R2L: Routing with Reinforcement Learning

Joao Reis
Department of Electronic and Electrical Engineering
University College London
London, United Kingdom
Email: ucakjmq@ucl.ac.uk

Truong Khoa Phan
Department of Electronic and Electrical Engineering
University College London
London, United Kingdom
Email: t.phan@ucl.ac.uk

Morteza Kheirkhah
Department of Electronic and Electrical Engineering
University College London
London, United Kingdom
Email: m.kheirkhah@ucl.ac.uk

Fan Yang
Department of Electronic and Electrical Engineering
University College London
London, United Kingdom
Email: fan.yang.15@ucl.ac.uk

David Griffin
Department of Electronic and Electrical Engineering
University College London
London, United Kingdom
Email: d.griffin@ucl.ac.uk

Miguel Rocha
Centre of Biological Engineering
Department of Informatics
University of Minho
Braga, Portugal
Email: mrocha@di.uminho.pt

Miguel Rio
Department of Electronic and Electrical Engineering
University College London
London, United Kingdom
Email: miguel.rio@ucl.ac.uk

***Abstract*—In a packet network, the routes taken by traffic from sources to destinations can be determined according to predefined objectives. For example, a video stream application might benefit from a path where packet loss is reduced, whereas a low delay path might be better for an interactive online game. Assuming that the network conditions remain static and the defined objectives do not change, mathematical tools such as linear programming could be used to solve this routing problem. However, networks can be dynamic or the routing requirements may change. In that context, Reinforcement Learning (RL), which can learn to adapt in dynamic conditions and offers flexibility of behavior through the reward function, presents as a suitable tool to find good routing strategies. In this work, we train an RL agent, which we call R2L, to address the routing problem. The policy function used in R2L is a neural network and we use an evolution strategy algorithm to determine its weights and biases. Evolution strategies have shown to be able to have a performance on par with gradient based methods, while offering some advantages, such as being simple to parallelize. Finally, we tested how R2L optimized routing for different objectives and how it performed under dynamic network conditions.**

I. Introduction

A large portion of the traffic in the current Internet follows a one-size-fits-all solution where all packets to the same destination follow the same route independently of the application requirements. It is often the case that between a source and a destination there is more than one path available. These paths may differ in some aspects, such as delay or presence of noise. Furthermore, network resources may be shared among different flows of traffic. Consequently, for a network operator, determining the routes for traffic within its network can be seen as a joint optimization problem, which takes into account the traffic and network operator requirements. To illustrate, a network operator such as an Internet Service Provider is perhaps interested in reducing congestion in its network. Therefore, it tries to load balance the traffic by preferring less congested paths. On the other hand, looking at the individual flows, the ones from a delay-sensitive application (e.g. an online game) might be routed through a low delay path, whereas others from a bandwidth-sensitive application (such as video-on-demand streaming) may use a less noisy path.

Assuming the traffic demand and the network properties (e.g. topology) are known, a solution can be obtained by modeling a multi-commodity flow problem and using a mathematical programming tool to solve it [1]. However, this approach has some drawbacks. Firstly, it returns a solution for some given network conditions, but these conditions may change (e.g. a network link failure). Having to rerun the algorithm each time a change does occur can be an expensive operation. Secondly, it can be cumbersome to adapt to different routing objectives.

To address the first drawback, different approaches have been proposed. For example, in [2], the authors propose to optimize routing for multiple representative traffic demand profiles (e.g. day demand and evening demand). Another approach, taken by [3], is to precompute more than one path from source to destination and then change the proportion of traffic sent through each path according to the network conditions.

In some cases, however, the second drawback would still persist as these do not address the fact that the routing objective may be coupled with the model or the algorithm to solve it.

In this context, Reinforcement Learning (RL) can be a suitable tool for network routing. Indeed, routing can be seen as a task where sequential decisions are taken, in order to optimize performance according to a pre-defined objective function. An RL agent can then be used to determine these decisions in a way that aligns with the desired objective. A motivation for this approach is that it allows us to switch the routing objective by changing the reward function (and possibly the state representation) and retrain the model without major modifications to the workflow. Furthermore, it does not require to introduce any additional mechanism to address changes in the network or traffic. Instead, this can possibly be learned by the agent through interaction with a changing environment and then generalized from that.

