Applying Reinforcement Learning to Packet Routing in Mesh Networks

George Horrell
Stanford University
ghorrell@stanford.edu

Eli Echt-Wilson
Stanford University
eliew@stanford.edu

Kevin Chang
Stanford University
kevchang@stanford.edu

Abstract

In this paper we explore reinforcement learning approaches for packet routing in mesh networks and situations where nodes do not have access to global network information. We also implement extensions to Q-routing in an effort to improve performance, including packet dropping penalties and dual-reinforcement learning. We also propose a hybrid algorithm for transitioning standard protocols to reinforcement learning, mitigating the poor performance of the "learning period" of Q-routing. We find that reinforcement-learning techniques successfully and consistently outperform standard routing protocols for both static and dynamic mesh networks.

1. Introduction

Decentralized networks, such as mesh networks, are becoming increasingly viable and attractive alternatives to existing networks due to the increased connectivity of devices, looming privacy concerns with highly centralized networks, and the sheer amount of latent and unused computing power available in everyday devices. Current industrial applications being explored include Google Loon [3] and Facebook's "Next Generation Data Network", while general applications include connecting isolated locations, developing countries and swarms of intelligent devices [4].

Routing packets through mesh networks is a difficult problem since each node in the network only has information about its directly-neighbored neighbors and there exists no central machine that has global information about the network. In this paper, we explore using reinforcement learning algorithms to "learn" about the network topology and make intelligent routing decisions under uncertainty. The goal is to create a reinforcement learning algorithm capable of outperforming standard routing protocols.

2. Modeling Mesh Networks and Reinforcement Learning

For the purposes of this paper, we decided to model our mesh networks using software-based simulations rather than setting up a real mesh network with hardware components. The software-based approach is preferable to setting up a system of routers given our time and budget constraints. Nevertheless, we invested significant work into making the model as reflective of a real mesh network as possible.

Listed below are properties of mesh networks which make them extremely suitable for reinforcement learning (RL) based approaches to packet routing. When designing our model, we kept the following unique properties of mesh networks in mind.

1. **Highly connected.** Standard network topologies, such as the internet, form a star where a small minority of centralized nodes handle the majority of traffic and route packets to the rest of the network. While the internet is strictly a mesh structure, it follows a power-law distribution where the vast majority of nodes have very few connections, allowing routers placement at high-degree nodes to be a viable mechanism for path-routing. The above does not hold for most mesh networks such as those employed in Project Loon, or those that supply internet to remote locations. These graphs can be modeled as a Watts-Strogatz random graph [7] and possess relevant "small work qualities" with short average path lengths and high clustering. Since there are a number of short paths between nodes, we hope that our RL approach will more accurately optimize for transmission time over number of hops.

2. **Low-to-medium end hardware.** ISPs and other centralized packet switching hubs use high throughput

hardware to power incredibly fast packet processing and switching. In contrast, hardware for mesh networks is often less powerful, subject to strict constraints on power usage, size and heat output since the packet routing hardware might be embedded in a balloon, or running in a civilian's house. Our model is sensitive to over-loading specific nodes: algorithms that consistently exploit certain routes will fare worse than algorithms that distribute load. An well-tuned RL algorithm is capable of more evenly distributing load throughout the network.

3. **High variability edge costs.** Mesh networks communicate through a range of media, including radio, infrared and wired connections. All of these media have very different transmission speeds and can suffer from high variability. This variability is a compelling reason for why reinforcement learning is suitable for mesh network packet routing. Traditional packet routing algorithms, such as RIP, compute a shortest hop path but will fail to learn which edges are consistently reliable. Reinforcement learning algorithms can more accurately learn which edges should be routinely exploited.

4. **Node reliability.** Within traditional star topology networks (the internet) it is highly unlikely that high throughput switches will go offline. These are industrial machines that are relied upon to provide connectivity to thousands of leaf devices. It is also unlikely that new high-throughput non-leaf nodes will be added to the network with high frequency. However, in a mesh network, it is much more likely that nodes will both go online and offline regularly, and with relatively high frequency. It is then critical that the packet routing algorithm is capable of responding to these changes quickly, something traditional algorithms like RIP are not capable of doing well. However, reinforcement learning algorithms can continue to adapt to changes within the network.

3. Baseline Algorithms

In order to assess the efficacy of our reinforcement learning solutions, we implemented two baseline packet routing algorithms that do not rely on the learning of network parameters.

