial Flow Arrival Time Generation for Demand-Oblivious Data Center Networks

Sebastian Schmidt, Johannes Zerwas, Wolfgang Kellerer
Chair of Communication Networks
Technical University of Munich, Germany
{sebastian.a.schmidt, johannes.zerwas, wolfgang.kellerer}@tum.de

***Abstract*—Researchers developing new architectures and algorithms for data center networks (DCNs) face the challenge of producing meaningful evaluations of their contributions. Traditional evaluation methods like traffic traces and parametric models can fail to reveal weak spots in DCNs. The concept of adversarial inputs shapes traffic data, making it challenging for a DCN to serve it. Adversarial traffic can provide insight into performance issues of a DCN that might go unnoticed with traces and models. This paper presents AdFAT, a genetic algorithm-based system for automated adversarial input generation for DCNs. While previous work focuses on reordering flow volumes or individual packets, our system uses flow arrival times as the adversarial traffic dimension. By creating adversarial flow arrivals for a demand-oblivious RotorNet topology, we show that AdFAT not only finds traffic that causes 22.64% higher mean flow completion times than traffic with uniform random arrival times but is also sensitive to the inherent periodicities and connection patterns of RotorNet. The results indicate AdFAT can find and exploit temporal and structural properties of dynamic and demand-oblivious topologies in an automated way.**

***Index Terms*—Data Centers, Genetic Algorithms, Performance Evaluation, Adversarial Machine Learning**

I. Introduction

The ever-increasing number of cloud-based services on the internet requires larger and larger amounts of storage and computational power [1]. Data center networks (DCNs) are a crucial part of the technical infrastructure that enables these services. To satisfy the increasing requirements, research tries to make DCNs more performant with the help of new topologies and network algorithms to interconnect the servers of a data center [2]–[7]. Further, data-driven approaches in network design and operation (often referred to as self-driving networks) grow in popularity [8]–[10]. They aim to reduce the manual and error-prone configuration of DCNs by introducing data-driven and automated management and analysis.

The evaluation of new DCN architectures and algorithms is traditionally conducted with either data sampled from traffic traces [2], [6] or by randomly sampling traffic from parametric models, as done in [3]–[5]. Trace data usually stems from a different use case and is, therefore, highly specific and not particularly well suited for evaluating other DCNs. Samples from distributions might be unbalanced and not represent the characteristics of the models due to chance, and require the researchers to select the parameters. It can not be assumed that either of the aforementioned traffic creation methods produces traffic that can provide deeper insight into the performance of topologies and algorithms in challenging scenarios [11], [12]. Only some specific parts of the input region might cause the network algorithms to behave poorly, but there is no guarantee that these regions are contained in the data used for evaluation. Further, to make new DCNs future-proof and robust, evaluation should be conducted with traffic outside regions covered by traces. Creating challenging evaluation data sets by hand is also complex and sometimes impossible even for experts [13].

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 438892507.

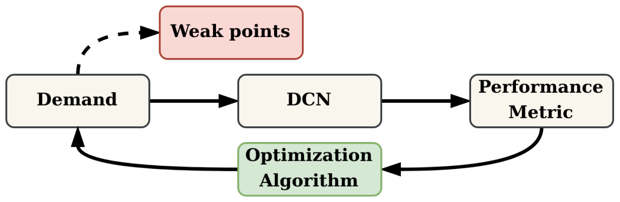

Fig. 1: Adversarial demand generation. After the lower loop converges, the found demand can help to find weak spots in the DCN.

When weak spots or failure modes of algorithms stay hidden during evaluation, it can lead to over-estimation of performance and oversight of implementation issues. This can drastically impact the reliability and security of a system [14]. The problem of identifying weak spots of DCNs gives rise to the idea of tailoring the used input data to the application, thus creating more reliable evaluation results [11], [12]. Borrowing from the concept of adversarial examples in machine learning systems, input data that is challenging for a DCN can be referred to as adversarial demand [13]. Analyzing demand that leads to decreased performance can create valuable insights into how the DCN behaves for specific traffic, help find implementation errors, and provide guidance on further improvements.

