Per-Host DDoS Mitigation by Direct-Control Reinforcement Learning

Kyle A. Simpson, Simon Rogers, Dimitrios P. Pezaros, *Senior Member, IEEE*,
School of Computing Science, University of Glasgow, Glasgow, G12 8RZ, UK
k.simpson.1@research.gla.ac.uk, Simon.Rogers@glasgow.ac.uk, Dimitrios.Pezaros@glasgow.ac.uk

***Abstract*—DDoS attacks plague the availability of online services today, yet like many cybersecurity problems are evolving and non-stationary. Normal and attack patterns shift as new protocols and applications are introduced, further compounded by burstiness and seasonal variation. Accordingly, it is difficult to apply machine learning-based techniques and defences in practice.**

***Reinforcement learning* (RL) may overcome this detection problem for DDoS attacks by managing and monitoring *consequences*; an agent's role is to learn to optimise performance criteria (which are always available) in an online manner. We advance the state-of-the-art in RL-based DDoS mitigation by introducing two agent classes designed to act on a per-flow basis, in a protocol-agnostic manner for any network topology. This is supported by an in-depth investigation of feature suitability and empirical evaluation.**

Our results show the existence of flow features with high predictive power for different traffic classes, when used as a basis for feedback-loop-like control. We show that the new RL agent models can offer a significant increase in goodput of legitimate TCP traffic for many choices of host density.

***Index Terms*—Security services, distributed denial-of-service, software-defined networking, machine learning, reinforcement learning.**

I. Introduction

Network anomaly detection and intrusion detection/prevention are continually evolving problems, compounded by the partial, non-*independent and identically distributed* (IID) view of data at each point in the network. Attacks and anomalous behaviours evolve, becoming more sophisticated or employing new vectors to harm a network or system's confidentiality, integrity, and availability without being detected [1]. These attacks and anomalies have measurable consequences and symptoms which allow a skilled analyst to infer new signatures for detection by misuse-based classifiers, but unseen attacks may only be defended against after-the-fact. This issue is inherent to *misuse-* or *signature-based* intrusion detectors, and it has been long-hoped that *anomaly-based* detectors would surpass this by making effective use of statistical measures [1].

While *machine learning* (ML) approaches seem like a sensible fit for this problem, in 2010 Sommer and Paxson identified the 'failure to launch' of ML-based anomaly detection systems—a distinct lack of real-world system deployments [2]. To quite a large extent, this remains the case today. They posit that their use is made difficult due to significant operational differences from standard ML tasks, including: the high cost of errors and extraordinarily low tolerance for false positives inherent to network intrusion detection [3]; a general lack of recent, openly available (and high-quality) training data; and diversity of network traffic across varying timescales combined with significant burstiness [4]. Above the aggregate level, the constant deployment of new services and protocols means that traffic is *non-stationary* and displays an evolving notion of normality. Learning is made harder still by the challenges encountered with unlabelled (often partial) data. All of these factors greatly inflate the difficulty of the detection problem.

For certain classes of problem e.g., volumetric *distributed denial of service* (DDoS) attacks, *reinforcement learning* (RL) offers another perspective. RL agents operate by following a *policy* to interact with or control a system, while at the same time using observed performance metrics and deliberate exploration to dynamically improve this policy. In this way the role of a RL agent differs from that of a standard classifier, adaptively reacting to threats by assuming the role of a feedback loop for network optimisation, typically to safeguard service guarantees. In a sense, this allows us to "overcome" some of the difficulties of the detection problem by monitoring *performance characteristics and consequences* in real-time; by looking for (and controlling) the effect rather than the cause. Long-term, we expect that the value of RL-based defence systems will be to augment what existing misuse-based solutions can provide, by automatically alerting, recording and controlling what are believed to be illegal system states. The goal of this work is much less general; we aim to prevent volume-based DDoS attacks with the aid of RL-based techniques (an important goal in its own right), while bringing to light the flexibility and applicability of these techniques in the security domain.

To date, there have been few applications of this class of algorithms towards intrusion detection and prevention which make use of their full potential for online control, rather than using them as the basis for a classifier. We aim to take steps to redress this and establish their proper capabilities, beyond simple "blind application". What approaches do exist are aimed towards the task of adaptive online DDoS mitigation, and rely upon learning to control probabilistic packet drop.

We find that the existing work for this task [5] fails to account for congestion-aware traffic (i.e., TCP) and environments with high host density per egress point, achieving poor results due to an overly coarse view of the network. To remedy this, we make throttling decisions on a per-source basis and present the engineering decisions this mandates: updating RL agents from multiple traces per timestep, timed random sequential action computation and a supporting *software-defined network* (SDN) architecture. In tandem with the development and evaluation of an effective state space and model, we provide the design of a second model inspired by past work on algorithmic DDoS prevention, as an example of the integration of domain-specific knowledge. Our introduction of per-source decisions improves substantially upon the state-of-the-art when acting upon most internet traffic (i.e., congestion-aware protocols), and we show that our second model achieves excellent performance for high host density in this case. Crucially, both models remain protocol- and content-agnostic to offer future-proofing against the rollout of future protocols like QUIC [6].

A. Contributions

This paper contributes two source-level granularity approaches to RL-driven DDoS prevention (*Instant* and *Guarded* action models), improving upon past aggregate-based models (section III). These are designed to make effective decisions irrespective of protocol, and act on individual flows at the edge of any network topology. We offer an in-depth investigation into suitable features for automatic DDoS mitigation, with qualitative and quantitative justification (section IV). These features have been suggested by past studies, and independently tested in their own contexts. Our study is the first attempt to quantify the individual efficacy of each in an RL setting.

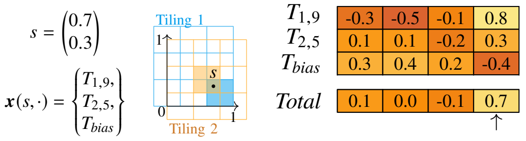
We implement reactive simulations of HTTP and VoIP web-server traffic, designed to test system characteristics that packet trace playback fails to capture (section V). To our knowledge, this is the first attempt to study or replicate Opus-based VoIP traffic, which has become commonplace since the codec's release in 2012. These new traffic models inform an empirical evaluation of our new models against the state-of-the-art in RL-based DDoS mitigation using (section VII), alongside a discussion of security concerns and real-world deployment (section VIII). We additionally compare our work against SPIFFY [7], reuniting two divergent strands of research and grounding the study of RL-based DDoS defences.

II. Background and Threat Model

A. Distributed Denial of Service

Distributed denial of service (DDoS) attacks are concentrated efforts by many hosts to reduce the availability of a service, typically to inflict financial harm or as an act of vandalism. Attackers achieve this by either exploiting peculiarities of operating system or application behaviour in *semantic attacks* (e.g., *SYN flooding attacks*), or overwhelming their target through sheer volume of requests or inbound packets (*volume-based attacks*) [8]. Hosts often participate unwillingly, typically having been recruited into a *botnet* by malware infection to be orchestrated from elsewhere [9].

Although there are variations of each class of attack, flooding attacks are the most relevant to our work. *Amplification attacks* exploit services who eagerly send large replies in response to small requests, where UDP-based services like DNS and NTP are most exploitable [10], [11]. Malicious hosts send many small requests, spoofed to appear as though they originated from the victim, causing many large replies to be sent to the intended target—significantly increasing a botnet's throughput while masking the identity of each participant. *Transit-link/link-flooding attacks* have been the subject of recent attention, wherein malicious traffic is forwarded across core links needed to reach a target (but not to the target itself) [12], [13].


$s = \begin{pmatrix}0.7\\0.3\end{pmatrix}$
$x(s,\cdot) = \begin{Bmatrix}T_{1,9},\\T_{2,5},\\T_{bias}\end{Bmatrix}$
Tiling 1
Tiling 2
1
0
1
s
$T_{1,9}$ -0.3 -0.5 -0.1 0.8
$T_{2,5}$ 0.1 0.1 -0.2 0.3
T_{bias} 0.3 0.4 0.2 -0.4
Total 0.1 0.0 -0.1 0.7

(a) Tile Coding

(b) Value Estimation and Action Selection

Figure 1. An example of tile coding for 2-dimensional state and 4 actions, using 2 tilings, 3 tiles per dimension, and a bias tile. All components of s are clamped to $[0, 1]$. For simplicity, we denote $\boldsymbol{x}(s, \cdot)$ as a list of indices and represent the values of all actions at each tile with a vector. (a) The state s activates the bias tile and exactly one tile in each tiling. (b) The action values of active tiles are summed to produce the current value estimate for each action available in s—for this state, local knowledge ensures that action 4 is chosen by the greedy policy despite typically being a poor choice elsewhere.

B. Reinforcement Learning

Reinforcement learning (RL) is a variant of machine learning principally concerned with training an agent to choose an optimal sequence of actions in pursuit of a given task [14]. We assume the agent has a certain amount of knowledge whenever a decision must be made: at any point in time t, it knows which *state* it is in ($S_t \in \mathcal{S}$), the set of *actions* which are available to it ($\mathrm{A}(S_t) \subseteq \mathcal{A}$) and a numeric *reward* obtained from the last action chosen ($R_t \in \mathbb{R}, A_{t-1} \in \mathrm{A}(S_{t-1})$). This model of system interaction is a *Markov Decision Process* (MDP). RL methods combine this information with a current *policy* π to determine which action should be taken: such a choice need not be optimal if an agent needs to further explore some region of the state space. The policy is refined by updating value estimates for state-action pairs or via policy gradient methods, meaning that RL-based approaches learn adaptively and online if reward functions are available in the environment they are deployed in. In practice, this means that agents are able to automatically adapt to evolving problems without operator intervention or a new, custom-built training corpus.

From any point in a sequence of decisions, we may describe the sum of rewards yet to come as the *discounted return*, $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$, choosing the discount factor $\gamma \in [0, 1)$ to determine how crucial future rewards are vis-à-vis the current state. Formally, an agent's goal is to choose actions which maximise the *expected discounted return* $\mathbb{E}_\pi[G_t]$.