3.1. Random Routing

The first baseline algorithm that we implemented was random routing. This algorithm selects a neighbour at random from the current node and continues to make hops through the network until the destination node is reached. While this naive approach does not typically result in fast transmission times, it does distribute load evenly throughout the network.

3.2. Routing Information Protocol

Our second baseline algorithm is far more effective than random routing, and is one of the most persistent routing protocols in existence. Routing Information Protocol, or RIP, relies on a table pre-computed by the Bellman-Ford equation [2]. This protocol was used for the first generation of computer networking, ARPANET, and is still in use today. For each node in the network i, a table T of size n (where n is the number of nodes) is computed. If we consider that the shortest path from node i to node j is $k_1, k_2, \ldots, j$, then:

$$T_i(j) = k_1$$

Following this algorithm provides a shortest-hop path from i to j, however it does not accurately account for stochasticity in edge cost, and the other factors discussed above. To improve on this naive algorithms for mesh network packet routing, we explore reinforcement learning approaches.

4. Reinforcement Learning Algorithms

4.1. Q-Routing Algorithm

The Q-routing algorithm that we started with uses the Q-learning framework to make routing decisions at a node based on routing information at the neighboring nodes [1]. The algorithm maintains a table of Q values which estimate the quality (or approximate travel time) of the available neighboring nodes at a given node. We update the Q values each time we send a packet to a neighboring node, based on the travel time incurred by traversing the edge. As the network routes more and more packets, it is able to gradually approximate and improve global information about the network. Additionally, it is able to react and adapt to changes in the local network topology.

4.2. Vanilla Q-Routing

In vanilla Q-routing, we are sending each packet P from a source node s to a destination node d. At each step in the process, we route P from some node x to a neighboring node y. We can then approximate the time remaining t for the packet with:

$$t = \min_{z \in N(y)} Q_y(z, d)$$

with $N(y)$ being the set of neighbors of node y. Given this remaining time estimate t and the time spend traveling from x to y (call this s, we can perform the following incremental update for Q_x:

$$Q_x(y,d) = Q_x(y,d) + \alpha(t + s + Q_x(y,d))$$

where α is the learning rate hyperparameter.

Nodes in a mesh network occasionally drop packets, which means that a packet traveling from node x to node y fails to reach node y. We account for this in our Q-routing algorithm by detecting when a packet is dropped from x to y. If the packet is dropped, we revise our estimate of time remaining to be the minimum travel time from the source node s instead of y, since the packet will have to restart at its source node:

$$t = \min_{z \in N(s)} Q_s(z,d)$$

With this addition, the Q-routing algorithm should penalize nodes that drop packets more often and learn to avoid nodes with higher drop rates.

In order to balance exploration and exploitation in the Q-routing algorithm, we use an epsilon-greedy approach. With probability $\epsilon = 0.05$, we pick a random neighbor to route a packet to from node x. With probability $1 - \epsilon$, we route the packet to the neighbor of x with the smallest Q value (smallest estimated time remaining).

4.3. Dual Reinforcement Q-Routing

As an extension to vanilla Q-routing, we implemented an extended version of Q-routing motivated by dual-reinforcement learning, which was developed in the late 20th century for satellite communication systems. While standard Q-routing performs forward exploration only to update our table of estimated travel times, dual-reinforcement Q-routing tries to learn good estimates more quickly by also performing backward exploration. This means that when we route a packet from node x to y, we perform two updates of the Q table. The first update is the standard Q-routing update (detailed above), which improves the estimate of travel time for a packet going from node x to the destination node through node y. For the backward exploration update, we seek to improve the estimate of travel time for a packet going backward from node y to the source node through node x (since the edge latency between pairs of nodes should be the same in both directions). We first approximate the estimated time from node x to the source node with:

$$t_b = \min_{z \in N(x)} Q_x(x,s)$$

with N(x) being the set of neighbors of node x. Given this time estimate t_b and the time spent traveling from x to y (which we call s_b), we then perform the backwards exploration update for Q_y:

$$Q_y(x,s) = Q_y(x,s) + \alpha_b(t_b + s_b + Q_y(x,s))$$

with α_b being the backwards learning rate. This algorithm is intended to converge faster than vanilla Q-routing by performing multiple Q-updates each time we route a packet. [5]

4.4. Hybrid RIP / Q-Routing

Q-Learning noticeably suffers from performance degradation in its learning phase. What we propose is to interpolate RIP routing and Q-Learning by relying more heavily on RIP routing earlier in the learning period, and gradually weaning off and relying on Q-Learning as Q-values converge.

