Packet Routing using Reinforment Learning-theory and simulation

Lecturer: Yishay Mansour *Writer: Shlomy Boshy*

1.1 Packet Routing Using Reinforcement Learning: Estimating Shortest paths in Dynamically changing networks

1.1.1 Abstract

This paper describes work done in the field of packet routing in dynamically changing networks,using reinforcement learning methods.
It summaries three major papers in the field and discusses the Q-Routing algorithm introduced by Littman and Boyan [1,2].
Other variants and improvements offered by Bates[3] are presented next.
Later various improvements to the algorithm are considered : Predictive Q-Routing [4] and and Dual Q-Learning [5], and their properties are discussed.

A packet routing algorithm chooses the best neighbor to send data packets through, in order to minimize transfer time in the network.
A reinforcement learning packet routing algorithm uses a distributed reinforcement learning module embedded into each node, to reach minimal overall routing times using only local information. The algorithm "learns" to switch to less-crowed paths and thus becomes superior to any static algorithm when used in a dynamically changing networks or in high load conditions.
There is a discussion on experiments and results from the papers, compared to the precomputed shortest path algorithm.

The paper concludes with a full implementation of a distributed network simulation using the reinforcement learning algorithms presented.

1.1.2 Introduction

Anyone who had been surfing the Internet lately knows that one of the most crucial performance issues in computer science applications today is efficient transport of

information through communication networks. Information must be passed between nodes quickly and reliably.
Farther more, the network is huge and dynamically changing. The task of finding static shortest paths in the network can no longer be achieved. The service time of servers, or even the existence of new paths, can change dramatically within short periods of time.

Most of network activity today is passed using "packets", which are small segments of messages which can be passed from computer to computer until the destination is reached. At each node, the decision must be made to which neighbor the packet must be sent next. This process is called *Packet Routing* .
The delivery time of a packet depends mainly on the total time it spent on message queues in the way .An inefficient routing policy can have a drastic effect on a network's performance. Congestion can result in huge delays.
The papers [1-3] present a dynamic algorithm, which "learns" the network and performs the routing policy. Papers [4-5] present algorithms, which improve its performance. These algorithms use a learning method called *Reinforcement Learning* .

Reinforcement learning algorithms enables an agent to change its behavior due to experiment results and improve it's performance. The agent gets a reinforcement signal with which he can update its behavior. Such an algorithm can adjust to a dynamically changing network by finding new, statically longer but now empirically shorter, paths thus avoiding congestion in the older shortest-paths.

A global observer could have an optimal policy of always sending the packet through the path with minimum total routing time. But in real network such a global approach is not practical. A node usually has only local information about it's neighbors.
Thus, the aim is to minimize total routing time using only local information by experimenting with different routing policies and collecting statistics on which policies did minimize total routing time.
The algorithms are compared to the static shortest-paths algorithm under various loads.

Some issues making this problem difficult are worth noting:

- We aim to achieve global minimum (the average packet delivery time of all packets) using only *local* information.
- The algorithm has no (or little) knowledge of the environment.
- There is no immediate "training signal" until the packet arrives its final destination. Even than, such a signal would have to travel back all the nodes to update estimates by it, thus consuming a lot of network resources.

- It is not known which particular decision in the routing sequence deserve credit for the total performance, and how much. This is called *the Credit Assignment problem* .

The network is constructed from a distributed collection of learners, each of which is responsible for solving a local part of the problem. The information "propagates" through the network , hoping to give a global solution for the whole network. The paper would consider the tradeoff between exploration and exploitation idea for avoiding getting "stuck" in a local minimum.
Note that the learning algorithm has no (or little) knowledge of the environment. Here the network topology and the input rates are not known and can even change dynamically during a run. The algorithm will adapt to the new settings. This generality makes the reinforcement learning approach very powerful in a huge dynamic network, such as the Internet, for example.

Apprioric knowledge of the environment can be given to a reinforment learning algorithm in order to improve results in specific networks (for example a small intra-net for a company). The learning is performed online, and uses online-policy algorithms. This means the algorithm can choose the policy taken for each state (which way to send a packet reaching it's node). These two features combined with the distributed implementation, using only local knowledge to reach a global understanding of the environment, make this approach a realistic routing policy for huge and dynamic networks.

1.1.3 Presenting Packet Routing as a Reinforcement Learning problem

The aim is to decide for each node, to which adjacent node to send a packet in order to get it as quickly as possible to its final destination.
The decision is made using only local information contained in a node.
Each node contains a message queue, which could be implemented in various ways (FIFO-queue, priority queue,etc.). The time messages spend in message queues of crowded nodes is the time the learning algorithm can save by choosing a new, currently better, path for packets.
Since the evaluation is measured by total delivery time, the node has no immediate reward (reinforcement signal). The node does not know if it made the right decision until the packet reached its destination.
We overcome this problem by "propagating" the signal information throughout the network.
Let us define the network routing problem as a reinforcement learning problem:

The state space

The state contains:

- The node in which we are in when receiving the packet.
  A distributed policy will have an agent in each node solving the local problem for that node.

- The final destination node of the packet

This means a node x will keep an estimate for every neighbor/destination pair (y, d) of how long a packet with destination d will take to get there,if sent through neighbor node y.
The estimate will be denoted as $Q(y, d)$. Note that $Q(d, y)$ (in opposite order of indices for comparison only) has the same meaning here as the $Q(s, a)$ in the *Q-Learning* algorithm in reinforcement learning[6].
Here the destination d (and the node we are in) represent the state, and the chosen neighbor y is the action.
The lookup table $Q(y, d)$ can be represented as a neural network for large state spaces.
Here, however, it is represented in a table.
Other parameters such as the time of day or local queue size could be added to the state and improve results in real networks.