Previous work on applications of RL to network routing has differences in the number of agents used, the action performed by the agent(s) and, the routing objective. Multi agent routing algorithms, such as Q-Routing [4], typically have each router learn a value function, whereas single agent ones train one agent for the whole network [5]. Algorithms also differ in the scope and actions taken by the RL agent. At one extreme, some acted on each packet in the network by deciding its next hop [6] [7]. On the other hand, others took a decision for the first packet of a flow and then kept the remaining ones on the same path [8] [9] [10]. Finally, some determine a routing strategy for the whole network given the traffic between each source-destination pair [11]. In terms of optimization, some have focused on directly minimizing the average delivery time of packets [12], while others minimize the maximum link utilization in the network, which in turn contributes to reduce congestion [11]. A third optimization objective seen in previous work is a utility function, typically a weighted sum of different components, which can include the average delivery time but also other metrics related to the Quality of Service (QoS), such as the number of packets delivered [10].

In this work we present R2L, an RL solution for routing that takes next hop decisions with a central single agent approach. Compared with previous work, R2L takes per packet decisions that determine the next hop like [4], [12], [6] or [7]. We believe that, relative to actions such as determining network-wide routing strategies [11] or choosing split ratios for pre-computed paths, [5] a per packet next hop approach may allow for more fine-grained decisions. Furthermore, R2L trains a single agent only. Compared with approaches that have one agent per router [4] [6] [7] [12], a centralized single agent has the advantage of having a global view of the network. A centralized approach, however, comes with certain challenges. One of them is ensuring that the controller has an updated view of the network. To avoid this, in R2L we assume instead a network view that is partially updated each time a router queries the controller for a decision and gives it its local network measurements.

The remaining of this work is structured as follows. In Section II we describe the network model used, explain what is routing and how it can be addressed with Reinforcement Learning. Next, in Section III, we introduce R2L by explaining the model, how it is trained, and general workflow. After that, in SectionIV, we present and discuss the results of the experiments done, and finally, in Section V, we conclude and list possible future directions.

II. Background

A. Network Model

We model a network as a directed graph $G(V, E)$ where each router i corresponds to a node $n_i \in V$ and each link connecting a router i to another j corresponds to an edge $l_{i,j} \in E$. Inside the network there are packets p that we want to route from a source node $src^p \in V$ to a destination node $dst^p \in V$. The path, or route, R^p taken by p is the sequence of nodes $(r_1^p, r_2^p ..., r_{|R^p|}^p), r \in V$ visited by p from source $src^p = r_1^p$ to destination $dst^p = r_{|R^p|}^p$. For the path to be valid, consecutive nodes $r_i^p, r_{i+1}^p \in R^p$ must be connected by an existent link $l_{r_i^p, r_{i+1}^p} \in E$. Furthermore, a link has an associated transmission rate, measured in bits per second, which determines the time it takes for a node to put a packet into the link. The link also has an associated propagation delay, measured in seconds, which is the time it takes for a bit to travel from one end of the link to the other one.

In a node n_i, for each of its outgoing neighbours $\mathrm{out}(n_i) = \{n_o \in V : l_{i,o} \in E\}$, there is a queue $q_{i,o}$, with capacity $\mathrm{cap}(q_{i,o})$, to where the packets are sent before being transmitted. If a packet is sent to a full output queue, then it is dropped and considered lost.

When a packet is at a node r_i^p, the following is done to send it to the next hop r_{i+1}^p:

1) r_i^p determines the packet's next hop out of its outgoing neighbours $\mathrm{out}(r_i)$.
2) The packet is sent to the corresponding output queue. If the queue is full then the packet is dropped. If the queue is not full but not empty then the packet stays there until it gets to the top of the queue.
3) Once the packet reaches the top of the queue, the node starts transmitting it into the link. The time it takes to complete this is dependent on the link $l_{r_i^p, r_{i+1}^p}$ transmission rate and the packet size.
4) The packet travels from the end of the link to the other for the time given by $l_{r_i^p, r_{i+1}^p}$ propagation delay.
5) The packet arrives to r_{i+1}^p. If r_{i+1}^p is the packet's destination dst^p, then the packet is consumed and is considered to have been delivered. Otherwise, the steps above are repeated.

