Confidence based Dual Reinforcement Q-Routing: an On-line Adaptive Network Routing Algorithm

Shailesh Kumar

Report AI98-267 May 1998

skumar@cs.utexas.edu
http://www.cs.utexas.edu/users/skumar/

Artificial Intelligence Laboratory
The University of Texas at Austin
Austin, TX 78712

Copyright

by

Shailesh Kumar

1998

Confidence based Dual Reinforcement Q-Routing: an On-line Adaptive Network Routing Algorithm

by

Shailesh Kumar, B.Tech.

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Arts

The University of Texas at Austin

May 1998

Confidence based Dual Reinforcement Q-Routing: an On-line Adaptive Network Routing Algorithm

Approved by
Supervising Committee:

Risto Miikkulainen

Raymond J. Mooney

Acknowledgments

I am very grateful to my supervisor Dr. Risto Miikkulainen for his insightful suggestions in both the content and presentation of this thesis. It was his encouragement, support and patience that saw me through and I am ever grateful to him.

I would like to thank Dr. Mooney for his comments and suggestions as a reader of the thesis.

I would also like to thank Dr. M. G. Gouda, Dr. G. de Veciana and specially Dr. Marco Schneider for their useful suggestions.

Thanks are also due to J. A. Boyan and M. Littman of Carnegie Mellon University and S. P. M. Choi and D. Y. Yeung of Hong Kong University of Science and Technology for their help and suggestions.

I am also very grateful to Dr. Joydeep Ghosh for his financial support and understanding throughout my work on this thesis.

Finally, I must thank Manish Misra for his tireless efforts in editing this thesis.

Shailesh Kumar

The University of Texas at Austin

May 1998

Confidence based Dual Reinforcement Q-Routing: an On-line Adaptive Network Routing Algorithm

Shailesh Kumar, M.A.
The University of Texas at Austin, 1998

Supervisor: Risto Miikkulainen

Efficient routing of information packets in dynamically changing communication networks requires that as the load levels, traffic patterns and topology of the network change, the routing policy also adapts. Making globally optimal routing decisions would require a central observer/controller with complete information about the state of all nodes and links in the network, which is not realistic. Therefore, the routing decisions must be made locally by individual nodes (routers) using only local routing information. The routing information at a node could be estimates of packet delivery time to other nodes via its neighbors or estimates of queue lengths of other nodes in the network. An adaptive routing algorithm should efficiently explore and update routing information available at different nodes as it routes packets. It should continuously evolve efficient routing policies with minimum overhead on network resources.

In this thesis, an on-line adaptive network routing algorithm called Confidence-based Dual Reinforcement Q-Routing (CDRQ-routing), based on the Q learning framework, is proposed and evaluated. In this framework, the routing information at individual nodes is maintained as Q value estimates of how long it will take to send a packet to any particular destination via each of the node's neighbors. These Q values are updated through exploration as the packets are transmitted. The main contribution of this work

is the faster adaptation and the improved quality of routing policies over the Q-Routing. The improvement is based on two ideas. First, the quality of exploration is improved by including a confidence measure with each Q value representing how reliable the Q value is. The learning rate is a function of these confidence values. Secondly, the quantity of exploration is increased by including backward exploration into Q learning. As a packet hops from one node to another, it not only updates a Q value in the sending node (forward exploration similar to Q-Routing), but also updates a Q value in the receiving node using the information appended to the packet when it is sent out (backward exploration). Thus two Q value updates per packet hop occur in CDRQ-Routing as against only one in Q-ROUTING. Certain properties of forward and backward exploration that form the basis of these update rules are stated and proved in this work.

Experiments over several network topologies, including a 36 node irregular grid and 128 node 7-D hypercube, indicate that the improvement in quality and increase in quantity of exploration contribute in complementary ways to the performance of the overall routing algorithm. CDRQ-Routing was able to learn optimal shortest path routing at low loads and efficient routing policies at medium loads almost twice as fast as Q-Routing. At high load levels, the routing policy learned by CDRQ-Routing was twice as good as that learned by Q-Routing in terms of average packet delivery time. CDRQ-Routing was found to adapt significantly faster than Q-Routing to changes in network traffic patterns and network topology. The final routing policies learned by CDRQ-Routing were able to sustain much higher load levels than those learned by Q-Routing. Analysis shows that exploration overhead incurred in CDRQ-Routing is less than 0.5% of the packet traffic. Various extensions of CDRQ-Routing namely, routing in heterogeneous networks (different link delays and router processing speeds), routing with adaptive congestion control (in case of finite queue buffers) and including predictive features into CDRQ-Routing have been proposed as future work. CDRQ-Routing is much superior and realistic than the state of the art distance vector routing and the Q-Routing algorithm.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Communication networks range from scales as small as local area networks (LAN) such as a university or office setting to scales as large as the entire Internet (Tanenbaum 1989; Huitema 1995; Gouda 1998). Information is communicated to distant nodes over the network as packets. It is important to route these in a principled way to avoid delays and congestion due to over flooding of packets. The routing policies should not only be efficient over a given state of the network but should also adapt with the changes in the network environment such as load levels, traffic patterns, and network topology.