The actions

The action would be choosing one of the neighbor nodes and sending the packet through it.
A node x would choose, for a packet with destination d ,the neighbor which minimizes the estimated delivery time to d.
Hence, node x will send a packet with destination d through:

$$\bar{y} = argmin_{y \in neighbors\ of\ x}[Q_x(y, d)]$$

The neighboring node is probably closer to the final destination, so its estimate is considered better and thus can be used to update the sending node's delivery time estimate.

Updating The Estimates

$Q_x(\bar{y}, d)$ is node x's estimate for the minimum delivery time to d (through $\bar{y}$).
When sending a packet to $\bar{y}$,x immediately gets back $\bar{y}$'s estimate for the remaining delivery time.
This estimate will be achieved using the same minimizing mechanism. Node $\bar{y}$ will choose a neighbor $\bar{z}$ that minimizes delivery time to d:

$\bar{z} = argmin_{z \in neighbors\ of\ y}[Q_y(z,d)]$

Note that the estimate could be sent back to x using a separate control channel, thus not affecting packet routing performance.

Now node x can update the estimated delivery time to d through $\bar{y}$, by combining the new result with the privously known knowledge:

$$Q_x(\bar{y},d) \longleftarrow Q_x(\bar{y},d) + \alpha[\overbrace{Q_{\bar{y}}(\bar{z},d) + t + q}^{new\ estimate} - \overbrace{Q_x(\bar{y},d)}^{old\ estimate}]$$

when t is the transmission time between nodes x and $\bar{y}$, and q is the time the message spent waiting in the queue.

We can consider the network as a graph in which an edge directed from node x to node y is weighted with the transmission time plus time spent in the queue, i.e. $t + q$.
In this network the optimal routing policy follows the shortest (weighted) paths through the graph. Hence the routing policy achieved can be compared to results of the static shortest-paths algorithm.
α is the learning rate, which is discussed next.

The Learning Rate

α is the "learning rate" , which is the "weight" we give the new "experiment" when combining it with the known estimate.
Many algorithms in reinforcement learning tend to use a learning rate which decreases as time passes, because the known estimates become better in time, as they are based on more samples.A single (new) sample has an error (variance) bigger than a well-established estimate over many samples.
This, although, is not the case here. The network is constantly changing, so we have to keep learning in the same rate. The known estimate's samples could have been taken from a different network, creating a biased,incorrect estimate.
So a new sample should always have the same weight. Thus we use a constant α .
Littman and Boyan, for example, used $\alpha = 0.7$ or $\alpha = 0.5$ in their experiments.

Note that the reinforcement learning approach to the routing problem could be seen as using a form of the Bellman-Ford shortest-paths algorithm, in which the relaxation steps are asynchronous and online.
Another difference between the algorithms is that here we measure total delivery time, and not the number of hops in the delivery.

1.1.4 The Q-Routing Algorithm

The Q-Routing algorithm is a variant of the Q-Learning reinforcement learning algorithm[6], used for the network routing problem.
Each node in the network contains a packet-routing module.
The routing module:

- routes messages arriving at the node to the neighbor with the smallest total delivery time estimate (as explained above).
- when a packet arrives, sends an estimate of the remaining delivery time back to the node which sent it.
- when receiving a reply from a neighbor, updates it's estimate for that neighbor, using the new estimate received (as explained above).

The estimates for the current node are kept in a lookup table $Q(y, d)$, which denotes the estimated total delivery time to destination d through neighbor y.

The Q-Routing basic algorithm

The routing module performs the following algorithm:

while (forever)

1. select a packet from the queue (using queue policy:FIFO,priority, ...).
   Let d be the packet destination.
2. select $\bar{y}$, such that $\bar{y}$ has minimal $Q(y, d)$ from all neighbors y of the current node.
3. wait for a reply from node $\bar{y}$
4. Update $Q(\bar{y}, d)$ (in the current node) using the new estimate from $\bar{y}$:
   $Q(\bar{y}, d) = Q(\bar{y}, d) + \alpha[Q_{\bar{y}}(\bar{z}, d) + t + q - Q(\bar{y}, d)]$
   (as explained above)

end while

When the node receives a packet it performs (as an interrupt to the while):

1. Receive message(packet) p (from node s)
2. select $\bar{z}$, such that $\bar{z}$ has minimal $Q(z, d)$ from all neighbors z of x.
3. send the value $Q(\bar{z}, d)$ back to node s.

4. place message p at the end of the message queue (using queue policy)

The "receiving packet" event will interrupt the while loop. This is done to minimize the time spent by nodes waiting for a reply (an estimate) from the current node. The reply is sent *before* the message enters the message queue. Thus the time waiting for a reply is neglected in this algorithm.
In my simulation I noticed that this time, and also reply sending times, can not immidiatly neglected.
Note that the distributed implementation should synchronize the events such the whenever the "receive" event occurs the Q table and the message queue are in a consistent state, thus reading correct values. This is not hard, however, since the update of the Q table is entry-local and can be made "atomic" to events.

The papers used a synchronic network, in which there are defined "time steps" in which each node performs an update, and a global "clock". In my simulation I used a distributed algorithm, assuming nothing on the responding time of other nodes. This assumption seems more realistic in huge networks today.

Estimates of current delivery times propagate rapidly through the network, passing as replies from node to node.
The space needed is little more than used to represent a complete routing policy. A clever implementation should keep for a node x the table $Q(y, d)$ only for neighbors of x and destinations that were already used.The hashtable implementation for lookup tables in my simulation presents such an implementation.
This idea becomes crucial in huge networks, since it is not possible for a node to have a table with an entry each node in the network.

