Reinforcement Learning for Network Routing

by

Hema Jyothi Yalamanchi

Major advisors:
Dr. Thinh Nguyen
Dr. Alan Fern

Committee:
Dr. Bella Bose

A PROJECT
submitted to
Oregon State University

in partial fulfillment of
the requirements for the degree of

Master of Science

Completed March 30, 2007
Commencement June, 2007
# Table of Contents:

Abstract .................................................................3
Introduction .............................................................4
Q-Routing .................................................................5
Predictive Q-Routing ..................................................6
Confidence-Based Q-Routing .........................................8
Dual Reinforcement Q-Routing ......................................10
Confidence-Based Dual Reinforcement Q-Routing ...............11
Experiments and Results ..............................................12
Conclusion ..............................................................18
References ..............................................................19
Abstract:

Efficient routing of information packets in dynamically changing communication networks requires routing policies that adapt to changes in load levels, traffic patterns and network topologies. Reinforcement Learning (RL) is an area of artificial intelligence that studies algorithms that dynamically optimize their performance based on experience in an environment. RL, thus, is a promising framework for developing adaptive network routing algorithms and there have been a number of proposed RL-based routing algorithms. In this project, we developed an infrastructure for evaluating RL-based routing mechanisms and use it to evaluate and compare a number of existing algorithms under various network conditions.
**Introduction:**

In communication networks information is passed on to distant nodes in the form of packets. Routing strategies play an important role in delivering these packets to avoid delays and congestion due to packet flooding. The routing policies should be able to adapt with the changes in network conditions such as load levels, traffic patterns and network topology.

In this project we show that the reinforcement learning framework helps improve adaptive routing algorithms. The algorithms learn on-line as they route packets so that any changes in the network conditions are reflected in the routing policies.

We developed a framework and code base for evaluating the performance of adaptive network routing algorithms that have already been proposed. The framework was developed using C# and it facilitates users to plug in different agents and tune the parameters in the network. Some of the reinforcement learning agents that have been implemented in this framework will be discussed further in the report.

**NOTE:** The algorithms discussed and explained in this report have been proposed by different authors (please refer the references section of the report). We have just implemented existing algorithms on our framework to verify their performance and evaluate our framework. In the conclusion the scope of the project and an idea for improving the performance of the network based on the algorithms is mentioned briefly.
**Q-Routing:**

Q-Routing is an adaptive routing algorithm that was proposed by Boyan and Littman in 1993/1994 to improve packet routing in communication networks. It is a distributed reinforcement learning scheme for packet routing in computer networks. It applies Q learning to adaptive network routing techniques to improve overall performance of the network in terms of average delivery time of packets under high traffic loads.

In this self-adjusting routing algorithm, Q learning modules are embedded in each node of the switching network. Local information is exchanged between nodes to keep track of estimated delivery time to nodes and updates are made on a regular basis. Routing policies lead to finding paths which require minimum delivery time. The policy’s performance is evaluated by the total time taken to deliver a packet.

The policy described by Boyan and Littman is as follows:

Each node keeps an estimate of how long it will take to reach a destination from each of its neighbors. This is implemented by maintaining a table at each node. The table holds values for every neighbor/destination pair \((y,d)\) and the values held tell us how long it will take for a packet \(P\) to reach destination \(d\) if sent through neighbor \(y\).

When a node \(x\) is asked to route a packet to destination \(d\), it sends the packet to the neighbor \(y’\) which is estimated to have the lowest delivery time to \(d\). After which \(x\) updates its policy, \(x\) queries \(y’\) to find out the estimated time that \(y’\) would take to deliver the packet to \(d\). Since \(y’\) is presumably closer to \(d\), its estimate is considered more accurate and thus is used to update \(x\)’s delivery time estimate.

