Strategic Games and Zero Shot Attacks on Heavy-Hitter Network Flow Monitoring

Francesco Da Dalt
ETH Zürich
francesco.dadalt@inf.ethz.ch

Adrian Perrig
ETH Zürich
aperrig@inf.ethz.ch

***Abstract*—Heavy–hitter detection underpins line-rate DDoS mitigation and rate-limiting, yet its resilience against adaptive adversaries is largely unexplored. We build an end-to-end evaluation framework that embeds heavy-hitter detection logic in a switch-level simulator, and auto-tunes its parameters using reinforcement learning to rate-limit elephant flows in the network. We subsequently confront the protection system with an adaptive adversary that learns to maximize throughput while evading detection and show that it manages to breach the configured bandwidth cap by up to 299%, exposing systematic blind spots. To harden the monitoring system we apply a form of joint adversarial training: detector and adversary co-evolve and reach an attack-defense Nash equilibrium in which the attacker's ability to exploit network bandwidth has been reduced by a factor 2.2×. Lastly, we show that it is possible to use machine learning to create smart packet-synthesizers which are able to perform bandwidth exploits on 8 out of 9 tested systems, without any prior knowledge on the targeted detection system. We refer to this as a zero-shot attack as it does not require knowledge about the targeted heavy-hitter detection system to perform its function. Our open-source framework helps quantify underilluminated attack surfaces and provides a constructive approach towards adversarially robust data-plane flow monitoring.**

I. Introduction

The resources of any real computer network are limited and therefore managing the sharing of resources among network entities is essential for the system's operation [1], [2], [3], [4]. The particular security aspect considered in this paper is the problem of ensuring that packet streams emitted by entities comply with bandwidth limitations [5]. This problem arises in the context of protecting network infrastructure against Denial-of-Service (DoS) attacks, but also when ensuring fair resource sharing and improving network QoS [6], [7], [8]. In general, to enforce compliant behavior of network flows requires to observe and monitor flows [9], [10], [11], where a flow is a sequence of packets which all share some common flow-identifier (e.g., source and destination addresses and ports).

In large-scale network systems, flow monitoring cannot simultaneously achieve low cost, high accuracy, and low latency [5], [12]. The volume of flows is too large compared to the resources available to the average network device, thereby forcing monitoring systems to make trade-offs [13]. A modern solution approach for this problem is to ignore the small flows and focus on the heavy-hitters [14], [15], [16], [17], [18]. Heavy-hitters are flows which consume a significantly larger amount of bandwidth compared to the average. The concrete specification of the flow-identifier defines the semantics of what a heavy hitter is: For example, grouping all packets with the same destination under one common identifier ID means that a heavy hitter will indicate a potential Distributed Denial-of-Service (DDoS) attack. In contrast, grouping packets by source allows locating potentially unfair or malicious traffic emitters that consume a larger amount of bandwidth than what would be fair, thereby identifying potential DoS attacks.

State of the art DDoS-protection methods [16], [17], [18] make use of specialized Heavy-Hitter-Detector (HHD) algorithms which have the ability to identify large flows in the network traffic using only a fraction of the memory and compute resources required by a naive approach. The tradeoff is that HHDs may report false-positives or false-negatives and their theoretic accuracy guarantees are often probabilistic [19], [10], [20], [21]. In particular since both HHD algorithms as well as DDoS protection systems are typically benchmarked and evaluated on static packet-traces that have either been collected or synthesized, not much focus has been put on whether traffic-patterns generated by an adapting adversary have the ability to avoid detection by the HHD algorithm and thus circumvent defensive rate-limiting systems.

This paper explores the extent to which smart adversaries are able to circumvent HHD-based protective rate-limiting systems by means of training smart agents using Reinforcement Learning (RL). The focus lies in particular on the heavy-hitter detection part of the defense structure.

Contributions: This work demonstrates that while HHD algorithms provide theoretic guarantees on their accuracy, evasive packet-patterns can be generated when considering more practical implementations. By means of joint training of adversary and defender we are able to improve the robustness of HHDs to evasive strategies, as well as generate artificial traffic patterns that are able to exceed the allowed bandwidth cap without any prior knowledge on the rate-limiting system.

We first describe the problem and environment setup in Section II and apply RL in order to automatically tune HHD hyperparameters (e.g., thresholds, refresh periods, decay-rates)

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.241301
www.ndss-symposium.org

to maximize the rate-limiting performance.

In Section III we introduce Neural Traffic Agents (NTAs) which are trained to generate traffic patterns which (partially) circumvent the rate-limiting systems in order to consume a larger amount of bandwidth than what would be allowed. Evaluations show that for all HHD algorithms it is possible to design evasive traffic patterns, ranging from 10% to 299% overuse using synthetic background traffic or from 78% to 671% when evaluated using captured background traffic.

Section IV-B demonstrates how combined adversarial training of attacker and defender can be used to improve the resilience of a HHD algorithm, e.g., ELASTICSKETCH [15], to evasive traffic patterns, reducing the exploitability from 299% to 139% by a factor of 2.2 and improving rate-limiting performance form 60.8 to 102.4 by a factor of 1.7.

Lastly, we show in Section V that it is possible for an NTA to develop sophisticated evasive traffic pattern strategies, without ever having seen any of the HHD algorithms we consider in this paper. We achieve this by creating a highly flexible surrogate HHD algorithm called Neural Monitor Agent (NMA) which serves as a training partner for the NTA. NTA and NMA train against each other in an attack-defense setup and learn increasingly better counter-strategies by interacting with the respective opponent. The final trained NTA performs successful zero-shot attacks on 8 out of 9 HHD algorithms, where zero-shot means that the attacking NTA does not know which HHD algorithm it is trying to evade, and it also cannot adjust its strategy once the attack has started.

All code associated with this paper is open-source [22].

Related Work: Defensive systems such as POSEIDON [16], JAQEN [17], and INDDOS [18] approach DoS from multiple angles. POSEIDON performs traffic monitoring and rate-limiting entirely in a P4 data plane and uses sketch-based compressive data-structures to measure flows. JAQEN uses special universal-sketches [23] to aggregate network-wide measurements for improved HHD detection, while INDDOS uses a special data-plane algorithm BACON to quickly detect and subsequently protect DDoS victims. All of these works however only evaluate against staticly generated attack-patterns. Methods such as NIDSGAN [24] and DEEP-PACKGEN [25] have shown that specially-designed traffic patterns can severely reduce the accuracy of traffic- and flow-classifiers, thereby posing a security risk. NIDSGAN uses a Generative Adversarial Network (GAN) [26] to perturb malicious traffic such that it gets misclassified by a Network Intrusion Detection System (NIDS). DEEP-PACKGEN employs a RL-based framework for finding adversarial traffic patterns that fool a target NIDS. Other approaches such as PAC-GAN [27] use GANs to generate realistic packet-level traffic, but do not consider the goal of generating antagonistic behavior.

II. RATE LIMITING HEAVY HITTERS IN THE DATA PLANE

In this work we regard the task of using HHD in order to rate-limit flows that exceed some set amount of bandwidth. HHDs are useful in this context as they allow reducing the number of flows that need to be monitored by a token-bucket system by focusing the attention on the heavy-hitters, thereby not requiring a token bucket for every flow in the packet stream.

Algorithm 1 Packet Processing Pipeline

```
1:  requires hhd                                   ▷ A HHD algorithm
2:  requires TBs                                   ▷ Token-buckets for rate-limiting
3:  procedure processPacket(p, time_now)
4:      hhd.update(p.ID, p.size, time_now)
5:      isHeavyHitter ← hhd.query(p.ID)
6:      if isHeavyHitter then
7:          TBs.insert(p.ID)
8:              ▷ If full, token buckets are freed in LRU order
9:      isAllowed ← TBs.withdrawTokens(p.ID, p.size)
10:     if isAllowed then
11:         p.forward()
12:     else
13:         p.drop()
```

A. Problem Environment

We consider a network system consisting of up to n concurrent flows whose packets pass through a simulated data-plane with an ingress-queue, a HHD algorithm, and a token-bucket based rate-limiter. Packets that are successfully forwarded by the data-plane arrive at a receiver which explicitly acknowledges received packets to the source.

1) Traffic Generation: We generate traffic pseudo-randomly such that the duration of individual flows follows an exponential distribution with mean 1 second, and the throughput per flow follows a Weibull distribution with mean $\mu = 20\ MTU/s$ where MTU is the maximum transmission unit. Since we simulate the environment, the MTU does not have to be concretely instantiated but it lies typically between $1.5\ kB$ and $65\ kB$ in real network systems. The number of active flows at any point in time is randomly distributed between 0 and n with mean $n/2$. Flows have an average rate of μ and a 10% chance to exceed the rate-limit threshold $2\mu = 40\ MTU/s$. This limit is chosen to not be too low to the point where it cannot be considered an excess anymore, and not too high which would allow too much unfair behavior to not be considered as bad by the monitor. The top 10% of flows have an expected rate of 2.87μ, while the bottom 90% have an expected rate of 0.65μ. Packet sizes are uniformly distributed between $0.01\ MTU$ and $1\ MTU$. Figure 1 visualizes statistics collected from a simulation trace. Except for experiments in Section V, we use $n = 1000$ as the maximum number of concurrent flows. We use freshly generated synthetic traffic in favor of captured traces in order to avoid overfitting of machine-learning methods to specific packet captures, thereby skewing results. For evaluation purposes we also use packet-captures to replay more realistic traffic. In particular we use data provided by MAWI [28], [29] (see Appendix F). Throughout this work we exclusively use synthetic data for training purposes.

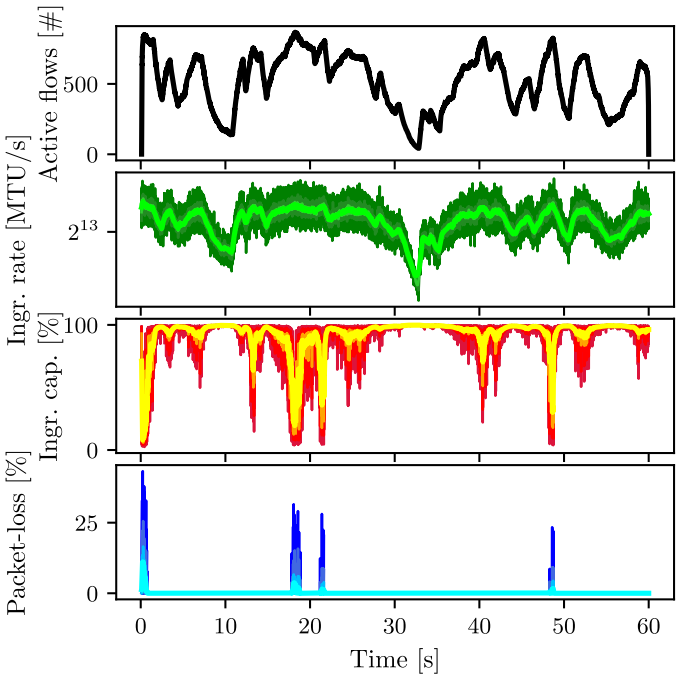


Fig. 1: Synthetic data trace: Color and brightness indicate the strength of the smoothening filter applied to the time-series.

2) Data Plane: The precise data-plane logic we simulate is as follows: An incoming packet lands in the ingress queue. The queue has a processing rate of $\mu \cdot n$ and will therefore become congested under high load. Packets that make it into the queue get processed by the HHD algorithm which decides whether to perform token-bucket based rate limiting on the flow or not. The number of available token-buckets is limited and we perform Least-Recently-Used (LRU) based eviction when necessary. The token buckets are configured to have a refill rate of 2μ and a capacity of 4μ. When a packet belonging to a flow that is in a token bucket does not have enough tokens, the packet is dropped. Otherwise the packet is forwarded. Algorithm 1 describes the high-level logic in pseudocode. The rate-limiter is constrained to 10 token-buckets, and the memory of the HHD algorithm is limited to $n/10$ memory cells, where a "memory cell" can hold either a number, a timestamp, or a flow-ID.

In summary, the HHD acts as a filter for which flows should be rate-limited using a token bucket. By construction, packets can only be lost due to congestion or rate-limiting (which has no false negatives). The data-plane logic is designed based on the common high-level characteristics of different state of the art rate-limiting systems [16], [17], [15], [14].