Let i indicate the current iteration in the Q-learning phase, ϵ be the probability that we explore, and ω be some iteration weighting factor. We will define π_{random} as the policy of randomly exploring neighbors, π_{RIP} as the RIP policy (following routing table), and π_Q as the Q-learning policy (finding neighbor with minimal Q). Further, let r be a randomly generated number where $r \in [0,1]$ and c be the current node.

$$\pi_{hybrid}(c,i,r) = \begin{cases} \pi_{random}(c), & r < \epsilon \\ \pi_{RIP}(c), & \epsilon \leq r < \frac{\omega - i}{\omega}(1-\epsilon) \\ \pi_Q(c), & r \geq \frac{\omega - i}{\omega}(1-\epsilon) \end{cases}$$

5. Implementation

For every set of simulations, we initialize a mesh network with specific node parameters and edge parameters and a set number of packets with specific packet parameters.

Importantly, the use of this network is persisted across multiple simulations of different packet routing algorithms. This ensures that networks behave similarly across simulations while still retaining stochastic behavior.

For example, if an edge drops a packet in the run of one packet routing algorithm, while it isn't guaranteed to drop a packet in the run of another packet routing algorithm on the same network, it will guarantee packet dropping on that edge with the same probability distribution.

Simulations of different packet routing algorithms are then run on this network by sending groups of packets in batches (2,000 at a time).

5.1. Edges

Each edge e_{ij} in the network is assigned a randomly generated base weight μ which represents the average transmission delay over the link - this ranges from 0ms to 300ms. Every time the edge is traversed, we generate a weight from the normal distribution (with $\sigma = 5$ms) as follows:

$$l_{ij} = N(\mu, 5)$$

Edges are also assigned a drop rate. Every time a packet traverses the edge, it is dropped with probability d_{ij}.

$$d_{ij} = U(0, 0.05)$$

5.2. Packets

Each packet is generated with a source node and destination node, and keeps track of its total time in transit thus far, as well as a history of nodes its traversed as it's been routed through the network. For the purposes of our simulation, the source node and destination node are never droppable-nodes (explained in 5.3).

5.3. Network

Each network is initialized with some proportion of nodes designated as ”constant-nodes” (generated using a Watts-Strogatz model) and some proportion nodes designated as ”droppable-nodes”.

After each batch of packets is sent out and processed, a droppable-node is selected and dropped (all its edges to other nodes removed). For the purposes of our simulations, it is guaranteed that none of the packets generated have sources or destinations that are droppable-nodes.

Each network is then initialized with three parameters:

1. **Number of Nodes**. Number of nodes that are in the network.
2. **Droppable Node Proportion**. Proportion of nodes in the network that are droppable-nodes.
3. **Droppable Node Connectivity**. The percentage of constant-nodes that each droppable-node will be connected to. For example, a drop node connectivity $c = .5$ means each droppable-node will have edges to 50% of all constant-nodes.

5.4. Simulation Algorithm

Refer to `https://github.com/kevhcha/q-routing` for codebase. Overview of implementation in Algorithm 1.

Algorithm 1 Network Routing Algorithm

```
1: function ProcessBatch(batch)
2:     for each $pkt \in batch$ do
3:         queues[pkt.src] $\leftarrow$ Enqueue($pkt$)
4:     while queues not empty do
5:         for each $node \in nodes$ do
6:             ProcessNode($node$)
7: function ProcessNode($node$)
8:     pkt $\leftarrow$ Pop($queues[node]$)
9:     next $\leftarrow$ RoutePacket($pkt, node$)
10:    if packet.time $\leq$ TIME_OUT then
11:        queues[next] $\leftarrow$ Enqueue($pkt$)
12:    if queues[cur] empty then
13:        Delete($queues[cur]$)
14: function RoutePacket(pkt, $cur$)
15:    next $\leftarrow$ GetNext($cur$)
16:    pkt.time $\leftarrow$ pkt.time + edge latency
17:    if edge drops packet then
18:        pkt.dropped $\leftarrow$ true
       return next
```

6. Results

6.1. Methodology

To measure the performance of our reinforcement learning algorithms, we compare them against the baseline random routing and RIP routing protocols. For each algorithm, we measure the metrics of average transit time, average path length, number of dropped packets, and number of timed-out packets.

6.2. Vanilla Q-Routing

We find that, with enough packets sent, Vanilla Q-Learning performs consistently better than random routing and RIP protocols both on static and dynamically changing networks (with and without dropping nodes).