The automated detection of algorithmic and structural weak spots in DCNs is especially important for the idea of self-driving networks [9], [10]. Autonomous networks need the capability to gain insight into their performance for a specific workload and draw conclusions on how to better adapt to it. In this context, the automated creation of adversarial traffic patterns and subsequent improvement upon the found information is a promising path to enhance the self-monitoring and self-managing capabilities of future DCNs. Figure 1 depicts this idea: By optimizing the demand to degrade a performance metric, the resulting adversarial demand can contain information about weak spots of the DCN under attack. Prior work in Toxin [12] solves this optimization problem with a Genetic Algorithm (GA). However, it neglects the arrival times as a dimension of the adversarial traffic. This work extends Toxin's basic idea of GA-based adversarial input generation. Our contributions are:

1) We present ADFAT that freely controls the arrival time of flows, thereby incorporating the temporal behavior of DCNs into the search for challenging traffic for the first time.
2) We formulate the optimization problem of flow-based adversarial demand generation for dynamic DCNs.
3) We present a statistical evaluation of the adversarial flow arrival times highlighting the resulting adversarial patterns.
4) We show that the GA successfully exploits implementation details and parameters of RotorNet to create adversarial arrival times.

II. Related Work

Other works concerned with adversarial traffic and attacks in the context of DCNs include: Meier et al. [10] showed that communication networks provide multiple possible attack points for adversarial attacks. Lin et al. [15] create adversarial UDP packets for network algorithms using generative adversarial networks. The packets target vulnerabilities by causing the execution of erroneous or inefficient code, thereby causing high resource utilization and exposing security risks. Nasr et al. [16] implement a neural network-based traffic generator that alters packet timing and sizes, and introduces new packets to a communication stream. The proposed system is able to defeat state-of-the-art traffic analysis techniques, like flow correlation and website fingerprinting.

The previous two approaches rely on data (either given or sampled from the target system) that describes the nature of adversarial traffic. In contrast, the following works create samples of adversarial input themselves by directly optimizing the traffic presented to the system under attack.

NetBOA [11] uses Bayesian optimization to create challenging packet parameters that trigger high CPU usage on software-defined switches. The closest work, Toxin [12], is based on permuting the volumes of flows sampled from a predefined parametric model. It describes itself as proof-of-concept and introduces a GA as a method to create adversarial demand for DCNs. The results suggest that a GA can effectively create adversarial flow-based traffic, but additional experiments, especially considering other dimensions of flows, like arrival times, are needed. The evaluation of Toxin describes single adversarial demand samples, but no statistical evaluation that could lead to more expressive demand patterns is conducted. Our work addresses both issues and statistically investigates adversarial flow arrival times created by a GA.

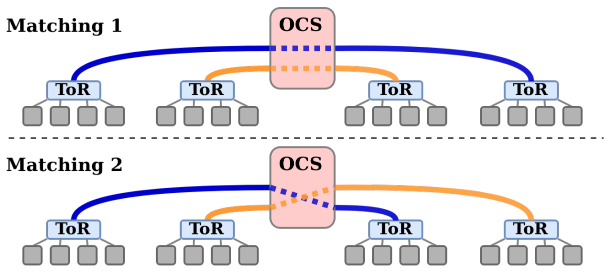

Fig. 2: A ToR-based topology. Gray boxes represent hosts connected to the ToRs. Two different matchings via the OCS are depicted.

III. Problem Formulation

This section describes the models used for network, traffic and routing algorithms, and the resulting optimization problem of creating adversarial flow arrival times.

A. Network, Traffic & Routing Algorithm Model

The DCN is modeled as a directed weighted multigraph $G(\mathbf{V},\mathbf{E})$. The n_T nodes $n_i \in \mathbf{V}$ model top-of-rack switches (ToRs), and the edges $e_i \in \mathbf{E}$ represent links. Edge weights describe the capacity of the links. In a dynamic topology like RotorNet [2] or Duo [5], the available links between ToRs change over time. To include this in the problem formulation, a topology is represented as a mapping $\mathcal{T} \in \Upsilon$,

$$\mathcal{T} : \mathbb{R}_+ \mapsto \mathcal{G}$$

from time to a network graph describing the topology at that time. $\mathcal{G}$ is the space of all possible graphs allowed by the dynamic DCN, and Υ is the space of all such possible mappings.