B. Tuning Heavy Hitter Detectors

In this work we perform experiments on nine different HHD algorithms: ELASTIC-SKETCH (ES) [15], HEAVY-KEEPER (HK) [14], MISRA-GRIES (MG) [30], SPACE-SAVING (SS) [31], COUNTMIN-HEAP (CM) [19], COUNTBAYES-HEAP (CB) [21], ALBUS (AL) [20], COUNT-HEAP (CH) [32], and HASHPIPE (HP) [33]. These nine methods cover a range of different approaches to HHD: CH, CM, and CB are sketch-based and estimate the size of *all* flows in the packet stream, and use a min-heap to track heavy hitters. SS, HP, and MG use a counter-based approach which focuses on measuring the size of a *few large* flows with higher accuracy. AL and HK extend on the counter-based approach and implement special approaches for decaying the counters in order to remove the need to manually refresh the counters over time. ES combines sketch-based and counter-based approaches to strike a better balance between measuring small and large flows.

These algorithms all have *some* hyperparameters that need to be tuned in a realistic deployment. Some hyperparameters are common across multiple methods:

- Sketch- and heap-based methods (e.g. CM and SS) require periodic refreshes in order to expel stale information from memory. The Refresh-Time (RT) is tuneable and has significant performance implications: A short RT reduces the time the HHD requires to identify new heavy hitters, while a longer RT increases the accuracy of the HHD as it aggregates more information into its decisions.
- Decay-based approaches (AL and HK) need to tune their decay-rates and related parameters in order to balance responsiveness and accuracy.
- All HHDs require a tuneable threshold parameter in order to decide whether some detected heavy-hitter is large enough to be subject to token-based rate limiting.

We use Reinforcement Learning (RL) in order to optimally set these parameters. RL is a type of machine learning in which an agent, in this case the HHD algorithm, improves itself by interacting with a training-environment and learning which hyperparameter configurations are best by observing the rewards it receives.

C. Optimizing HHD Hyperparameters using RL

Considering the scope of this paper, the two most relevant aspects of RL for the purpose of HHD tuning are: The policy π which decides which actions to take, and the reward function ρ which assigns rewards to the entity being trained.

$\pi^{\mathcal{H}}$ is a function that outputs a choice of hyperparameters for the HHD algorithm $\mathcal{H}$ (e.g. refresh-period, threshold, decay-rate, etc.). We implement this function as a 1-layer feedforward neural network with input dimension 0, i.e. a constant function that has no arguments, but which can still be trained and optimized using deep-learning based backpropagation. The output of a policy-function like $\pi^{\mathcal{H}}$ is referred to as an *action* and is fed into a running HHD simulation.

ρ^{HHD} is a function that gives rewards to the HHD depending on the execution trace of the simulation. Concretely, we define ρ^{HHD} to do the following: Every time that the HHD processes a packet of size s and with id ID, and marks the flow ID for token-bucket based rate-limiting, the HHD receives a reward of $-s$. Whenever the token-bucket drops a packet of size s' due to rate-limiting, the HHD receives a reward of $2 \times s'$. To summarize, the HHD receives a penalty for inserting flows into the rate-limiter, but it receives rewards whenever the rate-limiter drops a packet.

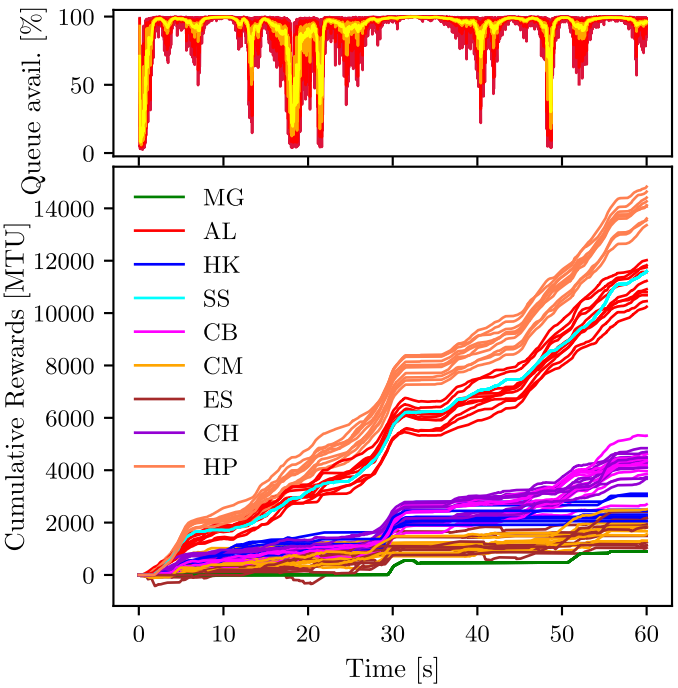


Fig. 2: Cumulative rewards of tuned HHDs: Shown are 10 evaluations for each HHD method over the same packet-trace. Each run randomizes the used hash-functions which is why sketch-based methods (e.g., CB) exhibit random fluctuations, while hash-less methods (e.g., MG and SS) are deterministic.

This reward-scheme serves the purpose to penalize the HHD for marking too many flows as heavy-hitters, while rewarding it whenever a reported heavy-hitter exceeds its allowed rate. The choice of this reward function is motivated in Section V. The reward function ρ^{HHD} is the same for all HHDs we tune.

1) Training Procedure: We use Proximal Policy Optimization (PPO) [34] in order to learn the best choice of $\pi^{\mathcal{H}}$ for each HHD $\mathcal{H}$. We simulate the training environment natively as a MDP. Each training step begins by gathering execution traces of 32 independent (randomized) simulations, where each simulation spans 6 seconds. We then perform 100 steps of PPO based on the gathered data. Each step of PPO optimizes $\pi^{\mathcal{H}}$ such that it maximizes the cumulative received rewards. Every 10 training steps (i.e. 1000 PPO steps) we evaluate $\pi^{\mathcal{H}}$ on longer simulations of 60 seconds (again over 32 independent random runs). The evaluation-performance of $\pi^{\mathcal{H}}$ is measured by the average Reward per Second (RpS) it receives, higher being better. We use the RpS measurements to checkpoint the best trained policy $\pi^{\mathcal{H}}$. If the best RpS has not been surpassed in the last 100 training steps, we reduce the learning-rate to allow for better fine-tuning. Training stops if the learning-rate reduction brought no improvement. For more details we refer to Appendices D and E.

2) Tuned HHD Performance: The training procedure described in the preceding section yields optimal policies π^{CM}, $\pi^{AL}, \ldots$ for each HHD algorithm. We evaluate the methods by measuring the *cumulative rewards* earned over 60 seconds time

	π^{AL}	π^{CB}	π^{CM}	π^{ES}	π^{HK}	π^{MG}	π^{SS}	π^{CH}	π^{HP}
RpS ($\mathfrak{S}$)	177.4	64.5	30.0	22.2	39.5	15.0	189.5	63.3	227.4
RpS ($\mathfrak{R}$)	1116.2	941.7	140.3	122.1	33.1	699.8	950.2	946.2	943.1

TABLE I: Average Reward per Second (RpS) achieved by different HHD algorithms after RL-based hyperparameter tuning on synthetic ($\mathfrak{S}$) and real ($\mathfrak{R}$) traffic.

and display the results for a single packet-trace in Figure. 2. Table I provides statistically more significant measurements collected over 320 independent runs. Rewards are significantly higher on the real compared to the synthetic traces because in real data streams there are fewer but more extreme outliers.

A first surprising observation is that the different methods, despite all having a common goal of identifying heavy-hitters, exhibit vastly different performance when measured based on the gathered rewards. The reason why this is possible is that the HHD algorithm must cooperate with the rate-limiter in order to gather rewards which requires three characteristics:

- Responsiveness: The HHD must quickly identify heavy-hitters while they are still hitting. Detecting a heavy hitter that has already elapsed 90% of its lifetime is of little value.
- Magnitude: To correctly prioritize which flows to rate-limit, the magnitude of flows must be inferrable from the HHD. It is not sufficient to know whether a flow is a top-k heavy-hitter; we require to know which of the heavy hitters are the biggest and which we can expect to exceed the allowed rate in order to maximize the impact of the rate-limiter.
- Selectivity: In order for a token-bucket to perform its task and generate value, it requires a minimum amount of time assigned to a single flow in order to deplete the heavy-hitter's token capacity and start dropping its packets. The HHD therefore needs to be selective with which flows it marks as heavy-hitters as it will be unable to receive positive rewards if it switches too quickly.

These qualities determine the ability of HHDs to succeed in the experimental setup we consider in this paper. From the measurements we see that for example AL and HP perform considerably better than the other methods and we attribute this to a combination of good selectivity and magnitude estimation: AL can be highly selective by tuning its rate-parameter and only allowing the highest-rate flows to be marked for rate-limiting. HP on the other hand provides very stable magnitude estimates by counting the aggregate volume of a few flows. According to the collected data, what appears to set these methods apart from conceptually similar approaches such as MG, ES, and HK is the ability to estimate the magnitude under different traffic conditions: AL and HP have the ability to estimate the absolute rate of individual flows independent of how much other traffic is passing through. On the other hand, MG for example decrements the counters at a rate proportional to the amount of traffic. This helps it expel stale information quicker, but it comes at the cost of coupling the estimated magnitude of heavy-hitters to the overall link-rate, making it more difficult to decide whether a

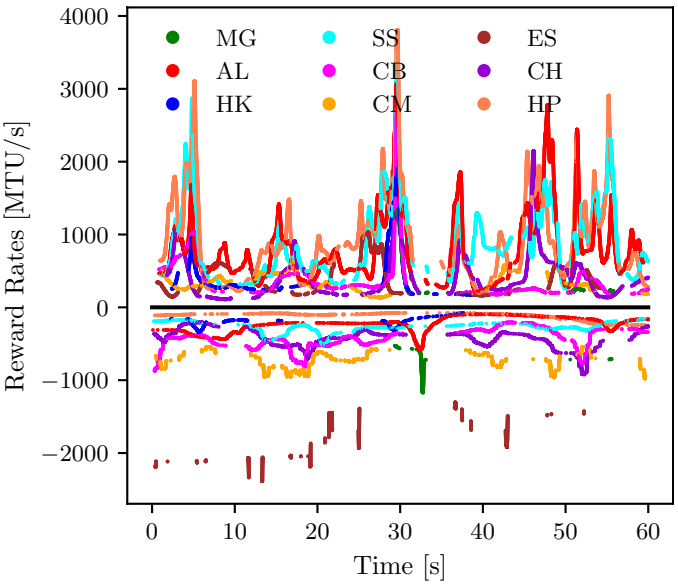


Fig. 3: Reward rates of tuned HHDs: The positive and negative axes indicate the (smoothed) rate at which HHDs earn rewards (positive axis) and pay penalties (negative axis) respectively.

flow is exceeding its allowed rate. This behavior can be seen in Figure 3 which plots the rate of reward gain and loss over time. At 32 seconds we see a sudden increase in penalties paid by MG which is due to the low amount of traffic at that point in time leading to almost no counter-decay and thus inflating the magnitude estimates of flows.

CM suffers from the same issue as decay-based methods except in this case the flow estimates generally increase with higher traffic. The related sketch-methods CH and CB partially counteract this problem by de-biasing the flow estimates, thereby reducing the influence of the current traffic-rate on the scale of the flow magnitude estimates.

III. Circumventing Rate Limiting with Learned Attacks

The previous section has described how we tune HHD hyperparameters in order to maximize the effectiveness of a limited number of token-bucket rate limiters in a simulated setting. In this section we will now show how an adaptive attacker can learn to circumvent flow monitoring systems and in particular HHDs through smart packet synthesis, and achieve in some cases a disproportionally high bandwidth consumption by partially evading detection.