Note that this basic algorithm is completely "greedy", and no exploration is done. Improvements to the algorithm discussed next will use various exploration methods to avoid getting "stuck" in a local minimum.

1.1.5 Exploration vs Exploitation

The basic Q-Routing algorithm is completely "greedy", because it always acts to exploit what it already knows (or believes it knows).
The algorithm only gets new information by actually "vising" a neighbor (sending a packet to it), meaning that it had decided that this neighbor has the minimal total delivery time. If another neighbor suddenly (by changes in the network topology or a congestion in the former path) becomes a better target for a packet, the basic algorithm would not notice it. It would keep sending packets only to the "best" (as he believes) neighbor, not updating

estimates for the others.
Only when performance of the "best" neighbors falls below the other nodes current estimate will the algorithm consider switching to the other paths.This behavior may cause the algorithm to settle for a suboptimal policy (a local minimum), thus damaging performance and learning ability.
One must remember, although, that obtaining further information about the network (by exploration) has a cost. Sending packets to gather information can result in increased congestion and thus result in a suboptimal,biased, routing policy.
A number of solutions to this problem were introduced.

Choosing some routing actions randomly

Route packets randomly with a small probability(say ϵ), and "greedy" in other cases. This approach is called an ϵ-greedy algorithm. The drawbacks are that the network is changing, so the learning period never ends, and random traffic has a very bad influence on congestion. Random packet also slow down other, greedy, packets. Because the nodes make decisions based on local information, the increased congestion may *change the problem* the nodes are trying to solve.

Constant pinging

in constant pinging,offered by Bates[3], each node receives an update from all its neighbors at every time step.
The node uses the updates as if they were replys in the greedy algorithm, updating the Q-values by the new estimates.
This approach achieves very rapid propagation of information but causes a big increase in the network load. It is not a practical method when a node can have a very large number of neighbors.
Note that update every "time step" has no meaning in an a-synchronic network.

a similar approach called *"full echo"* was offered by Boyan and Littman, but in full echo updates from all neighbors were made only when a packet was processed (and not in each time step). A node sending a packet also sends update-requests to all it's neighbors.
When possible, echo should be done on a separate channel, thus not affecting congestion in the network.

Variable pinging

In variable pinging a neighbor node y is queried for its estimates after a certain amount of time t has past since the last update from that node was received.

This method can reduce the amount of communication needed to gather information, while still gathering enough information on all neighbors to recognize better ones to switch to. Note that when t is small, the algorithm resembles constant pinging, and when t is large it resembles "greedy" Q-routing.

Confidence intervals in Q-routing

We could keep,in addition to the estimates for each neighbor, an estimate on the *confidence* we have in the estimate.
Using the confidence we can build an interval determining that the actual value is in that interval with some arbitrary probability.
A node can make a desicion based on the lower bound l of that interval, as determined by:
$l(x,d) = m(x,d) - t^{n-1}_{\alpha/2}\sigma(x,d)$
where $m(x,d)$ is the mean of samples of replies from node x and destination d, t is Student's function (as an approximation to the normal distribution),and $\sigma(x,d)$ is the samples's standard deviation.
We keep a list of past replies that are used as a sample. Limited space and time cause us to use only a constant "window" (for example, last 100 replies for each (x,d)).
This approach assumes that the samples are in a normal distribution.

The basic Q-routing algorithm can use this approach computing $l(x,d)$ of each neighbor each time a packet arrives, and choosing a neighbor that minimizes $l(x,d)$ (instead of $Q(x,d)$).
When a reply arrives a new sample would be added to the "window" for (x,d), updating the value of $l(x,d)$ in the future.

1.1.6 Predictive Q-routing

The Q-Routing basic algorithm has two problems:

1. The inability to fine-tune routing policies under low network load (because a shorter path may not be explored at all)

2. The inability to learn new optimal policies under decreasing load conditions (getting "stuck" on longer paths, that were less crowded when network load was high , instead of turning back to the shortest paths when load decreases)

Predicate Q-Routing[4] tries to solve those problems by keeping *best* experiences learned and reusing them by predicting the traffic trend.
The problems mentioned seem to come from the fact that basic Q-routing does no exploration. Littman and Boyan found random exploration to have a negative effect on

congestion. Predicate Q-Routing presents a way of exploration, without the cost of slowing down the network.
The idea of the algorithm is that shortest paths are a good solution when network load is low. When load increases we can learn better solutions, and when network load decreases we should check back if the shortest-paths are a good solution again.
The algorithm keeps the best Q-values and "probes" them occasionally (as controlled exploration), thus adapting to network trends. The probing frequency depends on the congestion level and the recovery rate of a path. The congestion level is reflected by the current Q-value, and the recovery rate is estimated in the learning process.

The predictive Q-Routing algorithm

definitions:

- $Q_x(d, y)$ - estimated delivery time from x to d via neighbor y
- $B_x(d, y)$ - BEST estimated delivery time from x to d via neighbor y
- $R_x(d, y)$ - recovery rate for path from x to d via neighbor y
- $U_x(d, y)$ - last update time for path from node x to node d via neighbor y

routing policy (in node x for a packet with destination d):

1. for each y compute: $\Delta t_y = current\ time - U_x(d, y)$
2. $y = argmin_y[max(Q_x(d, y) + \Delta t_y R_x(d, y), B_x(d, y))]$
3. send packet through neighbor y

when receiving a packet (when receiving reply from y at x):

1. $\Delta Q = [t + q + min_z(Q_y(d, z))] - Q_x(d, y)$
2. $Q_x(d, y) = Q_x(d, y) + \alpha \Delta Q$
3. $B_x(d, y) = min(B_x(d, y), Q_x(d, y))$
4. if $(\Delta Q < 0)$
   then
     $\Delta R = \Delta Q / (current\ time - U_x(d, y))$
     $R_x(d, y) = R_x(d, y) + \beta \Delta R$
   else
     if $(\Delta Q > 0)$ then $R_x(d, y) = \gamma R_x(d, y)$
   endif

5. $U_x(d, y) = current\ time$

Note how the recovery rate $R_x(d, y)$ is updated with each reply received, as the Q-value difference divided by the time passed.t is the transmission time, and q is the queueing time. There are three learning parameters in the algorithm:

- α - the Q function learning rate (as in basic Q-routing). In PQ-routing it is adviced that $\alpha = 1$ to avoid affecting the recovery rate.
- β - learning parameter for recovery rate. $\beta = 0.7$ was used in experiments.
- γ - used to discount the values of the recovery rate. $\gamma = 0.9$ was used in experiments.