There is immense variation in *how* policies and/or values may be learned, reliant upon the learning environment, problem and required convergence guarantees. In particular, we focus on methods which choose actions according to their value estimates from the current state: let $\mathrm{q}(s, a) \in \mathbb{R}$ be the estimate of action a's value if it were to be taken in state s. Exact (tabular) representations require that we store a value estimate for each action in every state—if state is real-valued or high-dimensional, then computation and storage quickly become infeasible. To cope with a continuous state and/or action space, one valuable technique is to employ linear function approximation backed by *tile coding* [14, pp. 217–221].

Tile coding is a form of feature representation which converts a state-action pair into a sparse boolean feature vector $\mathbf{x}(s, a)$ by subdividing a d-dimensional subset of the space into a number of overlapping grids with an optional bias component. Each tile corresponds to an entry of $\mathbf{x}(s, a)$ which is set to 1 if the state-action pair lies within it. Figure 1a demonstrates the

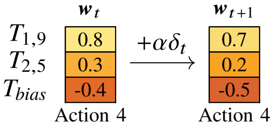


Figure 2. The update step for fig. 1, given an observed TD error $\delta_t = -0.2$ (indicating a lower observed reward than the expected long-term value of 0.7) and $\alpha = 0.5$. Action 4's value is thus reduced in the tiles associated with state s, but remains the most likely choice; the negative δ_t may have arisen from noise in the reward signal. For illustrative purposes, we choose an unrealistically high α (typically, $\alpha \leq 0.05$ is a more reasonable choice).

process for a 2-dimensional state space, and that the numbers of tilings and tiles per dimension control feature resolution and generalisation. Moreover, to capture combinatorial effects or create multi-scale representation we may combine codings by concatenating individual feature vectors. We may then approximate an action's value with respect to a policy parameter vector $\boldsymbol{w}$, defining some $\hat{\mathrm{q}}(s, a, \boldsymbol{w}) \approx \mathrm{q}(s, a)$:

$$\hat{\mathrm{q}}(s, a, \boldsymbol{w}) = \boldsymbol{w}^\top \mathbf{x}(s, a) \tag{1}$$

As each component of $\boldsymbol{w}$ is the value estimate of the corresponding tile, learning an effective policy is equivalent to learning $\boldsymbol{w}$. Given a learning rate $\alpha \in \mathbb{R}$ and initialising $\boldsymbol{w}_0 = \mathbf{0}$, we may continually update $\boldsymbol{w}_t$ using the *1-step semi-gradient Sarsa* algorithm [14, pp. 243–244]:

$$\delta_t = R_{t+1} + \gamma\, \hat{\mathrm{q}}(S_{t+1}, A_{t+1}, \boldsymbol{w}_t) - \hat{\mathrm{q}}(S_t, A_t, \boldsymbol{w}_t), \tag{2a}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\delta_t \nabla\hat{\mathrm{q}}(S_t, A_t, \boldsymbol{w}_t), \tag{2b}$$

where δ_t is known as the *temporal-difference* (TD) error, and the vector gradient ∇ is taken with respect to $\boldsymbol{w}$.

Computing the approximate value of every available action forms the basis of a policy. Actions with maximal value can be chosen each time (the *greedy* policy), we might modify this by taking random actions with probability ϵ to encourage early exploration (the *ϵ-greedy policy*), or we might use some other mechanism. Figure 1b extends the prior working example to show how the value of each action is computed (and which action would be chosen by a greedy policy), combining a global estimate (T_{bias}) with knowledge particular to each state.

This combination of algorithm and coding strategy is well-optimised, if actions are discrete; this allows a particularly efficient (vectorised) implementation of the policy and update rules by storing a vector of action values for each tile. Action values for any state are then obtained by summing the weight vectors from all activated tiles—taking $|\mathcal{A}|(n_{tilings} - 1)$ floating point additions per decision. Observing that $\nabla\hat{\mathrm{q}}(s, a, \boldsymbol{w}) = \mathbf{x}(s, a)$, further optimisations arise by considering that a tile-coded feature vector is a binary vector of constant Hamming weight (and so is amenable to representation as an array of indices, s_{list}). This means that we need only perform $n_{tilings} + 2$ additions and 2 multiplications per model update:

$$\boldsymbol{w}_{t+1}[i][\mathrm{index}(A_t)] = \boldsymbol{w}_t[i][\mathrm{index}(A_t)] + \alpha\delta_t, \forall i \in s_{list}. \tag{3}$$

Figure 2 shows how this applies to our prior example. If desired we may define a state space with an arbitrary number of tiles per dimension (higher-resolution, lower generalisation), yet having constant-size state vectors and constant action computation cost ($O(n_{tilings})$). Beyond this, we need not store action values for tiles which have not yet been visited, conserving memory. A caveat of tile coding remains, in that the value of α must be reduced according to the number of tilings to prevent divergence at the expense of slower learning ($\alpha \leftarrow \alpha / n_{tilings}$).

C. Motivation

Moving beyond the overt benefits of choosing RL-based defences for coping with non-stationary problems, we believe that there are concrete reasons for their use here. We have seen that for other domains in particular, misclassification is a serious problem, which can introduce *collateral damage* in the context of DDoS prevention. In theory, the feedback-loop-like model allows us to monitor flows *after* an action is taken to allow forgiveness of mistakenly punished flows. This does rely upon the ability to take a flow-by-flow view of the state space, but if we can combine knowledge of current state with the last applied action, then perhaps a flow which falls off identically to a legitimate flow can be rescued.

Other studies suggest that there are particularly useful features which make the task of online DDoS flow identification feasible. Aggregate network load observed at various locations suggests the overall health of a network [5], and the ratio of correspondence between pair flows can suggest asymmetry and in many cases illegitimacy [10]. Generic volume-based statistics (counts, counts per duration, average packet sizes) have seen effectiveness in such as k-nearest neighbours classifiers trained to detect DDoS attacks in progress [15]. Most importantly, there is evidence showing behavioural changes in response to bandwidth expansion [7], suggesting similar artefacts might arise after throttling, packet drop, or other interference.

D. Threat Model

An attacker's goal is to minimise the fair-share bandwidth allocation that a server can give to hosts, and they are expected to act rationally in its pursuit. Threat actors are external and act intentionally, aren't expected to be *advanced persistent threats*, and likely range from hacktivists to moderately funded adversaries. We assume that attacks will be volumetric DDoS attacks with the structure of an *amplification attack*, and that traffic aggregates at the target (unlike in a transit-link attack). The addresses of the set of unwitting reflector nodes are visible to the target, though any bots taking part in an attack or the machines those bots control are not revealed to the target without communication with 3rd party organisations such as upstream ISPs. The discovery of any reflector by some defence system does have a cost to the attacker—there is a particularly large (yet finite) supply of viable reflector nodes [10], but the constraints that each has a large upstream bandwidth and support for high-amplification-factor protocols narrow this pool.

We do not assume that an attacker has white-box access to an agent's policy, or that they will attempt to intelligently modify flow/system state to indirectly control an agent [16]–[19]. While they may be able to perform some degree of reverse engineering by observing the health of their own legitimate canary flows, "stealing" the policy through observation [20], investigating whether perturbations would persist in volatile network traffic statistics falls outside of the scope of this work. The same observation extends to the possibility of poisoning attacks [21]. These are APT-level capabilities, whose exploration presents a rich source for future work.

III. DDoS Mitigation with Per-flow Reinforcement Learning

Our main hypothesis is that the best method for advancing past the current shortcomings of RL-based DDoS mitigation is to design agents such that filtering decisions are computed per flow. However, these alterations must account for computational constraints imposed by the deployment environment—the amount of flows passing over an agent is unbounded. We describe and justify our approach, our algorithmic improvements, and present two action models, one of which draws on domain knowledge introduced by SPIFFY [7].

A. System Design and Assumptions

A deployment environment is a network with a set of *ingress/egress points* from its domain of control, through which traffic can flow, and a set of protected *destination nodes*. These nodes may be services, servers, or in the case of Autonomous Systems (ASes) and transit networks, egress points leading to other networks. Agents are co-located with each egress switch (i.e., k ingress points from other ASes require k agents), all employ the same action model/design, and control the proportion of upstream packets from each external host to discard. Each destination node s has a maximum capacity, U_s.

We assume that the deployment environment is a moderately complex software-defined network, because the paradigm offers features which can directly benefit RL agents acting within. The OpenFlow protocol allows a controller (or other authorised hosts) to install complex actions, forwarding rules and logic into a switch at runtime. Furthermore, networks of this kind more naturally enable the future use of *network function virtualisation*, a technology which could allow relocation and easy installation of learners (e.g., as examined by Jakaria *et al.* [22]). Agents communicate with their co-hosted OpenFlow-enabled switches—running a modified version of *Open vSwitch* (OVS) [23]—to install probabilistic packet-drop rules.

Our system design applies to both software-defined and traditional networks of arbitrary shape and size. Only the ingress/egress nodes from a network need to be OpenFlow-enabled, as it is advantageous to perform filtering as close to a flow's source as possible. In a traditional network, each agent has exclusive control over its switch's tables. In a fully software-defined network, these agents require exclusive control over the first table, forwarding legitimate packets to subsequent tables managed by the network's controller. The main difference is that a traditional network needs this additional hardware, and does not allow an operator to dynamically determine where the "edge" of their network lies through vNF relocation.

B. Algorithm

To make decisions cheaply and at low latency, we use *semi-gradient Sarsa with tile coding* as described in eq. (2) and section II-B, rather than using neural networks or more complicated function approximators. Exploration is introduced via ϵ-greedy action selection, linearly annealing ϵ to 0 over time. Each agent has its own internal parameter vector $\boldsymbol{w}$, and agents do not share their weight vector updates with one another (but may share experience and traces with one another).

Although the choice of a classical RL method likely brings lower theoretical performance, there are significant reasons to favour such methods; these include lower latency decision-making, lower energy usage, reduced model complexity (and training time), the availability of necessary hardware, and simpler decision boundaries. This aligns with our goal of quick online learning, and faster adaptation to aggregate changes in traffic without introducing dedicated tensor processing hardware to networks. Simpler decision boundaries reduce the risk of overfitting and unexpected behaviour, and we expect that the simplicity of tile-based policy computation will also greatly aid interpretability of anomalous action choices.