**Algorithm Description**[1][2]:

Let, \(Q_x(y’,d)\) be the time that node \(x\) estimates it takes to deliver a packet \(P\) to destination \(d\) via neighboring node \(y’\), including the time that the packet spends in node \(x\)’s queue. When the packet is sent to its neighboring node \(y’\), \(y’\) sends back an estimate to \(x\), of the time remaining to reach the destination through \(y’\)’s best neighbor.
\[ Q_y(z',d) = \min Q_y(z,d), \]
where \( z \in \text{neighbors of } y' \)

Node \( x \) revises its estimate based on the time spent by the packet in its queue \( (q) \).

\[ \Delta Q_x(y',d) = \eta (Q_y(z',d) + q - Q_x(y',d)), \]
where,

- \( Q_y(z',d) + q \) is the new estimate,
- \( Q_x(y',d) \) is the old estimate and
- \( \eta \) is the learning rate parameter, which is set to 0.7 in the experiments.

The \( q \) value in node \( x \)'s \( Q \) table is updated as follows after node \( y \) receives the packet from node \( x \):

\[ Q_x(y,d) \leftarrow Q_x(y,d) + \Delta Q_x(y',d) \]
where, \( y = y' \).

This way the learning algorithm discovers an efficient routing policy without need for a centralized routing control system.

**Predictive Q-Routing [3]:**

Predictive Q-Routing was proposed by Choi and Yeung in 1996 to overcome the shortcomings of the basic Q-Routing algorithm. Q-Routing could not learn new policies under decreasing load conditions and could not adapt well to varying traffic patterns, topological changes and network load conditions.

Under low network load, the optimal policy would be the shortest path routing policy. However, when the load level increases, packets start queuing up along the shortest paths and this routing policy will no longer perform well. If these congested paths are not used for a period of time, the number of packets in their queues will decrease with time and the nodes will recover. These recovered nodes will become good candidates again and can be
used. Therefore, such nodes should be probed once in a while by sending packets to them, to check for their recovery. This controlled exploration frequency is important, as frequent probing would lead to increase in congestion along the already congested paths. The main factors that the probing frequency would depend on are the congestion levels, which are reflected by the Q-values in the nodes tables, and the processing speed or recovery rate of the path, which is a part of the learning process.

In this algorithm the best estimate and recovery estimate for every corresponding Q value are stored. These values help in deciding whether a route would recover with time at a given time step.

**Choi and Yeung’s Predictive Q-Routing Algorithm:**
PQ-learning is identical to Q-learning in the way that the Q function is updated, but the policy varies by the fact that it uses recovery estimates along with Q-values to pick the next best neighbor.

**Algorithm Description:**
Every node in the network maintains these four tables:
- \( Q_x(d,y) \) - delivery time estimate from node \( x \) to destination \( d \) via neighboring node \( y \).
- \( B_x(d,y) \) - best delivery time estimate from node \( x \) to destination \( d \) via neighboring node \( y \).
- \( R_x(d,y) \) - recovery rate for a path from node \( x \) to destination \( d \) via neighboring node \( y \).
- \( U_x(d,y) \) - last update time for path from node \( x \) to destination \( d \) via neighboring node \( y \).

These tables are updated after a packet is received by node \( y \) from node \( x \) as follows:

\[
\Delta Q_x(d,y') = \alpha \text{ (transmission delay} + Q_{y'}(d,z') + q - Q_x(d,y'))
\]

where,

- \( Q_{y'}(d,z') = \min Q_{y'}(d,z) \), and where \( z \in \text{neighbors of } y' \).
- \( \alpha \) is the learning factor set to 1 for the accuracy of the recovery rate.
- transmission delay is set to zero for simplification.

Here \( q \) is the queuing time at node \( y \). (Note that it does not affect the policy whether \( q \) is the queuing time at node \( x \) or node \( y \).)