PQ-routing updates the Q-values the same as Q-routing, but it also keeps best estimates. Better current estimates for Q-Values are achieved by adding the network changing rate factor. The best estimated values are automatically used if they are better that the Q-values estimates.
The exploration here is done by ocassionally "probing" the shortest paths.
PQ-routing needs more memory for recovery rate estimation (but only for the visited states). The memory is still in the same order for the original Q-routing memory requirements.

1.1.7 Dual Q-routing

Q-routing learns by forward exploration (when sending a packet forward it gets a reply and updates Q-values). Dual Q-routing tries to also learn by *backward exploration* - learning from information added to packets that the node receives from neighbor nodes.
When a node x sends a packet to a node y, the packet carries some value information of node x. When node y receives the packet, it can update it's estimates according to it.
This way a node performs backward exploration and forward exploration (with replies from nodes that y sent a packet to, as in Q-routing), thus increasing the speed information propogates through the network, and getting better estimates faster. The only overhead is a slight increase in the size of a packet.

Let s denote the original source of packet p, which is currently in node x. This packet carries to node y the estimated delivery time to s through x, that is $Q_x[\bar{z}, s]$, defined as:
$Q_x(\bar{z}, s) = min_{z \in neighbors\ of\ x}[Q_x(z, s)]$
with this information, node y can update it's own estimate:
$$Q_y(x, s) = Q_y(x, s) + \alpha[\overbrace{Q_x(\bar{z}, s) + t + q}^{new\ eestimate} - \overbrace{Q_y(x, s)}^{old\ estimate}]$$
where α is the learning time, q is time spent in the queue, and t is transport time.

The information about the path the packet traversed so far is used to update the Q-value of the receiving node. The source node s plays the same part as the destination d in forward exploration.

There are two advantages to using Dual Q-routing:

1. Updating Q-values in both forward and backward exploration increases learning speed.

2. Backward exploration is *more accurate*than forward exploration, because it propogates information on a path already traversed, instead of estimates of the remaining path.

1.1.8 Experiments and Results

The Q-Routing algorithm and its variants were tested by several researchers. The experiments and results are given here:

Network topologies

Boyan and Littman tested the basic algorithm on a variety of network topologies, including a 7-hypercube and a 116-node LATA telephone network. But specific results are only given for an *irregular 6X6 grid* . Bates [3] had only mentioned the 6X6 grid.
The 6X6 irregular grid has a two paths connecting the two left and right sides of the network: a shorter path and a longer path.A learning algorithm should at start use the shorter path, but when the load increases and this path gets crowded, the algorithm should be able to find the statically-longer but currently-shorter path and switch to it.
Dynamically changing topologies were tested by disconnecting links from the network during simulation.

Network Load

The network load was varied by changing the number of packets created in the network in each time step. For every network load the average delivery time for packets in the system was computed (after learning has settled on a routing policy), and compared with static routing by shortest-paths algorithm in the same topology and load.
Tests were made with traffic moving between upper and lower halves of the network, and between left and right halves of the network.
The total number of nodes in the network was limited (generally to 1000).
A packet sent directly to its destination node was immediatly removed from the network.

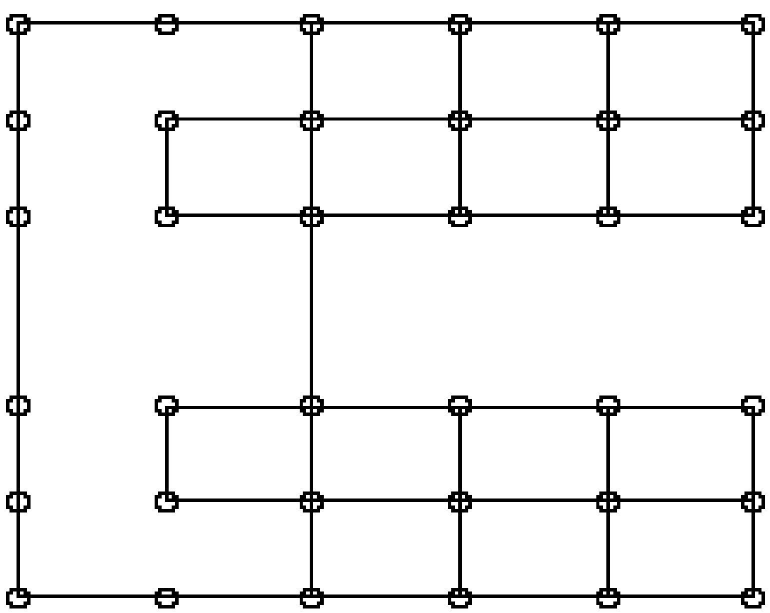

Figure 1.1: The irregular 6X6 grid topology

Bates assumed that transmissions between two nodes were instantaneous, so the queues could have an unlimited size.Network loads were varied from one packet added (at random) to the network every ten time steps, to four every single step.