All communication in the network is modeled as flows f_i. A flow describes a transmission request for a specified amount of data. The data should be transferred from one node in the network to another. Packets and related concepts like transport protocols are not modeled. A Flow f_i is defined by an ordered tuple (s_i, d_i, v_i, t^a_i) where $s_i \in \mathbf{V}$ is the source node and $d_i \in \mathbf{V}$ is the destination node of the flow. $v_i \in \mathbb{R}_+$ is the volume of the flow, $t^a_i \in \mathbb{R}_+$ is its arrival time, meaning the time the transmission request becomes active in a network. To complete a flow, the volume v_i needs to be reduced to zero. To do so, the flow is assigned a route through the network and a transmission rate by the routing algorithm. When the flow volume reaches zero, the flow is referred to as *serviced*, and the corresponding time is called the service time t^s_i of a flow f_i. Together with the arrival time t^a_i the flow completion time (FCT) fct_i of a flow is defined as $fct_i = t^s_i - t^a_i$. It is a commonly used performance metric for DCNs, where lower values indicate faster transmission [2]–[4].

The demand $\mathbf{F}$ is represented by an ordered tuple of ϕ flows, where each flow is defined according to the model presented above:

$$\mathbf{F} = (f_1, f_2, f_3, \ \ldots \ , f_\phi) \, .$$

In this work, ϕ is fixed to be $n_\mathrm{T} \cdot (n_\mathrm{T} - 1)$, the number of distinct and directed connections between n_T ToRs. $\mathbf{F}$ stems from $\mathbf{\Phi}$, the space of all possible demands obeying this constraint.

Furthermore, a routing algorithm is responsible for coordinating the transmission of flows. It is formalized as a mapping

$$\mathcal{A} : \Upsilon \times \mathbf{\Phi} \mapsto \mathbb{R}_+{}^{\phi},$$

where the routing algorithm $\mathcal{A}$ maps a topology and a demand to an ordered tuple of ϕ FCTs $\in \mathbb{R}_+$. So, for a topology $\mathcal{T}$ and a demand $\mathbf{F}$,

$$\mathcal{A}\left(\mathcal{T}, \mathbf{F}\right) = \left(fct_1, fct_2, fct_3,\ \ldots\ , fct_\phi\right).$$

B. Optimization Problem

To quantify how challenging a scenario is, a function $\mathcal{Q}$ maps a complete scenario (consisting of routing algorithm $\mathcal{A}$, topology $\mathcal{T}$ and demand $\mathbf{F}$) to a single value $q \in \mathbb{R}_+$: $q = \mathcal{Q}\left(\mathcal{A}, \mathcal{T}, \mathbf{F}\right)$. The function $\mathcal{Q}$ can be any performance measure constructed from the scenario and the FCTs obtained by the mapping $\mathcal{A}$. Details for the concrete function used in ADFAT are presented in Section IV, but in general, demand that produces high values of q is considered challenging in the sense of $\mathcal{Q}$. The system alters the demand $\mathbf{F}$ to find adversarial traffic. It tries to find the most challenging demand

$$\mathbf{F}^* = \underset{\mathbf{F}}{\arg\max} \quad \mathcal{Q}\left(\mathcal{A}, \mathcal{T}, \mathbf{F}\right).$$

This optimization problem describes creating adversarial demand for a single DCN topology and algorithm.

Since ADFAT focuses on creating adversarial arrival times and their impact on the performance, we set the volumes of all flows in the demand to a fixed value V. For simplicity, we focus on traffic with only one flow from rack i to rack j and no flows with the same source and destination rack. $|\mathbf{F}_{ij}|$ is the number of flows from rack i to rack j in $\mathbf{F}$. To limit the duration of one scenario, create comparability, and ensure that the flows overlap in the time domain and compete for bandwidth, a time window is defined for the flows to arrive in the network. Only arrival times in $[0, t^a_{max}]$ are considered. With this, the complete optimization problem is formulated as:

$$\begin{aligned} \mathbf{F}^* = \ & \underset{\mathbf{F}}{\arg\max} \quad \mathcal{Q}\left(\mathcal{A}, \mathcal{T}, \mathbf{F}\right) \\ & \text{subject to} \\ & \quad v_i = V, \qquad \forall f_i \in \mathbf{F}, \\ & \quad t^a_i \leq t^a_{max}, \forall f_i \in \mathbf{F}, \\ & |\mathbf{F}_{ij}| = 1, \qquad \forall (i,j) \in \{1, \ldots, n_\text{T}\}^2, i \neq j \end{aligned} \tag{1}$$

This problem is equivalent to presenting the network with the same volume for every possible flow between the ToR pairs and adjusting the arrival times of the flows to find challenging traffic. Therefore, the formulation isolates the influence of when a flow becomes present in the network.