\( Q_x(d,y) \leftarrow Q_x(d,y) + \Delta Q_x(d,y') \) where \( y = y' \).
\[ B_x(d,y) \leftarrow \min(B_x(d,y), Q_x(d,y)) \]
\[
\text{if } (\Delta Q < 0) \text{ then } \\
\quad \Delta R \leftarrow \Delta Q / (\text{current time} - U_x(d,y)) \\
\quad R_x(d,y) \leftarrow R_x(d,y) + \beta \Delta R \\
\text{else if } (\Delta Q > 0) \text{ then } \\
\quad R_x(d,y) \leftarrow \gamma R_x(d,y) \\
\text{end if} \\
U_x(d,y) \leftarrow \text{current time} \\
\]

The next hop of the packet is decided based on the routing policy described below:
\[
\Delta t = \text{current time} - U_x(d,y) \\
Q'_x(d,y) = \max(Q_x(d,y) + \Delta t R_x(d,y), B_x(d,y)) \\
y \leftarrow \arg \min_y \{ Q'_x(d,y) \} \\
\]

where,
\[
\beta \text{ is used to learn recovery rate and is set to 0.7 and} \\
\gamma \text{ is used to control decay of the recovery rate and is set to 0.9} \\
\]

The recovery rate is never positive, otherwise it might increase the predicted Q value without bound and the path can never be used again.

We will see in our experiments and results section that this algorithm performs better than the basic Q-Routing.

**Confidence-Based Q-Routing [4]:**

Confidence-Based Q-Routing is an extension of the basic Q-Routing algorithm and was proposed by Shailesh and Risto in 1998 to improve the quality of exploration. This is done by maintaining confidence values (C values) for each of the Q values. These C values measure how closely the corresponding Q values represent the current state of the
network. The learning rate in this algorithm is computed as a function of the confidence values for the old and estimated Q values for each update. If the old Q value has low confidence or the estimated Q value has high confidence, the learning rate is high. For example, say if a Q value of the neighbor has not been updated for a long time, decisions made based on this Q value might not be reliable, since it would have a low confidence value associated with it, the learning rate would be low as opposed to being high. The amount of update in a Q value depends on the learning rate and the learning rate varies as it is computed as a function of confidence values which again vary with time. These variable learning rates improve the quality of exploration.

**Confidence Measures:**

Each Q value Q_y(y,d) is associated with a confidence measure C_x(y,d). These measures hold values between 0 and 1. A value of 1 means that there is a maximum confidence on the corresponding Q value, since the Q value would have been updated recently and hence is likely to reflect the current state of the network. On the other hand, the value 0 means that the Q value is random and it has never been updated and does not reflect anything about the current state of the network. With this information, it can be said that C_x(y,y) = 1 (base case), since there is a full confidence on Q values of the form Q_x(y,y) (This is because these Q values are constant). The other C values are initialized to 0.

The update rules are given by:

\[ \Delta Q_x(y,d) = \eta(C_{old},C_{est}) \cdot (Q_y(z',d) + q - Q_x(y,d)), \]

where,

\[ C_{old} = C_x(y,d) \]
\[ C_{est} = C_y(z',d) \]

And the learning rate function is chosen based on the following rule:

**Rule 1:** Learning rate is high if confidence in the old Q value is low or/and confidence in the estimated Q value is high.

The function is implemented as follows:

\[ \eta(C_{old},C_{est}) = \max(C_{est}, 1-C_{old}) \]

where, \( \eta \) is set to 0.85.
Confidence estimates are updated based on the following rules:
All confidence values except the base cases are updated every time step to reflect the reliability of their corresponding Q values.

Rule 2(a): If the corresponding Q values are not updated in the last time step, every C value (except the base case values) decay with some decay constant \( \Lambda \in (0,1) \), in our experiments this value is set to 0.95.

\[
C_{\text{upd}} = \Lambda C_{\text{old}}
\]

Rule 2(b): If a Q value has been updated in the last time step, then the corresponding C value is updated based on the C values corresponding to the Q values used in the Q value update.

\[
C_{\text{upd}} = C_{\text{old}} + \eta(C_{\text{old}},C_{\text{est}})(C_{\text{est}} - C_{\text{old}})
\]

The idea and the Q updates are the same as the Q-Routing except that the learning rate in this routing technique varies increasing the exploration.