To that end we create a Neural Traffic Agent (NTA) with the ability to generate and emit packets based on decisions taken by a Neural Network (NN). The NTA itself is modeled as a RL agent that can learn based on the actions it takes. In particular, we set up its reward function ρ such that for every packet with size $s \in [0.01MTU;\ 1MTU]$ that the NTA emits, it pays a penalty of $-s$. If the packet makes it through the HHD and rate-limiter without being dropped or lost, the NTA receives an ACK and a reward of $2 \times s$. Thus for every packet it emits it will end up either receiving in total $+s$ or $-s$ based on whether the packet arrives at its destination or not. For a brief look into the effects of reward see Appendix C. Informally, this reward function incentivizes the NTA to send as much traffic as possible over the link, while avoiding packet-loss (in particular due to token bucket rate-limiting) as much as possible. Analogously to the HHD methods we measure performance in terms of average RpS. In particular, the RpS of the NTA captures the rate of successfully transmitted data minus the rate of lost data (in MTU/s). If the NTA achieves an RpS higher than 40 it is exceeding its allowed rate set by the token-buckets, see Section II-A1.

Algorithm 2 Neural Traffic Agent

requires $M \in \mathbb{R}^{20\times 6}$ ▷ History matrix
requires NN ▷ Time-series transformer
$time_{last} \leftarrow time_{now}$
procedure PERFORMACTION($time_{now}$)
  $a \leftarrow getAcksSince(time_{last})$
  $time_{last} \leftarrow time_{now}$
  $M[0, 2] \leftarrow a$
  $(t, n, s) \leftarrow NN(M)$ ▷ t: wait, n: count, s: size
  $M[:, 0] \leftarrow M[:, 0] + 1$
  $M \leftarrow shiftRowsDown(M)$
  $M[0] \leftarrow [0, time_{now}, \_, t, n, s]$
  $sendPackets(t, n, s)$
  ▷ Send n packets of size s within the next t seconds
  $sleep(t)$

A. Threat Model

In the context of this work the NTA acts as an "attacker" while the HHD is the "defender". The capabilities and threat model of the NTA are identical to those of other flows in the network and it can therefore not spoof its address. It has the ability to receive ACKs for the successfully transmitted packets and can emit packets with sizes in the range between $0.01MTU$ and $1MTU$ with arbitrary frequency. The latency from NTA to the switch is $100ms$, from the switch to the receiver $200ms$ and from the receiver back to the NTA $100ms$.

B. Enabling Smart Packet Synthesis

The premise of the NTA is for it to control the timing and size of packets such that it is able to at least partially evade detection by the HHD and thus consume a larger amount of bandwidth than allowed. To enable this, we require the NTA to have the ability of generating non-trivial traffic patterns based on past observed events such as received ACKs.

To that end we employ a transformer-based NN architecture as the NTA's policy function π^{NTA} [35]. The policy π^{NTA} outputs a vector $[t, n, s]$ of 3 elements : 1) how much time t should elapse before π^{NTA} is evaluated again, 2) the number of packets n we want to emit in the next t seconds, and 3) the size s of the next n packets we emit.

The function π^{NTA} takes as input a 20×6 dimensional matrix M where the i-th row of M contains information about the i-th last action taken. More precisely, the i-th row of M

Training	ACKs		~~ACKs~~			
Evaluating	ACKs		~~ACKs~~			
($\mathfrak{S}$)	NTA RpS	HHD RpS	NTA RpS	HHD RpS	NTA RpS	HHD RpS
$\pi^{NTA\times AL}$	73.8	181.9	52.5	186.7	64.1	190.2
$\pi^{NTA\times CB}$	71.2	101.5	62.0	65.1	63.1	118.3
$\pi^{NTA\times CM}$	95.2	56.1	51.4	30.2	85.6	57.5
$\pi^{NTA\times ES}$	159.4	60.8	45.4	22.9	67.6	126.7
$\pi^{NTA\times HK}$	44.0	48.4	42.4	44.6	42.4	56.1
$\pi^{NTA\times MG}$	54.9	27.2	23.1	15.3	48.2	29.5
$\pi^{NTA\times SS}$	67.0	189.7	47.3	187.2	82.1	191.7
$\pi^{NTA\times CH}$	66.6	102.3	67.9	100.4	61.4	112.7
$\pi^{NTA\times HP}$	61.7	233.2	68.7	225.0	73.4	227.6

TABLE II: Reward rates of NTA and HHD algorithms: Values in row $\pi^{NTA\times\mathcal{H}}$ give the score of the trained NTA policy $\pi^{NTA\times\mathcal{H}}$ and of the HHD algorithm $\mathcal{H}$ running policy $\pi^{\mathcal{H}}$. ACKs and ~~ACKs~~ indicate whether or ~~not~~ the NTA receives ACK-packets during training and/or evaluation.

Training	ACKs		~~ACKs~~			
Evaluating	ACKs		~~ACKs~~			
($\mathfrak{R}$)	NTA RpS	HHD RpS	NTA RpS	HHD RpS	NTA RpS	HHD RpS
$\pi^{NTA\times AL}$	75.9	1100.7	51.6	1105.9	71.1	1100
$\pi^{NTA\times CB}$	114.1	962.8	65.0	959.1	107.8	974.0
$\pi^{NTA\times CM}$	170.5	155.1	47.3	132.8	116.8	167.2
$\pi^{NTA\times ES}$	308.2	125.1	46.0	239.1	106.6	257.3
$\pi^{NTA\times HK}$	31.6	280.7	45.0	30.8	30.3	323.4
$\pi^{NTA\times MG}$	71.1	659.1	23.2	745.5	61.3	693.5
$\pi^{NTA\times SS}$	75.0	954.1	47.8	955.3	86.4	953.0
$\pi^{NTA\times CH}$	104.5	979.5	107.6	968.6	110.2	981.0
$\pi^{NTA\times HP}$	72.9	948.0	70.6	944.0	76.0	946.8

TABLE III: Evaluation of NTAs and HHDs on real traffic.

contains the following data: 1) i itself indicating the order of this data-point, 2) the time t elapsed between the i-th past action and now, 3) how many ACKs have been received between the i-th and $i-1$-th past actions, and 4) the output of π^{NTA} at the i-th past action. In simpler terms, M can be understood as a time-series of length 20 with information about the past performed actions and received ACKs. Algorithm 2 described the NTA's procedure in pseudocode.

We apply a time-series transformer to our input M by performing positional encoding of the the temporal data-features of M, embedding the remaining features of M using a fully connected neural network, adding the two and giving it as input to a transformer with 2 layers, 4 heads, and 16-dimensional input.

C. Training and Evaluating the Neural Sender Agent

As we have seen in Section II-C2, the characteristics of HHDs vary greatly which is why we train a custom attack-policy $\pi^{NTA\times\mathcal{H}}$ for each mentioned HHD method $\mathcal{H}$, where $NTA\times\mathcal{H}$ generally means "NTA trained versus $\mathcal{H}$".

Tables II and III present the evaluation of the trained NTA policies $\pi^{NTA\times\mathcal{H}}$ on synthetic ($\mathfrak{S}$) and real ($\mathfrak{R}$) traffic

($\mathfrak{S}$)	$NTA\times HHD_{pre}$	$NTA\times HHD_{post}$	$NTA\times HHD_{rel}$
Lin. Corr.	0.21 (p=0.29)	0.44 (p=0.022)	0.41 (p=0.033)
Rank Corr.	0.27 (p=0.16)	0.47 (p=0.014)	0.18 (p=0.34)

TABLE IV: Correlation between NTA and HHD performance: The first column measures correlation between NTA RpS from Table II and HHD RpS from Table I (i.e. pre-attack). The second column measures correlation only from Table II (i.e. post-attack). The third column measures correlation between NTA RpS and the HHD's relative RpS gain between pre- and post-attack. Statistical significance is indicated by p.

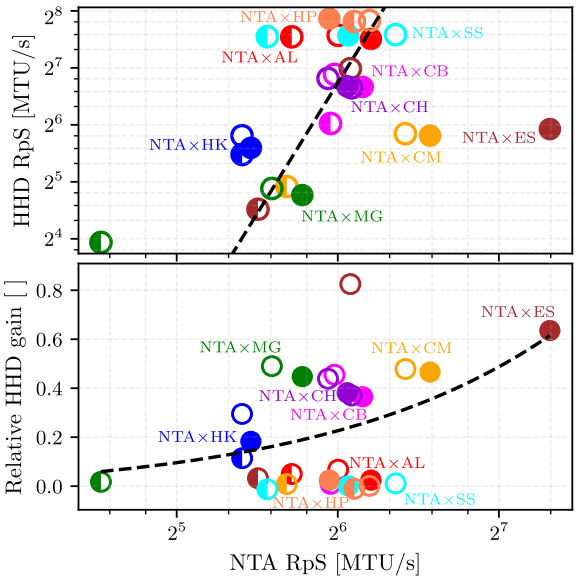
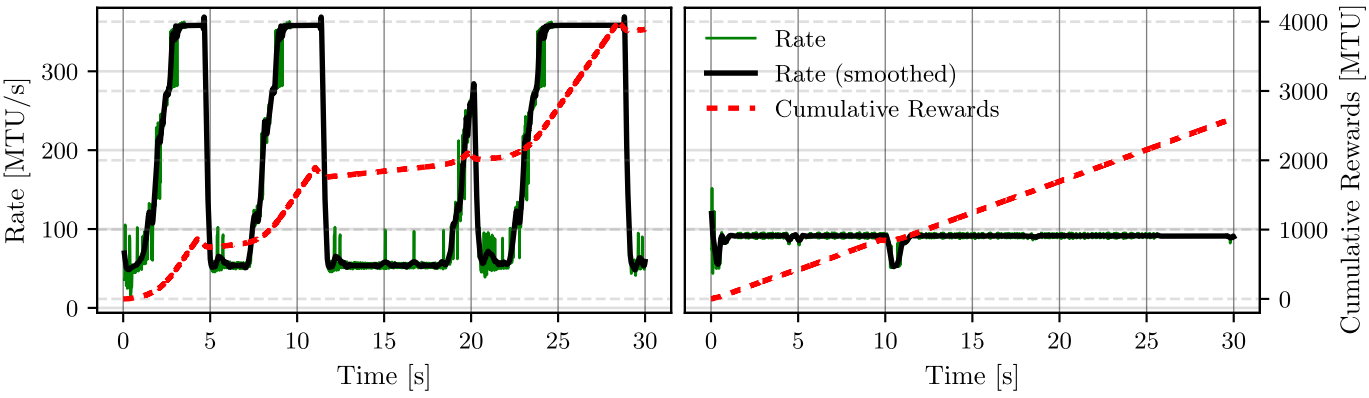


Fig. 4: Table II visualized. A filled left/right side of the marker indicates the NTA received ACKs during training/evaluation. The dashed lines indicate the principal linear correlation axes. The relative HHD gain is measured between Table I and II.

respectively. The first two rows indicate whether the NTA was trained/evaluated with the ability to receive ACKs or not. The other rows are structured such that for example $\pi^{NTA\times MG}$ indicates that we evaluate the learned NTA policy $\pi^{NTA\times MG}$ against the MG heavy-hitter detector with previously learned policy π^{MG}. The NTA's and HHD's performances are indicated in terms of RpS in their respective columns. The first thing we would like to point out is that the NTA achieves an RpS greater than 40 against all HHDs when the ACK-configuration matches between training and evaluation which means that in all cases it has found a way to exceed its allowed rate of 40 MTU/s, even when it cannot receive feedback information through ACKs. Secondly, by comparing with Table I we see that all HHDs gain significantly higher rewards when pressured by the NTA compared to when no smart flow emitter is present in the system. This makes intuitive sense as the NTA's behavior of aggressively grabbing bandwidth makes for an additional opportunity for the HHDs to collect rewards by imposing a rate-limit on it. Furthermore, the attacks learned by the NTA on synthetic data also transfer



(a) Visualizing $\pi^{NTA \times ES}$ evaluated against π^{ES}

(b) Visualizing $\pi^{NTA \times AL}$ evaluated against π^{AL}

Fig. 5: Traffic patterns generated by the NTA evaluated against the respective targeted HHD (ES and AL). The rate is computed as the quotient of packet-size and inter-packet interval. The cumulative rewards indicate the aggregate sum of payload volume that bypassed the HHD minus the volume that was dropped by the HHD.

to the experiments with real background traffic, and in fact perform even better than in the synthetic case since it can mask its behavior behind other extreme heavy-hitters that appear more prominently in the real traffic than the synthetic one. A phenomenon which surprised us at first is that there seems to be no significant negative correlation between the HHD performance before the NTA's attacks (Table I) and the NTA's attack scores (Table II). This is at first glance unintuitive in so far as that one would expect the NTA to achieve better results when matched against HHDs that have low performance in terms of RpS. Table IV presents quantitative numbers on this and we see that the correlation between pre-attack HHD and NTA is insignificant and if anything weakly positive. As a followup we also measure that there is significant positive correlation between NTA and HHD score post-attack. This is visualized in Figure 4.