B. Routing

This work focuses on the routing problem, illustrated by step 1 above. From the point of view of a router, we abstract routing as a black box where the inputs are the packet header and the node's local view of the network (e.g. queue sizes), and the output is the packet's next hop. Different routing protocols build this black-box in different ways and may have different inputs.

For example, in Shortest Path based routing, each link in the network is given a cost/weight and the traffic is routed through the valid path that minimizes the sum of individual link costs. A widely used routing protocol, OSPF [13], computes (in a decentralized manner) these shortest paths between source/destination pairs in the network. In practice, each router only needs to know the next hop and not the full path between itself and the destination of a packet. This information can, therefore, be stored in routing tables which act as the black-box mentioned above, taking the packet destination and returning the next hop.

In this family of protocols, the routing objectives are achieved by setting the appropriate weights. For example, when the objective is to reduce congestion, a commonly used heuristic is to set the weights to the inverse of the link transmission rate. An alternative provided by [14], also to reduce network congestion,

uses a local search heuristic to find weights that perform well given a Demand Matrix (DM) containing the traffic demand between each node in the network (in this problem, they show that finding the optimal weights is NP-Hard)

An alternative to Shortest Path based routing is to explicitly define the paths rather than weights. These paths can then be used with a protocol such as MPLS [15]. One possible way to determine the paths would be to model routing as Multi Commodity Flow problem and find the optimal solutions (for a given DM) using tools such as Linear Programming.

A challenge when optimizing routing in a network is accounting for changing network conditions, e.g. a link failing, and traffic patterns, e.g. a new traffic flow. In a Shortest Path setting, [2] proposes an algorithm that addresses this by finding weights to reduce network congestion for a set of DMs and a convex combination of them. A different approach, TeXCP [3] precomputes a set of possible paths between sources and destinations. The proportion of traffic sent through each path is then adjusted according to the current network conditions and traffic.

C. Reinforcement Learning Routing

Routing can be modeled as a Markovian Decision Process (MDP), where an agent trained with Reinforcement Learning takes the routing decisions by observing the network. A motivation for an RL approach is that it offers a framework where changes in the network conditions or traffic can be dealt implicitly by learning different patterns. Furthermore, adapting routing to different objectives can be done by retraining the model with a different reward function.

One of the design aspects in which existing works differ is the degree of centralization in the routing decisions. To illustrate this, in [12] they train a different agent for each node, which acts at the node level by observing the incoming packet's destination and deciding on their next hop. On the other hand, [5] uses a single agent that receives a set of connections and allocates, for each connection, the proportion of traffic sent among precomputed paths.

An advantage of a central controller is that it can make more informed decisions, as it has a global view of the network, potentially producing better results. However, ensuring that a controller keeps an updated state of the network is itself a challenge that may require a significant traffic overhead.

III. R2L

Packet routing in R2L works as follows. Each time a packet arrives at a router, the router queries a central controller. As inputs, it gives the packet and some local network measurements. The controller then gathers this information updates its view of the network, builds a state and, queries the RL agent for the next hop, which then relays to the router.

A. Model

a) State: The state built by the controller consists of two components, one determined from the routers local network measurements and another from the packet header, as well as an identifier for the router querying the controller and a flag for the final state.

The first component is obtained by caching and combining the local network measurements, e.g. queue sizes, of all the routers in the network. This information is collected and updated when a router queries the controller and gives it its local network measurements. With caching, we can provide the RL agent with a global vision (albeit possibly outdated) of the network without requiring every router in the network to send all its local network measurements each time one router queries the controller. The second component comes from the packet header. This should include, for example, the packet's source and destination.

b) Action: The agent decides the outgoing port to which the packet should be sent. We employ a stochastic policy, so the policy neural network output is used to parameterize a distribution. This provides a mechanism to execute stochastic routing strategies which could be useful, for example in traffic load balancing.

In this context, having a single agent for the whole network means that the policy network has to be able to output all the outgoing ports in the network. This has the issue of some actions being invalid. To address that, we only consider the policy output from the possible valid outgoing ports and use those to parameterize the action distribution.

c) Reward: The reward depends on the routing objective. For example, if we are interested in maximizing the number of packets delivered then we can give a reward of 1 for each packet delivered. Another possible objective could be to minimize the average delivery time, in this case, a negative reward of the average delivery time during the episode could be a good choice.

Some of the local measurements sent by the routers to the controller are used to compute the reward. Therefore, these local measurements should be consistent with the objectives set. For example, in the scenario where we want to maximize the number of packets delivered, each node would need to tell the controller how many packets it has consumed so far.

d) Policy Function: The policy function is a neural network with one hidden layer with 16 neurons and an output layer with the size equal to the number of outgoing ports in the network, as each output neuron has a corresponding outgoing port. The activation function of the hidden layer is a ReLU and no activation function is used for the output layer. As mentioned above, to determine a valid action we only consider the portion of the output corresponding to outgoing ports of the node querying. We then take the absolute values, normalize them so they add up to 1 and sample the action among them using their values as the probability of being picked.

B. Training

The training is not done with a gradient based approach, but instead we with an evolutionary algorithm. This procedure is commonly called Neuroevolution [16]. More precisely, we use an *evolution strategy* algorithm called Covariance Matrix Estimation Evolution Strategy (CMA-ES) [17].

In general, evolution strategies work by first sampling a set of candidate solutions from a parameterized search distribution. This set is referred to as a generation and each solution as an individual. Next, each individual is tested and assigned a fitness score. After that, the search distribution parameters are recalculated by taking into account a subset of individuals chosen according to their fitness score. The procedure is repeated until a termination criterion is met.

In particular, the search distribution in CMA-ES from which the individuals are sampled is a Multi Variate Gaussian distribution $\mathcal{N}(\mathbf{m}, \sigma^2\mathbf{C})$ and at each generation the parameters updated are the the mean $\mathbf{m}$, the step-size σ and the covariance matrix $\mathbf{C}$.

In Neuroevolution for Reinforcement Learning, each individual is a neural network, where a gene corresponds to a specific parameter. The fitness score for an individual is the sum of the rewards obtained by running an episode using it as the policy function.

One of the advantages stated in [18] for using black-box optimization methods, such as evolutionary algorithms, to find policies for an RL agent, stems from the fact that we only consider the aggregated value of the rewards and not each one individually. This has the consequence that the way rewards are distributed during an episode does not affect the algorithm. In this work, we are interested in training an agent that can optimize for different objective functions. Some of these, such as the number of packets delivered, can be seen as a sum of rewards that can be given during the episode. On the other hand, objectives that depend on averages, such as the average delivery time, are perhaps better modeled as sparse rewards that are only given at the end of an episode. We believe that an algorithm that is agnostic to this is beneficial for the flexibility we aim for.

Another motivation [18] given for using evolution strategies is that they are simple to parallelize since the fitness score of each individual can be computed independently. In our case, where the training time is mainly determined by the simulation duration (as opposed to other tasks, e.g. updating the model parameters), this can lead to a significant decrease in training time.

IV. Experiments and Results

To test R2L, we ran two sets of experiments with a network model and routing as described in Sections II-A and II-B, respectively. In the first set of experiments, with static network conditions, the aim was to understand if R2L was capable of delivering good routing strategies for different objective functions. In the second, we focused on how R2L performed when there was some event that changed some network conditions.

A. Static network conditions

In this experiment, we used the 16-node grid topology shown in Figure 1, inspired by the one used in [4] but smaller and with some links removed. The traffic consisted of two flows, one from node 1 to node 16 and the other from node 16 to node 1. All the packets had the same size, 100 bytes, and the interarrival time between each packet followed an exponential distribution with a mean of 1 second. Furthermore, the links had a transmission rate of 500 bits per second. Regarding the propagation delay, the dashed links in Figure 1 had 0.5 seconds and the remaining ones 5 seconds. The goal was to have a "low propagation delay" path with a relatively high number of intermediate nodes such that a simple shortest path approach would not be optimal when optimizing for delay.

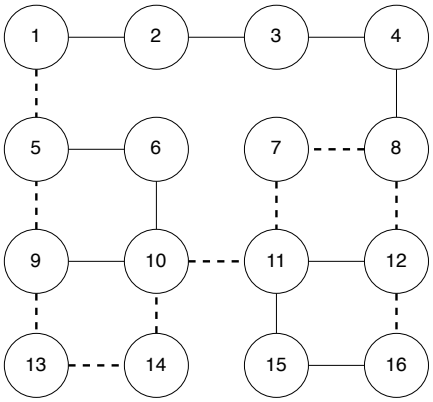


Fig. 1: 16-node topology. Dashed links have a 0.5s propagation delay.

As benchmarks, we used two routing strategies, Shortest Path and Load Balancing. The Shortest Path routed all the traffic from node 1 to node 16 through the path of nodes $(1, 5, 9, 10, 11, 12, 16)$ and the traffic from 16 to 1 through the reverse path. The Load Balancing strategy sent traffic from node 1 to 16 along two possible paths $(1, 5, 9, 10, 11, 15, 16)$ and $(1, 2, 3, 4, 8, 12, 16)$ and the traffic from 16 to 1 through the reverse paths. In cases where the packet interarrival rate is higher than a link's transmission rate, the advantage of balancing the traffic across more links is that it can potentially reduce the queuing delay and the number of packets being dropped.

We trained three R2L agents, each with a different reward function:

- **Throughput oriented:** A reward of 1 for each packet delivered.
- **Delay oriented:** A positive reward of 0.2 for each packet delivered and a negative reward of 0.8 times the average delivery time (the latter is only given once, at the end of the episode).
- **Mixed:** The reward for all transitions is 0, except for the last one. In the last transition, a negative reward of the average delivery time is given or, if the number of packets delivered is lower than 80, the reward is -1000 times the difference. We chose 80 as the threshold because some preliminary experiments showed that R2L Throughput was able to deliver approximately 140 packets and R2L Delay approximately 20 packets, so 80 seemed a reasonable choice for a mixed strategy.

For training, we used the CMA-ES implementation in [19]

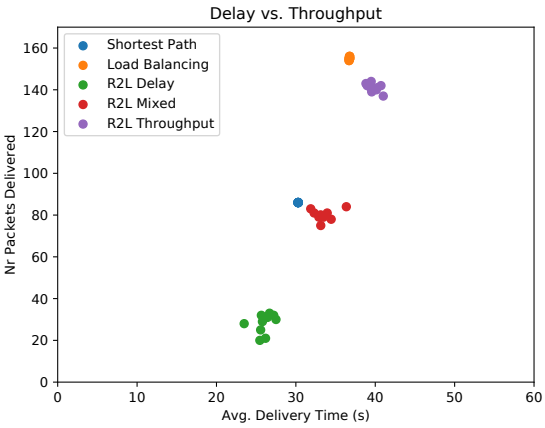


Fig. 2: Delay vs Throughput scatter plot.

with the default hyperparameters except for the population size which was set to 50. Each agent was trained for 300.000 episodes, where each episode lasted for a simulation time of 90 seconds, and then the best agent from the last generation was picked and tested for 10 episodes.

The results are shown in Figure 2, where each point corresponds to an episode. The number of packets delivered in the episode is given by the x-axis and the average delivery time by the y-axis. Although the R2L Throughout does not have the best performance in terms packets delivered, it is close to the Load Balancing strategy. On the other hand, the R2L Delay does have the best delay performance however at the expense of delivering fewer packets. We also note that R2L Mixed was able to find a balance between delay and throughput, however, the Shortest Path strategy was able to achieve a lower delay with a similar throughput. This plot, however, hints that R2L is capable of learning different routing strategies that align with the objectives set. Although in this case all the traffic is treated as equal, the results we obtained are promising for an approach that would use a reward depending on the traffic requirements. For example, we could introduce an additional state parameter to describe the type of traffic, e.g. throughput- or delay-sensitive and give a reward based on that.

B. Dynamic network conditions

We considered a topology with five nodes, as shown in Figure 3. All links had a transmission rate of 800 bits per second and propagation delay of 1 second. The exception was the links $2 \leftrightarrow 3$ which had a transmission rate of 600 bits per second. All queues had a maximum capacity of 20 packets. As in Section IV-A, there were two flows of traffic, in this case from 0 to 4 and 4 to 0 and the packets had an interarrival rate of 1 second.

At a certain point in the simulation, the transmission rate of links $1 \leftrightarrow 4$ was reduced to 2.5 bytes per second. The time of this event followed a uniform distribution $\mathcal{U}(0, 1000)$. The link went back up again to 800 bits per second after a number of seconds, which followed a $\mathcal{N}(400, 200)$. We refer to the

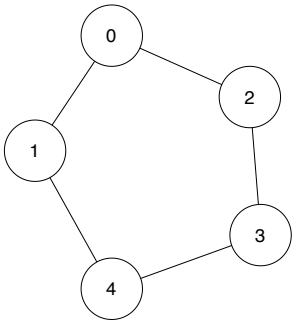


Fig. 3: 5-node topology.

period when the links $1 \leftrightarrow 4$ transmission rate is reduced as the event.

As benchmarks, we used three other routing strategies:

1) **Shortest Path:** Route all traffic $0 \leftrightarrow 1 \leftrightarrow 4$. With this strategy, there should not be a significant number of packet drops before and after the event, given that the incoming rate at node 1 is the same as the outgoing rate (however some queue could build-up due to the randomness in the packet generating process). However, during the event, almost all traffic should be dropped.
2) **Longest Path:** Route all traffic $0 \leftrightarrow 2 \leftrightarrow 3 \leftrightarrow 4$. This strategy is not affected by the event, but its throughput is not as high as Shortest Path and Load Balancing because it routes traffic through links $2 \leftrightarrow 3$ which have a lower transmission rate.
3) **Load Balancing:** Route 50% of traffic as Shortest Path and 50% as Longest Path. Before and after the event this is the best strategy of the three benchmarks since it load balances the traffic in a way that there are almost no queues building up and therefore no packets being dropped. During the event, almost all traffic that tries to goes through links $1 \leftrightarrow 4$ is dropped and therefore almost half of all traffic is dropped.

The training was done for 50000 episodes each lasting for 1000 seconds of simulation time. In Figure 4a we can observe the agent and benchmarks performance during training. The y-axis is the moving average of the sum of the rewards, with a window size of 1000 episodes. After approximately 20000 episodes the R2L can deliver on average more packets than Load Balancing.

We then used the model that had the best performance in the last generation and tested it with four different average durations for the event: 0, 200, 400 and, 600 seconds. The maximum event starting time was set to 1000 and therefore followed a $\mathcal{U}(0, 1000)$. Each scenario was run 10 times. In Figure 4b we plot the average packets delivered per strategy, for the different scenarios. As expected, the Shortest Path performance is the one that is affected the most by an increase in the event duration. Moreover, the Longest Path strategy performance is not affected by the change in link transmission rate. Although R2L performance is affected by an increase in the event duration, the decrease in the number of packets

delivered is lower than with the other strategies.

To better illustrate why does R2L performance differs from Load Balancing, we picked one of the trials with event duration of 400 and plotted a timeseries of packet deliveries in Figure 4c. We can observe that R2L agent has a performance similar Load Balancing in the beginning of the simulation but when the event starts a gap appears between the two. Finally, after the event the performance is again similar, as indicated by the slope of the two lines. This suggests that R2L was able to learn a routing strategy that can have a good performance even the network conditions change.

V. Conclusion and Future Work

In this work, we presented R2L, a solution to route packets in a network using Reinforcement Learning. R2L takes a per packet approach deciding the next hop using a single agent that lives in a central controller. This agent is given a global view of the network that is collected by caching the routers' local network measurements when they query the controller. Our experiments show that R2L was able to learn different routing strategies consistent with the desired objectives. Additionally, R2L found a routing strategy that performed well in a scenario where the network conditions were not static. In future work, we plan to perform more extensive experiments to study how can a reward that depends on the type of traffic, and leads to routing strategies where paths are tailored to the application needs, be designed to also ensure that all competing traffic flows receive a fair amount of shared resources. Furthermore, we plan to remove the dependency from local network measurements such that the state is determined only by historical packet header information. We see this approach as akin to use raw pixel data in image or video tasks. A benefit is that it would help to decrease the amount of information that routers have to collect and transmit, not only saving bandwidth costs but also reducing the system complexity.

References

[1] D.G. Cantor and M. Gerla. Optimal routing in a packet-switched computer network. *IEEE Transactions on Computers*, 23(10):1062–1069, Oct 1974.

[2] B. Fortz and M. Thorup. Optimizing ospf/is-is weights in a changing world. *IEEE Journal on Selected Areas in Communications*, 20(4):756–767, May 2002.

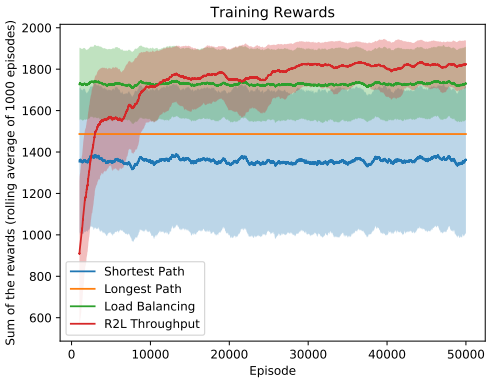
[3] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. Walking the Tightrope: Responsive Yet Stable Traffic Engineering. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, pages 253–264, New York, NY, USA, 2005. ACM.

[4] Justin A. Boyan and Michael L. Littman. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, pages 671–678. Morgan-Kaufmann, 1994.

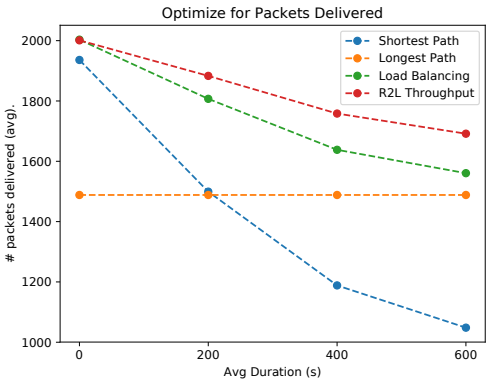
[5] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-driven Networking: A Deep Reinforcement Learning based Approach. *arXiv:1801.05757 [cs]*, January 2018.

[6] Shailesh Kumar and Risto Miikkulainen. Dual Reinforcement Q-Routing: An On-Line Adaptive Routing Algorithm. 1997.

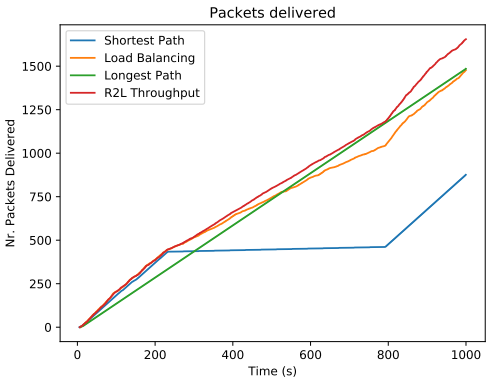
[7] Samuel P. M. Choi and Dit-Yan Yeung. Predictive Q-Routing: A Memory-based Reinforcement Learning Approach to Adaptive Traffic Control. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 945–951. MIT Press, 1996.



(a) Training rewards.



(b) Average packets delivered for different new flow durations.



(c) Packets delivered over time.

Fig. 4: Throughput based dynamic experiment.

[8] Ramy E. Ali, Bilgehan Erman, Ejder Baştuğ, and Bruce Cilli. Hierarchical deep double q-routing. *arXiv:1910.04041 [cs, math]*, Mar 2020. arXiv: 1910.04041.

[9] Erik Einhorn and Andreas Mitschele-Thiel. RLTE: Reinforcement Learning for Traffic-Engineering. In David Hausheer and Jürgen Schönwälder, editors, *Resilient Networks and Services*, Lecture Notes in Computer Science, pages 120–133. Springer Berlin Heidelberg, 2008.

[10] S. Lin, I. F. Akyildiz, P. Wang, and M. Luo. QoS-Aware Adaptive Routing in Multi-layer Hierarchical Software Defined Networks: A Reinforcement Learning Approach. In *2016 IEEE International Conference on Services Computing (SCC)*, pages 25–33, June 2016.

[11] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. Learning to Route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pages 185–191, New York, NY, USA, 2017. ACM.

[12] L. Peshkin and V. Savova. Reinforcement learning for adaptive routing. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, volume 2, pages 1825–1830 vol.2, May 2002.

[13] J. Moy. Ospf version 2. Technical Report 2178, RFC Editor, Jul 1997.

[14] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 2, pages 519–528 vol.2, March 2000.

[15] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. Technical Report 3031, RFC Editor, Jan 2001.

[16] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv:1712.06567 [cs]*, Apr 2018. arXiv: 1712.06567.

[17] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv:1604.00772 [cs, stat]*, Apr 2016. arXiv: 1604.00772.

[18] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv:1703.03864 [cs, stat]*, Sep 2017. arXiv: 1703.03864.

[19] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, February 2019.