**Dual Reinforcement Q-Routing [5]:**

Dual-Reinforcement Q-routing was proposed by Shailesh and Risto in 1999. This algorithm uses backward exploration in addition to the forward exploration (as that of Q-routing), which speeds up the learning process. The backward exploration here is implemented appending information to the packets that are received by the nodes from their neighbors.

Backward exploration makes the adaptation more effective since:

1. It leads to two-fold increase in exploration as two Q values are updated with every hop that the packet makes, instead of one. This increased exploration leads to an increased speed of learning.
2. Also backward exploration is more accurate than forward exploration since in backward exploration, information about the path already traversed is propagated, instead of estimates of the remaining paths.

Say a packet $P$ is generated at node $s$. This packet carries to node $y$ (next hop/neighbor) the estimated time it takes for a packet destined for node $s$ from node $x$. $Q_x(z',s)$ is defined as:

$$Q_x(z',s) = \min Q_x(z,s),$$

where $z \in$ neighbors of $x$

The updates made are as follows:

$$\Delta Q_x(y,d) = \eta_f(Q_y(z',d) + q_x - Q_x(y,d))$$

just like in Q Routing and $\eta_f$ is set to 0.7.

and

$$\Delta Q_y(x,s) = \eta_b(Q_x(z',s) + q_y - Q_y(x,s))$$

where,

$Q_y(x,s)$ is the old estimate and

$Q_x(z',s) + q_y$ is the new estimate and $z'$ is the neighbor of $x$

$\eta_b$ is the backward learning rate set to 0.9 and

$q_y$ is the time spent in $y$’s queue.

We can clearly see from the updates that Dual Reinforcement Q-Routing’s learning speed will be faster than the basic Q-Routing.

**Confidence-Based Dual Reinforcement Q-Routing [6]:**

Confidence-Based Dual Reinforcement Q-Routing uses the advantages provided by the Confidence-Based Q-Routing and the Dual-Reinforcement Q-Routing. This leads to an improved quality and increased quantity of exploration.

The updates made are the same as Dual Reinforcement Q-Routing but with the learning rate modeled as a function of confidence values like in the Confidence-Based Q-Routing.
Experiments and Results:

Experiments have been carried out using the framework (discrete event simulator) that has been built. It models the routing of packets in the network. Packets are periodically introduced into the network at a random source node to a random destination node. Multiple packets at a node are stored in an unbounded FIFO queue. The total number of live packets in the network is set for different experiments. This determines the traffic load in the network.

At every time step, each node in the network takes a packet from the top of its queue and examines the destination and chooses the neighboring node based on the routing algorithm, and sends the packet to the chosen neighbor. A packet sent to the destination node is removed from the network.

Delivery time of a packet is the time it takes for the packet to get from the source to the destination. Average delivery time of the packets in the network are computed every 40 time steps. This is used as a measure to monitor the network performance while learning takes place. The performance of different routing algorithms is tested on a 6X6 grid network as shown below. Experiments have been conducted for different network loads and changing network topology to show the performance of the algorithms.
**Base Case scenario:**

In the legend:

1. CQ Routing is Confidence-Based Q-Routing
2. DRQ Routing is Dual Reinforcement Q-Routing
3. CDRQ Routing is Confidence-Based Dual-Reinforcement Q-Routing
4. PQ Routing is Predictive Q-Routing.

The total number of live packets maintained in the network is 100 and there are no changes in the network conditions (steady state).

All Q values for these experiments are initialized to 0.

In PQ-Routing, after 1500 time steps (after the initial exploration phase) we initialize values of the B-Table to the values of the Q-Table to get a good estimate of the best values possible for each route.
Comparing Q-Routing against CQ-Routing and DRQ-Routing:

From the graph above we can say that DRQ-Routing learns and converges a little faster than Q-Routing and CQ-Routing because of the increase in information exchanged about the network state.