Queues

All papers used only FIFO queues.
Boyan and littman used different local queue sizes. Bates used an unlimited local queue size.

Q-routing and Shortest Paths Comparison

According to Boyan and Littman's results, Q-routing was better than shortest paths underunder high loads. Under conditions of low loads the shortest paths algorithm gave better results.
Shortest-paths was proved optimal under low loads, But since Q-routing updates by the Bellman-Ford recurrence for shortest-paths we would expect similar results under low loads. The difference comes from the fact that Q-routing updates (in a relaxation stage) <u>only the best</u> estimate. In addition, QRouting has much more overhead.
A learning algorithm also needs an initial inefficient period, in order to learn the topology of the network.
As network load increased, shortest-paths ceases to be optimal - it ignores congestion and floods the popular paths with packets, waiting in endless queues. Q-routing, however, is able to switch to longer but less crowded paths and achieve better results under high network load.
Note that after a further significant increase in network load, the Q-routing algorithm also suffers from congestion.
The Q-routing algorithm addapts well to dynamically changing the network (by disconnecting links).It also addapted well to any traffic direction.
Bayes noticed that at the point the static shortest-paths algorithm "blows up" the Q-routing algorithm has a momentary rise, but but it learns a new policy and reaches better results.
When network load raised, basic Q-routing was able to adapt and switch to better paths. But when the load was lowered back, it got "stuck" in the local minimum, and didn't switch back to the shorter path. This was due to the fact (discussed earlier) that basic Q-routing is "greedy" and has no exploration machanisms.

Using "full echo" for exploration brought Q-routing to perform as well as shortest paths under low loads, and better than shortest paths in higher loads. In higher loads, however, "full echo" was not as good as basic Q-routing. The reason is exploration increases

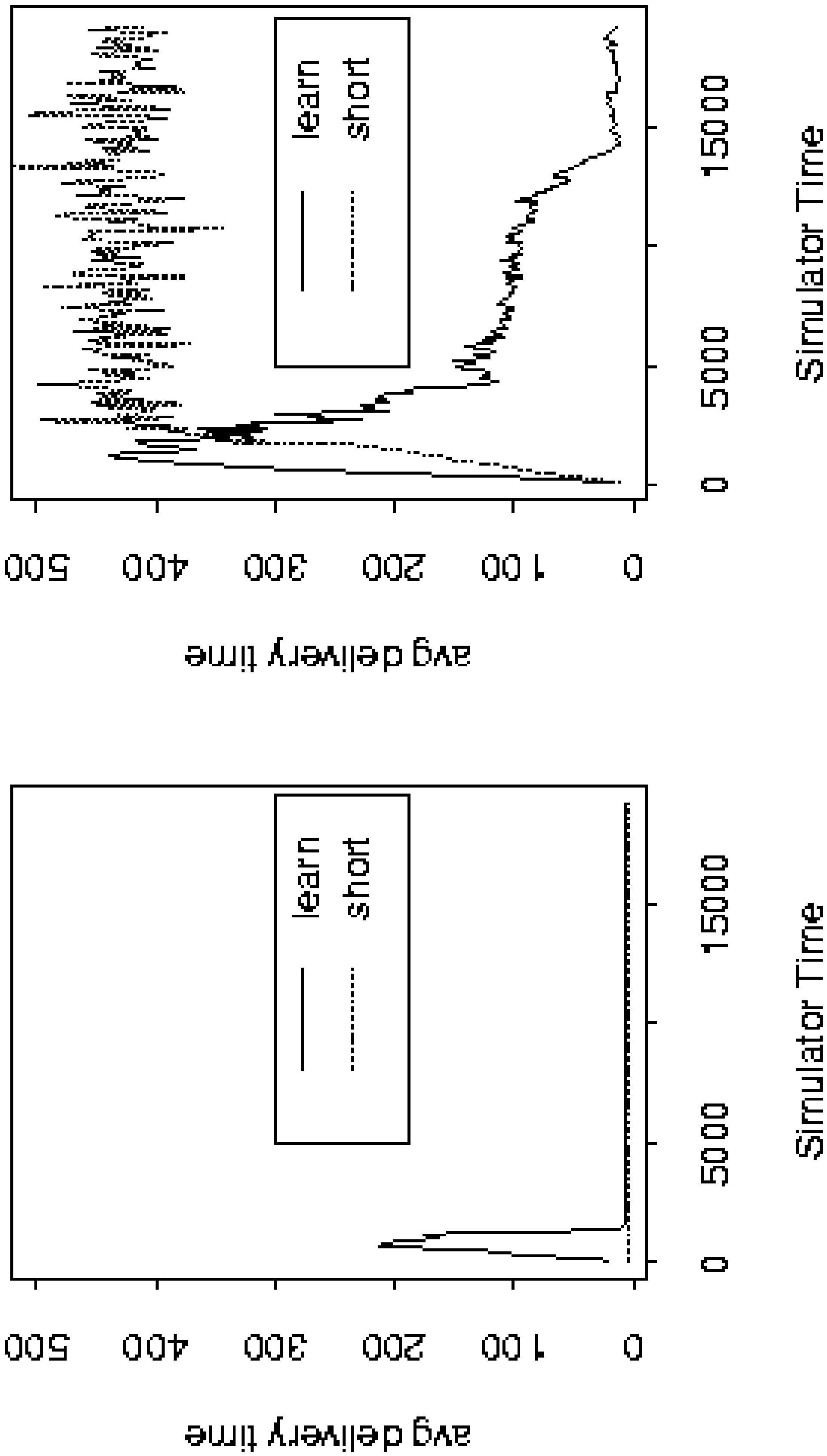
learn
short
avg delivery time
Simulator Time
0
100
200
300
400
500
0
5000
15000
learn
short
avg delivery time
Simulator Time
0
100
200
300
400
500
0
5000
15000