In Fig. 1 and Fig. 2, we compare one iteration each of RIP routing and Q-routing on the same static network of 30 nodes and 500,000 packets without node dropping. Notably, Vanilla Q-Routing outperforms RIP Router in the limit, but performs dramatically worse while it is still learning. Since the learning inefficiency is a one-time cost, however, Vanilla Q-Learning will continue to outperform RIP as the number of packets grows.

We see that by tuning our learning rates, we can decrease the duration of the inefficient learning period, achieving convergence with a learning rate of .05 nearly three times faster than with a learning rate of .09 (Fig. 3).

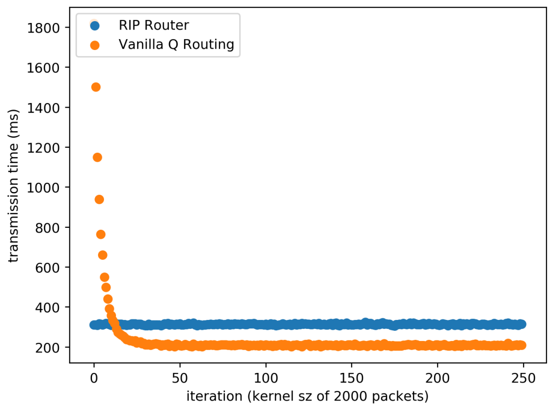

Fig. 1. RIP vs. Q-Routing, no dropped nodes.

	RIP	Q-Routing
Avg. Path Length	2.71	4.39
Avg. Transit Time	314.73	198.88
Timed Out Packets	0	0.00076
Dropped Packets	0.056	0.107

Fig. 2. Random vs. RIP vs. Q-Routing, no dropped nodes

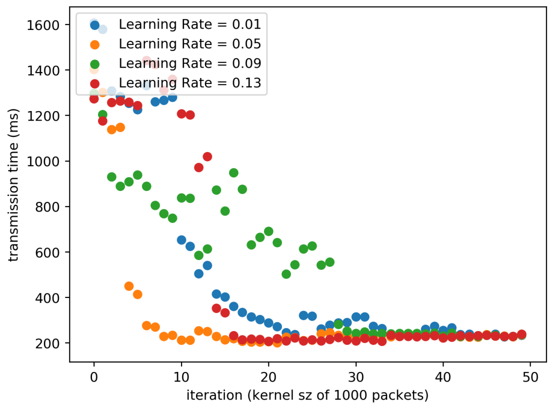

Fig. 3. Vanilla Q-Learning with different learning rates.

6.3. Dynamic Networks

The advantages of Q-Routing really begin to shine in more realistically-modeled networks, i.e. with dropped nodes. As seen in Fig. 4 and Fig. 5, Vanilla Q-Routing decreases transit time by nearly 27% as compared to RIP, and over 200% reduction in time out rates.

This makes sense intuitively since RIP routing follows a fixed policy derived from the initial network structure without the ability of adapting policies to changes in the network structure, such as dropped nodes.

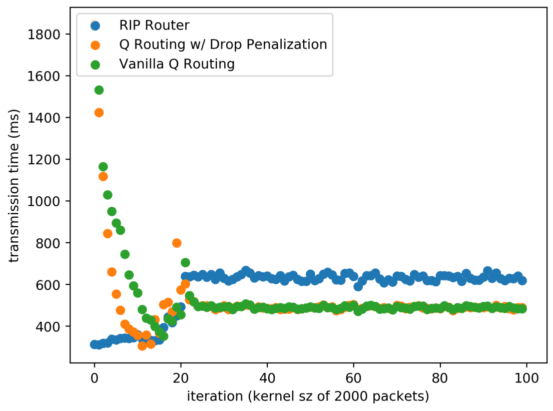

Fig. 4. RIP v. Penalized Q-Routing v. Vanilla Q-Routing

	RIP	Vanilla	Penal.
Path Length	3.623	3.721	3.729
Transit Time	705.124	508.977	505.071
Time Out Rate	0.046	0.002	0.001
Drop Rate	0.108	0.090	0.081

Fig. 5. RIP vs. Q-Routing with drop node penalty

As discussed earlier in 4.2, we can further optimize Vanilla Q-Routing by introducing a penalty for dropping nodes by rerouting packets back to their source node when dropped. We see that this Penalized Q-Routing provides a slight improvement over the Vanilla Q-Routing (Fig. 5).

6.4. Dual-Reinforcement Q-Routing

We find that, while dual-reinforcement Q-routing outperforms the baseline RIP routing protocol, it converges at the same rate and performs slightly worse than vanilla Q-routing (Fig. 6).