When choosing a learning algorithm, we compared against Q-learning as well as methods based on *eligibility traces* such as Watkins's Q(λ) [14, pp. 312–314] and Sarsa(λ) [14, pp. 305]. Preliminary experiments found that Sarsa offered the best performance and behaviour.

1) Action rate

We adapt the algorithm to prioritise rapid response to changes in network state and to visit as many state-to-state transitions as possible for effective learning. To this end, we allow agents to make many decisions per timestep. We maintain the last state-action pair associated with each source IP and destination, and calculate any actions for the flows which still exist. Finally, we update $\boldsymbol{w}$ using each available trace and the reward signal from the relevant destination. As exploration still occurs for each action, this approach reduces ϵ multiple notches every timestep. In turn, we increase the annealing window for ϵ by a factor of 2.67 so as to preserve exploration over time, by accounting for the greater volume of decisions being made.

2) Per-tile updates

While the standard formulation of eq. (2) updates the value of all tiles identically (by a scalar $\alpha\delta_t$), we found it more effective to compute a different temporal difference value *for each tiling*. While we make use of the sum of all tiles' action value estimates when making decisions, each tiling is updated using only its own contribution, allowing us to set α to a higher value without divergence. A crucial observation is that value updates to each tile can move by different values in different directions, converging on effective estimates sooner.

3) Decision narrowings

When learning control on the basis of a high-dimensional, tile-coded state space, assignment of credit for each decision is difficult (because all tiles have identical gradient). To combat this, with probability ϵ an agent will mark a flow as being governed by a subset of the state space for the next 5 decisions. Each agent chooses actions on that source/destination pair using one element of local state, the global state, and the bias tile—we include the latter two to strike a balance between and accuracy and correct credit assignment.

C. Feature Space

Our state space combines elements of global state (network link load observations) with per-flow measurements. Each is

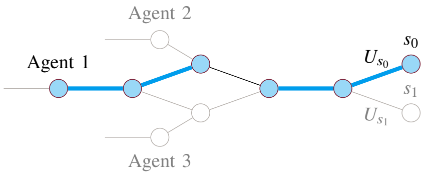


Figure 3. Global state selection for a flow between an external host and server s_0 which passes over Agent 1. All nodes in the path taken through the defended network are filled in blue, and all link load measurements which are chosen for action computation are indicated with a thick blue line.

tile-coded with 8 tilings and 6 tiles per dimension, using the windows described in table I.

Global state is a vector of load values in $\mathbb{R}^4$ (Mbit/s) depending upon the bandwidth measurements regularly received from monitors in the environment. For any flow, an agent then computes the path it would take through the network. The incoming load recorded along the first hop, last hop, and tertiles of the path may then be tile-coded together. In the event that the path from an agent to its destination is shorter than 4 hops, we duplicate (in order of preference) the load measurement of a middle hop or the last hop. Figure 3 illustrates the process.

We build global state in this way to offer compatibility with multipath, multi-destination networks, offering support for diverse deployment environments from endpoint servers to transit ASes. Computing the path from agent to destination is not computationally expensive. Multipath routing is often fast since typical *equal-cost multipath* (ECMP) routing algorithms simply hash a packet's flow key, and are deterministic to provide consistent quality-of-service to hosts.

We describe and analyse each of the per-flow features included in the state vector throughout section IV. Each feature is tiled separately, with the exception of packet in/out count (per-window and total), mean in/out packet size, and Δ in/out rate, which are combined with the last action taken. Rather than having the network push the data to an agent, the agent requests this information about active flows periodically to isolate it from non-control-plane traffic and to eliminate the risk of resource exhaustion by excessive requests.

D. Reward Function

Each destination node s generates a reward signal, $R_{s,t}$, at every timestep t. Assume, for now, that each destination has access to some classification function $g(\cdot)$ which estimates the volume of legitimate traffic received, and expects to receive $\mathit{traffic}_s$. Denoting the upstream, downstream and combined loads $\mathrm{load}_t^{\uparrow}(s), \mathrm{load}_t^{\downarrow}(s), \mathrm{load}_t^{\updownarrow}(s)$ at this node:

$$\mathrm{c}_{s,t} = [\max(\mathrm{load}_t^{\uparrow}(s), \mathrm{load}_t^{\downarrow}(s)) > U_s], \tag{4a}$$

$$R_{s,t} = (1 - \mathrm{c}_{s,t}) \frac{g(\mathrm{load}_t^{-}(s))}{\mathit{traffic}_s} - \mathrm{c}_{s,t}, \tag{4b}$$

replacing $\mathrm{load}_t^{-}(s)$ in eq. (4) with whichever directional load is prioritised according to the traffic characteristics of the deployment environment, where $\mathrm{c}_{s,t}$ represents the "overloaded" condition at destination s. We choose $\mathrm{load}_t^{\uparrow}(\cdot)$ for our UDP-based models and $\mathrm{load}_t^{\downarrow}(\cdot)$ for HTTP, though we expect that $\mathrm{load}_t^{\updownarrow}(\cdot)$ would be the most suitable for general deployment or heterogeneous traffic patterns. These choices reflect where the bulk of transmitted bytes in each traffic model are observed (and the lack of this knowledge in the general case).

While our use and definition of $g(\cdot)$ appears nebulous, there are many ways to infer this quantity in practice. End-host servers may use canary flows or other active measurements, or employ existing quality-of-experience metrics in the case of VoIP services such as lost packets, reorderings, and jitter. ASes and transit networks may make use of reports received from downstream networks, i.e. over the *DDoS Open Threat Signalling* (DOTS) protocol [24]. Even if such heuristics or perfect knowledge aren't available in deployment, a sufficiently well-trained agent needs only to greedily follow the policy it has learned from training, allowing pre-training by a simulated environment (with perfect knowledge) to transfer to reality.

If a network is believed to be vulnerable to indirect attacks, such as link-flooding attacks, we may use the following reward:

$$R_{s,t}^{Cross}(\beta) = \beta R_{s,t} + (1 - \beta) \min \{R_{s',t} | s' \neq s\} \tag{5}$$

where the collaboration parameter $\beta \in [0, 1]$ models the expected degree of interference between flows, and s, s' are protected destination nodes in the network. The key insight underpinning LFAs is that flows can affect a target *without communicating with that target*. β then acts as a tunable parameter which can incentivise agents to remove flows which harm overall system health, by including the performance of the worst-performing destination. However, such attacks (and the effectiveness of $R_{s,t}^{Cross}$) are not examined by our work.

E. Action Space

When monitoring a source-destination pair, an agent uses its state vector to decide which proportion of that flow's *inbound* traffic should be dropped. This is implemented by installing an action via OpenFlow, instructing its host switch to drop each relevant packet with probability p. We choose to drop packets rather than impose traffic limits as it offers us a discrete action space without prior knowledge of traffic characteristics or measurement. Furthermore, we need not consider burstiness, fairness or tuning (such as per-flow bucket sizes) which could limit scalability. We offer two models on how to choose p:

1) Instant control

Each agent directly chooses $p \in \{0.0, 0.1, \ldots, 0.9\}$, giving a discrete, static action set which cannot completely filter traffic. These choices ensure that the rate reduction imposed on a source IP may never be permanent or irreversible. Since this model needs no forward planning, we found it best to set the discount factor $\gamma = 0$ (making agents purely myopic).

2) Guarded control

The measurements of Kang *et al.* [7] suggest that bot attack flows cannot scale up to match an increase in available bandwidth. We apply their observations within the RL paradigm by constraining how an agent treats each flow using a simple finite state automaton: we restrict $p \in \{0.00, 0.05, 0.25, 0.50, 1.0\}$. The action set is then simply to *maintain*, *increase*, or *decrease* p for a flow in single steps. We choose these potential values for p to add complete filtering to a steady progression of rate-limiters (25 % increments for UDP traffic). The outlier,

$p = 0.05$, corresponds to roughly a 50 % rate reduction for TCP flows in our test topology. This uneven spread of choices for p allows light and heavy rate reduction to be applied to both congestion-aware and congestion-unaware traffic as required.

To enable temporary bandwidth expansion in all deployments, every flow is initially placed under light packet drop ($p = 0.05$); this is chosen above the equivalent for UDP due to TCP's higher prevalence. Most importantly, an agent must now choose to punish a flow multiple times in succession to cause rapid degradation, reducing variance while allowing an agent to see how a host reacts to structured changes in the environment.

As each agent now requires the capability to plan ahead, we require a discount factor $\gamma \neq 0$, allowing the value of future states to influence state-action value updates. We found the setting $\gamma = 0.8$ to be the most effective choice for this hyperparameter during exploratory testing.

3) *Risks*

Our mode of action means that each agent is in control of pushback [25], and so carries a risk of introducing collateral damage into the network. This is particularly severe when handling TCP traffic: the Mathis equation [26] states that TCP bandwidth is proportional to $1/\sqrt{p}$ (noting that p is nonzero in any real network) while constant bitrate (CBR) UDP traffic is proportional to $1 - p$. This weakness is still present in modern TCP flavours, such as TCP Cubic which in turn has bandwidth proportional to $1/p^{0.75}$ [27]. This is of particular importance due to the prevalence of TCP and other congestion-aware protocols within the Internet. Our own analysis of CAIDA datasets [28] shows that congestion-aware traffic makes up at least 73–82 % of packets, corresponding to 77–84 % of data volume[1]. QUIC, a future congestion-aware protocol, comprises 2.6–9.1 % of traffic observed on backbone links, depending on location and typical workload [29].

This further justifies our focus on per-flow decisions—real-world deployments see many flows pass over any egress point, making global actions (such as those chosen by Malialis and Kudenko [5]) more likely to inflict collateral damage. Given the probability that a host is legitimate, $P_G \in [0, 1]$, it follows that a host will be malicious with probability $P_B = 1 - P_G$. Defining *imperfect service* to mean any case where all n hosts connecting over a switch do not share the same classification (i.e., a mixture), then the probability that a switch is delivering imperfect service is $P_{M,n} = 1 - (P_G^n + P_B^n)$.

Theorem 1. *As the host/learner ratio n increases, it is more likely that a throttling switch will exhibit imperfect service:* $\forall n \in \mathbb{Z}^+, P_{M,n} \leq P_{M,n+1}$.