Our understanding of this phenomenon is that if a HHD such as AL achieves higher performance than MG it suggests that it has managed to tune its policy π^{AL} to fit very well to the general traffic distribution, whereas π^{MG} is unable to adjust perfectly to the packet stream statistics. It appears however that the high degree of specialization of π^{AL} leads to more vulnerabilities in terms of allowing an adaptive adversary such as NTA to find exploitative traffic patterns since π^{AL} relies more on the traffic characteristics than π^{MG}, and therefore is not prepared to handle traffic patterns outside of its narrow comfort zone.

The possibility of this phenomenon arising should serve as a word of caution when deciding on which HHD algorithm to deploy as high performance and accuracy achieved when benchmarked on static network traces does not necessarily translate into protecting well against adaptive adversaries.

D. Analyzing Smart Traffic Patterns

To give some more concrete insight and context into the adversarial traffic-patterns generated by the NTA we analyze two synthesized packet-traces in this section. Figure 5 shows the strategies developed by $\pi^{NTA \times ES}$ and $\pi^{NTA \times AL}$ against ES and AL respectively. We see that there are high-level similarities between the two shown strategies. Both attack patterns boil down to alternating between a high sending rate (360 MTU/s and 91 MTU/s respectively) and a low sending rate (53 MTU/s for both). In essence, the NTA has learned to use a very aggressive sending rate when it is not monitored by the HHD, and to use a low sending rate when it is being rate-limited. Interestingly, the NTA has learned in two independent training sessions against different HHD algorithms, that 53 MTU/s seems to be an optimal choice as low sending rate. At first glance this seems odd since the rate limit is 40 MTU/s, meaning that even when holding back, the NTA attempts to use 13 MTU/s more bandwidth than allowed, meaning that it chooses to receive $40 - 13 = 27$ RpS opposed to the 40 that it could get by following the rate-limit. Since this is a clearly suboptimal strategy in the short-term, the NTA must see a long-term benefit in this. Our understanding is that the NTA uses the packet-loss incurred by the rate-limiting to sense the earliest opportunity when it can switch to the high sending rate again. If the NTA were to send at the allowed rate of 40 MTU/s, it would never be subject to rate-limiting but it would also never learn whether it is being monitored by a token bucket or not. On the other hand if the rate is too high, the penalties paid may outweigh the benefit of acquiring the knowledge of when to strike with the high rate. Ultimately the tradeoff depends on how long the NTA thinks that it may take to be evicted from the rate limiter. Intuitively, if the time is expected to be long, say 10 seconds, it is not worth performing aggressive probing since the gains do not outweigh the expected costs one incurs for the duration of the probing. On the other hand if we expect to be evicted from the rate limiter within 0.1 seconds, probing can be done more aggressively since we do not expect it to last for long. A further similarity between the two distinct NTA strategies is the occurrence of steps in the rates. This can be seen more clearly in Figure 6 which plots the traces

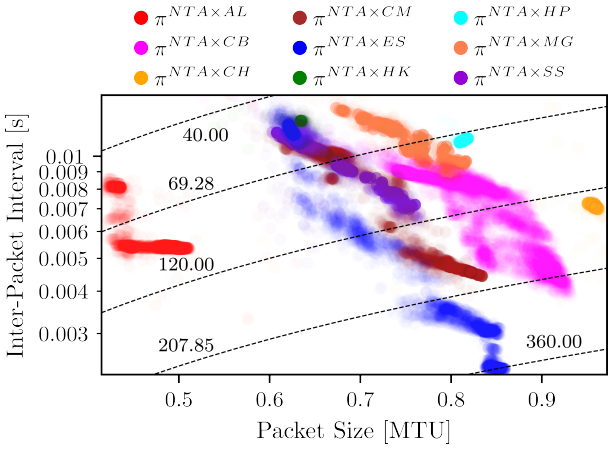


Fig. 6: Phase-space diagram of NTA policies. Every dot represents an action taken by the NTA in a collected trace. The dashed lines are isolines indicating equal sending-rate.

in phase-space where the time dimension has been removed. For $\pi^{NTA\times AL}$ one observes four clusters: two strong ones at sending rates 53 and 91 as already discussed, but also a minor cluster at 69 as well as a fourth cluster at 82 which is connected to the cluster at 91 and therefore hard to distinguish. For $\pi^{NTA\times ES}$ it is more clear and it is possible to clearly separate at least four clusters at the following levels: 53, 120, 310, and 360. These jumps can also be seen in Figure 5a as whenever the NTA goes from the low to the high sending rate, it does not ramp up its rate instantly, but rather does so in what seems to be 4 steps. These steps are not involuntary but serve the purpose of checking whether the NTA has for sure been evicted from the rate limiter or not. Evidence for this is that each step takes around $400ms$ which corresponds precisely to the NTA's roundtrip-time which suggests that the NTA waits for ACKs confirming that the step-up in sending rate was successful before considering a further increase. Figure 6 tells us that these steps are a common pattern among the many learned strategies as can be seen based on the clear clusters of $\pi^{NTA\times CM}$ and $\pi^{NTA\times SS}$.

IV. Robustifying Heavy Hitter Detection using Multi-Agent Reinforcement Learning

The preceding section has shown that generating adversarial traffic patterns that circumvent HHDs and rate-limiters is possible. The extent to which an NTA can avoid the effects of being rate-limited depend on the concrete HHD algorithm used. An odd observation was that highly performant HHD systems like AL can provide a larger attack surface for adaptive attackers despite its robust appearance, compared to for example HK which achieves consistently lower scores. In this section we show that it is possible to increase the robustness of HHD algorithms like AL to adversarial attacks, as well as greatly enhance the effectiveness of low-performers like ES at rate-limiting flows echibiting antagonistic behavior. We achieve this by means of applying multi-agent reinforcement learning and allowing attacker (i.e. NTA) and defender (i.e. HHD) to learn from each other in an iterative manner, thus creating increasingly more robust attack and defense policies. We demonstrate this approach in applied to ES and AL.

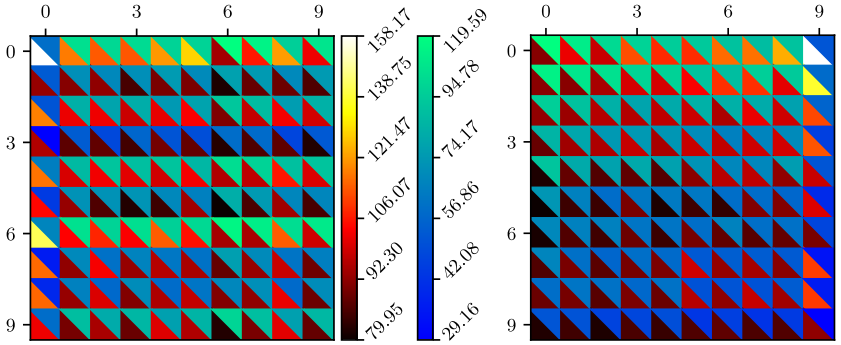
Algorithm 3 PSRO applied to NTA×ES

requires $\pi^{NTA\times ES}$ ▷ Initial NTA policy
requires π^{ES} ▷ Initial ES policy
procedure TRAIN(env, n)
  $\mathcal{P}^{NTA\times ES} \leftarrow \{\pi^{NTA\times ES}\}$
  $\mathcal{P}^{ES} \leftarrow \{\pi^{ES}\}$
  for i in $range(n)$ **do**
    $G \leftarrow evaluatePairwise(\mathcal{P}^{NTA\times ES}, \mathcal{P}^{ES})$
      ▷ G is the game between NTA and ES
    $(N^{NTA}, N^{ES}) \leftarrow NashEquilibrium(G)$
      ▷ N^{NTA} is the distribution over $\mathcal{P}^{NTA\times ES}$
      ▷ N^{ES} analogously over $\mathcal{P}^{ES}$
    $env.set(HHD) \leftarrow random(\mathcal{P}^{ES}, N^{ES})$
    $\pi^{NTA\times ES}_{new} \leftarrow env.train(NTA)$
    $env.set(NTA) \leftarrow random(\mathcal{P}^{NTA\times ES}, N^{NTA})$
    $\pi^{ES}_{new} \leftarrow env.train(HHD)$
    $\mathcal{P}^{NTA\times ES} \leftarrow \mathcal{P}^{NTA\times ES} + \pi^{NTA\times ES}_{new}$
    $\mathcal{P}^{ES} \leftarrow \mathcal{P}^{ES} + \pi^{ES}_{new}$
  return $\mathcal{P}^{NTA\times ES}$, $\mathcal{P}^{ES}$

A. Population-based Multi-Agent Reinforcement Learning

The key idea we describe in this section is based on the observation that while $\pi^{NTA\times ES}$ may be very successful at finding exploits in π^{ES}, there may also exist a new ES policy $\tilde{\pi}^{ES}$ which is very successful at detecting and limiting $\pi^{NTA\times ES}$. In general, for both NTA as well as ES, there is always the possibility of the opponent finding a new strategy that counters your own strategy. By iteratively finding more and more counter-strategies for both NTA as well as ES we are able to train better and better policies that leave less room for counter-policies.

The training method called Policy-Space Response Oracles (PSRO) [37] formalizes this idea into a concrete method. We have reformulated the core of the training procedure within the context of of this paper and noted in Algorithm 3. On a high level, the training procedure works by iteratively growing a population of NTA policies $\mathcal{P}^{NTA\times ES}$ and ES policies $\mathcal{P}^{ES}$. The populations are expanded by training new policies (e.g. $\pi^{NTA\times ES}_{new}$) against a mixture of different opponent policies (e.g. a mix over $\mathcal{P}^{ES}$) in order to allow the policy being trained to learn how to counter multiple strategies. The mix of the opponents policies is not chosen arbitrarily but rather based on the opponent's game-theory optimal strategy, i.e. the Nash equilibrium [38]. The output of PSRO are the two populations $\mathcal{P}^{NTA\times ES}$ and $\mathcal{P}^{ES}$. By probabilistically combining all policies within $\mathcal{P}^{ES}$ according to the optimal Nash equilibrium, we obtain a more robust HHD that provides a well-rounded defense due to having experienced all possible adversarial attacks by an adaptive packet emitter during the PSRO training procedure. We use mixed-integer linear programming [39] to compute Nash equilibria.



(a) NTA and ES policies evaluated and ordered based on the PSRO iteration (0 to 9) in which they were trained.

(b) Policies from Figure 7a ordered using nearest-point hierarchical (row- and column-) clustering [36].

	NTA RpS		HHD RpS	
$(\mathfrak{S})$	π^{ES}_{cot}	π^{ES}	π^{ES}_{cot}	π^{ES}
$\pi^{NTA\times ES}_{cot}$	95.6	124.2	102.4	58.5
$\pi^{NTA\times ES}$	100.7	159.4	115.8	60.8
$(\mathfrak{S})$	π^{AL}_{cot}	π^{AL}	π^{AL}_{cot}	π^{AL}
$\pi^{NTA\times AL}_{cot}$	65.5	70.8	183.6	178.8
$\pi^{NTA\times AL}$	66.0	73.8	191.7	181.9

(c) Policies produced by the NTA co-training with ES and AL evaluated against each other as well as against strategies from Sections II and III.

Fig. 7: The matrix-plots show the evaluation of NTA and ES policies trained by PSRO. Each column represents one policy from $\mathcal{P}^{ES}$, while each row is one of the attack-patterns in $\mathcal{P}^{NTA}$. The lower-left triangles measure NTA's RpS while the upper-right measure ES's RpS. For example consider the cell in row 6 and column 7 of Figure 7a. The dark red triangle indicates the NTA receiving ≈ 92 RpS, while the green triangle means that ES achieves ≈ 100 RpS. These two scores capture the performance of the 6th NTA and 7th ES policies generated by PSRO when evaluated against each other.

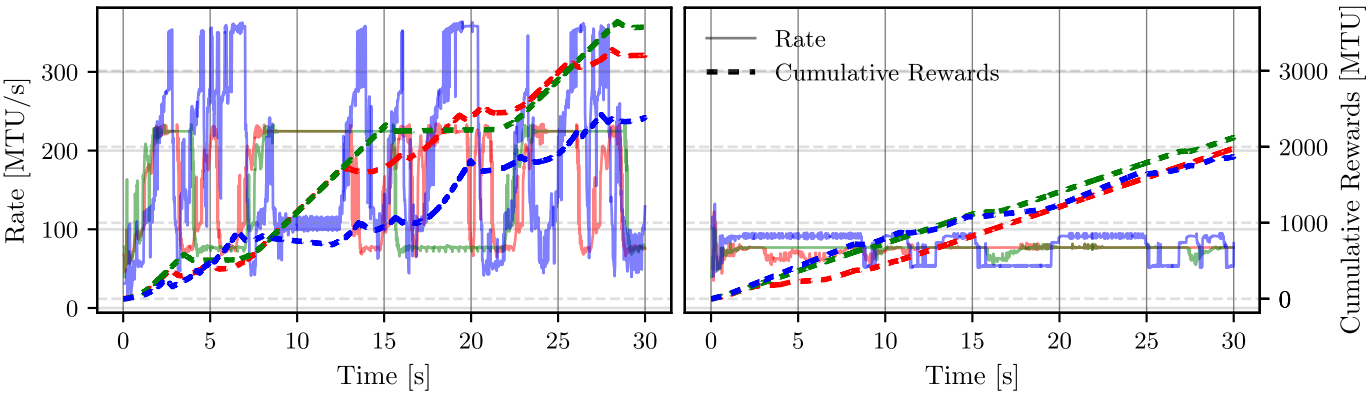
1) Intuition on Merging Policies for Robustness: Consider a hypothetical scenario in which $\mathcal{P}^{NTA\times ES}$ and $\mathcal{P}^{ES}$ each contain 3 policies (call them "rock", "paper", and "scissors"). The NTA picks to play a policy (i.e. and adversarial traffic pattern) from $\mathcal{P}^{NTA\times ES}$, while the HHD picks a configuration for ES from $\mathcal{P}^{ES}$. Assuming the usual rules of rock-paper-scissors (i.e. for example if NTA picks $\pi^{NTA\times ES}_{rock}$ and the HHD picks π^{ES}_{paper}, then the NTA achieves -1 RpS while the HHD gets $+1$ RpS) the optimal strategy for the HHD is to pick uniformly at random between the three available policies for ES as this is the best choice which minimizes the ability for the NTA to cause damage. This way we have combined three HHD strategies (rock, paper, and scissors) which individually can each be exploited with a targeted attack, into a single (probabilistic) policy that perfectly defends against any attacks from the NTA. While such simple and clear-cut examples usually do not occur in practice, the core idea remains very much applicable.

B. Evaluation of Adversarially Robust Policies

We have performed 9 iterations of PSRO applied to the adversarial setting between NTA and ES, resulting in two policy populations $\mathcal{P}^{NTA\times ES}$ and $\mathcal{P}^{ES}$ of size 10 each; 1 initial policy plus 9 new policies each. The resulting strategic game between the attacker-entity NTA and defender-entity ES is shown in Figure 7. The first Subfigure 7a displays the game with policies ordered based on which iteration of PSRO they were trained, and looks therefore quite unstructured. In the second Subfigure 7b we order the policies based on their similarity instead and we discover significant structure and similarity amongst both attack- as well as defense-strategies. For example, the first 3 columns (ES strategies) in Subfigure 7b all exhibit very similar behavior. Likewise, the rows 7 and 8 form a group of similar NTA policies. To link this observation back to the rock-paper-scissors analogy from Section IV-A1, it would be like having two instead of one "rock" options: The rocks may be different but they essentially perform the same task.

We have furthermore performed an analogous PSRO-based training run between NTA and AL, yielding strategy populations $\mathcal{P}^{NTA\times AL}$ $\mathcal{P}^{AL}$ of size 10 each. In both runs, the Nash equilibria $N^{NTA\times ES}$, $N^{NTA\times AL}$, N^{ES}, and N^{AL} appearing in the training procedure (see Algorithm 3) can happen to both pure as well as mixed. The final output however has a pure equilibrium, meaning a single strategy from each of $\mathcal{P}^{NTA\times ES}$, $\mathcal{P}^{ES}$, $\mathcal{P}^{NTA\times AL}$, and $\mathcal{P}^{AL}$ is chosen as the best one. These strategies are indicated with subscript cot and evaluated in Table 7c.

1) Quantifying Robustness Increase: Section III-C has previously shown that an adaptive adversary such as NTA can achieve an average RpS of 159.4 when matched against a rate-limiter using ES for heavy-hitter detection. 159.4 RpS corresponds to an overuse of 299% compared to the allowed rate of 40 MTU/s. From Table 7c we see that after improving ES through PSRO-based adversarial training, the highest score achievable by the NTA is 95.6 RpS, which corresponds to an overuse of 139%. The adversarial training has thus allowed to reduce the exploitability of bandwidth by adaptive adversaries by a factor of $2.2 = 299\% \div 139\%$. At the same time, we have improved the performance of ES from 60.8 to 102.4 RpS by a factor of 1.7. Both old ($\pi^{NTA\times ES}$) as well as new ($\pi^{NTA\times ES}_{cot}$) attacks perform significantly worse on ES after co-training (π^{ES}_{cot}) compared to before (π^{ES}). Regarding the co-training between NTA and AL we observe qualitatively the same results except that the exploitability is reduced by a smaller amount from 85% to 64% by a factor of 1.3. The reason for this is that AL has a stronger baseline performance, therefore leaving less room for improvement by means of adversarial co-training.



(a) Traces with ES as HHD

(b) Traces with AL as HHD

Fig. 8: The left and right plots are attack traces resulting from PSRO-based adversarial co-training between NTA and ES / AL respectively. In blue we have the attacks from Section III ($\pi^{NTA\times ES}$ and $\pi^{NTA\times AL}$) pitched against π^{ES}_{cot} and π^{AL}_{cot} respectively. Red and green indicate the co-trained NTA attacks $\pi^{NTA\times ES}_{cot}$ and $\pi^{NTA\times AL}_{cot}$, but in the green case it is against the old HHD policies (π^{ES} and π^{AL}), while in red it is against the co-trained HHD policies π^{ES}_{cot} and π^{AL}_{cot}.

2) Visualizing the Effects of Multi-Agent Adversarial Training: In Figure 8 we visualize the final attack-traces generated by the adversarial co-training. The blue indicates the old attack while green and red are the new one evaluated against the old HHD defender policy from Section II as well as the newly co-trained one. Generally we see that the red and green traces also jumpy between high and low sending rates as we have discussed in Section III. The difference to the old blue strategies is that the blue ones have larger high sending rates than the red and green ones. We see from Subfigure 8a that blue reaches peaks of 360 MTU/s while red and green anly go as high as 220 MTU/s. Subfigure 8b indicates the same qualitative observations with blue reaching 91 MTU/s while red and green staying lower at 80 MTU/s. Red and green surpass blue in terms of RpS, which suggests that the old blue attacks were overconfident in their ability to overuse large amounts of bandwidth due to the HHD's inability to defend against these unseen traffic patterns. Thanks to the adversarial co-training however, ES and AL are both prepared for such overconfident strategies and are therefore able to intercept the attacks more regularly and quickly.

Algorithm 4 Neural Monitor Agent

requires Cs ▷ Array of numbers of length n
requires IDs ▷ Array of flow IDs of length n
requires TSs ▷ Array of timestamps of length n
requires NN ▷ Transformer-based neural network
procedure ProcessPacket($size$, ID, $time_{now}$)
  $IDs' \leftarrow [ID, IDs]$
  $Cs' \leftarrow [size, Cs]$
  $TSs' \leftarrow [time_{now}, TSs] - time_{now}$
  $isID' \leftarrow IDs == ID$
  $isEmpty' \leftarrow IDs == 0$
  $(isHH,\ prio) \leftarrow NN(Cs', TSs', isID', isEmpty')$
    ▷ $isHH$ is a 1-d boolean value
    ▷ $prio$ is a $(n+1)$-d vector
  $i \leftarrow findInsertIndex(isID', isEmpty', prio)$
  if $i > 0$ **then**
    if $IDs[i-1] == ID$ **then**
      $Cs[i-1] \leftarrow Cs[i-1] + size$
    else
      $Cs[i-1] \leftarrow size$
      $IDs[i-1] \leftarrow ID$
      $TSs[i-1] \leftarrow time_{now}$
  return $isHH$

V. Generating Zero-Shot Attacks on Data-Plane Heavy-Hitter Detection

The preceding section has shown how applying adversarial multi-agent reinforcement learning can help find more robust configurations for HHD. The success of the method hinges in particular on the ability for the NTA to generate a rich population of traffic patterns that cover possible exploits in a thorough way. In this section we show that the NTA policies generated by PSRO not only make good "training-partners" for improving the robustness of HHD algorithms like ES, but that they also are effective adversarial traffic patterns that work on HHD algorithms not encountered during training. We refer to this as a "zero-shot attack" based on the concept of "zero-shot learning" in the context of machine learning, where a model is trained on data of type A but evaluated on data of type B that differs significantly from A [40].

A. Requirements for Successfully Learning Zero-Shot Attacks

The idea is to apply the PSRO adversarial training procedure to NTA and some HHD algorithm $\mathcal{G}$. The NTA will learn evasive traffic patterns $\mathcal{P}^{NTA}$ by interacting with $\mathcal{G}$ and when training is done, $\mathcal{P}^{NTA}$ is evaluated against a HHD algorithm $\mathcal{H} \neq \mathcal{G}$. To ensure $\mathcal{P}^{NTA}$ contains good strategies, it is essential for the training-partner $\mathcal{G}$ to provide resistance and flexibility in order to act as a useful proxy for unseen HHD

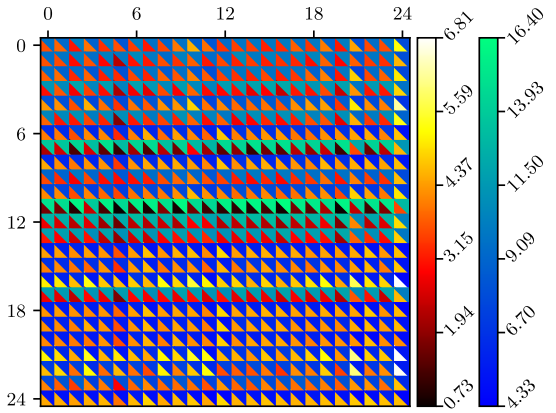


Fig. 9: Strategic game displaying the matchups between NTA and NMA after finishing adversarial training based on PSRO. Lower-left and upper-right triangles measure NTA's and NMA's RpS respectively.

$(\mathfrak{S})$	π^{AL}	π^{CB}	π^{CM}	π^{ES}	π^{HK}	π^{MG}	π^{SS}	π^{CH}	π^{HP}
π^{NTA}_{zsa}	59.9	73.1	85.2	86.8	31.8	50.9	63.4	74.2	55.2
π^{NTA}_{best}	67.0	78.1	104.5	116.2	32.9	61.5	72.8	80.6	71.1
$\pi^{NTA\times\mathcal{H}}$	73.8	71.2	95.2	159.4	44.0	54.9	67.0	66.6	61.7

TABLE V: Evaluation of the NTA's zero-shot attacks π^{NTA}_{zsa} on HHD configurations learned in Section II. The values in the table indicate the NTA's RpS score. π^{NTA}_{best} is the best attack the NTA has learned among the 25 from Figure 9.

$(\mathfrak{R})$	π^{AL}	π^{CB}	π^{CM}	π^{ES}	π^{HK}	π^{MG}	π^{SS}	π^{CH}	π^{HP}
π^{NTA}_{zsa}	81.3	92.1	90.1	113.9	104.8	92.1	99.4	91.1	87.9
π^{NTA}_{best}	100.0	117.3	212.3	175.8	191.5	144.1	107.6	116.9	166.2
$\pi^{NTA\times\mathcal{H}}$	75.9	114.1	170.5	308.2	72.9	71.1	75.0	104.5	72.9

TABLE VI: Evaluation of ZSA's on real traffic.

algorithms $\mathcal{H} \neq \mathcal{G}$. If $\mathcal{G}$ is too rigid and unable to capture a wide range of behavior (e.g. $\mathcal{G}$ can not perform similar heavy-hitter detection as an algorithm $\mathcal{H}'$ no matter how it is configured), it is likely that the NTA will exploit this characteristic. But by doing so, it will not learn strategies that work against $\mathcal{H}'$ and thus the zero-shot attack will fail.

B. The Neural Monitor Agent

Since all HHD algorithms we have discussed until now are fairly rigid, with at most 3-4 tuneable parameters, they are not suitable training-partners $\mathcal{H}$ for the NTA. For this reason we design a new HHD algorithm from the ground up with increased flexibility in mind. We call this new algorithm Neural Monitor Agent (NMA) and it combines concepts from HHD algorithms like AL and SS with deep neural networks in order to provide a lot of tuneable parameters that can aid it in covering a wide range of behaviors. The pseudocode for the NMA is given in Algorithm 4. It works by keeping an array of flow-IDs IDs, counters Cs, and timestamps TSs and letting a neural network NN process these arrays to produce two outputs: Firstly, a prediction on whether a flow is a heavy-hitter or not. And secondly, a priority-value for each flow ID in IDs. The priorities are then used to decide which is the lowest priority flow that will be evicted from the data-structure. This method therefore borrows a priority-heap-like approach from SS and MG, in addition to time-stamp functionality from AL. The neural network NN has to perform the priority management and decide at what point a flow should be reported as heavy-hitter. We would like to emphasize that the NMA is not designed to be a compute-efficient algorithm to be deployed in in favor of other HHD algorithms. Instead, it is designed to purely act as an adversarial training partner for the NTA in a simulated training environment. We refer to Appendix B for a discussion on the network model design.

1) Challenges in Training the Neural Monitor Agent: Since the NMA has the same interface as any other HHD algorithm mentioned in this work, the procedure we use to train its neural network is the same we use for all other HHD algorithms whose hyperparameters we tune. In particular, we refer to a configuration of the NMA's neural network as its policy π^{NMA}. We have encountered two challenges when training the NMA: scaling and reward engineering.

- The scaling-related problem is that simulating the NMA requires evaluating a neural network for every packet that arrives at the NMA. With 32 training environments and each environment simulating 6 seconds and 1000 flows sending 20 packets per second, we arrive at 3.8 million data-points for one learning step, which is too expensive. We therefore reduce the training scale for the NMA by lowering the maximum number of concurrent flows to 100 and only running 4 training environments in parallel.
- The other problem regards the rewards we give the NMA in order to learn to perform heavy-hitter detection. The initial idea was to give positive rewards only, namely when the rate-limiter drops packets from overusing flows as a consequence of the NMA's detection. We found however that the NMA would resort to marking almost every flow a heavy-hitter because it incurs no risk in doing so, therefore providing little incentive towards more accurate detection. For this reason we introduced a penalty to be paid for every packet that is marked as belonging to a heavy-hitter, and offset this initial penalty by providing twice the amount of rewards when the rate-limiter drops a packet. This is the reward scheme presented at the beginning in Section II-C.

C. Zero-Shot Attack Analysis

Figure 9 shows the evaluation of the strategic game between NTA and NMA after completing the adversarial training procedure. This section analyzes how well the attack patterns learned by the NTA work against HHD algorithms it has never encountered before. As is the case with the adversarial training between NTA and AL in Section IV-B, several of the learned policies $\mathcal{P}^{NTA}$ are redundant and can thus be pruned.

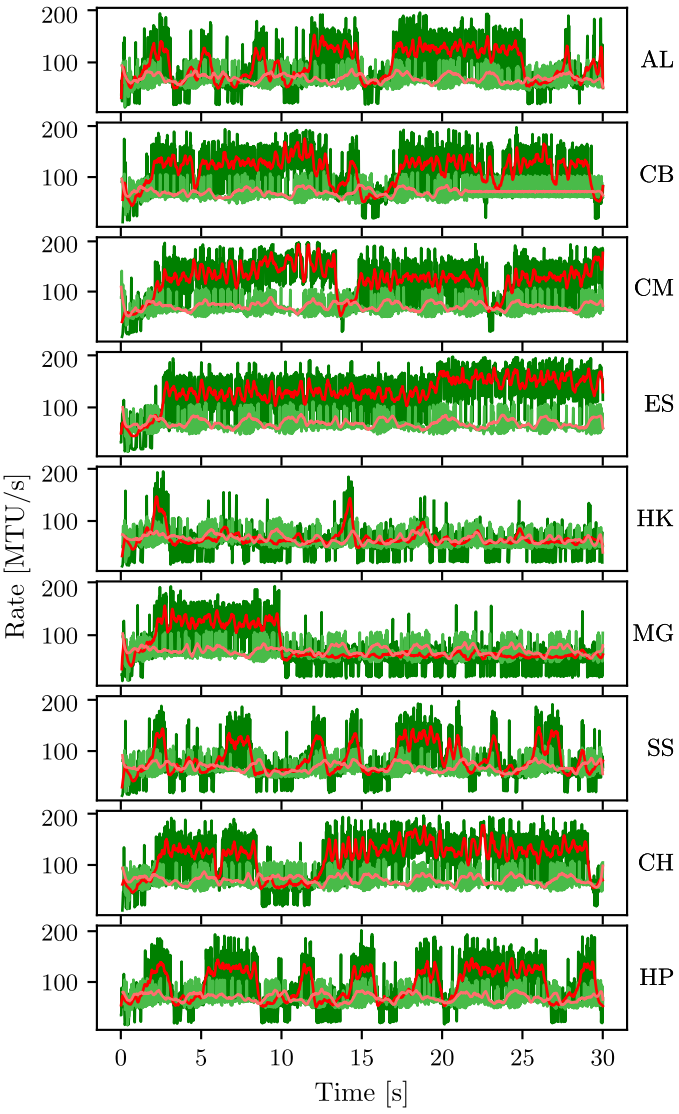


Fig. 10: Zero-shot attack π^{NTA}_{zsa} trace when confronted with different HHD-based rate limiters. Red indicates the smoothed curves. Each plot contains a mix of two attack traces; the darker one with weight 0.66, while the paler one with weight 0.34 according to the Nash equilibrium computed for π^{NTA}_{zsa}.

To evaluate the ability of the NTA to perform a zero-shot attack, it must decide on what mix of attack strategies from $\mathcal{P}^{NTA}$ to perform *before* we decide which HHD algorithm to match it against. Assuming a rational HHD configuration, the HHD maximizes its own rewards to the best of its ability. We use the NMA as a proxy and select the Nash-optimal strategy for the final zero-shot attack. Concretely, this results in a zero-shot attack consisting of two strategies from $\mathcal{P}^{NTA}$ mixed in a ratio of 0.34 to 0.66. We denote the final rationality-assuming zero-shot attack as π^{NTA}_{zsa}.

1) Evaluation: Based on its training against the NMA, π^{NTA}_{zsa} expects to be able to achieve a score of 45.7 RpS against a rational opponent. This value is calculated based on the expected rewards from the Nash equilibrium. We evaluate

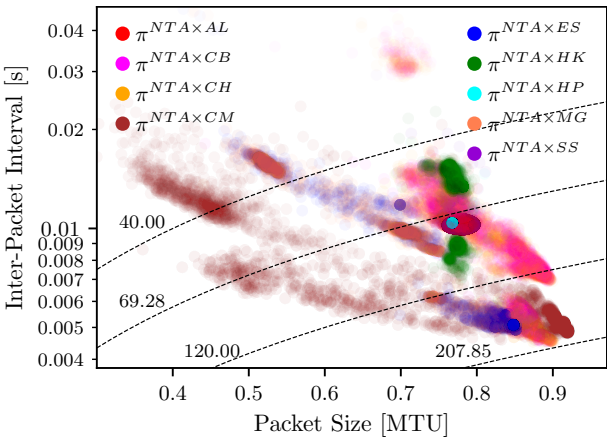


Fig. 11: Phase-space plot of π^{NTA}_{best} evaluated against various HHD methods. The dashed lines are isolines indicating equal sending-rate.

π^{NTA}_{zsa} against each of the 9 HHD algorithms from Section II and show the results in Table V. We see that π^{NTA}_{zsa} is able to generate traffic patterns that overuse the allowed rate of 40 MTU/s in 8 out of 9 cases. Furthermore, π^{NTA}_{zsa} surpasses its expected score of 45.7 RpS in 8 out of 9 cases. Figure 10 shows traces of π^{NTA}_{zsa} evaluated against each of the 9 HHD algorithms and we see that π^{NTA}_{zsa} behaves differently from scenario to scenario based on how well the rate-limiter is able to restrict its overuse.

We additionally evaluated the π^{NTA}_{zsa} in an environment with replayed real packet captures and the results are in Table VI. Here we see that the ZSAs exceed the allowed rate in 9 out of 9 cases and achieve significantly higher scores than on the synthetic data. The explanation for this is as with the experiments in Section III that the synthetic background traffic is less favorable for the NTA due to the reduced occurrence of extreme background traffic that may overshadow the NTA.

So the zero-shot attacks developed by the adversarial training between NTA and NMA have shown to work in almost all cases except when matched up against the HK algorithm. The approach therefore works in principle but some challenges limit the capability of zero-shot attacks. In particular, two aspects contribute to the low score against HK:

- HK is a decay-based HHD algorithm which has the ability to decrease the counters in its data-structure based on incoming packets. The designed NMA does not have such capability and thus struggles to replicate the behavior of an algorithm like HK, which is why the NTA's zero-shot attack is not prepared to confront the HK-based rate-limiting system.
- Due to computational cost, the adversarial training between NTA and NMA has been performed in a smaller training environment than the simulation environment in which we evaluate the NTA's attack strategies. Therefore there is a small mismatch between the environment in which the attacks were designed, and evaluated in.

Lastly, we consider allowing the attacker to pick the best policy from $\mathcal{P}^{NTA}$ for each HHD defender individually, we

denote these with π_{best}^{NTA} and results are in Tables V and VI. Surprisingly, the attack strategies learned by the NTA through training with the NMA exceed even the scores of the targeted attacks $\pi^{NTA \times \mathcal{H}}$ from Section III in 8 out of 9 cases. The reason is that the flexibility of the NTA's training partner NMA helps it explore more diverse attack strategies compared to the case in which the NTA trains against a single HHD algorithm and therefore has less incentive to explore. Figure 11 shows the phase-space plot of π_{best}^{NTA} and while we do observe recurring patterns such as prominent clustering as in Figure 6, one notable difference is that the co-training between NMA and NTA has for example taught π_{best}^{NTA} the value of sending less than $40\ MTU/s$, which did not happen in Section III.

D. Generalization Limits

Quantifying generalizability of RL models such as the NTA is difficult due to the many variables present in the environment. This work has for example shown that despite being trained exclusively on synthetic data, the learned strategies also work when deployed in an environment with real packet captures acting as background traffic. A contributing factor is that the real traffic makes for a more noisy environment from the point of view of the HHD, therefore reducing its ability to detect the NTA. In other words, the NTA has been trained in a more challenging environment and thus when deployed in an easier one, performs very well. However if for example the NTA were to be the only flow in the system, it would perform worse since the difficulty of evading detection by a HHD that is solely focused on the NTA, is beyond what the NTA is used to from its training.

VI. Conclusion

This work has shown that while Heavy-Hitter Detection (HHD) algorithms may provide provable guarantees, deploying these methods as part of a more sophisticated traffic processing system such as a DDoS defense system makes it difficult to carry over the theoretic guarantees due to the reliance on other components such as token-buckets to translate the HHD's predictions into concrete actions.