Comparing DRQ-Routing against CQ-Routing and DRQ-Routing:

We know that CQ-Routing and DRQ-Routing are both better than Q-Routing during the learning phase. We can also say that CDRQ-Routing will perform better than CQ-
Routing and DRQ-Routing because both the components (CQ and DRQ Routing) contribute independently to the learning process in CDRQ-Routing. This can be proved from the previous graph; we can say that CDRQ-Routing learns an effective routing policy almost 35% faster than Q-Routing.

Comparing PQ-Routing against Q-Routing:

The above graph shows that PQ-Routing performs slightly better than Q-Routing even under steady state. In the following sections we will see that it outperforms Q-Routing when there are changes in the network conditions.

Implementing Epsilon and Boltzman Exploration techniques with Q-Routing:
In Epsilon Greedy Exploration, we select a small random value for ε and at every time step we select a random action with probability ε and a greedy action with a probability of (1-ε). In this experiment the value of ε is set to 0.01.

In Boltzmann Exploration, we select an action with probability as given below:

$$\Pr(a|s) = \frac{\exp \left( Q(s,a) / T \right)}{\sum_{a' \in A} \exp \left( Q(s,a') / T \right)}$$

where,
(T is the temperature and the exp is raised to the power of negative Q-values as smaller Q values are better). The value of T is set to 1 here.

From the above graph we see that there was not much difference when Epsilon and Boltzman exploration techniques are added to the Q-Routing algorithm. These techniques did not perform any better even under changing network conditions.

**Changing Topology:**

To show that Confidence-Based Q-Routing adapts faster than Q-Routing to changes in network topology, the following experiment has been conducted:

A link was added between node 3 and node 34 in the 6X6 grid network. The routing algorithms were allowed to learn for the first 5000 time steps until they converged. At time step 5000 the link between node 12 and node 25 was disconnected. The network load or the total number of live packets maintained in the network is around a 100 packets during this experiment.
We see that as soon as the link went down the average delivery time for both the algorithms increase as they try to adapt to the change in the network topology. However CQ-Routing adapts faster and settles down in about 4000 time steps. Whereas, Q-Routing takes about 8000 time steps to settle down, which is almost twice as much time as what CQ-Routing takes. The main reason for this is that alternative routes apart from node 12 to node 25 which were used to route to the destination were learnt faster in CQ-Routing due to improved quality of exploration.

**Load Changes and changes in traffic patterns:**
In this experiment the following changes to the network parameters were made with time: The first 2000 time steps were given for the initial exploration at a low load of 50 packets (i.e. the total live packets maintained in the network are 50). After this time period packets were sent from the bottom left to the bottom right of the 6X6 grid network. At time step 3000 packets were sent from both the bottom left to bottom right as well as bottom right to bottom left of the network and the total number of active packets maintained during this period was 100.
Here we see that whenever the traffic patterns change PQ-Routing is able to adapt faster than Q-Routing. In PQ-Routing, nodes would probe different/alternative paths and this would lead to an increase in exploration. The faster convergence after changes in traffic patterns can be attributed to this fact.

We also saw that DRQ-Routing fails wherever Q-Routing fails as the algorithm is the same except that it exchanges more information about the network for every hop that a packet takes. Hence, CDRQ-Routing does not outperform CQ-Routing when there are changes in the network. It performs well only under steady state conditions.

**Conclusion:**
We implemented some of the RL based routing algorithms in the literature using our framework. There is a consensus between the results. These algorithms prove to work well for some network conditions. However we noticed that some of the algorithms (PQ-Routing) are very sensitive to the way the parameters are initialized, which makes them hard to use and these might not work in all network conditions.
From reading and analyzing the above algorithms, it can be said that by combining the advantages of CDRQ-Routing with PQ-Routing there are chances that the performance might be further improved and can be used for different changes in the network conditions.

References: