Improving Direct-Control Reinforcement Learning for Network Intrusion Prevention (WIP)

Kyle A. Simpson
FelixMcFelix `https://mcfelix.me`

3rd September, 2018

University of Glasgow

Introduction

- Network IDS/IPS backed by machine learning haven't taken off as hoped—particularly anomaly-based work.
- Detection problem tricky in this domain:
    - Evolving: usage shifts, new protocols, new applications.
    - Burstiness, seasonal variation.
    - Need for correctness, almost no false-positive tolerance.
    - Labelling issues.

Introduction: Part II

- Classes of problem like flooding-based DDoS attacks manifest as a service degradation.
    - Can these be controlled via feedback loop?
    - “Overcome” the difficulties of the detection problem by monitoring and adapting to *performance characteristics and consequences* in real-time
- Goal: augment signature-based approaches to provide a last line of defence.

RL: The Main Idea™

- Underlying theory: systems as (discrete-time) Markov Decision Processes—states, actions, rewards and transition probabilities.
  - I.e., choosing action a_t from a policy in state s_t, $a_t \sim \pi(s_t)$, induces the next state s_{t+1} and an associated reward r_{t+1}.
  - Generalises to value $Q(s, a)$—how much reward can we *eventually* expect from choosing each action currently available?
- Goal: train an agent to make optimal decisions based on observed state.
  - Formally, learn a policy to maximise the expected discounted reward[1].

[1] Sutton and Barto, *Reinforcement Learning: An Introduction*.

RL: The Main Benefits™

- We can learn the optimal policy without modelling the world ourselves.
- Formulation allows learning adaptively and online, so long as a reward signal is available.
- Variation in available algorithms, update mechanisms, function approximations, dependence on value functions, action selection, exploration…
    - Orthogonal concerns, allowing tunable algorithm design.

Where has RL succeeded in networks?

- Data-driven networking. Effectively applied to intra-domain routing[2], task allocation[3], traffic optimisation[4] and more, each with general and domain-specific insights.
- In anomaly detection? Optimising information sharing in distributed statistical model training[5].

---

[2] Valadarsky *et al.*, 'Learning to Route'.
[3] Mao *et al.*, 'Resource Management with Deep Reinforcement Learning'.
[4] Chen *et al.*, 'AuTO: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization'.
[5] Xu, Sun and Huang, 'Defending DDoS Attacks Using Hidden Markov Models and Cooperative Reinforcement Learning'.

Multiagent RL for DDoS prevention

- Reimplementing (and poking holes in) MARL[6].
- Network model
  - Hosts have a fixed probability of being benign/malicious.
  - n hosts per learner, i learners to a team, j teams, one server.
  - Per-team rewards: coordinated team learning.
  - Action: (per-timestep) choose p, s.t. each learner drops p% of external traffic.
- Implemented in mininet with Ryu controller, traffic generated by replaying traces.
  - Packet content unimportant—only need accurate load stats/queuing behaviour.
  - Alternate model featuring live HTTP traffic.

[6] Malialis and Kudenko, 'Distributed response to network intrusions using multiagent reinforcement learning'.

Multiagent RL for DDoS prevention

- Algorithm: Semi-gradient Sarsa, linear fn approx, ϵ-greedy selection.
- Actions: Drop [0, 10, ... 90]% upstream traffic.
- State: load vectors of agent and parents ($\mathbb{R}^4$) $\rightarrow$ tile-coded (fixed-weight binary vector).
- Rewards: −1 if $load > U_s$, else fraction of surviving legit traffic.

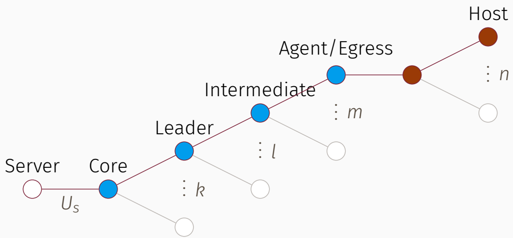


Figure 1: *Network topology diagram. Red nodes are external, blue nodes feature in the state vector. Any packet drop occurs when forwarding packets from an egress switch to its parent (intermediate) switch.*

The algorithm?

A little terse, but the main update rule is:

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \nabla_{\mathbf{w}} \hat{\mathsf{q}}(S_t, A_t, \mathbf{w}_t) \tag{1b}$$

with linear function approximation:

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

$$\nabla_{\mathbf{w}} \hat{\mathsf{q}}(s, a, \mathbf{w}) = \mathbf{x}(s, a) \tag{1d}$$

The case for finer granularity

- Learner/host ratio (action/host ratio) affects host QoS.
- Reduced service guarantees by nature of *pushback* model.
  - Worse with good-faith TCP congestion avoidance.

