DeepRoute: Herding Elephant and Mice Flows with Reinforcement Learning

Mariam Kiran, Bashir Mohammed, Nandini Krishnaswamy

DeepRoute: Herding Elephant and Mice Flows with Reinforcement Learning *

Mariam Kiran[1], Bashir Mohammed[2], and Nandini Krishnaswamy[2]

[1] Energy Sciences Network (ESnet), Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[2] Scientific Data Management, Lawrence Berkeley National Laboratory, Berkeley, CA
{mkiran,bmohammed,nkrishnaswamy}@lbl.gov

Abstract. Wide area networks are built to have enough resilience and flexibility, such as offering many paths between multiple pairs of end-hosts. To prevent congestion, current practices involve numerous tweaking of routing tables to optimize path computation, such as flow diversion to alternate paths or load balancing. However, this process is slow, costly and require difficult online decision-making to learn appropriate settings, such as flow arrival rate, workload, and current network environment. Inspired by recent advances in AI to manage resources, we present DeepRoute, a model-less reinforcement learning approach that translates the path computation problem to a learning problem. Learning from the network environment, DeepRoute learns strategies to manage arriving elephant and mice flows to improve the average path utilization in the network. Comparing to other strategies such as prioritizing certain flows and random decisions, DeepRoute is shown to improve average network path utilization to 30% and potentially reduce possible congestion across the whole network. This paper presents results in simulation and also how DeepRoute can be demonstrated by a Mininet implementation.

Keywords: Reinforcement learning · route optimization · path computation

1 Introduction

The rise of data hungry services such as mobile, video streaming and Cloud/Internet applications are bringing unprecedented demands to underlying network backbones [9]. Wide area networks (WANs) are investigating intelligent and efficient network management techniques to do load balancing, improve used bandwidth and overall optimize network performance. Traffic congestion can directly cause performance deterioration, such as when links are oversubscribed causing bottlenecks [10]. Many services rely on having high-throughput transfers and need high capacity links such as 100s Gbps. However, even for the busiest link, the current average utilization is only between 40-60%, to account for unanticipated

* This work is funded under DOE ASCR Early Career Grant Deep Learning: FP00006145.

peaks [1]. Traffic engineering and path computation techniques such as MPLS-TE (Multiprotocol Label Switching Traffic Engineering) [5], Google's B4 [18] and Microsoft's SWAN (Software Driven WAN) [1] have proposed manners in which routers can greedily select routing patterns for arriving flows, both locally and globally, to increase path utilization. However, these techniques require meticulously designed heuristics to calculate optimal routes and also do not distinguish between arriving flow characteristics.

Path computation has a number of real-world networking implications. Examples such as load balancing, minimizing congestion and utilizing maximum-bandwidth as some cases that can be explored as a reward. WAN networks al-low a number of pathways to exist between pairs of end-hosts.

Internet and WAN traffic usually contains a mixture of flow characteristics, such as long and short flows, which if on the same path, can have detrimental effects on each other. Known as bulk long-living file transfers (elephant flows) are bandwidth-sensitive and short transfers (mice flows) are latency-sensitive, significantly impacting user experience and require innovative ways to manage them [34]. WAN traffic usually contains a 80%:20% flow distribution (mice to elephant ratio) and if mixed on same paths, can cause queuing delays impacted by high latency and low throughput [2]. Isolating flows to dedicated routes [38] is challenging in real-time [34][32] and has led to under-utilized paths. Researchers have attempted to recognize arriving flows to efficiently manage them [22]. In data center networks, these are often prioritized, such as mice latency [7][6] or elephant throughput [4] to improve data center network performance. In WAN, path computation uses optimization to calculate paths taken between end-hosts.

Leveraging research in software defined networking (SDN) and reinforcement learning, we explore this problem of path computation and finding optimal routes in general, as a path resource allocation problem. *We use machine learning algorithms to learn and provide viable solutions for dynamic flow management for both elephant and mice flows.*

Network routing and machine learning is not new. Broadly speaking, there are two main approaches used here [33]:

1. optimize routing configurations by predicting future traffic conditions depending on past traffic patterns or
2. optimize routing configurations based on number of feasible traffic scenarios with aim to improve performance parameters.

While SDN's centralized control offers great promise, these calculations cause overhead for high-performance networks and need global management to work, which is difficult in a large network [12]. Recent success of machine learning in complex decision making problems such as Alpha-Go [29], cooling datacenters and self-driving cars [37] suggest feasible applications to our problem. Particularly reinforcement learning (RL), actively being applied in robotics [31], allows agents to learn how to make better decisions by interacting directly with the environment. Using concepts of rewards and penalties it can learn through experience to optimize its objective function.

Revisiting the path computation challenge, in this paper we use RL to find optimal paths (or routes) as a resource management problem. First, the system learns optimal paths by repeatedly selecting available paths between source and destination, given the arriving flow distribution. RL directly interacts with the network environment and learns best paths to minimize the flow completion time given the current network conditions. Our RL approach (DeepRoute) is developed using Q-learning as a value-based gradient reinforcement learning [39].

Our experiments are focused on WAN scenarios, with two implementations,

1. We simulate an environment with synthetic data set of 100 flows, with 80:20 (elephant:mice) distributions, being allocated on 4 possible paths. We compare DeepRoute to either randomly selecting paths or prioritizing one flow type over the other. Here we use the analogy of 100 flows waiting to be allocated on paths and aim to quickly empty the wait queue.
2. We translate the experiment on Mininet network emulator environment. Doing so, we remove some assumptions we drew in the simulation and also understand how DeepRoute will work in a real network environment.

In both cases, we train DeepRoute via experience generating abundance of flow interaction data with the network. Evaluation of the agent's performance is done on test data set, which is previously unseen flow data, to see how well DeepRoute fares against other techniques.