IV. Genetic Algorithm

The optimization problem above is not guaranteed to be analytically solvable since it involves the potentially highly nonlinear mapping between the flows and the corresponding FCTs caused by algorithms and topology. Like Toxin, ADFAT relies on a GA as a black-box-optimization method to obtain near-optimal solutions by repeatedly selecting and recombining the *fittest* solutions to a problem [17]. The only requirement for the application of a GA is that $\mathcal{Q}$ can be *calculated* for demands $\mathbf{F}$. We solve this by using simulation. Even though the GA does not have any guarantees on the quality of the found solutions and how close they are to the real $\mathbf{F}^*$, the found near-optimal solutions can still be of interest. Investigating $\tilde{\mathbf{F}}^*$ the approximations of $\mathbf{F}^*$ created by the GA

$$\mathbf{F}^* \approx \tilde{\mathbf{F}}^* = \mathcal{GA}\left(\mathbf{P}\right),$$

where $\mathbf{P}$ stands for an optimization problem of the form described in (1), can provide information about structures found in the true optimum $\mathbf{F}^*$ and therefore still be used as adversarial demand, that can help to improve the DCN.

Central for using a GA to find adversarial arrival times is encoding them in chromosomes $\mathbf{C}$.

$$\mathbf{C} = \left(c_{12}, c_{13}, \cdots, c_{1n_\text{T}}, c_{21}, c_{23}, \cdots, c_{2n_\text{T}}, \cdots, c_{(n_\text{T}-1)n_\text{T}}\right),$$

where the entries c_{ij} correspond to the arrival times t^a_{ij} of the flow between rack i and j, and n_T is the number of ToRs or racks in the topology. The c_{ij} are defined as $\in [0, 1]$, so by multiplying them with t^a_{max}, the resulting arrival times fall into the valid range. Note that the coding only allows for demands $\mathbf{F}$ that satisfy the constraints from the optimization objective in (1). The demand created from a chromosome consists of $n_\text{T} \cdot (n_\text{T} - 1)$ flows with the same volume V and the arrival times created from the chromosome:

$$\mathbf{F}_\mathbf{C} = \left\{ f(i, j, V, c_{ij} \cdot t^a_\text{max}) \;\middle|\; \forall (i,j) \in \{1, \ldots, n_\text{T}\}^2, i \neq j \right\}.$$

To measure the *fitness* of an individual demand, the mean of the fct_i of the flows $f_i \in \mathbf{F}$ is used. $\mathcal{Q}_{\overline{\text{FCT}}}$, the mean flow completion time fitness function is defined as

$$\mathcal{Q}_{\overline{\text{FCT}}}\left(\mathcal{A}, \mathcal{T}, \mathbf{F}\right) \;=\; \frac{1}{|\mathbf{F}|} \cdot \sum_{i=1}^{|\mathbf{F}|} fct_i.$$

It is used to select the high-performing chromosomes and is the objective function of the optimization in ADFAT.

The remaining parts of the GA are designed as follows: We use a random uniform initialization where all $c_{ij} \sim \mathcal{U}(0, 1)$ for the initialization of the GA population. The parents selected for crossover and creation for the next generation are picked by greedily selecting the top 10% of the solutions of the previous generation. To combine parents, an order-preserving random k-point crossover [18] operator is used. The mutation step is a combination of resampling n_{mut} random genes c_{ij} with probability p_re and, with probability p_add, adding samples from $\mathcal{U}(-1, 1)$ to n_mut different genes and afterwards truncating them to stay in $[0, 1]$.

V. Experiments

This section evaluates ADFAT as a method to create adversarial flow arrivals. It briefly describes the RotorNet topology, the simulator we use for calculating $\mathcal{Q}$, the GA parameters, and the resulting adversarial arrival times.

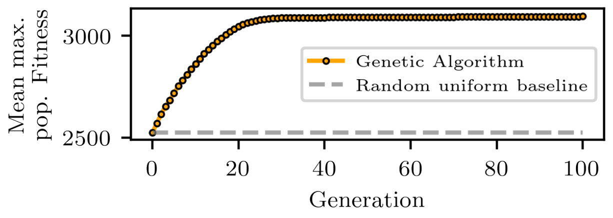

Fig. 3: Mean maximum score per generation. Confidence interval ($\alpha = 0.99$) not visible

A. Example: RotorNet

As a case study, we create adversarial arrival times for the dynamic RotorNet DCN. Because it is demand-oblivious, it does not alter its structure based on the demand it needs to serve [2]. All n_{T} racks of the topology are connected to n_{OCS} optical circuit switches (OCSs). The OCSs cycle through *matchings* where every rack is directly connected to exactly one other rack. A matching is held for one *slot* of duration t_{slot}. After this, the OCS links are reconfigured to the matching of the next slot. Figure 2 shows two exemplary matchings via a single OCS. RotorNet provides all-to-all connectivity between n_{T} racks over the course of $n_{\mathrm{T}}-1$ slots, referred to as a *matching cycle*. Routing decisions are made at the beginning of a slot to transmit directly, if possible, or indirectly via two-hop paths if no direct path is available. Because indirectly transmitted volume is stored on the intermediate rack until it can be transmitted to its final destination, this way of transmission is less effective and introduces delay [2]. The periodic behavior of RotorNet allows us to investigate if the GA is sensitive to the temporal changes of the topology structure defined by the matchings between the racks. Also, since two transmission qualities (direct and indirect) exist, the GA can potentially exploit the differences between them to construct adversarial demand. Further, the decision on which flows will transmit during a slot is made at the beginning of the slot. With this, the reaction of the GA to the time points of the routing decisions can be analyzed.

B. Settings

We use a custom flow-level simulator to evaluate the performance of a demand for the RotorNet topology. The simulation is conducted on a ToR level. It includes no transport layer effects, and multiple flows sharing a link do not interact. This abstract simulator is chosen to focus on the impact of the topology reconfiguration and routing.

The GA uses a population size of 500, $k=9$ for the crossover operation, and the mutation parameters are $p_{\mathrm{add}}=0.5$, $p_{\mathrm{re}}=0.1$, and $n_{\mathrm{mut}}=5$. For RotorNet, $t_{\mathrm{slot}}=100\,\mu\mathrm{s}$, $n_{\mathrm{OCS}}=1$, and $n_{\mathrm{T}}=8$. The optimized demand is obtained by selecting the best solution after 100 epochs of the GA. All results represent the accumulation of 100 differently seeded runs of the GA.

C. Results

ADFAT produces adversarial arrival times. Figure 3 shows the mean maximum population fitness over 100 generations in orange. The gray dashed line represents a uniformly sampled baseline. After less than 30 generations, the GA is converged and the resulting adversarial arrival times increase the mean FCT by 22.64% compared to the initial, uniformly sampled arrivals.

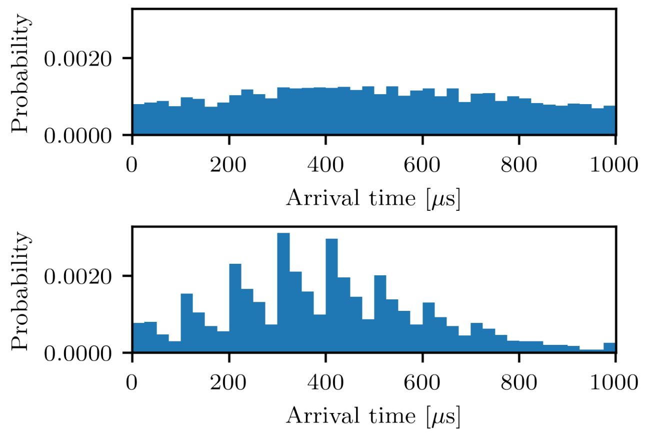

Fig. 4: Histograms of the arrival times of all flows accumulated over 100 seeds. The top figure shows the histogram of the best solutions of the first generation, the bottom one of the last generation.

ADFAT is sensitive to the periodicity of routing decisions. To investigate how the flows arrive between 0 and $t^a_{max}=1000\,\mu\mathrm{s}$, Figure 4 shows histograms of the arrival times of all flows created over 100 runs. The bin width is $25\,\mu\mathrm{s}$, and all bins exclude the lower boundary while including the upper. On top are the initial arrival times before optimization with the GA. As expected from the random uniform initialization, the arrival times are evenly distributed in the interval $[0\,\mu\mathrm{s}, 1000\,\mu\mathrm{s}]$. In contrast, the lower histogram shows the arrival times of the found adversarial demand. A sawtooth pattern with a periodicity equal to t_{slot} of RotorNet ($100\,\mu\mathrm{s}$) is visible. In each time window of $(n{\cdot}100, (n{+}1){\cdot}100]$, the distributions are skewed so that it is much more likely for flows to arrive exactly after multiples of $100\,\mu\mathrm{s}$. This is explained by the routing algorithm of RotorNet, which only considers flows in the system at multiples of $100\,\mu\mathrm{s}$ for routing and transmission in the following slot. A flow that arrives slightly after this has to wait until the next reconfiguration and routing step happens at the beginning of the next slot. This increases the FCT of a flow that arrives directly after the routing decision by almost one slot length.

ADFAT exploits structural changes of the topology and produces connection-specific arrival times. The previous paragraph discusses the arrival times of all flows. However, in RotorNet, the FCT of a flow depends on when related to the matching cycle it arrives in the system because this influences the ratio between direct and indirect transmission. This observation motivates the analysis of the arrival time not on the global level but on the level of flows with a fixed source and destination. Figure 5 shows time on the x-axis and 8 selected source-destination pairs on the y-axis. Each colored dot represents the arrival time of an adversarial flow with the source and destination ToR corresponding to the y-value. The

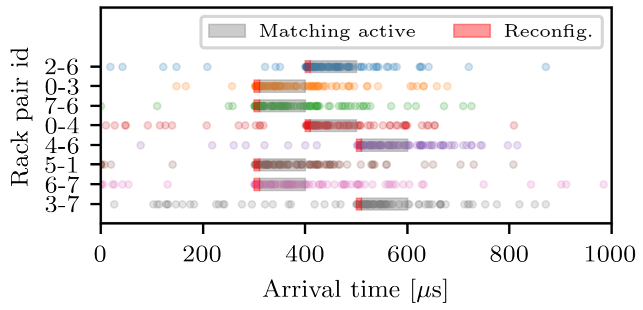

Fig. 5: Selected ToR pairs with adversarial flow arrival times.

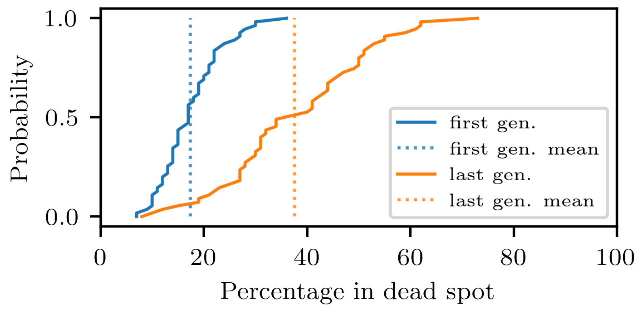

Fig. 6: ECDF of the percentage of flow arrival times in dead spot.

gray areas mark when a direct optical connection between a ToR pair is active, and the red areas mark reconfiguration. The optimized traffic shows clusters that overlap with the times during which the flow's source and destination are directly connected. Arrivals in times before and after the direct connections are reduced. If a flow arrives while its direct connection is active, it is not routed during the slot of direct transmission and can only be sent via indirect paths in the following slots. During the indirect transmission, it competes with the transmission of other flows, direct and indirect, further increasing the FCT.

A flow arriving during the corresponding matching can be referred to as arriving in a *dead spot* because arriving there is highly sub-optimal. To investigate if the tendency to place flows in dead spots is visible for all ToR pairs, the percentage of flows arriving in their corresponding dead spot is calculated per connection. Since we only create a single flow for a ToR pair per run, this percentage aggregates the 100 runs. Figure 6 shows the empirical CDF of the 56 percentages (one sample per ToR pair). The blue line represents the CDF before and the orange after applying the GA. After optimization, the percentage of flows arriving in their dead spot is higher for all ToR pairs. On average, the percentage increased by 20.14 percentage points from 17.45% to 37.59%.