Figure 1.2:

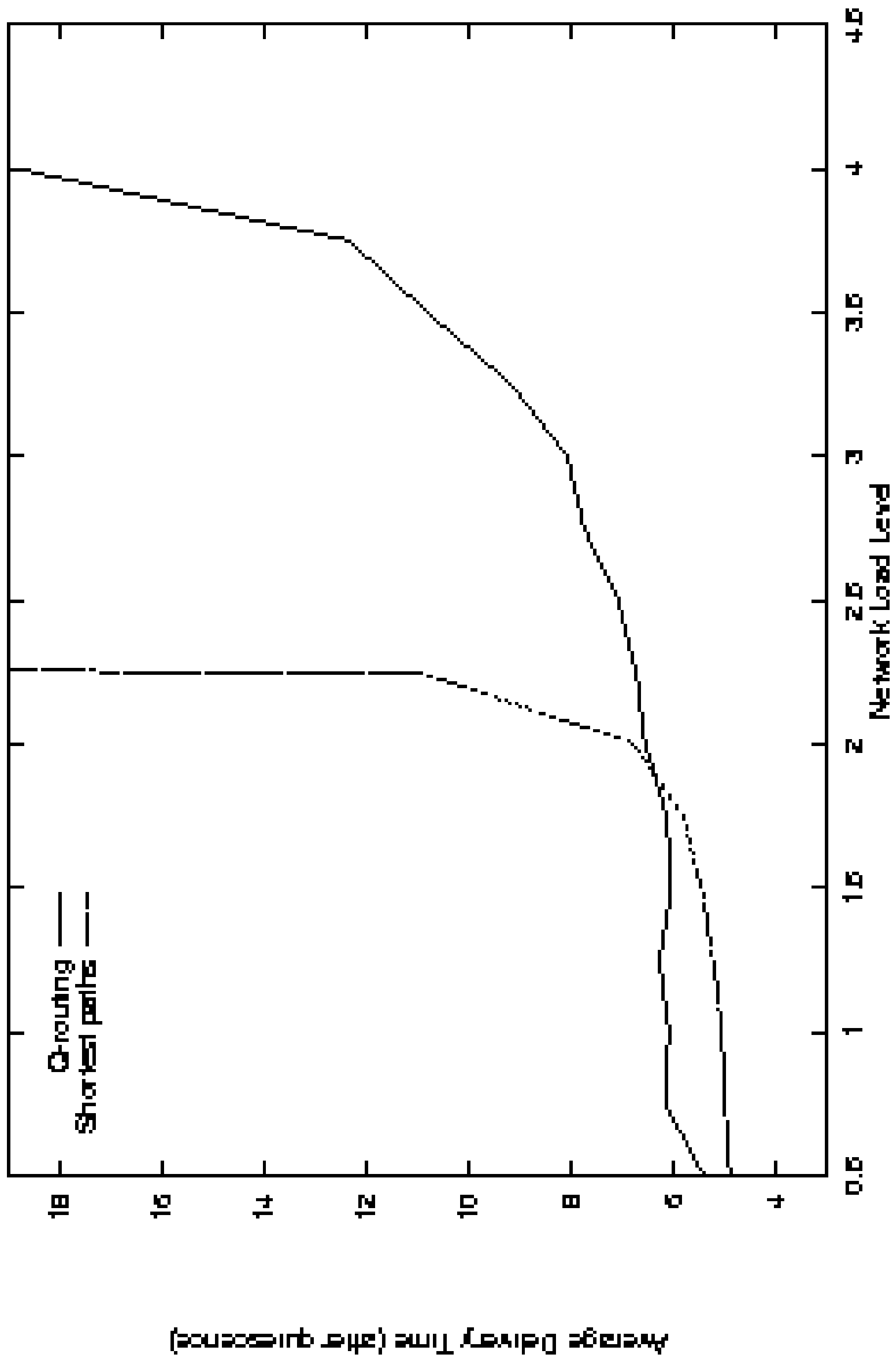

Figure 1.3: Delivery time at various loads for Q-routing and shortest paths

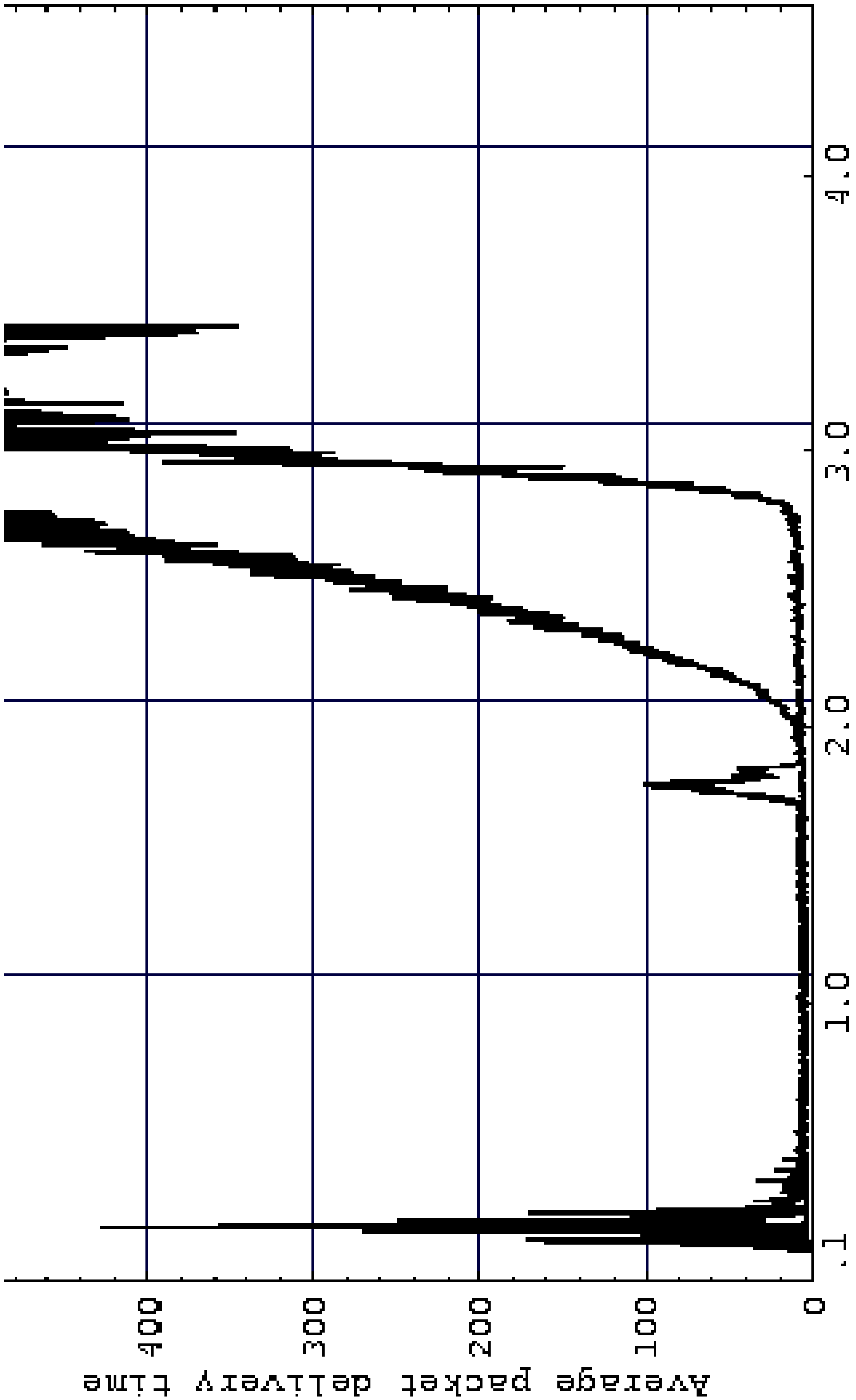

Figure 1.4: Baye's comparison of Q-routing and shortest-paths in various loads

congestion and takes time from greedy exploitation.
It was noted that "full echo" Q-routing constantly changed its policy under high loads, causing a "ringing" behviour.
Bate reports his "Constant pinging" algorithm to have better results than Q-routing. He explains the difference in the results from "full echo" (a similar approach) by stating that propagating at each single step (and not only only when a packet is sent) prevents the instabilities in routing policy selections, and thus gives better results.

Variable pinging had given good results when t was low (in this situation it resembles the constant pinging approach), but performence decreased when t became bigger, and it performed even worse than basic Q-routing.
Using confidence intervals (Bates[3]) yielded mixed results.It was able to sustain a higher load than Q-routing, but was slower to learn and adapt new paths. When network changed fast, the algorithm sometimes got "stuck" on a policy, requiring a long time to recover. When network was quite static it had equal or better results than greedy Q-routing.

PQ-routing [4] was shown to perform better than Q-routing after a short exploration phase (25 time steps), under low loads and under decreasing loads. The difference was bigger in periodic packet generation than in an aperiodic generation.
Dual Q-routing[5] experiments were done with unbounded FIFO queues, using periodic random insertion of packets to the network, in the 6X6 irregular grid topology. It was able to learn routing policies more than twice as fast as Q-routing at low loads, and learn routing policies as twice as good as Q-routing under high load (in terms of average packet delivery time). Further, it was able to sustain higher network loads.

1.1.9 Distributed Simulation of Network Activity: Implementation and Results

My simulation contains a distributed network structure which is used to simulate network activity and routing policies under different routing algorithms, network topologies and input pattern. The modular structure of the simulation enables "pluging-in" of different topologies,algorithms and queue implementations with minimal changes in code. Currently I have implemented three routing algorithms: QRouting, Dual-QRouting and Bellman-Ford shortest-paths algorithm.
A system of debug flags system was added to give the ability to view execution on various aspects: view only final result, debug on routing phases only,debug on nodes and message transmissions and debug on result estimates.

Simulation Design

The network simulation is made of these modular Java classes:

Network.java - The main class which loads the network topology, runs the network nodes and packet creators and stops the nodes when the simulation ends.
Topology.java - network topology abstract interface. Two topologies are implemented: GRID6X6 (as in the articles) and a small demo 3-node network.
Node.java - A thread for each node. contains queues, a routing module and a lookup table for the node. Performs routing and sends replies back.
PacketCreator.java - A thread that creates packets in a node,in three possible input patterns: random source and destination, to specific destination and from specific source to specific destination. Packets are sent it very high rate, to make queueing the most important issue in delivery time estimate.
lookupTable.java - A synchronized lookup table for Q-values. An abstract interface which two (and any other) classes implement: HASHlookupTable (with big overhead but only local knowledge) and ARRAYlookupTable (more efficient, but knows the number of nodes in the network,and demands more memory).
The reason to implement these two was to see if decreasing the overhead of learning would be critical for its success. The result, however, showed that changing input pattern was much more critical (in comparison to shortest-paths).
Queue.java - queue abstract interface. FIFOQueue class is implemented.
Router.java - routing module interface - gets best routing neighbor for destination and performs updates to Q-Values. QROUTINGRouter.java (which also DUAL-QRouting uses) and BellmanFordRouter.java are implemented.
BellmanFordEstimator.java - makes shortest-paths estimates.
Message.java - class implementing messages
The network does not change in my experiments, but adding or removing nodes is very easy to implement, given the current structure.