Proof. Base case: $P_{M,1} = 0, P_{M,2} = 1 - P_G^2 - P_B^2 > 0$. *Inductive step:* Assume that the theorem holds for n. Observe that $P_G^n \geq P_G^{n+1}$ (resp. P_B). It then follows that:

$$\begin{gathered} P_G^n + P_B^n \geq P_G^{n+1} + P_B^{n+1} \\ 1 - (P_G^n + P_B^n) \leq 1 - (P_G^{n+1} + P_B^{n+1}) \\ P_{M,n} \leq P_{M,n+1} \end{gathered}$$

□

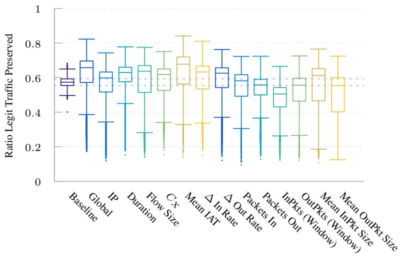


Figure 4. Learned performance of Instant Control agents when benign traffic is UDP-like, using only a single feature as a basis for decisions. Mean IAT, inbound packet sizes, and global state offer the best predictive performance, while most features offer marginal advantage over the unprotected baseline.

Corollary 1.1. *Restricting* $P_G \in (0, 1)$ *so that both* P_G *and* P_B *are non-zero ensures strict inequality:* $P_{M,n} < P_{M,n+1}$.

When considering that many hosts have an especially adverse reaction to our main means of control, flow-level granularity becomes an obvious choice.

F. *Systems Considerations*

Taking many actions per timestep means that any agents are assigned a larger, and potentially unbounded, set of tasks to perform every time they receive load and flow statistics from the network and their parent switch. This introduces some potential issues: the inability to respond to unexpected changes in flow state, delayed service of new flows, and risks that flow states become outdated. At their worst, these risks present additional attack surface to an adversary. To adapt to these problems, we make use of *timed random sequential* updates.

Each agent begins with an empty work list. For the set of flows active in any timestep, we shuffle the list and perform as many action calculations and updates as possible, within a set time limit. Uncompleted work is passed on to the next timestep, until the list is emptied, at which point it is repopulated using the set of available measurements. To ensure that flow control actions are made with recent information, we combine state vectors for unvisited flows in the current work set, and replace the stored vector for all others. State vector combination is done by summing deltas and packet counts, updating means via weighted sums, and replacing all other fields. Following Chen *et al.*'s observations concerning short flows [30], we maintain a deadline of 1 ms—in tests, an agent is typically able to process around 3 flows in this time. We expect this should be tuned based on the frequency at which statistics arrive. Naturally, this implies that an agent must carry work forward (and coalesce state updates) when *host density* is $n > 3$ (section VI); this behaviour is not explicitly a property of network size.

IV. Rethinking the State Space

The main element required by a per-source model is a feature set with high predictive power, so that behavioural differences are apparent to an agent. Elaborating on the statistics discussed in section II-C which others have shown to be effective, we believe the following features to be useful (and humanly justifiable), and investigate their use alongside different traffic types:

Global state: This is the vector of load measurements along a flow's path introduced in section III-C. These values indicate the overall health of the network, and crucially are all

[1] https://github.com/FelixMcFelix/caida-stats

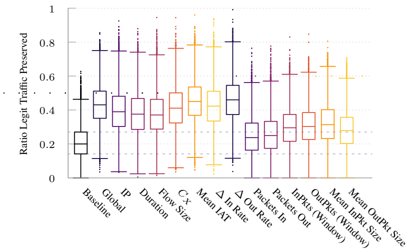


Figure 5. Learned performance of Instant Control agents when benign traffic is TCP-like, using only a single feature as a basis for decisions. All of the chosen features can offer a marked improvement over no protection at all. Global state and Mean IAT still offer the greatest improvement above baseline, but packet-level statistics are considerably less effective for this class of traffic.

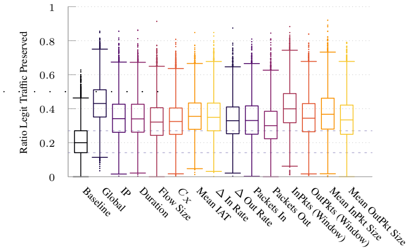


Figure 6. Learned performance of Instant Control agents when benign traffic is TCP-like, combining each feature with the last action taken as a basis for decisions. This combination causes a significant improvement in the effectiveness of packet-level and per-window statistics.

measurements which an agent directly controls.

Source IP address: While trivial to spoof (and thus of limited use for many classes of attack), reflectors are themselves legitimate services being abused by spoofing attackers. As a result, they communicate with attack victims using their own IP address. In real-world scenarios the addresses of reflector nodes might exhibit similarity due to network uncleanliness [31], e.g., unhardened services exposed by a single organisation.

Last action taken: This encodes an agent's current belief in the maliciousness of a flow. This feature also potentially allows forgiveness, serving as a reference point for determining whether a source mistakenly marked as malicious exhibits different falloff behaviour after punishment. It's important to note that this feature only makes sense once combined with another flow feature, and never appears individually tile-coded.

Flow duration and size: Features which describe the length of time a connection has been active, and the amount of data transferred within that time. An extraordinarily long flow, having sent a lot of data, could be more likely to be an amplifier: though most (62 %) waves of amplifier traffic last shorter than 15 min [32], this is considerably longer than the typical length of an HTTP request/response.

Correspondence ratio: The ratio between upstream and downstream traffic for a source IP. We define this to be $C_X = \min(\text{load}_t^{\uparrow}(\cdot), \text{load}_t^{\downarrow}(\cdot)) / \max(\text{load}_t^{\uparrow}(\cdot), \text{load}_t^{\downarrow}(\cdot))$, where a value close to 0 indicates strong asymmetry.

Δ Send/receive rate: The change in traffic rates caused by the last action. Behavioural changes induced by bandwidth expansion/reduction are expected to be most visible here.

Mean inter-arrival time (IAT): A measure of how often packets arrive at the agent's parent switch; low IATs indicate a high number of packets per second, and can be a possible marker of malicious behaviour. We only make use of the mean IAT of *inbound* traffic.

Table I
TILE CODING WINDOWS FOR EACH FEATURE.

New Feature (unit)	Range
Load (Mbit/s)	$[0, U_s]$
IP	$[0, 2^{32} - 1]$
Last Action (%)	$[0, 1]$
Duration (ms)	$[0, 2000]$
Size (MiB)	$[0, 10]$
Correspondence Ratio	$[0, 1]$
Mean IAT (ms)	$[0, 10\,000]$
ΔIn/Out Rate (Mbit/s)	$[-50, 50]$
Packets In/Out	$[0, 7000]$
Packets In/Out Window	$[0, 2000]$
Mean In/Out Packet Size (B)	$[0, 1560]$

(Per-window) packet count: The amount of packets sent to/from a source over a flow's lifetime (or the current window of measurement), similar in use to flow size and mean IAT.

Mean packet size per window: Legitimate flows, both TCP- and UDP-based, often transmit packets with a distribution of sizes. Attack traffic is not likely to be so diverse: we might expect solely max-size packets in the case of amplification attacks, or minimum-size packets in other flooding attacks.

The exclusion of features such as source/destination ports or protocol numbers is a deliberate choice. If *QUIC* (or a similar protocol) were to become ubiquitous, then these fields would have little to no correlation with the class of traffic a flow might contain. Our aim was to design around this constraint as a form of future-proofing.

All of the above features, save for global state, are 1-dimensional. Figure 4 shows the effectiveness of each feature for UDP (resp. fig. 5 for TCP), on a single-destination topology (section VI-A) with $n = 2$ hosts per egress point averaged over 10 runs. Figure 6 demonstrates how feature accuracy varies when tiled alongside *last action*, with similar trends observed when applied to UDP traffic (omitted). The plots show that different protocols and traffic classes are best defended by different features—as such, every feature presented has value in a complete model. All features converge to their highest-observed performance within around 4000 timesteps. In general, some of the most effective features are the global state, mean IAT, mean inbound packet size and Δ rates.

V. Traffic Modelling

We contribute network models built around live testing of reactive TCP and UDP traffic in an SDN-enabled environment, which is adaptable to arbitrary topologies, with an explicit focus on preserving their real-time dynamics in a way that trace-based evaluation cannot. First and foremost, we are interested in representative load and packet inter-arrival characteristics and in how these characteristics evolve in response to actions. We introduce these models because we are interested in capturing interactive, correlated back-and-forth exchanges associated with live HTTP traffic; mainly because of the particular interactions between the application-level dynamics, congestion awareness at the transport level and the nature of control signal used.

A. Network Design

We make use of a fully software-defined network, built using OpenFlow-aware switches in mininet alongside a controller based on *Ryu* [33]. All internal routers are primed with knowledge of the shortest path to each internal host, while new inbound flows register the "way back" for each hop used, to ensure consistent traffic conditions for each flow. If several ports offer different (equal-length) paths to a destination, a consistent random port is chosen from the flow-hash by an OpenFlow *Group action* (in *select* mode). If such information is lost, perhaps expiring due to inactivity, it suffices to forward an outbound packet on a random outbound port, as we assume that any external IP is reachable through any of the test network's egress ports (i.e., that it is not connected to any stub ASes). The controller is also responsible for computing how switches respond to ARP requests: this need arises due to the reliance upon Linux's networking stack for live applications, and wouldn't need to be considered for trace-based evaluation.

B. TCP (HTTP) Traffic Model

To model legitimate TCP traffic, server nodes run an nginx v1.10.3 HTTP daemon, serving statically generated web pages alongside various large files and binaries. Benign hosts run a simple libcurl-based application written in Rust, repeatedly requesting resources from the server. Hosts and clients both use TCP Cubic [27]. Each host's download rate is limited to match the maximum bandwidth assigned to it, and requests several random files known to exist within a website, followed by any dependent resources for each (stylesheets, images, etc.) as a browser might. On completion, a host changes its IP to generate separate statistics per-flow, while minimising downtime. This presents a balanced distribution of flow duration and size, with large files included to model elephant flows.