VI. Conclusion & Future Works

The automated creation of adversarial traffic can help to gain insight into hidden weak spots of DCNs. In this paper, we present ADFAT that alters the arrival times of flows to find challenging demand, formulated the corresponding optimization problem, and evaluated the created adversarial arrival times for a RotorNet topology. The results show a GA can produce adversarial flow arrival times, and the algorithm is sensitive to temporal effects in DCNs. Periodicities of the routing algorithm and topology reconfigurations are identified and exploited in generating adversarial demand. Weak points in the temporal behavior of RotorNet are not only found on a global level, but the GA also finds challenging arrival times for specific connection pairs. The presented method can provide insight into arrival time patterns that lead to performance issues of RotorNet.

However, the flow-based simulator in this work is highly abstract. Connecting the system to more detailed packet-level simulation or real-world testbeds can create solutions containing information about how, e.g., protocol or firmware effects in DCNs can be exploited to create adversarial inputs. In this context, the scaling and efficiency of ADFAT concerning the cost of fitness function evaluations and topology sizes need investigation. Further, we plan to extend the chromosome coding to allow for multiple flows per rack pair and the joint optimization of arrival times and volumes.

References

[1] N. Taleb and E. A. Mohamed, "Cloud computing trends: A literature review," *Academic Journal of Interdisciplinary Studies*, vol. 9, Jan. 2020.
[2] W. M. Mellette *et al.*, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proc. ACM SIGCOMM*, p. 267–280, 2017.
[3] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter, "Expanding across time to deliver bandwidth efficiency and low latency," in *17th USENIX NSDI*, pp. 1–18, 2020.
[4] H. Ballani *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proc. ACM SIGCOMM*, pp. 782–797, 2020.
[5] J. Zerwas, C. Györgyi, A. Blenk, S. Schmid, and C. Avin, "Duo: A high-throughput reconfigurable datacenter network using local routing and control," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 7, mar 2023.
[6] C. Griner and C. Avin, "Cachenet: Leveraging the principle of locality in reconfigurable network design," *IFIP Networking*, pp. 1–3, 2021.
[7] M. N. Hall, K.-T. Foerster, S. Schmid, and R. Durairajan, "A survey of reconfigurable optical networks," *Opt. Switch. Netw.*, vol. 41, 2021.
[8] A. Blenk, P. Kalmbach, W. Kellerer, and S. Schmid, "O'zapft is: Tap your network algorithm's big data!," in *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, p. 19–24, 2017.
[9] W. Kellerer, P. Kalmbach, A. Blenk, A. Basta, M. Reisslein, and S. Schmid, "Adaptable and data-driven softwarized networks: Review, opportunities, and challenges," *Proceedings of the IEEE*, vol. 107, pp. 711–731, 2019.
[10] R. Meier *et al.*, "(self) driving under the influence: Intoxicating adversarial network inputs," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, p. 34–42, 2019.
[11] J. Zerwas *et al.*, "Netboa: Self-driving network benchmarking," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, pp. 8–14, 2019.
[12] S. Lettner and A. Blenk, "Adversarial network algorithm benchmarking," in *Proc. ACM CoNEXT*, pp. 31–33, 2019.
[13] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *International Conference on Learning Representations*, 2015.
[14] E. Sit and R. T. Morris, "Security considerations for peer-to-peer distributed hash tables," in *International Workshop on Peer-to-Peer Systems*, 2002.
[15] Z. Lin *et al.*, "Towards oblivious network analysis using generative adversarial networks," in *Proc. ACM HotNets*, p. 43–51, 2019.
[16] M. Nasr, A. Bahramali, and A. Houmansadr, "Defeating dnn-based traffic analysis systems in real-time with blind adversarial perturbations," in *30th USENIX Security Symposium*, pp. 2705–2722, 2021.
[17] J. Shapiro, "Genetic algorithms in machine learning," in *Advanced Course on Artificial Intelligence*, pp. 146–168, Springer, 1999.
[18] J. L. Pachuau, A. Roy, and A. K. Saha, "An overview of crossover techniques in genetic algorithm," in *International Conference on Modeling, Simulations and Optimizations*, 2020.