6.5. Hybrid RIP / Q-Learning

In our tests, we found Hybrid RIP / Q-Learning to perform better than both RIP and Q-Routing, especially with fewer number of packets. Notably, it provided the minimum across all metrics: path length, transit time, time out rate, and drop rate (Fig. 7 and Fig. 8).

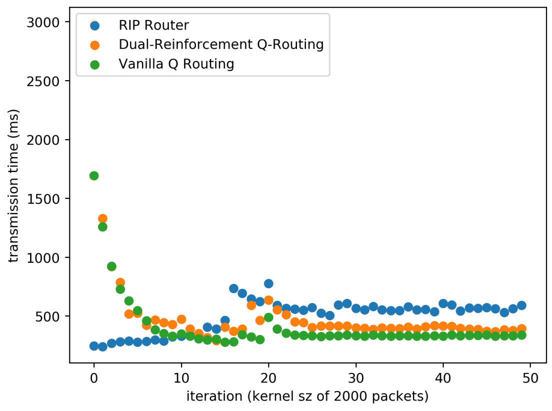

Fig. 6. RIP v. Dual-Reinforcement Q v. Vanilla Q

	RIP	Penal.	Hybrid
Path Length	4.19	6.34	3.77
Transit Time	457.08	395.50	312.58
Time Out Rate	.0074	.012	.00038
Drop Rate	.058	.076	.056

Fig. 7. RIP v. Penalized Q v. Hybrid Q

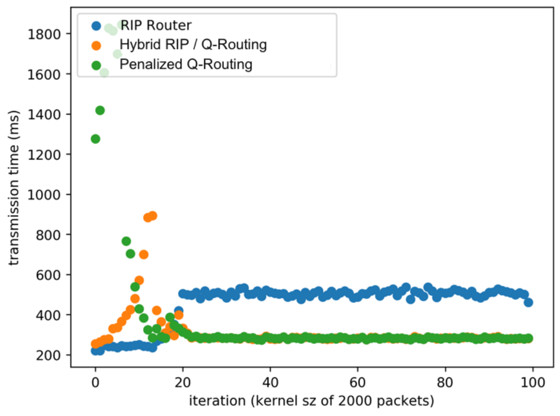

Fig. 8. RIP v. Penalized Q v. Hybrid Q

Conclusion

Our work considers and explores the viability of Q-learning for use on mesh networks as a more performant alternative to existing routing algorithms, and the preliminary results are promising. We have shown that Vanilla Q-learning is a promising alternative to RIP routing, and that with further modifications it can consistently outperform RIP routing in a variety of scenarios. Q-learning-based algorithms that are tailored to the constraints and behavior of specific mesh networks will likely perform even better.

Interesting future work include considering better interpolation mechanisms between Q-learning and RIP policies, extending to incorporate predictive Q-routing [6], and more holistic and accurate mesh network models.

Contributions and Acknowledgements

George: Led implementation. Optimized model to permit real testing, extended it to include packet load and dynamic graph structure. Implemented random routing, RIP-routing and Q-routing.

Eli: Led write-ups, including proposal, update and final paper. Wrote implementation of parallelized packet routing simulation, including re-routing dropped packets. Implemented dual-reinforcement Q-routing and penalizing packet dropping in Q-routing.

Kevin: Designed and implemented packet and mesh network architecture (modeling drop rate, latency, and time outs). Implemented random routing and RIP / Q-routing hybrid. Extended implementation to allow for easier data collection across multiple simulations.

References

[1] J. A. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in neural information processing systems*, pages 671–678, 1994.

[2] C. L. Hedrick. Routing information protocol. Technical report, 1988.

[3] S. Katikala. Google project loon. *InSight: Rivier Academic Journal*, 10(2):1–6, 2014.

[4] S. Katti, D. Katabi, W. Hu, H. Rahul, and M. Medard. The importance of being opportunistic: Practical network coding for wireless environments. 2005.

[5] S. Kumar and R. Miikkulainen. Dual reinforcement q-routing: An on-line adaptive routing algorithm. *Smart Engineering Systems: Neural Networks, Fuzzy Logic, Data Mining, and Evolutionary Programming*, 7, 1997.

[6] S. P. M. Choi and D.-Y. Yeung. Predictive q-routing: A memory-based reinforcement learning approach to adaptive traffic control. 8, 12 1999.

[7] D. J. Watts and S. H. Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440, 1998.