C. UDP (Opus/VoIP) Traffic Model

VoIP traffic exhibits very different characteristics to the above model; packet arrivals are highly periodic due to real-time requirements, flows have a constant bitrate, and do not react substantially to lost packets. Interestingly, DDoS attack traffic is known to share many of these characteristics, offering an interesting detection problem. We present a VoIP traffic model[2] based on Discord[3], a freely-available messaging and VoIP platform geared toward gaming communities. We chose Discord as our prototype due to its publicly documented API, many open source bot frameworks, large user base, and due to the lack of models for Opus-encoded traffic.

Hosts send RTP traffic with Salsa20 encrypted payloads—20 ms audio frames at 96 kbit/s. We generate similar traffic at hosts by replaying anonymised traces gathered in general use and tabletop RPG servers; each trace contains only the size of each audio payload, entries denoting missed packets, and the duration of silent periods. We trim these silent periods to a maximum 5 s due to the lengthy talk/silence bursts introduced by users in RPG servers, and estimate the size of missed packets by taking an exponentially-weighted moving average over known sizes. Hosts punctuate audio frames with a 4-byte keepalive every 5 s. All traffic passes over a central server which groups hosts into rooms, and is forwarded to other participants; we do not replicate pre-call Websocket traffic which would be used for authentication. There is no peer-to-peer traffic—the server acts as a TURN relay for all hosts. We find that each flow occupies an expected 52.4 kbit/s upstream bandwidth. To match the target upload rate assigned to each host, it runs enough individual sessions to meet the target data rate.

D. Attack Traffic Model

Malicious traffic is generated by use of the *hping3* program, generating UDP-flood traffic targeting random ports. We configure each instance of *hping3* to generate ethernet MTU-sized packets (1500 B) with a random source and destination port towards a target server, and configure the output rate r (in Mbit/s) by setting the inter-arrival time $t_{attack} = \frac{1500 \cdot 8}{r \cdot 10^6}$. This fulfils certain characteristics of many types of amplification DDoS traffic: it is congestion-unaware [10], and packets are larger than the minimum frame size and identically-sized (e.g., NTP amplification traffic is fragmented at the application layer into 482 B chunks [34]). We differ from NTP amplification in frame size so that inter-arrival times are larger, to keep emulation of the network feasible at high traffic rates.

VI. Evaluation

We compare our work most naturally against MARL, introduced by Malialis and Kudenko [5], the state-of-the-art in RL-based DDoS prevention. We are most interested in seeing how their approach contrasts with ours across different topologies and workloads. Different network environments will also impose different levels of host density, where popular web servers may have orders of magnitude more clients than egress points from their network—we aim to see how these characteristics affect performance and learning rate. Marl is known to outperform the AIMD [35] strategy, yet the state of the art has long since moved on. To paint a more current picture, we compare our work against an effective modern approach, SPIFFY [7]. SPIFFY tests a proportion of flows by routing them through an alternate path with higher bandwidth, observing how their speed changes some time later. This comparison lets us position our new agent designs against the state of the art, observing that SPIFFY has a similar mode of interaction to RL-based systems (taking action, observing an effect, and acting once again) and does not rely on protocol characteristics or signatures. We make the simplifying assumption that a suitable unused path exists (with identical bandwidth to the server's link). We test 10 % of active flows at a time (according to the authors' observation that there is a factor of 10 difference between the ideal and achieved bandwidth expansion), excluding flows below 50 kbit/s and requiring a 3× expansion from legitimate flows, making a judgement after 5 s.

To test this, we made use of both traffic models introduced in section V (OPUS and TCP), both topologies discussed below (1-dest vs Fat-Tree), and vary the amount of hosts typically communicating over each agent's ingress/egress node. Additionally, we evaluated our new models in multi-agent mode (*separate*, no model sharing), and in single-agent mode (*single*, 0-cost perfect information sharing). In each case, the algorithm's performance was averaged over 10 episodes of length 10 000 timesteps (setting each agent's $\boldsymbol{w} = \mathbf{0}$ between episodes). Host allocations at the beginning of each episode were generated pseudorandomly to ensure fairness between episodes—a host is malicious with probability P(*malicious*), and is benign otherwise. Benign hosts generate traffic from

[2] https://github.com/FelixMcFelix/opus-voip-traffic

[3] https://discord.gg

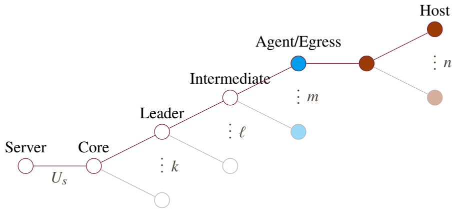


Figure 7. Network topology diagram, showing how the server and its core switch's k teams are structured, with ℓ intermediate routers per team, connected to m agents which each moderate n hosts beyond a single external switch. Red nodes are external, and each blue node hosts an agent.

either sections V-B and V-C depending on the experiment, while malicious hosts generate traffic according to section V-D (both at experiment-dependent rates).

All experiments were executed on Ubuntu 18.04.2 LTS (GNU/Linux 4.4.3-040403-generic x86_64), using a 4-core Intel Core i7-6700K (clocked at 4.2 GHz) and 32 GiB of RAM. All code underpinning these findings is available on a public repository[4].

A. Single Destination

The network is tree-structured, where one server s connects through a dedicated switch to k team leader switches, each connected to ℓ intermediate switches, which in turn each connect to m egress switches. We then have $N_{hosts} = k\ell mn$. Figure 7 demonstrates this. We configured the network topology using $k = 2$ teams, $\ell = 3$ intermediate nodes per team, $m = 2$ agents per intermediate node, and $n \in \{2, 4, 8, 16\}$ hosts per learner. This is a slight simplification of Malialis and Kudenko's 'online' experiment [5], choosing fewer teams but remaining as a single server with a fan-out network.

B. Multiple Destinations

The previous topology allows for direct comparison against the state-of-the-art, and indeed is illustrative of one way in which attack traffic might aggregate in the network. It is hard, however, to argue its relevance to specific classes of victim or to reason about the interactions it might have with dependent applications. In contrast, the fat-tree topology [36] sees regular use in real-world datacentres and scales well horizontally. We use a $k = 4$ fat-tree, with one pod hosting two servers s_0 and s_1. n external hosts connect through each core switch (where agents are hosted), and communicate with s_0, s_1 uniformly randomly. Both servers host identical services. We set $n \in \{6, 12, 24, 48\}$ hosts per learner (keeping N_{hosts} identical to each tier of the single-host topology), and restrict $U_{s_0} = U_{s_1} = U_s/2$.

C. Parameters

The algorithm parameters were set at $\alpha = 0.05$, linearly annealing $\epsilon = 0.2 \rightarrow 0$ by $t = 3000$ in the case of Marl (8000 actions per agent in the *Instant/Guarded* models).

Benign hosts each occupied 0–1 Mbit/s, and hosts were redrawn at each episode's start with P(*malicious*) = 0.4. Malicious hosts each sent 2.5–6 Mbit/s when attacking UDP traffic, though this was increased to 4–7 Mbit/s when using TCP-like traffic (to meaningfully impact benign flows). Given n and P(*malicious*), we see an expected malicious bandwidth 1.27–1.87 and 2.03–2.18×U_s respectively. For our choices of n in both topologies, we observe $N_{hosts} \in \{24, 48, 96, 192\}$, and an expected number of malicious hosts $\mathbb{E}[N_{attackers}] \in \{9.6, 19.2, 38.4, 76.8\}$. For the largest choice of n, we see an expected total attack traffic $\mathbb{E}[V_{attack}] = 334.05$ and 422.4 Mbit/s for Opus and HTTP traffic respectively.

Table II
AVERAGE REWARD FOR COMBINATIONS OF MODEL, HOST DENSITY AND TRAFFIC CLASS WITH A SINGLE DESTINATION.

Traffic	n	SPIFFY	Marl	Instant		Guarded	
				Separate	Single	Separate	Single
OPUS	2	0.043	0.628	**0.629**	0.448	0.430	0.629
	4	0.069	0.538	**0.653**	0.449	0.308	0.571
	8	0.065	0.468	**0.533**	0.516	0.398	0.507
	16	0.053	0.460	0.438	0.452	0.347	**0.504**
TCP	2	**0.799**	0.305	0.061	0.068	0.241	0.196
	4	**0.953**	0.359	0.191	0.097	0.278	0.504
	8	**0.995**	0.362	0.376	0.201	0.357	0.605
	16	**0.999**	0.320	0.316	0.302	0.478	0.708

Table III
AVERAGE REWARD FOR COMBINATIONS OF MODEL, HOST DENSITY AND TRAFFIC CLASS WITH MULTIPLE DESTINATIONS.

Traffic	n	SPIFFY	Marl	Instant		Guarded	
				Separate	Single	Separate	Single
OPUS	6	0.092	**0.382**	0.300	0.170	0.307	0.189
	12	0.096	0.217	0.322	0.275	**0.333**	0.235
	24	0.125	0.404	0.358	0.296	0.382	**0.461**
	48	0.110	0.430	0.418	**0.438**	0.427	0.428
TCP	6	**0.692**	–0.222	0.123	–0.018	0.121	0.116
	12	**0.896**	0.008	0.132	0.008	0.163	0.266
	24	**0.974**	0.063	0.130	0.024	0.337	0.390
	48	**0.995**	0.156	0.219	0.111	0.431	0.499

U_s was fixed at $N_{hosts} + 2$ Mbit/s (to account for burstiness), and each link had a delay of 10 ms. All links had unbounded capacity, save for each server-switch. These parameters match those of the original study to enable direct comparison, and many are (to the best of our knowledge) arbitrary, but we justify our range of n as capturing increasing scales of host activity.

VII. RESULTS

We now examine the performance of our two new models (*Instant*, *Guarded*) as compared against existing RL work (*Marl*) and *SPIFFY* under different traffic behaviour and topologies, varying the host-to-learner ratio n and environment. We present the average rewards for all combinations of these factors in tables II–III—providing a rough idea of expected performance, with the highest-performing model in bold and the best RL-based model underlined. Average rewards take into account any portions of time that an agent allows illegal system states. Several plots augment this, illustrating peak performance or the amount of time which an agent requires to learn.

A. Congestion-unaware traffic

In a single-destination network, we observe that Marl's performance degrades as n increases. Typically, our *Instant* agent design achieves the best performance in multi-agent mode, having lower collateral damage than the current state-of-the-art, but sharply degrades at low n when agents share experience. This trend reverses for the *Guarded* model, which improves as n increases and in single-agent mode—when $n \geq 4$, the single-agent variant offers consistent improvement. Figure 8 shows the preserved traffic in multi-agent mode. When defending multiple destinations, we see a sharp decrease in the effectiveness of all

[4] https://github.com/FelixMcFelix/rln-dc-ddos-paper

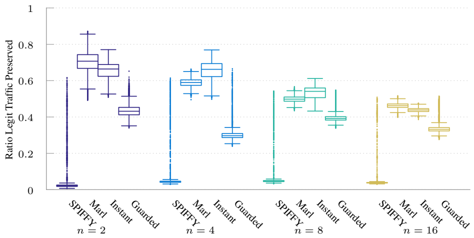


Figure 8. Online performance for Opus benign traffic in a single-destination network, multi-agent mode. *Instant* outperforms Marl for $n \in \{4, 8\}$ (with higher variance), but performs similarly to Marl at $n \in \{2, 16\}$. *Guarded* underperforms compared to the other agent designs in this problem variant.

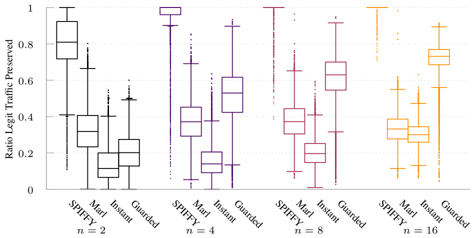


Figure 9. Online performance for HTTP benign traffic in a single-destination network, single-agent mode. *Instant* and *Guarded* exhibit similar efficacy at $n = 2$, protecting less traffic than Marl. Only *Guarded*'s performance rapidly increases with n, achieving a considerably better median and lower variance than the other models. The longer tails of outliers typically indicate the longer training time the new models require—we observe that *Guarded* typically has considerably lower variance once it has converged on a stable policy.

agent designs. Our new agent designs become more effective as n increases, while Marl's effectiveness is roughly constant (aside from the outlier at $n = 12$). Interestingly, SPIFFY is unable to effectively protect constant bitrate traffic.

B. Congestion-aware traffic

Table II shows that Marl offers a low (though fairly consistent) level of protection for TCP traffic, which the *Instant* agent offers no substantial improvement over. However, *Guarded* agents offer a remarkable improvement for this class of traffic, particularly when experience can be shared—offering a 2.21× improvement over the state-of-the art during training, which is made clearer in fig. 9. Figure 10 shows that this model can protect a peak 80 % of TCP traffic (2.5× improvement) after just 100 s, but also that all of the new models require considerably longer than Marl to learn their best-achieving policy. We observe that the same trends present themselves in the multi-destination topology: *Guarded* remains the best fit for TCP, in both training modes. Crucially, the rigid tree of learners and teams which define Marl, along with its lack of action granularity, seem to be a poor fit in this environment. In both cases, SPIFFY greatly outperforms the RL-based methods.

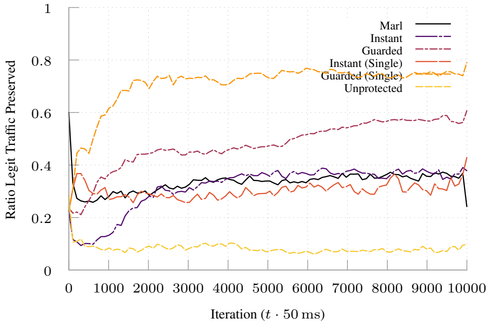


Figure 10. Online performance of standard and single-agent models in a single-destination network with $n = 16$ hosts per egress point, HTTP traffic. At this level of host density, *Guarded* reaches higher peak performance sooner and is considerably more consistent throughout the episode. *Guarded* benefits greatly from information sharing, converging to protect around 75 % of TCP traffic within 100 s. The *Instant* model converges to Marl's level of performance.

Table IV
AVERAGE REWARD VERSUS ATTACK VOLUME.

Factor	$\mathbb{E}[V_{attack}]$ (Mbit/s)	Reward
1.5	633.6	0.671
2.0	844.8	0.625
2.5	1056.0	0.620
3.0	1267.2	0.619
3.5	1478.4	0.600

C. Increased Attack Volume

To assess the effect of larger volumes of attack traffic, we increase an attacker's output by various factors, supposing $n = 16$ with HTTP traffic (*Guarded*, Single); table IV records the expected rate of attack and average performance. The initial increase in traffic volume causes the steepest reduction in performance (due to the increased cost of incorrect action), though performance levels out as attack traffic increases.

D. Computational Cost

Measurements from each of these experiments indicated that the cost of computing any action is typically within 80–100 µs per flow. This is reassuring when measured alongside the insights from other work. Chen *et al.* [30] observe that, ideally, actions must be computed and taken within 1 ms to have a meaningful affect on short flows. That our starting point falls significantly below this threshold allows us to safely consider more costly actions or larger state spaces, which would typically increase the computational cost. This cost is constant and independent of network size. As discussed in section III-F, we are able to judge 3 flows before this deadline: the difference is primarily accounted for by serialisation/communication delays and single-threaded processing in the Python language.

VIII. Discussion

Model performance: Of the results presented, *Guarded*'s unpredictable (often worse) starting performance is unexpected, given its far smaller action space. It's natural to expect that this would make the model easier to learn, but the additional state required appears to make the task *harder*, beyond even the value of choosing a non-zero discount factor (adding forward-planning to explicitly mitigate this effect). Accordingly, we see that this design performs best (and exhibits considerably lower variance) when agents learn from as much knowledge as possible: high n and single-agent training. To filter incoming traffic from a source, it must decide to degrade inbound traffic multiple times in a row, reducing the likelihood that a legitimate flow is punished by accident. Our belief is that *Guarded* is a considerably stronger model for these reasons, and its successes offer strong rationale to consider the best schemes for efficient information sharing. Paradoxically, *Instant* generally achieves the best performance for UDP traffic yet actively suffers when trained as a single learner—this may occur due to a roughly even spread of values between disparate actions, due to shared characteristics between legitimate and malicious flows.

Although we have improved upon Marl in both identified problem cases, the improvements are not quite on the order we'd expect for UDP traffic. The most likely explanation is that agents are converging to, and becoming stuck in, locally optimal (but globally sub-optimal) policies. The increased state space size makes this a more likely occurrence, as does the unclear effect of hyperparameters (α, γ) as we scale up the state space. We suspect that these difficulties may be exacerbated by the competitive nature of learning that these models embody: agents are learning action values for multiple features simultaneously, taking many actions at once (making it harder to observe the true value of each action), and controlling shared global state. Although our design does take steps to counteract such effects, these mitigations may not be enough. Moreover, benign UDP traffic shares many characteristics with attack traffic, suggesting that more training samples or some unknown feature might aid control, or that it may be worthwhile to extensively pre-train agents non-competitively on each feature using individual flows.

Most importantly, what we wish to impart is the knowledge that while the models and techniques we present here are a significant improvement over past RL-based work, this strand still trails behind existing (exact) DDoS flow detection mechanisms where TCP traffic is concerned. The ability to better protect VoIP traffic when compared against one of these approaches is a curious observation, which suggests that other (exact) protocol-agnostic approaches may carry hidden assumptions and is a promising direction for future investigation. Similar traffic makes up a significant fraction of network load today (18–27 %). Although we have conducted work to map the territory, there are still more advancements to be made before RL-based DDoS defence is truly competitive. The benefits we have at present are, however, substantial. What we offer above many of the approaches we discuss in section IX are potentially more flexible deployments, low-overhead and fixed-cost decision-making, without requiring active measurement or the network resources and capabilities that the most effective techniques rely upon. Moreover, our decision making processes are entirely agnostic of the protocol or content of traffic, offering future-proofing against the introduction of new transports.

Security concerns and vulnerability: Can an agent be flooded with new flows to reduce their ability to make decisions? One of the risks introduced by our policy update strategy is that so much work can be queued up that an agent is never able to act on some attack flows. The natural solution is to impose an upper bound on the amount of action computations/policy updates that can be performed before a work list is discarded completely. This removes the guarantee that all flows will be visited fairly often, but if updates occur regularly then this random sampling may be sufficient to achieve good performance.

Can an attack on the controller can impact our approach? This question hinges upon whether the deployment environment is a traditional network or is fully SDN-enabled—each agent is, in a sense, *a* controller alongside the network's controller. In a traditional network, only the agents act as controllers, but since they periodically request per-flow data (rather than continuously receiving it) no amount of flows generates more requests or messages to the agent. More work is generated, but we discuss how to handle this safely above. Accordingly, agents can never be stalled by request volume: their only remote communication (load measurements) comes from trusted nodes, is highly periodic, and has constant size. The same logic holds for a fully software-defined network. Recalling that we do not employ the network's controller to install filtering rules on edge switches, an agent's ability to act is unimpeded. Thus, the controller is made no more vulnerable than in any other SDN. The only necessary change for such a scenario is that a load measurement which has not been updated (due to a timeout or missed deadline) should be set at $R_t = -1$.

Machine learning algorithms have earned a reputation for eluding human interpretation, while being vulnerable to evasion and poisoning. Given the security risks associated with introducing such techniques, it is natural to be concerned with the interpretability of the models we have proposed. With the exception of global state, the tile coding parameters we make use of ensure that the set of outputs for each feature we add is relatively enumerable: for n tilings and c tiles per dimension there are $nc^{\dim f}$ individual action value vectors per feature f (48 for the new features we introduce, 10 368 for global state), though considerably more combinations thereof ($c^{n \cdot \dim f}$). Furthermore, system state which is dependent on many signals drawn from across a wide network (such as our global state) is difficult to exert precise control over. These signals' topological separation, in concert with their burstiness and unpredictability, may have substantial effects on an attacker's capabilities.

Real-world Deployment: Currently, we assume that switches support an extension to OpenFlow to enable remotely installable packet-drop rules, either by running a modified version of OVS on commodity hardware at these locations or through custom firmware for egress switches. Similar functionality could be employed by making use of OpenFlow's meter rules.

Where overheads are concerned, the state space sizes guarantee that an *Instant* agent's policy remains under 520 KiB, although in practice our sparse representation typically leads to far smaller policies: ~17.8 KiB from our experiments. *Guarded* policies are 30 % of this size. As we have described earlier, action updates require a constant number of floating point operations—160 floating point additions and 32 multiplications per update of $\boldsymbol{w}$ with per-tile updates, above the 160 additions required to choose an action. The vast majority of these operations can be vectorised trivially, if such hardware is present. Action computation for *Guarded* agents is cheaper still, requiring only 48 additions per action. Beyond this, we require that egress switches are capable of co-hosting an agent (i.e., through *network function virtualisation*), with the necessary hardware to support this. We believe that it may be possible to implement similar behaviour on standard commodity switches through application of *programmable data planes* [37].

Gathering and transmission of load/flow statistics would be difficult to perform as often as an emulated environment allows, without inadvertently affecting host traffic. However, the measurements acquired in such a scenario are likely to be less noisy (by being collected over longer periods of time), which could aid training. The main bottlenecks are likely in forwarding the load measurements from various aggregation points (which can be made more efficient through multicast) and in running some estimator $\mathrm{g}(\cdot)$ to condition the reward function. We expect that agents will be able to share policies for all features, which may help to offset the reduced rate of incoming experience. Regardless, it will take longer to achieve enough state-state transitions to converge on a good policy.

One limit of SDN-capable hardware is that OpenFlow rules occupy 6× the space of standard rules—commercial switches only have TCAM space for 2–20 k rules [38]. Our approach consumes a rule for each active flow (the host density), and by the end of an experiment a switch can accrue around 900 rules. While we use a default fallback action to maintain connectivity, eviction of high-value decisions which filter high-bandwidth attackers poses a significant risk. Given that most flows are small (with the majority of bytes coming from a few "heavy-hitters") [39], it may suffice to only apply RL-based analysis to larger flows. OpenFlow rules have an *importance*, controlling which rules may be evicted by a new entry (preventing entries from evicting those with higher importance). If an agent is to act on all flows, a solution is to assign an importance of 0 to mice flows, 1 to elephant flows, and 2 to total filtering (leaving agents to time out and remove elephant flow rules to prevent bloat). Given the high churn and prevalence of mice flows, eviction here is most likely to affect flows which are complete. In both cases, extra rules can be made available by upgrading rules which completely filter a flow into upstream blackholing (as in collaborative approaches [40]), having the agent remove this rule once blackholing is active.

IX. Related Work

DDoS Prevention: Braga *et al.* [41] examine the detection of flooding DDoS attacks through *self-organising maps*, using SDN to gather statistics effectively. Many of their features aren't overly relevant, as their focus is not active defence or discovering *which* hosts contribute to an attack. The closest available approach within this field is that of Malialis and Kudenko [5] (whom we have positioned our work against), and their contribution in applying RL to the task of intrusion prevention is significant: their work helps to show the viability of live, adaptive, feedback-loop-like control of the network to detect and prevent DDoS attacks. They create a tree overlay topology (subdivided into teams), where each agent applies packet drop to *all* flows inbound to a protected server. Our results show that their technique underperforms at high host density and when congestion-aware traffic dominates—that their results do not demonstrate this suggests an evaluation driven purely by traces (rather than live application dynamics).

SPIFFY [7] aims to remedy transit-link attacks by observing how flows from each source respond to a sudden increase in available bandwidth. Kang *et al.* realise that bots participating in an attack are often unable to match this bandwidth expansion (having already saturated the capacity of their outbound links), while legitimate flows typically speed up to match the new fair-share rate. A weakness of their approach is that computing a route to measure bandwidth expansion on real networks can be costly (up to 14 s for the Cogent topology), and that the low expansion factors in real network can require more "rounds" of filtering. By contrast, our approach takes a constant time to compute an action for a flow regardless of topology size. Their assumptions about traffic response to such bandwidth expansion do not hold for constant bitrate flows (e.g., VoIP) and may not extend to HTTP DASH flows, both of which make up a sizeable proportion of network traffic.

Athena [15] is a generalised SDN framework for intrusion detection, but has shown the use of a *k-nearest neighbours* classifier to detect individual attack flows. Although heavy-weight (and proven to be effective compared with Braga *et al.* [41]), their comparison against SPIFFY lacks the quantitative evidence required to understand how the system compares. Smith and Schuchard [42] use AS-level routing to tackle both transit-link and flooding-based attacks. This view is taken due to the perceived cost of per-stream classification and inherent sensitivity to adversarial examples. The approach is creative, relying upon BGP *fraudulent route reverse poisoning* to preserve traffic to a target AS, but unlike SPIFFY the approach doesn't actually *remove* the congestion. Because of this, flooding-based attacks aren't fully alleviated.

RL in Networks: Earnest, well-considered application of RL towards the challenge of intrusion prevention has seen comparatively little examination. Past work treats the paradigm as a traditional classifier for anomaly detection [43] and DDoS prevention [44]. Given that the main strengths of RL techniques are the ability to control ongoing interaction and adapt by observing the concrete effects of actions, such works don't apply the rich literature on the subject to its fullest potential.

For categorising how RL fits into solving problems, we label works as direct- or indirect-control RL. A *direct-control* RL problem is one where the RL agent(s) learn optimal control over a set of actions as the *primary* defence or decision-maker—requiring measurements, reward functions and action sets tailored for this purpose. To date, the best-fitting example we have encountered is that of Malialis and Kudenko [5]. An *indirect-control* RL problem is one where agents act in service to *another technique* responsible for decision-making, optimising or generalising aspects of its operation beyond that of hand-coded heuristics. A past example includes learning when best to share knowledge between *hidden Markov model* anomaly detectors [45]. This work is weakened by its reliance on the problematic 'DARPA99' dataset [2], but the idea itself is well-treated. Outside of intrusion detection, there has been growing interest in the use of RL in data-driven networking, such as for intra-AS route optimisation [46] and resource-constrained process allocation [47]. Mao *et al.* [48] employ client-side observations of network state and video performance with RL to optimise bitrate selection for multimedia streaming. *AuTO* [30] employs deep RL to perform traffic optimisation. Crucially, they find that the vast majority of flows are short-lived, requiring effective decisions in less than a millisecond. To overcome the high latency of action computation via a neural network, two agents are trained, handling aspects of short and long flows respectively. The first learns to optimise the flow size thresholds to demarcate long and short flows; these short flows are routed by ECMP. The second agent makes bespoke decisions about routing, prioritisation etc. for each of the remaining long flows.

X. Conclusions and Future Work

Through this paper, we have discussed reinforcement learning and its relevance to network intrusion prevention. We believe the potential to learn feedback loop-like control online and against non-stationarity makes it particularly suited to the problems endemic to the field. We identified weaknesses in past work, recommending an RL agent which acts per flow, and have outlined the algorithmic and engineering choices needed to make its deployment feasible. Supporting this, we've presented an in-depth examination of our feature space, offering quantitative and qualitative justification for our choices. Our evaluation shows that our new agent designs considerably advance the state-of-the-art in RL-based DDoS prevention, with *Guarded* agents showing the most promise for future evaluation.

The most direct improvements to be made lie in the correct protection of legitimate UDP traffic, which our agent designs have difficulty safeguarding. Outside of this, there is scope to test these new techniques against link-flooding attacks in large-scale topologies using reward functions such as eq. (5). Simulation is the most likely avenue for such evaluation.

The remaining weaknesses invite many improvements worth investigation. A problem we raised (without a clear solution) was the design of reward functions which do not rely upon heuristic estimates or a priori knowledge of benign traffic content. If true online learning is desired (i.e., coping with a non-stationary environment), then such reward functions are sorely needed. While $\mathrm{load}_t^{\updownarrow}(\cdot)$ is likely to be a good candidate for many deployments, we believe that finding an effective metric derived from the individual statistics we suggested serves as an interesting research problem.

Given that one of the advantages of RL methods is the ability to handle non-stationary problems, it is important to propose and test sensible simulations or captures of evolving networks. While it is known that DDoS attack strategies evolve in real time [49], evaluation is difficult at present since no works detail what patterns such evolution might take. Regardless, these scenarios present ideal circumstances to apply adaptive exploration [50], changepoint detection, or intelligent sampling methods to judge which flows are most worthy of consideration. For estimating *when* to explore, we believe that the intersection of signal processing and RL is as-yet unexplored.

Effective real-world deployment of RL-based defences cannot assume that switches in a network will support a custom version of OVS or other arbitrary software, introducing the question of whether agent training, execution and distribution may be possible when using *programmable data planes* [37]. We also expect it will be fruitful to look into *how* agents may share knowledge with one another.

Although we believe that the security landscape for classical RL models is not *identical* to that of neural-network based approaches (particularly with such noisy, volatile, and hard-to-control data), there is still immense value in determining the exact capabilities of a sufficiently powerful adversary as the risk of external control still exists. In particular, we believe that poisoning attacks and evasion attacks merit close consideration.

We hope it is clear that reinforcement learning holds promise and can inspire further innovation. It allows us to offer distinct advantages above existing works, such as protocol-agnostic DDoS flow detection, flexible deployment, and automatically learned low-overhead decision-making—without requiring many of the network resources or capabilities that other techniques rely upon. It's hoped that more research in this direction will open the door to works which *respect the complexity of the network*; evolving topologies, natural change in traffic and protocol distributions, and the mutation of attacks.

Acknowledgements

The authors would like to thank Colin Perkins, Mircea Iordache, Qianru Zhou, Charles Rutherford, Marco Cook and Cristian Urlea for their advice and technical assistance, and their anonymous reviewers. This work has been supported in part by the UK Engineering and Physical Sciences Research Council [grants EP/N509668/1, EP/N033957/1 and EP/P004024/1] and European Cooperation in Science and Technology (COST) Action CA15127: RECODIS—Resilient communication and services.

References

[1] M. H. Bhuyan *et al.*, 'Network anomaly detection: Methods, systems and tools', *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[2] R. Sommer and V. Paxson, 'Outside the closed world: On using machine learning for network intrusion detection', in *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, 2010, pp. 305–316.

[3] S. Axelsson, 'The base-rate fallacy and its implications for the difficulty of intrusion detection', in *CCS '99, Proceedings of the 6th ACM Conference on Computer and Communications Security, Singapore, November 1-4, 1999.*, 1999, pp. 1–7.

[4] W. E. Leland *et al.*, 'On the self-similar nature of ethernet traffic', *Computer Communication Review*, vol. 25, no. 1, pp. 202–213, 1995.

[5] K. Malialis and D. Kudenko, 'Distributed response to network intrusions using multiagent reinforcement learning', *Eng. Appl. of AI*, vol. 41, pp. 270–284, 2015.

[6] A. Langley *et al.*, 'The QUIC transport protocol: Design and internet-scale deployment', in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, 2017, pp. 183–196.

[7] M. S. Kang *et al.*, 'SPIFFY: inducing cost-detectability tradeoffs for persistent link-flooding attacks', in *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016.

[8] M. Jonker *et al.*, 'Millions of targets under attack: A macroscopic characterization of the dos ecosystem', in *Proceedings of the 2017 Internet Measurement Conference, IMC 2017, London, United Kingdom, November 1-3, 2017*, 2017, pp. 100–113.

[9] M. Antonakakis *et al.*, 'Understanding the mirai botnet', in *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, 2017, pp. 1093–1110.

[10] C. Rossow, 'Amplification hell: Revisiting network protocols for ddos abuse', in *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.

[11] M. Kührer *et al.*, 'Exit from hell? reducing the impact of amplification ddos attacks', in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, 2014, pp. 111–125.

[12] M. S. Kang *et al.*, 'The crossfire attack', in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, 2013, pp. 127–141.

[13] A. Studer and A. Perrig, 'The coremelt attack', in *Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings*, vol. 5789, 2009, pp. 37–52.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, Nov. 2018.

[15] S. Lee *et al.*, 'Athena: A framework for scalable anomaly detection in software-defined networks', in *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017, Denver, CO, USA, June 26-29, 2017*, 2017, pp. 249–260.

[16] N. Papernot *et al.*, 'The limitations of deep learning in adversarial settings', in *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, 2016, pp. 372–387.

[17] N. Papernot *et al.*, 'Sok: Security and privacy in machine learning', in *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, 2018, pp. 399–414.

[18] S. H. Huang *et al.*, 'Adversarial attacks on neural network policies', *CoRR*, vol. abs/1702.02284, 2017. arXiv: 1702.02284.

[19] N. Carlini and D. A. Wagner, 'Towards evaluating the robustness of neural networks', in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, 2017, pp. 39–57.

[20] F. Tramèr *et al.*, 'Stealing machine learning models via prediction apis', in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, 2016, pp. 601–618.

[21] Y. Han *et al.*, 'Adversarial reinforcement learning under partial observability in software-defined networking', *CoRR*, vol. abs/1902.09062, 2019. arXiv: 1902.09062.

[22] A. H. M. Jakaria *et al.*, 'A requirement-oriented design of nfv topology by formal synthesis', *IEEE Transactions on Network and Service Management*, pp. 1–1, 2019.

[23] The Linux Foundation. (2018). Open vswitch, [Online]. Available: https://www.openvswitch.org/ (visited on 02/05/2018).

[24] R. Dobbins *et al.*, 'Use cases for DDoS Open Threat Signaling', Internet Engineering Task Force, Internet-Draft draft-ietf-dots-use-cases-17, Jan. 2019, Work in Progress, 14 pp.

[25] R. Mahajan *et al.*, 'Controlling high bandwidth aggregates in the network', *Computer Communication Review*, vol. 32, no. 3, pp. 62–73, 2002.

[26] M. Mathis *et al.*, 'The macroscopic behavior of the TCP congestion avoidance algorithm', *Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[27] I. Rhee *et al.*, *CUBIC for Fast Long-Distance Networks*, RFC 8312, Feb. 2018. DOI: 10.17487/RFC8312. [Online]. Available: https://rfc-editor.org/rfc/rfc8312.txt.

[28] CAIDA. (2018). The CAIDA UCSD anonymized internet traces – 2018, [Online]. Available: http://www.caida.org/data/passive/passive_dataset.xml (visited on 11/05/2019).

[29] J. Rüth *et al.*, 'A first look at QUIC in the wild', in *Passive and Active Measurement - 19th International Conference, PAM 2018, Berlin, Germany, March 26-27, 2018, Proceedings*, vol. 10771, 2018, pp. 255–268.

[30] L. Chen *et al.*, 'Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization', in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018*, 2018, pp. 191–205.

[31] M. P. Collins *et al.*, 'Using uncleanliness to predict future botnet addresses', in *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference, IMC 2007, San Diego, California, USA, October 24-26, 2007*, 2007, pp. 93–104.

[32] L. Krämer *et al.*, 'Amppot: Monitoring and defending against amplification ddos attacks', in *Research in Attacks, Intrusions, and Defenses - 18th International Symposium, RAID 2015, Kyoto, Japan, November 2-4, 2015, Proceedings*, vol. 9404, 2015, pp. 615–636.

[33] Ryu. (2018). Ryu SDN framework, [Online]. Available: https://osrg.github.io/ryu/ (visited on 12/10/2018).

[34] (2014). Cisco event response: Network time protocol amplification distributed denial of service attacks, [Online]. Available: https://www.cisco.com/c/en/us/about/security-center/event-response/network-time-protocol-amplification-ddos.html (visited on 23/09/2019).

[35] D. K. Y. Yau *et al.*, 'Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles', *IEEE/ACM Trans. Netw.*, vol. 13, no. 1, pp. 29–42, 2005.

[36] M. Al-Fares *et al.*, 'A scalable, commodity data center network architecture', in *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Seattle, WA, USA, August 17-22, 2008*, 2008, pp. 63–74.

[37] S. Jouet and D. P. Pezaros, 'Bpfabric: Data plane programmability for software defined networks', in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2017, Beijing, China, May 18-19, 2017*, 2017, pp. 38–48.

[38] X. N. Nguyen *et al.*, 'Rules placement problem in openflow networks: A survey', *IEEE Communications Surveys and Tutorials*, vol. 18, no. 2, pp. 1273–1286, 2016.

[39] R. Pan *et al.*, 'Approximate fairness through differential dropping', *Computer Communication Review*, vol. 33, no. 2, pp. 23–39, 2003.

[40] S. Ramanathan *et al.*, 'SENSS against volumetric ddos attacks', in *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018*, 2018, pp. 266–277.

[41] R. Braga *et al.*, 'Lightweight ddos flooding attack detection using nox/openflow', in *The 35th Annual IEEE Conference on Local Computer Networks, LCN 2010, 10-14 October 2010, Denver, Colorado, USA, Proceedings*, 2010, pp. 408–415.

[42] J. M. Smith and M. Schuchard, 'Routing around congestion: Defeating ddos attacks and adverse network conditions via reactive BGP routing', in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, 2018, pp. 599–617.

[43] S. Shamshirband *et al.*, 'Anomaly detection using fuzzy q-learning algorithm', *Acta Polytechnica Hungarica*, vol. 11, no. 8, pp. 5–28, 2014.

[44] A. Servin and D. Kudenko, 'Multi-agent reinforcement learning for intrusion detection: A case study and evaluation', in *Multiagent System Technologies, 6th German Conference, MATES 2008, Kaiserslautern, Germany, September 23-26, 2008. Proceedings*, vol. 5244, 2008, pp. 159–170.

[45] X. Xu *et al.*, 'Defending ddos attacks using hidden markov models and cooperative reinforcement learning', in *Intelligence and Security Informatics, Pacific Asia Workshop, PAISI 2007, Chengdu, China, April 11-12, 2007, Proceedings*, vol. 4430, 2007, pp. 196–207.

[46] A. Valadarsky *et al.*, 'Learning to route', in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks, Palo Alto, CA, USA, HotNets 2017, November 30 - December 01, 2017*, 2017, pp. 185–191.

[47] H. Mao *et al.*, 'Resource management with deep reinforcement learning', in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks, HotNets 2016, Atlanta, GA, USA, November 9-10, 2016*, 2016, pp. 50–56.

[48] H. Mao *et al.*, 'Neural adaptive video streaming with pensieve', in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, 2017, pp. 197–210.

[49] M. S. Kang *et al.*, 'Defending against evolving ddos attacks: A case study using link flooding incidents', in *Security Protocols XXIV - 24th International Workshop, Brno, Czech Republic, April 7-8, 2016, Revised Selected Papers*, vol. 10368, 2016, pp. 47–57.

[50] M. Tokic and G. Palm, 'Gradient algorithms for exploration/exploitation trade-offs: Global and local variants', in *Artificial Neural Networks in Pattern Recognition - 5th INNS IAPR TC 3 GIRPR Workshop, ANNPR 2012, Trento, Italy, September 17-19, 2012. Proceedings*, vol. 7477, 2012, pp. 60–71.

Kyle A. Simpson received the MSci degree in computing science from the University of Glasgow in 2017. He is currently a PhD student within the Networked Systems Research Laboratory at the School of Computing Science, University of Glasgow. His research focusses on the use of machine learning and reinforcement learning techniques in cybersecurity and network management, with a core interest in evolving problems and defences.

Simon Rogers is a senior lecturer in the School of Computing Science, University of Glasgow. He received his PhD from the department of Engineering mathematics at the University of Bristol in 2005 and has been a permanent member of academic staff at the University of Glasgow since 2009. His work focuses on the development of machine learning and statistical methods for the analysis of complex data, particularly within the field of computational biology.

Dimitrios P. Pezaros (S'01–M'04–SM'14) received the B.Sc. and Ph.D. degrees in Computer Science from Lancaster University. He is currently (full) Professor and the founding director of the Networked Systems Research Laboratory at the School of Computing Science, University of Glasgow. He is also a visiting Professor at the University of Athens, Department of Informatics and Telecommunications. Professor Pezaros has published widely in the areas of computer communications, network and service management, and resilience of future networked infrastructures, and has received significant funding for his research from public funding agencies and the industry. He is a Chartered Engineer, and a senior member of the IEEE and the ACM.