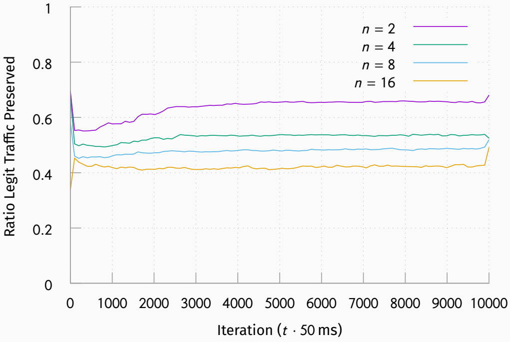
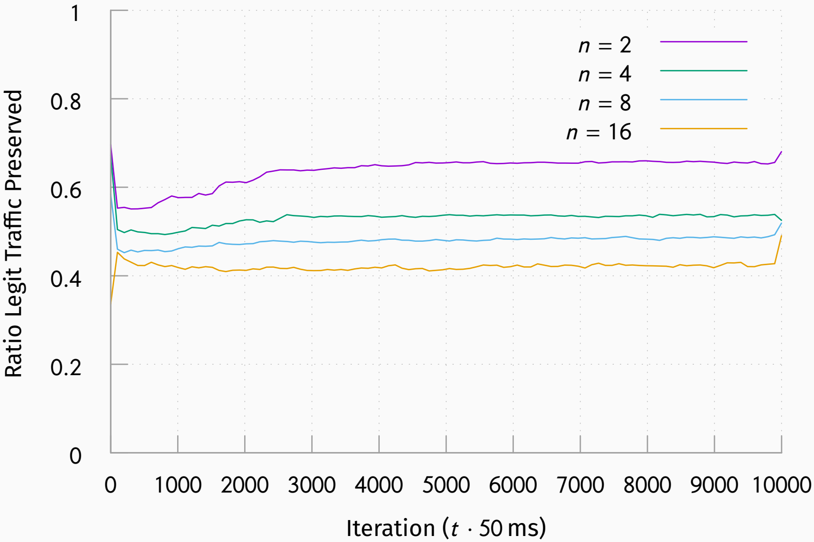


Figure 2: *Service quality decreases as actions become less granular (affecting n hosts at once).*

The case for finer granularity



On collateral damage

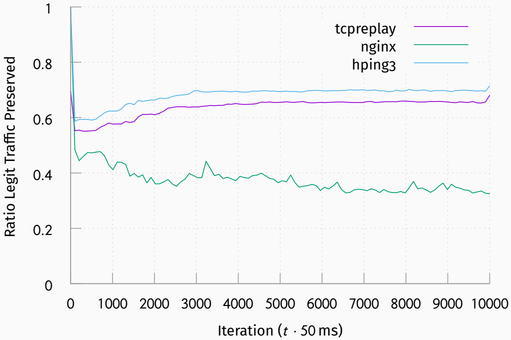
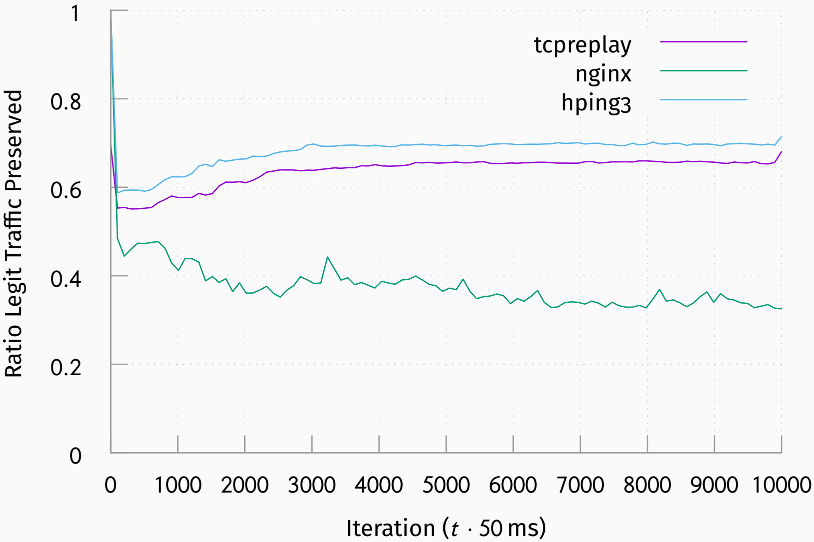


Figure 3: *Explicit UDP traffic matches replayed traces (hping3 vs tcpreplay). TCP traffic (nginx) is severely punished.*

- Is the simulation environment of past work complete?
- No. It's reliant on a numerical simulator, derived from observations of *traces*.
- UDP benign traffic similar trend to replayed TCP traces, which matches the original results.
- Live TCP responds very badly.

On collateral damage



And the replication reveals...

- Network topology has no basis in reality—admitted by its *own* source work[7].
- Action granularity causes more collateral damage than we'd like...
- ...and the picture is worse still for legitimate TCP flows.
- Reward function needs a priori knowledge/reliable estimates to learn online.
- But on the plus side, action computation is fast: 80–100 µs.

[7] Mahajan *et al.*, 'Controlling high bandwidth aggregates in the network'.

How can we use these observations? (The Immediate Future)

- Why not take actions on a per-flow basis?
  - Solves the granularity issues by construction.
  - Allows different treatment by flow features (i.e., protocol).
- Need to rethink state space: more costly computation, but we have room to work in.
  - We need any additions to be justified beyond just “more data”, since changes affect training time and execution time.
- How do we select flows to act upon?

A candidate state space

Global State The existing state space.

Local State (At least) the following:

- **Src/dst IP, Port, Protocol**—identification.
- **Flow size, duration, rate**—standard features.
- **Last action taken**—encode belief/forgiveness.
- **Correspondence ratio**—explicitly capture asymmetry.
- **Δrate**—model how a flow's behaviour changes post-action.
- Other features?

And then finding a suitable discretisation...

The Near Future

- Flow selection strategies (guided action calculation).
- Reward functions without dependence on ahead-of-time knowledge.
  - I.e., for certain distributions of communication we might want to maximise link utilisation in both directions.
- Deriving normal model behaviour from traces.
  - We only need to simulate specific behaviour to test these enhancements, but that can become more representative.
- Investigate other RL algorithms.
  - "Deep learning" probably not feasible.
  - TD(λ), actor-critic methods.

The Far Future

- Other problems.
  - New action spaces, careful consideration.
- Adversarial capabilities—evasion and poisoning attacks.
- Knowledge-sharing between agents: cost-modelling and optimisation.
- Test deployments in real networks.

Conclusion

We've looked at...

A quick introduction to RL, and its importance to future networks for optimisation and control of certain classes of problem.

A recent 'direct control' approach to intrusion prevention, and its significant weaknesses.

Intended improvements specifically targeting these weaknesses.

Questions?

References i

- Chen, Li, Justinas Lingys, Kai Chen and Feng Liu. '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*. Ed. by Sergey Gorinsky and János Tapolcai. ACM, 2018, pp. 191–205. DOI: `10.1145/3230543.3230551`. URL: `http://doi.acm.org/10.1145/3230543.3230551`.
- Mahajan, Ratul, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson and Scott Shenker. 'Controlling high bandwidth aggregates in the network'. In: *Computer Communication Review* 32.3 (2002), pp. 62–73. DOI: `10.1145/571697.571724`. URL: `http://doi.acm.org/10.1145/571697.571724`.

References ii

Malialis, Kleanthis and Daniel Kudenko. ‘Distributed response to network intrusions using multiagent reinforcement learning’. In: *Eng. Appl. of AI* 41 (2015), pp. 270–284. DOI: 10.1016/j.engappai.2015.01.013. URL: https://doi.org/10.1016/j.engappai.2015.01.013.

Mao, Hongzi, Mohammad Alizadeh, Ishai Menache and Srikanth Kandula. ‘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*. Ed. by Bryan Ford, Alex C. Snoeren and Ellen W. Zegura. ACM, 2016, pp. 50–56. ISBN: 978-1-4503-4661-0. DOI: 10.1145/3005745.3005750. URL: http://doi.acm.org/10.1145/3005745.3005750.

- Sutton, Richard S. and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. In progress. MIT Press, 2018. ISBN: 0262039249. URL: `http://incompleteideas.net/book/the-book-2nd.html`.
- Valadarsky, Asaf, Michael Schapira, Dafna Shahaf and Aviv Tamar. '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*. Ed. by Sujata Banerjee, Brad Karp and Michael Walfish. ACM, 2017, pp. 185–191. ISBN: 978-1-4503-5569-8. DOI: `10.1145/3152434.3152441`. URL: `http://doi.acm.org/10.1145/3152434.3152441`.

Xu, Xin, Yongqiang Sun and Zunguo Huang. '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*. Ed. by Christopher C. Yang, Daniel Dajun Zeng, Michael Chau, Kuiyu Chang, Qing Yang, Xueqi Cheng, Jue Wang, Fei-Yue Wang and Hsinchun Chen. Vol. 4430. Lecture Notes in Computer Science. Springer, 2007, pp. 196–207. ISBN: 978-3-540-71548-1. DOI: 10.1007/978-3-540-71549-8_17. URL: https://doi.org/10.1007/978-3-540-71549-8_17.