Distributed implementation

The simulation is a distributed collection of threads interacting with each other using messages.
Note that the articles used a global "clock" of time steps.In each step each node performs a routing step. The simulation,though, does not assume a global clock or time steps. A node sends a message to another node thus "interrupting" it to reply. No assumption is made on the response time of other nodes. This is a more realistic assumption in huge computer networks. I used RPC (a node calls another node's routine to place message) but any other implementation for 'sendMessage' would work. I have used the 'synchronized' feature of

routines in Java, making an object-lock which ensures only one thread is running the routine on the object at each time.

Reinforcement Learning Notes

A constant learning parameter $\alpha = 0.7$ was used. It is constant from reasons states above.
To gain a faster learning (and in comparison to shortest-path) I used the Bellman-Ford shortest-paths estimators as first estimates to the learning algorithms, which gave faster learning.
It seems that you have to know network structure to do it, but it is only a rough static transfer time shortest-path estimate, which can be given in realistic situation (for example, A web server in Israel has smaller transfer time from a server in Israel than a server in New-Ziland).
Anyway, the algorithm will also work with zero estimates at start,making it search through all possible routes before setting on a routing policy.

Simulation Results

Here are some results of the simulation.The numbers represent average packet delivery time, including queueing time and trasfer time (distance was set to 1 between each node to neighbor is my 6X6GRID). 3000 pakets were sent in a 6X6GRID topology. Note that the network load increases as time passes because packet-creator thread put packets into it much more quickly than they are delivered.
The aim was to make queueing the most important factor of the result.

Under random input pattern shortest-paths seemed to be much better:
Shortest paths=284.68
QRouting (using hashtable lookup table implementation)=740.69
DUAL-QRouting(with hashtable)=733.36
QRouting (using array lookup table implementation)=370.53
Note that DUAL-QRouting did not manage to help routing much in my simulation, and that changing lookup table implementation to decrease overhead helped to decrease average delivery time, but did not change the final results in comparison to shortest-paths.
I think the reason that shortest-paths was better under random packets is because not enough congestion was created at specific places in the network, so learning was not able to improve results,but its overhead was bigger.
The fact DUAL-QRouting did not help not improve result may be explained by the fact that DUAL-QRouting gives earlier estimates, which are thus based on less experiments.

When nodes were sent from one side of the network to the other (as in the articles),

QRouting algorithms performed much better than static shortest-paths, even with the less-efficient implementation of the lookup table:
(random sender,one destination)
Shortest paths=671.30
QRouting(hashtable)=532.48

(from the node at left-up to the node at right-down in the 6X6 grid)
Shortest paths=657.54
QRouting(hashtable)=48.37!!!

The reason learning became much better, is that when congestions appeared on specific "bridges", learning algorithms could learn to avoid those routes, while shortest-paths kept sending packets through the crowded path, increasing queueing time more and more.

1.1.10 Conclusion

In this paper I have presented learning algorithms which enable to improve packet routing in dynamic networks, without having to know in advance network topology and traffic patterns.
The algorithms performed packet routing control using reinforcement learning. By solving local per-node problems and propagating information it reached an efficient routing policy. Reinforcement learning algorithms were shown to be better than the static shortest-paths algorithm, espicially under high network loads.
The simulations done by Littman and Boyan were admitted to be unrealistic and used only specific topologies and traffic patterns. I tried to build a simulation that was a little more realistic, at least at being distributed with no global clock or "time steps". I still believe,as Littman and Boyan do, that the papers had shown that adaptive routing is a natural domain for reinforcement learning.

Experiments had shown that gathering more information every time priod (even at a cost of sending more packets) could have a drastic affect on improving the results. This result is familiar to reinforment learning: adding exploration helps to reach a "more global" minimum. Using confidence intervals, however, did not yet worth the additional overhead in Bayes's experiments.
Predictive Q-routing used exploration by "probing" the best (shortest known) paths, and thus improved results under low load, and lowered apdation time at decreasing network load.
Dual Q-routing added backward exploration to increase learning speed and provide more accurate estimates.It was shown to learn more than twice as fact as basic Q-routing,and sustain higher network loads, with a slight increase in packet size.I must note my simulation

should no improvement by using DUAL-QRouting.Maybe this was due to the fact that DUAL-QRouting gives earlier estimates, which are thus based on less experiments.

Farther investigation can include replacing the lookup tables with a function approximator (for example a neural network) and using it on actual huge networks. This would also allow adding more parameters to the decision process (such as the time of day, for example). Application of the presented algorithms could also be used to road traffic control and elevator control.

1.1.11 References

[1] J. Boyan and M. Littman, A distributed reinforment learning scheme for network routing. Technical Report CMU-CS-93-165, School of Computer Science, Carnegie Mellon University, 1993.
[2] J. Boyan and M. Littman, Packet routing in dynamically changing networks: A reinforcement learrning approach. Available for ftp from ftp.cs.cmu.edu, 1995.
[3] John W. Bates, Packet Routing and Reinforcement Learning: Estimating Shortest Paths in Dyanmic Graphs (jbates@encore.com),1995.
[4] Samuel P.M. Choi, Dit-Yan Yeung, Predictive Q-Routing: A memory- based Reinforcement Learning approach to Adaptive Traffic Control. Department of Computer Science, Hong Kong university of Science and Technology.
[5] Shailesh Kumar and Risto Miikkulainen, Dual Reinforcement Q-Routing: an on-line adaptive routing algorithm. The University of Texas at Austin , in Proceedings of Artificial Neural Networks in Engineering, 1997.
[6] C. Watkins , Learning from delayed Rewards, PHD thesis, King's College, Cambride,1989.