2 Background

2.1 Path Computation and Flow Management in WAN

We explain why path computation is a challenging problem:

- Network traffic demands are continuously changing and are often difficult to predict. Routing tables are continuously changing with new devices joining/leaving the network. However, paths between two endpoints, usually follows one optimized route. If there are too many flows, this leads to potential congestion on the path [13].
- Underlying network systems are complex and distributed, with multiple links between source-destination pairs. Understanding link properties such as bandwidth or latency is difficult to model accurately [14].
- Most flow management techniques are developed for data center networks [3]. WANs, on the other hand, prioritize performance metrics such as minimizing packet loss and bandwidth utilization for traffic diversion [36]. Google's B4 and Microsoft's SWAN, both, make decisions on application characteristics and heuristics.

Compared to approaches designed by B4 and SWAN [18][1], we leverage reinforcement learning to provide alternative to heuristic-based path computation problem. We aim to allow *networks to learn best routes such to minimize flow completion times of both elephant and mice flows.*

Path computation has a number of real-world networking implications. Examples such as load balancing, minimizing congestion and utilizing maximum bandwidth as some cases that can be explored as a reward. WAN networks allow a number of pathways to exist between pairs of end-hosts. These paths can have equal or different cost distributions such as settings for bandwidth, latency, throughput and more. These settings can determine how quickly the arriving flow will reach its destination, and can be allocated using different egress ports to choose path to take. ECMP routing is an example of this where it uniformly picks an egress port to reduce congestion on one path, however has seen to suffer from hash collisions [17] and unbalance on different cost distribution paths.

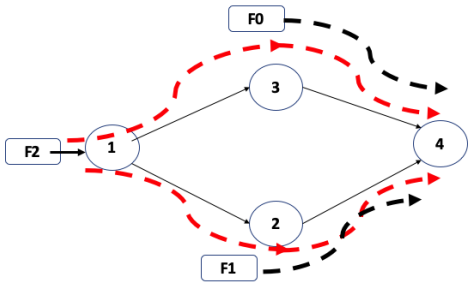


Fig. 1. Example of routing a new flow F1 from 1→4. There are two possible paths to take through Node-3 or Node-2.

Another problem is the changing traffic conditions in the network. Figure 1 shows a new flow F2 being allocated to one of the paths. However, there might be other previous flows running F0 and F1, already allocated on part of the paths. This means that while the links costs can be set in advance, the available bandwidth on the links is continuously changing and difficult to anticipate when selecting paths.

Traffic engineering such as MPLS-TE, B4 and SWAN use heuristics to design optimal allocations for paths using local and global optimization functions. For DeepRoute, we focus on WAN network path computation, particularly where we want flow-based routing decisions to improve flow completions times for mice and elephants that can allow all available paths to be chosen and utilized.

2.2 Reinforcement Learning

A reinforcement learning problem is formulated with an agent, situated in a partially observable environment, learning from past interaction data to make current decisions. The agent receives data in form of environment snapshots, processed in some manner, with specific relevant features. After receiving information and computing the value for future actions given the current state, an agent then acts to change its environment, subsequently receiving feedback on

its action in form of rewards, until terminal state is reached. The objective is to maximize the cumulative reward over all actions in the time agent is active. Reinforcement learning research has investigated multiple techniques such as in multi-armed bandit problems, resource allocation or finding routes through a maze [30]. Deep reinforcement learning builds upon classical models, replacing the learning with a neural network to approximate policy and value functions. Here, the function approximates the environment state space with actions and rewards. Particularly when the state space is too large to store, this approach has proved feasible in learning approximate conditions. In future, we plan to expand DeepRoute to adapt from Q-learning to neural network learning (deep Q-networks), but is currently out of scope of the work presented here.

Reinforcement learning can be expressed as a Markov Decision Process (MDP) involving sequential decisions. This is given as a tuple (S, A, R, P), where $s \in S$ set of states, $a \in A$ set of actions, $R(s, a, s')$ represents reward given for executing action a at s and moving to new state s'. There is probability P for executing action in state s.

In our problem, the network is modeled as a MDP. We model 4 paths and their current allocations as a state. The agent selects a path and receives a reward after a flow has completed. In the simulation model, this is delayed reward as flows take longer to finish. In the mininet model, we get the reward at every iteration. The reward act as signals to adjust the forwarding-link priorities to enhance or diminish the probability a specific next-hop is selected for the flow. Our agent learns to adjust path selection policies based on experience through continuous modification and rewards.

2.3 *Q*-learning Formulation

A Q-value represents state-action combinations. Better Q-values shows better chances of getting higher rewards which are earned at the end of a complete episode. The Q-value is calculated using a Q-function. It approximates the Q-value using prior Q-values, a short-term and a discounted future reward. This way the find optimal control policies across all environment states. Q-learning is an off-policy reinforcement learning algorithm that uses a table to store all Q-values with possible states and action pairs. This table is updated using the Bellman equation, allowing the action to be chosen using a greedy policy, given as with γ is discounting factor.

$$Q(s, a) = R(s, a) + \gamma \max_{a'} Q(s', a')) \tag{1}$$

Temporal Difference (TD) Learning. In our model, we enable the agent to learn in every action taken, despite it being end of the episode or not. We define an episode to end after 100 flows have been allocated. The TD learning factor updates current Q-value where α is the learning rate,

$$Q_t(s, a) = Q_{t-1}(s, a) + \alpha TD_t(a, s) \tag{2}$$

Therefore, our final equation becomes,

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha(R(s,a) + \gamma \max Q(s',a') - Q_{t-1}(s,a)) \quad (3)$$

Where $\gamma \in [0,1]$ represents discounting factor that scales importance of the immediate reward obtained for the action and rewards R obtainable for actions at the new state s'. The learning rate $\alpha \in [0,1]$ models the rate at which of Q-values are updated.

3 Related Work

Efficient selection of paths for short and long flows has also shown to reduce congestion [26]. Additionally, adaptive traffic management using congestion-based routing has proven to improve overall network utilization [8] [18]. However, adaptive algorithms in heavy traffic load, could cause oscillatory behavior and cause performance degradation [35]. OWAN was developed to optimize bulk transfers on WAN by re-configuring the optical layer showing a 4-times faster flow completion rate [20].

Flow completion time has been used as a vital metric to improve network congestion [27]. Additionally, separating elephant and mice flows can have a direct impact on network quality [15]. Other approaches have used calendaring to improve bandwidth utilization and reduce congestion [21].

In comparison to above approaches, machine learning has been used for network routing. The authors [33] show how learning can help improve the average congestion rate through softmin learning.

Reinforcement learning , in particular, has given interesting results in compute resource allocations such as CAPES [23] learning to optimize the Luster file system and DeepRM [24] learning to allocate jobs on processors. Within networks, implementations of reinforcement learning in internet congestion control [19] and developing intelligent TCP congestion algorithms [36] have shown game-changing results in managing bandwidth and bottleneck across individual links. But nearly all of these demonstrations have only been shown on small networks and simulation only. There is also little evidence on how these can be expanded to real WAN environments.

In this paper, we prove the usefulness of reinforcement learning for path computation, but this is presented as early work on how Q-learning approaches can provide benefit and be translated in WAN environments, which we will expand in future work.

4 Design of DeepRoute

We define a general network topology with unidirectional links denoted as paths $p \in P$ with varying bandwidth capacity and latency. Flows arrive at time step $t = 1$ and are assigned a fixed path, for example either $1 \to 2 \to 4$ or $1 \to 3 \to 4$

(in Figure 1). The arriving flows are generated with a specified size and duration allowing the flow to exist on the path for a time period (longer for elephant flows). Given every flow f_i from source s_i to destination d_i, is defined with flow size v_i and duration r_i. The flow is assigned to a path p_i which would have latency lat_{pi} and available capacity ac_{pi}. The latency influences the actual flow completion time $c_i = r_i + lat_{pi}$. At a given time, flows are allocated based on available bandwidth ac_{pi}, which changes depending on traffic patterns.

Similar to SDN [1], we assume a centrally existing routing control. The controller maintains information on current allocations across paths and makes routing decisions based on completion times of previous flows.

Table 1. Elephant and mice flow distributions used in simulation case.

Flow type	**Size of Flow (bandwidth units occupied)**	**Duration of Flow (time units occupied)**	**Distribution %**
Elephant	Range(3,5)	Range(5,8)	20%
Mice	Range(1,2)	Range(1,3)	80%

4.1 DeepRoute in Simulation Model

Figure 2 shows the network topology used in the simulation. The goal is to move all arriving flows from source to destination as quickly as possible. There are 4 possible paths with different bandwidth and latency settings. In the model, we assume 100 flows are arriving together in one timestep, and the controller allocates each flow to the next available path. The 100 flows contain a distribution of 80% mice flows and 20% elephant flows (Table: 1). Once allocated, the time step progresses to $t+1$, where the controller tries to allocate the remaining flows.

Objective. When a flow duration r_i (in time units) finishes, it computes its completion time c_i by adding its duration with path latency. We then inverse this, to give the flow's slowness rate by $l_i = c_i/r_i$. Similar to [24], we normalize this, to prevent skewing results for longer flows. The objective of DeepRoute is to get as many flows completed soon as possible.

Paths as Resources. We assume 100 flows arrive at Node-1 going to Node-5. There are 4 possible paths $1 \rightarrow 2 \rightarrow 5$, $1 \rightarrow 5$, $1 \rightarrow 3 \rightarrow 5$ or $1 \rightarrow 4 \rightarrow 5$, with each path having different bandwidths and latency attached. **bu** means total **bandwidth units** available to be allocated. This changes as flows are allocated on them consuming part of the bu equal to flow size. The allocations last for the flow's duration, and the completion time is computed summing path latency given in **tu (time units)**.

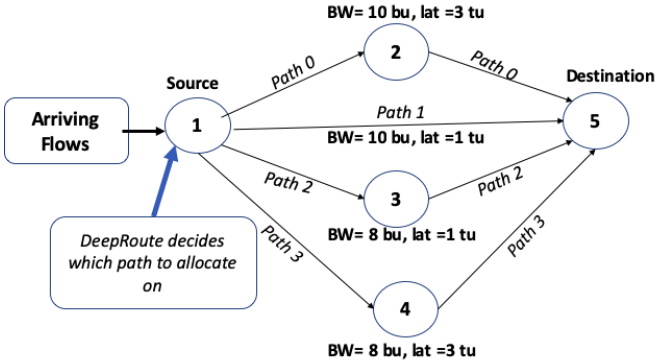


Fig. 2. Topology used in the Simulation Model (bu=bandwidth units occupied and tu=time units occupied).

State Space. The state of the environment is what DeepRoute learns against. We define this as current available bandwidth across all paths, size of the flow being allocated and current allocations on the paths. For example, after an allocation on path0 ($1 \rightarrow 2 \rightarrow 5$), the bandwidth availability is now $ac_{p0} = 10 - v_1$ and this part of state becomes $(ac_{p0}, ac_{p1}, ac_{p2}, ac_{p3})$.

Action Space. There are 4 paths so 4 possible actions. We assume one action is taken per one flow, within one time step. If there are no available paths, the controller skips a time step with no allocations. Once all 100 flows are allocated, it finishes a complete episode. The simulation is run for number of iterations containing many episodes. The total reward is calculated per episode when all 100 flows are allocated.

Reward Calculation. At the end of each episode, the RL agent calculates if any flows have finished and total completion time is recorded.

4.2 DeepRoute in Mininet

Figure 3 shows mininet topology forming multiple paths to chose from one end-host to the other. Here, we configure three metrics for the links - capacity, packet loss and latency. Mininet allows us to add delays on links. The capacities (bandwidth) are setup as 2 x 10 Gbps links for path0 and path1, and 2 x 8 Gbps for path2 and path3.

Sending flows. We ping from one end-host to the other and record the time in which the ping reaches the other host. The controller learns which egress port to use, deciding which path to take. This time is recorded as the reward against the path taken.

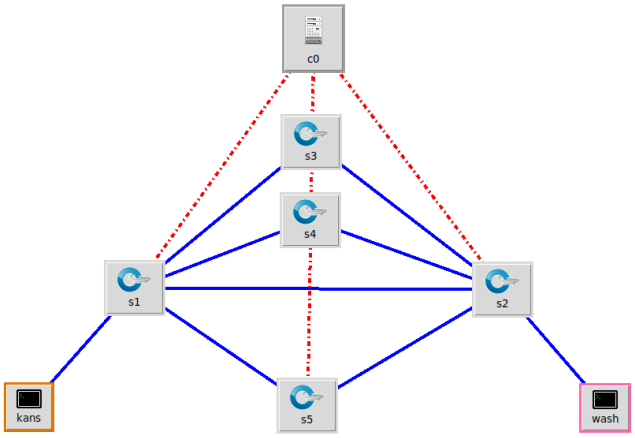


Fig. 3. Mininet Topology.

SDN Control. We emulate an SDN network using an Ryu OpenFlow switch. To have a global view, we configure a network controller, Openflow switches, linux hosts and network links. The hosts run a standard Linux kernel and network stack to emulate running real network applications. Traceroute measurements are recorded on the route taken and the specific gateway at each hop. We calculate the total time across each hop as a reward. In this case, every episode is represented by one flow (one ping).

State and Action Space. For dynamic multipath selection, we send a packet from source to destination via one of the four network paths, dynamically allocating flows on links. The flows go through paths: $1 \rightarrow 2 \rightarrow 5$, $1 \rightarrow 5$, $1 \rightarrow 3 \rightarrow 5$ and $1 \rightarrow 4 \rightarrow 5$.

For initial learning, we use Link Layer Discovery Protocol (LLDP) [11] to obtain link and switch states in the topology, it uses to advertise device identity and abilities, and other devices connected within. LLDP helps maintain a global view of network topology and also retains a multipath environment.

5 Training DeepRoute

The DeepRoute agent runs in an episodic fashion (with 100 flows in simulation and 1 flow in mininet). The episode terminates when all flows have been allocated.

A Q-value is added with each state and action taken and saved into a Q-table. We design our algorithm based on [30]. As the reinforcement learning algorithm uses Bellman's equation, there is a possibility of overfitting to ideal conditions. To prevent this, during the testing phase, we use ϵ for allowing DeepRoute to select random action rather than Q-table values.

Algorithm 1 Q-learning for Training DeepRoute

Initialize Q-table
for each Iteration: **do**
  for each Episode: **do**
    Generate 100 flows
    for each flow i=1,..., 100: **do**
      Get available bandwidth $(ac_{p1}, ac_{p2}, ac_{p3}, ac_{p4})$
      Get flow to allocate v_i
      Get current flow allocations across the 4 paths $m_{p1}, e_{p1}, m_{p2}, e_{p2}, m_{p3}, e_{p3}, m_{p4}, e_{p4}$
      State $s_i = ((ac_{p1}, ac_{p2}, ac_{p3}, ac_{p4}), v_i, m_{p1}, e_{p1}, m_{p2},$
      $e_{p2}, m_{p3}, e_{p3}, m_{p4}, e_{p4})$
      if randomnumber $< \epsilon$
      Select any action $a_i \in (a_1, a_2, a_3, a_4)$
      Else
      Check if Q-table has this state and select best action with highest Q-Value
      Update Q-value
      Check expired flows and add reward
      If (s_i, a_i) not found in Q-table, add new entry to Q-table.
    end for
  end for
end for
For each episode:
Print Reward

=0

5.1 Training in Simulation

DeepRoute is trained for multiple iterations as shown in Algorithm 1. Each iteration generates new 100 flows and DeepRoute learns by allocating these flows on the paths. We record the network state (available bandwidth on all 4 paths, current allocations, size of the flow to allocate), action taken (which path is chosen) and reward collected in the episode.

Training iterations. Training for more iterations, allows the size of the Q-table to grow (from 3532 for 50 iterations, 6238 for 100 and 21640 for 500 iterations). However, Figure 4 shows that the maximum score is achieved by 400 iterations. Therefore this is chosen as the ideal training iterations.

5.2 Training in Mininet

The implementation in Mininet removes some assumptions drawn in the simulation model. Here, along with using an OpenFlow switch, we use packet loss and latency, to calculate the reward for every ping received at other host. The

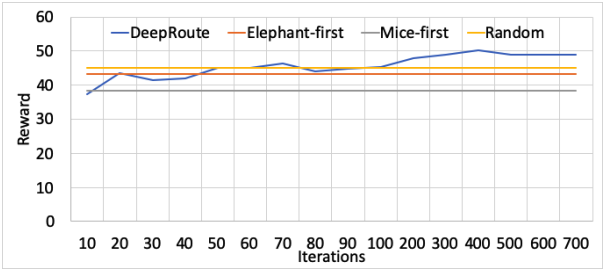


Fig. 4. Changing number of iterations during training gives different rewards.