Section III has shown that it is possible to synthesize traffic patterns that exploit some of the innate limitations of typical HHD deployments, namely limited memory and processing time, in order to systematically overuse the allowed bandwidth. Furthermore, our experiments show that strong empirical performance of HHD algorithms on static packet traces does not translate into robustness against adversarial attacks by smart traffic synthesizers. On the contrary, our evaluation in a simulated setting suggests that high-scoring HHD algorithms tend to be more susceptible to exploits compared to lower-scoring methods due to a higher degree of specialization. In Section IV-B we develop and evaluate a machine-learning and game-theory-based training framework which allows hardening a HHD algorithm with an initial exploitability of 299% down to 139%, a reduction by a factor of 2.2, as well as increasing its rate-limiting performance by a factor of 1.7. The framework is generic and can be repurposed for different scenarios and tasks, and is available as open-source [22]. Lastly, Section V demonstrates that the training procedure that can make HHDs more robust, can also be used to produce more sophisticated adversarial attack strategies that are able to evade 8 out of 9 HHD algorithms without ever having been exposed to them. We refer to this as a zero-shot attack due to the similarity with "zero-shot learning" from the domain of artificial intelligence.

A. Takeaways

We summarize the insights of this work into three points.

1) System Evaluation: Evaluating HHD algorithms in experiments with no adaptive adversary generally only reveals whether the proposed method works on generic traffic, but reveals little about what an adaptive adversary may achieve. Where applicable, techniques such as those presented in this paper can be used to complement existing frameworks such as fuzzing and formal verification to provide an additional tool for analyzing network system security.

2) Adversarial Co-Learning: Many aspects of network security have two characteristics; Firstly, multiple entities with selfish or antagonistic goals, and secondly a very large space of possible entity behaviors and events. These are prime application-areas for multi-agent RL as it can be used to explore the huge space of behaviors, and in addition exploit the antagonistic relationships in the system to generate increasingly more sophisticated and valuable training data.

3) Code: The developed code-base aims to reduce the entry-barrier for research on the intersection between multi-agent RL and networked systems. It is designed to be more natural to use for network systems researchers than for example libraries like TorchRL [41], but more light-weight and with more ML utility and features than for example ns3-ai [42].

B. Limitations and Future Work

The co-training approach between attacker and defender that we propose in this paper is computationally expensive. The compute requirements are both CPU- as well as GPU-heavy: CPU for simulation, and GPU for deep learning. Scaling up the training scope thus requires an increase in both resources. Therefore, while the smart attack patterns learned by the NTA in this paper perform as expected in our simulations, orders of magnitude more resources are required to generate NTAs that can generalize to arbitrary real-world systems.

Further opportunities for research lie in exploring how different reward policies shape the dynamics between rate-limiting systems and traffic-emitting adversaries, such as, for example, if the adversaries' goal switches from maximizing its own throughput to maximizing other network-participants' latency and jitter. Additionally, an interesting way of scaling up the complexity of the problem setting is to allow for communication between multiple adversaries and defenders and to make them develop cooperative strategies through machine learning.

VII. Ethical Considerations

As part of this paper we disclose the code used to generate the results, as well as the results themselves which consist of both trained neural networks for all RL agents discussed in this paper, in addition to all the execution traces from which we extract the data presented in this paper. We elaborate on the responsible disclosure of these three aspects.

A. Measurements

The execution traces contain no sensitive information and pose no security or privacy risk of any kind.

B. Trained Models

We disclose all trained policies and neural networks presented in this paper for the sake of reproducibility. They are designed to perform their function in our simulated environment but require additional engineering to be deployed as part of a live system in general due to the sim-to-real gap which may be very large depending on how the models are intended to be mis-used. Therefore, without significant further engineering these models cannot be mis-used to perform effective attacks on arbitrary real-world systems.

C. Program Code

The program code enables training smart attacker and defender agents in a multi-agent reinforcement learning framework. The code can be altered to train more advanced models with greater capabilities than those described in Section VII-B. This however requires advanced understanding of deep reinforcement learning and network security, as well as additional compute resources and engineering work.

References

[1] M. Nawrocki, R. Hiesgen, T. C. Schmidt, and M. Wählisch, "Quicsand: quantifying quic reconnaissance scans and dos flooding events," in *Proceedings of the 21st ACM Internet Measurement Conference*, ser. IMC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 283–291. [Online]. Available: https://doi.org/10.1145/3487552.3487840

[2] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, p. 64–74, Jul. 2008. [Online]. Available: https://doi.org/10.1145/1400097.1400105

[3] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 127–141.

[4] B. Hao and C. Michini, "Inefficiency of pure nash equilibria in network congestion games: the impact of symmetry and network structure," *ACM Trans. Econ. Comput.*, vol. 12, no. 3, Sep. 2024. [Online]. Available: https://doi.org/10.1145/3665590

[5] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270—-313, Aug. 2003. [Online]. Available: https://doi.org/10.1145/859716.859719

[6] I. Mahmud, G. Papadimitriou, C. Wang, M. Kiran, A. Mandal, and E. Deelman, "Elephants sharing the highway: Studying tcp fairness in large transfers over high throughput links," in *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 806–818. [Online]. Available: https://doi.org/10.1145/3624062.3624594

[7] S. Islam, K. Hiorth, C. Griwodz, and M. Welzl, "Is it really necessary to go beyond a fairness metric for next-generation congestion control?" in *Proceedings of the 2022 Applied Networking Research Workshop*, ser. ANRW '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3547115.3547192

[8] K. Khandeparkar, K. Ramamritham, and R. Gupta, "Qos-driven data processing algorithms for smart electric grids," *ACM Trans. Cyber-Phys. Syst.*, vol. 1, no. 3, Mar. 2017. [Online]. Available: https://doi.org/10.1145/3047410

[9] R. Wójcik and A. Jajszczyk, "Flow oriented approaches to qos assurance," *ACM Comput. Surv.*, vol. 44, no. 1, Jan. 2012. [Online]. Available: https://doi.org/10.1145/2071389.2071394

[10] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 164–176. [Online]. Available: https://doi.org/10.1145/3050220.3063772

[11] R. Ben Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Designing heavy-hitter detection algorithms for programmable switches," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, p. 1172–1185, Jun. 2020. [Online]. Available: https://doi.org/10.1109/TNET.2020.2982739

[12] M. Chiesa and F. L. Verdi, "Network monitoring on multi-pipe switches," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 7, no. 1, Mar. 2023. [Online]. Available: https://doi.org/10.1145/3579321

[13] M. Moshref, M. Yu, and R. Govindan, "Resource/accuracy tradeoffs in software-defined measurement," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 73–78. [Online]. Available: https://doi.org/10.1145/2491185.2491196

[14] J. Gong, T. Yang, H. Zhang, H. Li, S. Uhlig, S. Chen, L. Uden, and X. Li, "HeavyKeeper: An accurate algorithm for finding top-k elephant flows," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, Jul. 2018, pp. 909–921. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/gong

[15] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: adaptive and fast network-wide measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 561–575. [Online]. Available: https://doi.org/10.1145/3230543.3230544

[16] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," *the 27th Network and Distributed System Security Symposium (NDSS 2020)*. [Online]. Available: https://par.nsf.gov/biblio/10176415

[17] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A High-Performance Switch-Native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3829–3846. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/liu-zaoxing

[18] D. Ding, M. Savi, F. Pederzolli, M. Campanella, and D. Siracusa, "In-network volumetric ddos victim identification using programmable commodity switches," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1191–1202, 2021.

[19] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196677403001913

[20] S. Scherrer, J. Vliegen, A. Sateesan, H.-C. Hsiao, N. Mentens, and A. Perrig, "Albus: a probabilistic monitoring algorithm to counter burst-flood attacks," in *2023 42nd International Symposium on Reliable Distributed Systems (SRDS)*, 2023, pp. 162–172.

[21] F. D. Dalt, S. Scherrer, and A. Perrig, "Bayesian sketches for volume estimation in data streams," *Proc. VLDB Endow.*, vol. 16, no. 4, p. 657–669, Dec. 2022. [Online]. Available: https://doi.org/10.14778/3574245.3574252

[22] F. D. Dalt, "Paper code," 2025. [Online]. Available: https://github.com/FrancescoDaDalt/HHNFM.git

[23] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 101–114. [Online]. Available: https://doi.org/10.1145/2934872.2934906

[24] B.-E. Zolbayar, R. Sheatsley, P. McDaniel, M. J. Weisman, S. Zhu, S. Zhu, and S. Krishnamurthy, "Generating practical adversarial network traffic flows using nidsgan," 2022. [Online]. Available: https://arxiv.org/abs/2203.06694

[25] S. Hore, J. Ghadermazi, D. Paudel, A. Shah, T. Das, and N. Bastian, "Deep packgen: A deep reinforcement learning framework for adversarial network packet generation," *ACM Trans. Priv. Secur.*, vol. 28, no. 2, Feb. 2025. [Online]. Available: https://doi.org/10.1145/3712307

[26] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[27] A. Cheng, "Pac-gan: Packet generation of network traffic using generative adversarial networks," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2019, pp. 0728–0734.

[28] K. Cho, K. Mitsuya, and A. Kato, "Traffic data repository at the wide project," in *Proceedings of the USENIX 2000 FREENIX Track*, San Diego, CA, Jun. 2000.

[29] "Mawi samplepoint G traffic data (2020-03-18 14:00)," https://mawi.wide.ad.jp/mawi/samplepoint-G/2020/202003181400.html, accessed: 2025-11-25.

[30] J. Misra and D. Gries, "Finding repeated elements," *Science of Computer Programming*, vol. 2, no. 2, pp. 143–152, 1982. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167642382900120

[31] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-*k* elements in data streams," in *Proceedings of the 10th International Conference on Database Theory (ICDT 2005)*, ser. Lecture Notes in Computer Science, vol. 3363. Springer, 2005, pp. 398–412.

[32] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theoretical Computer Science*, vol. 312, no. 1, pp. 3–15, 2004, automata, Languages and Programming. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397503004006

[33] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17. New York, NY, USA: Associatio n for Computing Machinery, 2017, p. 164–176. [Online]. Available: https://doi.org/10.1145/3050220.3063772

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[36] G. N. Lance and W. T. Williams, "A general theory of classificatory sorting strategies: 1. hierarchical systems," *The Computer Journal*, vol. 9, no. 4, pp. 373–380, 02 1967. [Online]. Available: https://doi.org/10.1093/comjnl/9.4.373

[37] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Perolat, D. Silver, and T. Graepel, "A unified game-theoretic approach to multiagent reinforcement learning," 2017. [Online]. Available: https://arxiv.org/abs/1711.00832

[38] J. F. Nash, "Equilibrium points in ¡i¿n¡/i¿-person games," *Proceedings of the National Academy of Sciences*, vol. 36, no. 1, pp. 48–49, 1950. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.36.1.48

[39] T. Sandholm, A. Gilpin, and V. Conitzer, "Mixed-integer programming methods for finding nash equilibria," in *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'05. AAAI Press, 2005, p. 495–501.

[40] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, " Zero-Shot Learning—A Comprehensive Evaluation of the Good, the Bad and the Ugly ," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 41, no. 09, pp. 2251–2265, Sep. 2019. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TPAMI.2018.2857768

[41] A. Bou, M. Bettini, S. Dittert, V. Kumar, S. Sodhani, X. Yang, G. D. Fabritiis, and V. Moens, "Torchrl: A data-driven decision-making library for pytorch," 2023.

[42] H. Yin, P. Liu, K. Liu, L. Cao, L. Zhang, Y. Gao, and X. Hei, "ns3-ai: Fostering artificial intelligence algorithms for networking research," in *Proceedings of the 2020 Workshop on Ns-3*, ser. WNS3 '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 57–64. [Online]. Available: https://doi.org/10.1145/3389400.3389404

[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[44] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014. [Online]. Available: https://arxiv.org/abs/1406.1078

[45] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019. [Online]. Available: https://arxiv.org/abs/1711.05101

[46] ——, "Sgdr: Stochastic gradient descent with warm restarts," 2017. [Online]. Available: https://arxiv.org/abs/1608.03983

[47] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133 653–133 667, 2019.