Q-table is recorded by monitoring Wireshark logs across all interfaces on the controller.

Initially, the OpenFlow switch performs active and passive measurements across all egress links, building the Q-table. The utilization data across the interfaces is collected. Here, we also have a local Q-table at each network node, as well as a global aggregation table managed by the network controller, shown in Figure 5. The wireshark logs allow to collect data on latency, throughput and packet loss. Rewards or the completion time is added afterwards to populate the Q-values in the table constructed. Table 2 shows an example of the Q-table constructed in Mininet. Showing only 4 entries, 1 in each path. We transfer a mixture of flow distributions (8 and 16 GBytes). The arrival time (or completion time) is considered as the reward, added to the Q-table for that state and action.

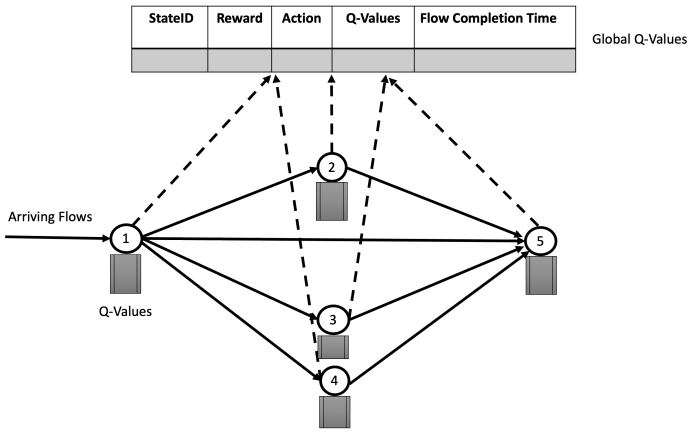
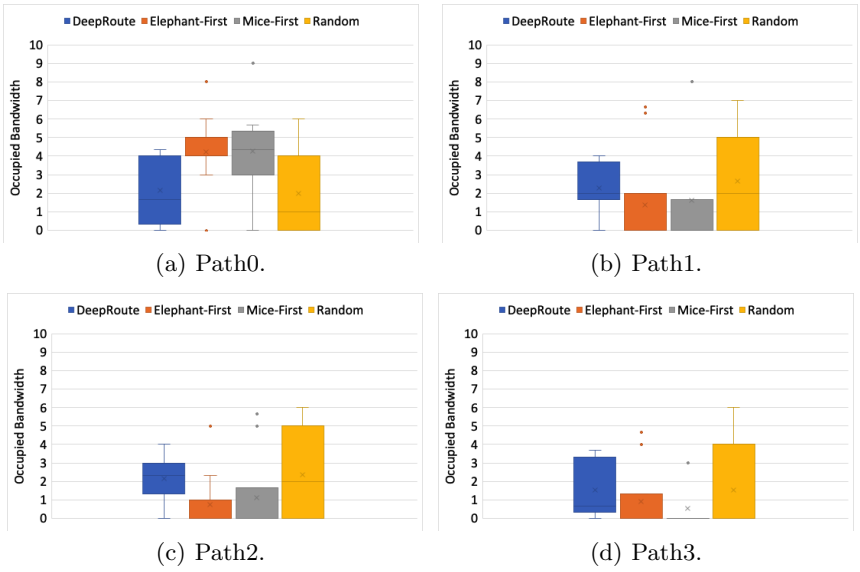


Fig. 5. Q-values for Local and Global levels.

Table 2. Sample Q-table showing capacity, latency and flow arrival time.

Capacity (Gbps)	Latency (ms)	Path	Transfer (GB)	Arrival Time (ms)
10	300	0	16	60290
10	100	1	8	42086
8	100	2	8	50819
8	300	3	16	59732



(a) Path0. (b) Path1.

(c) Path2. (d) Path3.

Fig. 6. Occupied bandwidth across the paths with arriving flows. Occupied bandwidth refers to the percentage bandwidth occupied on the path.

6 Evaluation

We evaluate DeepRoute for the following objectives:

- With a distribution of network path parameters, how does DeepRoute compare with other allocation techniques in simulation model?
- How can DeepRoute be translated into real network environments as in a mininet model?
- How much of utilized capacity improvements are observed for the overall network?

6.1 Simulation Model Results

Testing Data. We generate a new set of 100 flows of 80:20 (mice:elephant) distribution. This data is unseen by the DeepRoute during training phase. The test data is consistent for all other comparison schemes to validate the results.

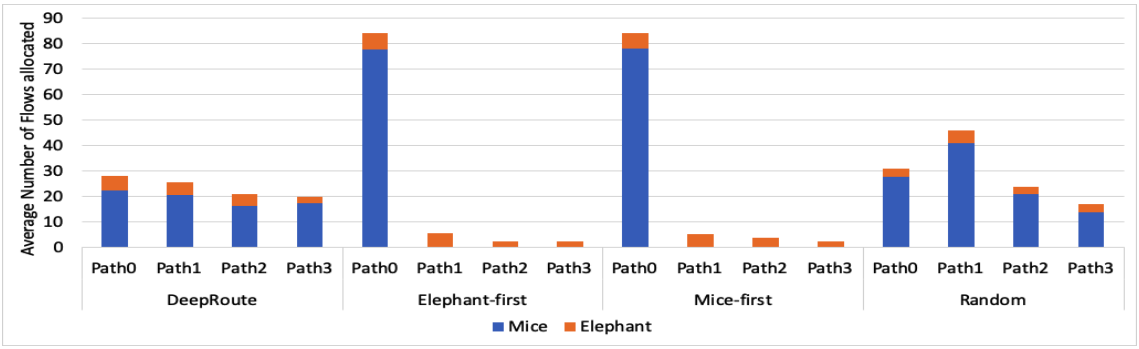


Fig. 7. Average flows allocated to all paths.

Comparing with other techniques. We compare DeepRoute with other algorithms - Random, to randomly assign flows on available paths, Prioritize-Mice, to allocate mice flows before allocating elephant flows, and Prioritize-Elephant, allocate elephant flows before mice flows. These have been published in path computation problems [3][32].

Average Path Utilization. Figure 6 shows the path capacity used during the testing phase. Here we see that while Path0 is the most used by the elephant- and mice-first techniques, DeepRoute is able to show a more spread of using other paths efficiently. It learns to use Path 1 and Path 2, with lower latency more efficiently. This is also shown in Figure 7, where paths are used more than in other techniques. The random technique just spread use across all paths.

Figure 8 shows that DeepRoute is able to completely utilize the network at stable 30% as compared to the other techniques as number of flows increase.

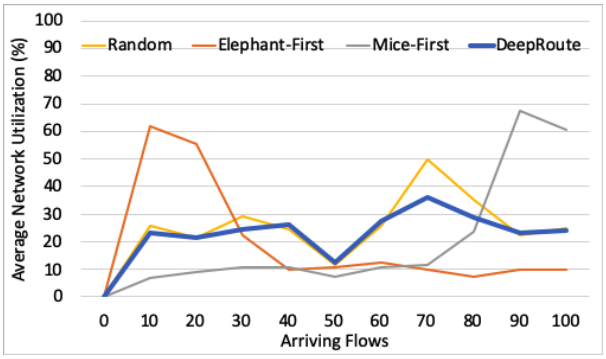


Fig. 8. Average network utilization.

Mice and Elephant allocations. Figure 7 shows that DeepRoute is able to spread elephant and mice flows more uniformly across all the paths. We also see that by spreading the load, there is less congestion on one path, which was the case with other techniques.

Comparing with shortest path route. Based on the path configurations, the network would use Path1 the most due to higher bandwidth and lower latency. However, in the simulation DeepRoute learns to allocate on alternate paths, based on the current allocations already active on the paths.

6.2 Mininet Model Results

After training, we configure the Ryu SDN controller to select egress based on Q-table. This adjusts forwarding rules dynamically as new flows arrive.

Testing data. We generate 100 pings from one end-host to the other and record the utilization across all the paths.

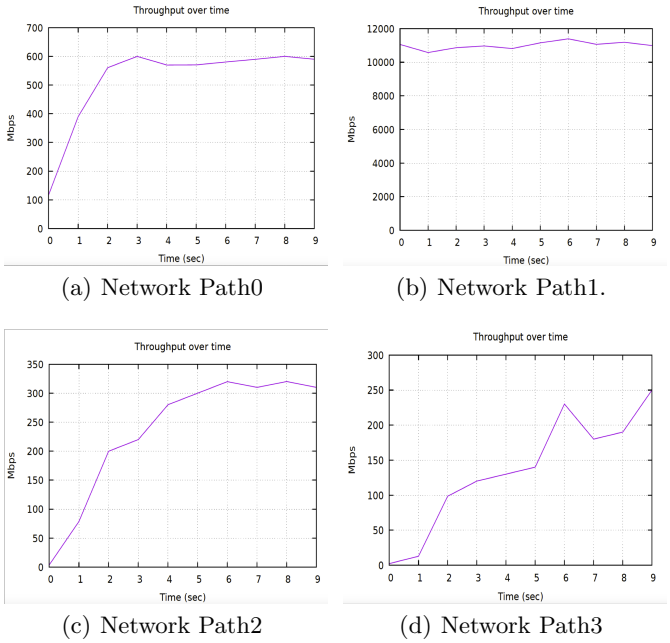
Throughputs recorded. Figure 9 shows the throughput measurements along the 4 paths. Here, because the mininet model is extremely simple, the states learned are the available paths, flow size sent and the time recorded. This allows the controller to learn the shortest, less cost path, Path 1, as the most optimal path to use. As a result, all future traffic goes through this path and less evenly distributed at other paths. This result shows, if the controller was able to learn more features of the current state, we could enhance the controller decisions. This will form basis for future investigations of implementing DeepRoute in real networks.

7 Discussion

7.1 Modelling realistic WAN flows

In this section we explained how we model realistic WAN flows, using different size parameters in mininet simulation. We use NETEM [16] which provides network emulation functionality for testing protocols by emulating properties of wide area networks. To emulate the real-world network scenario, with control on parameters that affect network performance, we change the path delay on all four paths to $(300ms, 100ms, 100ms, 300ms)$. In running our emulation, we consider four key metrics - link capacity (bandwidth), latency, transfer size and flow arrival time (presented in Table 2). To calculate latency, link delays were assigned paths.

For link capacities (bandwidth), the DeepRoute topology has a total link capacity of 36 Gbps linking the path 0 to path 3. This includes (2 X 10 Gbps) on path 0 and 1 as well as (2 X 8 Gbps) on path 2 and 3 respectively. Iperf3 [25]



(a) Network Path0 (b) Network Path1.

(c) Network Path2 (d) Network Path3

Fig. 9. Throughput measurement from Source-Destination.

is used for measuring performance characteristics, specifically, the TCP version of Iperf3. Each TCP Iperf throughput test is initiated for 60 seconds per path. All end hosts loop through this process for at least 100 flow rounds, thereby measuring throughput on all the network paths (path 0-3) with different flow distributions. We use wireshark [28], a free and open-source packet analyzer to capture packets on all interfaces.

7.2 Performance measurement with load and delay

As flow are sent, the network controller starts the learning process by performing active and passive measurements between all switches (S1-S5). The active measurements are used to measure latency between the switches, while passive measurements are used to obtain load and residual capacity in each link.

Having successfully deployed emulated WAN using NETEM and Mininet, we conducted some performance measurement. With a set of flows, we specify a normal distribution for delay, but since delays are not always uniform, we specify a Pareto distribution (non-uniform delay distribution). As a result, all packets leaving source to destination via path 1 and 2 will experience a delay time of 100ms, while those leaving via path 0 and 3 will experience 300ms. The final results show minimum, average, maximum and standard deviation of the Round-Trip-Time (RTT) and if packet loss is recorded.

7.3 Comparing to optimization approaches

The current implementation of Q-table shows that similar results could have been achieved by optimization techniques. However, this implementation is limited by modelling the system as a simple MDP approach. In a real network setting, conditions are much more dynamic with unseen environment conditions. This means the Q-table approach will have to updated the build approximate value functions, such as deep Q-networks, to make decisions in unseen conditions.

8 Conclusions and Future Work

Recent breakthroughs in deep learning research, made possible by accelerated hardware and big data, in many fields. However, there is still a lack of understanding on how this can be used in network routing research.

By utilizing Q-learning we allow the controller to learn from the environment about the paths and best hops between source and destination. With network environments being very dynamic, with possible packet loss and traffic congestion across some of the best paths, we explore how a DeepRoute controller can learn best possible combinations depending on the traffic arriving and the current network conditions to optimally utilize the network.

Commercial systems that promise improved network performance tend to focus on average typical flows and through exploring edges (when compared to average) press exploration of new traffic engineering models. Our work highlights the need to change approaches to path computation and flow management for new applications like hybrid cloud computing and other use cases cited. While networks are challenged to strike the balance between capacity, throughput, latency and cost, AI applications can have an impact on future deployments. Our results show promise on how DeepRoute can allow efficient use of path capacity and the mininet implementation shows how it can be adapted in a real network environment.

References

1. Achieving high utilization with software-driven wan. Tech. Rep. MSR-TR-2013-54 (August 2013), https://www.microsoft.com/en-us/research/publication/achieving-high-utilization-with-software-driven-wan/
2. Akella, A., Seshan, S., Shaikh, A.: An empirical evaluation of wide-area internet bottlenecks. In: SIGMETRICS. pp. 316–317 (2003). https://doi.org/10.1145/781027.781075, http://doi.acm.org/10.1145/781027.781075
3. Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., Vahdat, A.: Hedera: Dynamic flow scheduling for data center networks. In: USENIX Conf. Networked Systems Design and Implementation. pp. 19–19 (2010), http://dl.acm.org/citation.cfm?id=1855711.1855730

4. Alizadeh, M., Greenberg, A., Maltz, D.A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., Sridharan, M.: Data center tcp (dctcp). In: SIGCOMM. pp. 63–74 (2010). https://doi.org/10.1145/1851182.1851192, http://doi.acm.org/10.1145/1851182.1851192
5. Awduche, D., Malcolm, J., Agogbua, J., O'Dell, M., McManus, J.: Requirements for traffic engineering over mpls (1999)
6. Benson, T., Anand, A., Akella, A., Zhang, M.: Microte: Fine grained traffic engineering for data centers. In: COnference on Emerging Networking EXperiments and Technologies. pp. 8:1–8:12 (2011). https://doi.org/10.1145/2079296.2079304, http://doi.acm.org/10.1145/2079296.2079304
7. Carpio, F., Engelmann, A., Jukan, A.: Diffflow: Differentiating short and long flows for load balancing in data center networks. In: GLOBECOM. pp. 1–6 (Dec 2016). https://doi.org/10.1109/GLOCOM.2016.7841733
8. Chakravorty, R., Banerjee, S., Rodriguez, P., Chesterfield, J., Pratt, I.: Performance optimizations for wireless wide-area networks: Comparative study and experimental evaluation. In: International Conference on Mobile Computing and Networking. pp. 159–173 (2004). https://doi.org/10.1145/1023720.1023737, http://doi.acm.org/10.1145/1023720.1023737
9. Cisco: Trending analysis. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html (2019)
10. Clark, D., Bauer, S., claffy, k., Dhamdhere, A., Huffaker, B., Lehr, W., Luckie, M.: Measurement and Analysis of Internet Interconnection and Congestion. In: Telecomm. Policy Research Conf. (Sep 2014)
11. Congdon, P.: Link layer discovery protocol and mib. V1. 0 May **20**(2002), 1–20 (2002)
12. Curtis, A.R., Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P., Banerjee, S.: Devoflow: Scaling flow management for high-performance networks. SIGCOMM Comput. Commun. Rev. **41**
13. Domzal, J., Jajszczyk, A.: New congestion control mechanisms for flow-aware networks. In: Int. Conf. Communications (May 2008). https://doi.org/10.1109/ICC.2008.11
14. Floyd, S., Jacobson, V.: Link-sharing and resource management models for packet networks. IEEE/ACM Transactions on Networking **3**(4), 365–386 (Aug 1995). https://doi.org/10.1109/90.413212
15. Habibi Gharakheili, H., Sivaraman, V., Moors, T., Vishwanath, A., Matthews, J., Russell, C.: Enabling fast and slow lanes for content providers using software defined networking. IEEE/ACM Trans. Netw. **25**(3), 1373–1385 (Jun 2017). https://doi.org/10.1109/TNET.2016.2627005, https://doi.org/10.1109/TNET.2016.2627005
16. Hemminger, S., et al.: Network emulation with netem. In: Linux conf au. pp. 18–23 (2005)
17. Iselt, A., Kirstadter, A., Pardigon, A., Schwabe, T.: Resilient routing using mpls and ecmp. In: 2004 Workshop on High Performance Switching and Routing, 2004. HPSR. pp. 345–349 (April 2004). https://doi.org/10.1109/HPSR.2004.1303507
18. Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., Vahdat, A.: B4: Experience with a globally-deployed software defined wan. SIGCOMM Comput. Commun. Rev. **43**(4), 3–14 (Aug 2013). https://doi.org/10.1145/2534169.2486019, http://doi.acm.org/10.1145/2534169.2486019

19. Jay, N., Rotman, N., Godfrey, B., Schapira, M., Tamar, A.: A deep reinforcement learning perspective on internet congestion control. In: Int. Conf. Machine Learning. vol. 97, pp. 3050–3059 (09–15 Jun 2019), http://proceedings.mlr.press/v97/jay19a.html
20. Jin, X., Li, Y., Wei, D., Li, S., Gao, J., Xu, L., Li, G., Xu, W., Rexford, J.: Optimizing bulk transfers with software-defined optical wan. In: SIGCOMM. pp. 87–100 (2016). https://doi.org/10.1145/2934872.2934904, http://doi.acm.org/10.1145/2934872.2934904
21. Kandula, S., Menache, I., Schwartz, R., Babbula, S.R.: Calendaring for wide area networks. In: SIGCOMM. pp. 515–526 (2014). https://doi.org/10.1145/2619239.2626336, http://doi.acm.org/10.1145/2619239.2626336
22. Kiran, M., Chhabra, A.: Understanding flows in high-speed scientific networks: A netflow data study. Future Generation Computer Systems **94**, 72 – 79 (2019). https://doi.org/https://doi.org/10.1016/j.future.2018.11.006, http://www.sciencedirect.com/science/article/pii/S0167739X18302322
23. Li, Y., Chang, K., Bel, O., Miller, E.L., Long, D.D.E.: Capes: Unsupervised storage performance tuning using neural network-based deep reinforcement learning. In: Int. Conf. for High Performance Computing, Networking, Storage and Analysis. pp. 42:1–42:14 (2017). https://doi.org/10.1145/3126908.3126951, http://doi.acm.org/10.1145/3126908.3126951
24. Mao, H., Alizadeh, M., Menache, I., Kandula, S.: Resource management with deep reinforcement learning. In: ACM Workshop on Hot Topics in Networks. pp. 50–56 (2016). https://doi.org/10.1145/3005745.3005750, http://doi.acm.org/10.1145/3005745.3005750
25. Mortimer, M.: iperf3 documentation (2018)
26. Munir, A., Qazi, I.A., Uzmi, Z.A., Mushtaq, A., Ismail, S.N., Iqbal, M.S., Khan, B.: Minimizing flow completion times in data centers. In: IEEE INFOCOM. pp. 2157–2165 (April 2013). https://doi.org/10.1109/INFCOM.2013.6567018
27. Noormohammadpour, M., Srivastava, A., Raghavendra, C.S.: On minimizing the completion times of long flows over inter-datacenter wan. IEEE Communications Letters **22**(12), 2475–2478 (Dec 2018). https://doi.org/10.1109/LCOMM.2018.2872980
28. Orebaugh, A., Ramirez, G., Beale, J.: Wireshark & Ethereal network protocol analyzer toolkit. Elsevier (2006)
29. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. Nature **529**, 484 (01 2016), https://doi.org/10.1038/nature16961
30. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. A Bradford Book, USA (2018)
31. Tomar, M., Sathuluri, A., Ravindran, B.: Mamic: Macro and micro curriculum for robotic reinforcement learning. In: Int. Conf. on Autonomous Agents and MultiAgent Systems. pp. 2226–2228 (2019), http://dl.acm.org/citation.cfm?id=3306127.3332066
32. Trestian, R., Muntean, G., Katrinis, K.: Micetrap: Scalable traffic engineering of datacenter mice flows using openflow pp. 904–907 (May 2013)

33. Valadarsky, A., Schapira, M., Shahaf, D., Tamar, A.: Learning to route. In: Hot Topics in Networks. pp. 185–191 (2017). https://doi.org/10.1145/3152434.3152441, http://doi.acm.org/10.1145/3152434.3152441
34. Wang, W., Sun, Y., Salamatian, K., Li, Z.: Adaptive path isolation for elephant and mice flows by exploiting path diversity in datacenters. IEEE Trans. Network and Service Mng. **13 (1)**, 5–18 (2016). https://doi.org/10.1109/TNSM.2016.2517087
35. Wang, Z., Crowcroft, J.: Analysis of shortest-path routing algorithms in a dynamic network environment. SIGCOMM Comput. Commun. Rev. **22**(2), 63–71 (Apr 1992). https://doi.org/10.1145/141800.141805, http://doi.acm.org/10.1145/141800.141805
36. Winstein, K., Balakrishnan, H.: Tcp ex machina: Computer-generated congestion control. In: In SIGCOMM, Hong Kong (2013)
37. Xiao, J.: Learning affordance for autonomous driving. In: Wrkp on Smart, Autonomous, and Connected Vehicular Systems and Services. pp. 1–1 (2017). https://doi.org/10.1145/3131944.3133941, http://doi.acm.org/10.1145/3131944.3133941
38. Yan, Z., Tracy, C., Veeraraghavan, M., Jin, T., Liu, Z.: A network management system for handling scientific data flows. J. Netw. Syst. Manage. **24**(1), 1–33 (Jan 2016). https://doi.org/10.1007/s10922-014-9336-2, http://dx.doi.org/10.1007/s10922-014-9336-2
39. Zhu, C., Leung, H., Hu, S., Cai, Y.: A q-values sharing framework for multiple independent q-learners. In: Int. Conf. on Autonomous Agents and MultiAgent Systems. pp. 2324–2326 (2019), http://dl.acm.org/citation.cfm?id=3306127.3332099