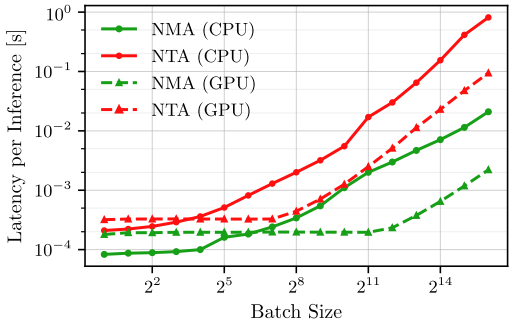


Fig. 12: The curves measure latency of batch inference as a function of batch size on both CPU as well as GPU.

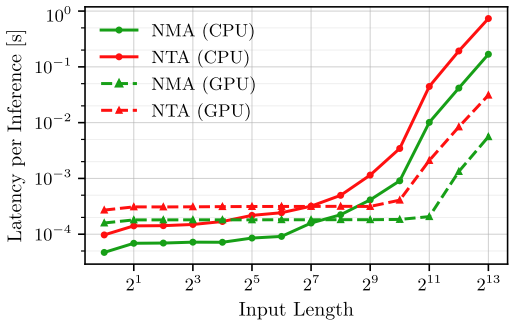


Fig. 13: The curves measure single-data inference latency as a function of input length on both CPU as well as GPU.

APPENDIX

A. NTA and NMA Inference Latency

We measure inference latency of NMA and NTA in our system with respect to two variables: input length and batch size. Input length has different meanings for the NMA and NTA respectively: The input size to the NTA is the length of the time-series of past actions which it takes into account to decide on what the next action should be. For the NMA on the other hand, the input length is proportional to the size of the data-structure which it must manage. The batch size refers in both cases to the standard definition present in the literature in which data which in theory is to be processed independently, gets batched together to allow SIMD/SIMT parallelism.

The measurements with respect to batch size are shown in Figure 12. The values of NMA (CPU) and NTA (CPU) at batch size 1 are important as they indicate the inference time during evaluation when the model is deployed. The data of NMA (GPU) and NTA (GPU) is more relevant for the training procedure where collected data can be batched and moved to the GPU in which cases we use batch-sizes of up to 32k and 40k for NTA and NMA respectively.

The measurements with respect to input length are shown in Figure 13. The primary observation here is that in contrast to Figure 12, the asymptotic inference cost grows quadratically with the input size as we can see that when going from an input length of 2^{11} to 2^{13} we incur a $\approx 10\times$ increase in latency, despite only a $2\times$ increase in size. The initial plateaus in Figures 12 and 13 occur due to SIMD/SIMT parallelism where more data can be processed at little to no additional cost due to the SIMD/SIMT pipelines not being under maximum load.

B. NMA Deep Neural Network Design

Designing the NMA required balancing flexibility, computational efficiency, and fixed memory constraints. We evaluated three architectural approaches before selecting a hybrid transformer model.

1) Recursive Neural Network (RNN): RNNs (e.g., LSTM [43], GRU [44]) were the initial choice due to their flexible memory. However, they were rejected for two reasons:

- Flow ID Recognition: RNNs struggle to embed and compare flow IDs while simultaneously tracking bandwidth usage.
- Training Difficulties: The RL environment is noisy. Optimizing RNNs to retain long-term dependencies and memorize IDs under these conditions proved infeasible. Modified architectures attempting to address this did not scale.

2) Time-Series Transformer: We considered using a transformer taking the last k packets as input. However, as the number of flows increases, packets space out, requiring k to grow linearly. Since transformer complexity scales quadratically with k, this approach was deemed unscalable.

3) Hybrid Models: We settled on a hybrid architecture: a manually defined counter-based data structure managed by a neural network. More precisely, a transformer. Unlike Fully Connected (FC) networks or convolutional networks (which would find no local structure in the data to exploit), transformers provide the necessary equivariance, ensuring that swapping memory entries swaps outputs equivalently.

The final design is a single-layer, single-head transformer-encoder combined with FC layers. The transformer ensures equivariance, while the FC layers handle feature transformation, memory update rules, and heavy-hitter predictions.

C. Qualitative Effects of Reward Shaping on NTA

In this paper we train all NTA policies using a reward scheme which rewards the agent with s for every packet of size s that passes the rate-limiter, and $-s$ if it gets dropped or lost instead. The choice was primarily based on the idea that packet loss should cause some cost or penalty at the sender, but also based on observations on the qualitative behavior of the NTA when we do not attribute negative rewards whenever packet loss occurs. We observed namely that generally NTA policies with packet-loss penalty exhibit more sophisticated behavior than their non-penalized counterpart. Figure 14 shows the phase-space plots of the NTA strategies where green is with loss-penalty and red is without. We see that the strategy without penalty has a consistently higher average sending rate than the one with penalty, which makes sense since it does not have to fear being explicitly punished for sending a lot of traffic. However what we also observe is that qualitatively the red strategies without loss-penalty have generally a simpler structure with fewer clusters, whereas we observe much more variability and the generation of multiple clusters for the green strategies with loss-penalty. NTA policies having more distinct clusters in the phase-space plot suggests a more sophisticated

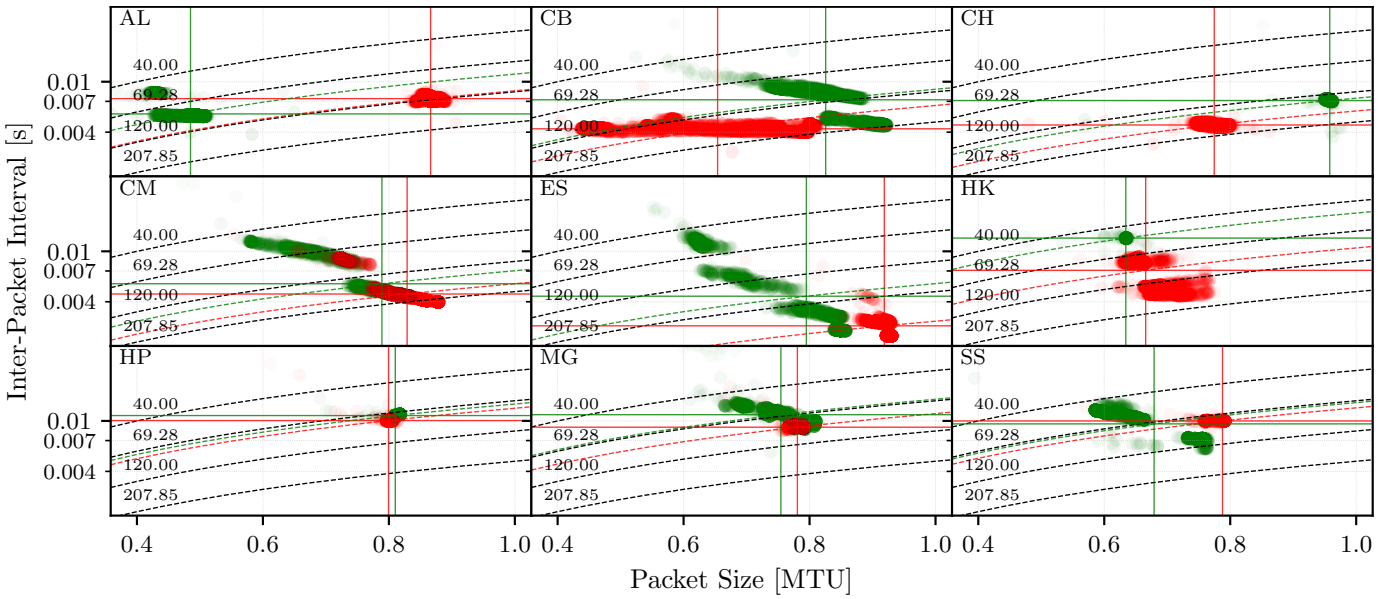


Fig. 14: Phase-space plot of π^{NTA} evaluated against various HHD methods. The dashed lines are isolines indicating equal sending-rate. In green we have the policies from Section III whereas in red we have NTA policies that have been trained without packet-loss penalty. The red and green lines pinpoint the average packet size, inter packet interval, and sending rate of each policy.

Metric	HHD	NTA	NMA
Total Time [s]	5288.629	19680.220	14481.064
Optimization	16.05%	41.92%	69.56%
Eval	43.73%	52.77%	17.29%
Test	40.22%	5.31%	13.15%
Max Score Time [s]	3508.226	10620.357	6665.188
50% of max	94.129	2222.602	1060.886
90% of max	278.461	3780.261	1708.581
99% of max	2422.100	10620.357	2359.489
Epochs (Major)	111	78	68
Minor epochs	1109	769	669
PPO steps	110900	76900	66900

TABLE VII: Training metrics for HHD, NTA, and NMA.

strategy and a more advanced understanding of the system, its dynamics, and how to act in it.

D. Training Resource Consumption

Minimizing the training time is not the main technical focus of this paper. We optimized the whole pipeline at a system-level considerably compared to existing frameworks but we did not go into too many details about customizing ML methods such as learning-rate adaptation to our use-case. In particular, in our experiments we typically prefer to let the training take additional steps just to be sure that it cannot improve anymore before we terminate it. In this section we provide data on the resource consumption of the training process.

We divide the training time into three parts: test-time which measures how much walltime is spent evaluating the agent in the 60 second simulation, eval-time which measures how much walltime is spent on the shorter 6 second simulations which are used to collect training data, and finally optimization-time which measures how much walltime was spent actually adjusting the agent policy. Test- and eval-time measure the CPU load while optimization-time consists of GPU load.

Generally the type of workload depends on which agent is being trained and we observed three categories: HHD agent training (such as AL), NTA training, and NMA training. Measurmeents are shown in Table VII.

E. Reducing Training Time

Training involves CPU-bound data generation and GPU-bound data consumption. The goal is to increase throughput for both types of workloads.

1) Increasing Data Generation Throughput: Throughput scales inversely with environment complexity. We increased it via two methods:

- System-Level: Using multiple parallel simulations, which provides a linear increase in throughput with available CPU resources.
- Model-Level: Reducing wasted work by minimizing non-essential system complexity. For example, packets are handled as simple objects holding minimal information (source, size) instead of fully serialized structures, as the training algorithm does not use the serialized data.

2) Increasing Data Consumption Throughput: The idea is to have the model absorb information from the data at the highest rate possible.

- System-Level: The primary objective is to reach 100% GPU utilization for inference, backpropagation, and weight updates, which was achieved in our framework.

- Model-Level: We selected robust and generic optimization algorithms: AdamW [45] for gradient descent and PPO [34] for reinforcement learning, with cosine-annealing [46] used to adjust the learning rate. For multi-agent learning, we used PSRO [37] because it is robust, has no hyperparameters, and enables combining smaller neural networks. It avoids the need for increased neural network capacity and the resulting slowdown that alternatives like Q-Learning [47] would require.

F. Packet Capture Data

To incorporate real data within our experiments we start off from a MAWI [29] packet capture. We extract 10000 (TCP or UDP) flows with at least 2 packets and compute the MTU over the packets to make it compatible with our framework. The longest extracted trace has 23150 packets, the average traffic rate is $90.5\ MTU/s$, while the average rate per flow is $27.1\ MTU/s$. Out of the extracted flows, 3.43% exceed the rate limit of $40\ MTU/s$. In order to avoid constant congestion of our simulated network link we downscale the packet sizes by the factor 0.22 in order to match an average rate of $20\ MTU/s$ which the link can handle. After scaling, only 0.73% of extracted flows exceed $40\ MTU/s$.

G. Hardware and Software Configuration

This section details the hardware and software used to perform all work in this paper.

1) Hardware: We use a single machine to do all our work. The machine has the following specifications:

- Motherboard: ROG STRIX X670E-F
- CPU: AMD Ryzen 9 7950X (16 physical / 32 virtual cores)
- GPU: NVIDIA AD104 [GeForce RTX 4070 SUPER] (12 GB GDDR6X)
- Memory: 2×16 GB DDR5 Kingston

In summary, we used high-end consumer-grade components.

2) Software: We use the following software:

- OS: Xubuntu 24.04.2 LTS
- C++: gcc 13.3.0
- libtorch: 2.7.0 for cuda 11.8
- MOSEK: 10.2
- openMP: 4.5