In this thesis the Q learning framework of Watkins and Dayan (1989) is used to develop and improve adaptive routing algorithms. These algorithms learn "on-line" as they route packets so that any changes in the network load levels, traffic patterns, and network topologies are reflected in the routing policies. Inspired by the Q-ROUTING algorithm by Boyan and Littman (1994), a new adaptive routing algorithm, CONFIDENCE-BASED DUAL-REINFORCEMENT Q-ROUTING (CDRQ-Routing), with higher quality and increased quantity of exploration, is presented. It has two components (figure 1.2). The first component improves the quality of exploration and is called the CONFIDENCE-BASED Q-ROUTING. It uses *confidence measures* to represent reliability of the routing information (i.e. the Q-values) in the network. The second component increases the quantity of exploration, and is called the DUAL REINFORCEMENT Q-ROUTING (DRQ-ROUTING) (Kumar and Miikkulai-

nen 1997). It uses *backward exploration*, an additional direction of exploration. These two components result in significant improvements in speed and quality of adaptation.

The organization of the rest of this chapter is as follows. In section 1.1 the basic model of communication network used in this thesis is described in detail. Section 1.2 formally defines and characterizes *the routing problem* and various approaches to this problem. In section 1.3 the motivation for adaptive routing is given. A brief overview of the main contributions of this work and the organization of the rest of the thesis is given in section 1.4.

1.1 Communication Networks

A communication network is a collection of nodes (routers or hosts) connected to each other through communication links (Ethernet connections, telephone lines, fiber optic cables, satellite links etc.) (Tanenbaum 1989; Huitema 1995; Gouda 1998). These nodes communicate with each other by exchanging messages (e-mails, web page accesses, download messages, FTP transactions etc.). These messages (which could be of any length) are broken down into a number of fixed length packets before they are sent over the links. In general, and most often, all pairs of nodes are *not* directly connected in the network and hence, for a packet $P(s,d)$ to go from a source node s to a destination node d, it has to take a number of hops over intermediate nodes. The sequence of nodes starting at s $(= x_0)$ and ending at d $(= x_l)$ with all intermediate nodes in between i.e. $(x_0, x_1, x_2, ..., x_l)$ is called the route (of length l) that the packet took in going from s to d. There is a link between every pair of nodes x_i and x_{i+1} in this route, or in other words, x_i is a neighbor of node x_{i+1} for all $i = 0...l-1$.

The total time a packet spends in the network between its introduction at the source node and consumption at the destination node is called *packet delivery time* (T_D), and depends on two major factors (Tanenbaum 1989):

1. **Waiting time in intermediate queues**, T_W: When a node, x_{i+1}, receives the

packet $P(s,d)$ from one of its neighboring nodes x_i, it puts $P(s,d)$ in its packet queue while it is processing the packets that came before $P(s,d)$. Hence, in going from s to d, this packet has to spend some time in the packet queues of each of the intermediate nodes. If q_i is the queue length of node x_i when the packet arrived at this node, then the waiting time of the packet in node x_i's queue is $O(q_i)$. The total waiting time in intermediate queues is given by:

$$T_W = O(\sum_{i=1}^{l-1} q_i). \tag{1.1}$$

Thus for optimal routing it is essential that the packet goes through those nodes that have small queues.

2. **Transmission delay over the links**, T_X: As the packet hops from one node x_i to the next x_{i+1}, depending on the speed of the link connecting the two nodes, there is a transmission delay involved. If $\delta_{x_i x_{i+1}}$ is the transmission delay in the link between nodes x_i and x_{i+1}, then the total transmission time T_X is given by:

$$T_X = \sum_{i=0}^{l-1} \delta_{x_i x_{i+1}}. \tag{1.2}$$

Hence the length of the route l is also critical to the packet delivery time, especially when links are slow. In this work, all δ_{xy} are assumed to be same. Transition delays in actual networks are same for all links if the links are similar. Generalization to heterogeneous networks when the links have different transmission delays is discussed in section 7.2.

The total delivery time T_D is given by $(T_W + T_X)$ and is determined by the state of the network and the overall route taken by the packet.

1.2 The Routing Problem

Depending on the network topology, there could be multiple routes from s to d and hence the time taken by the packet to reach d from s depends on the route it takes. So the overall

goal that emerges can be stated as: *What is the optimal route from a (given) source node to a (given) destination node in the current state of the network?* The state of the network depends on a number of network properties like the queue lengths of all the nodes, the condition of all the links (whether they are up or down), condition of all the nodes (whether they are up or down) and so on.

If there were a *central observer* that had information about the current state (i.e. the packet queue length) of all the nodes in the network, it would be possible to find the best route using the WEIGHTED SHORTEST PATH ROUTING algorithm (Dijkstra 1959). If q_i is the waiting time in the packet queue of node x_i and δ is the link delay (same for all links) then the cost of sending a packet $P(s, d)$ through node x_i will add $(q_i + \delta)$ to the delivery time of the packet. The weighted shortest path routing algorithm finds the route for which the total delivery time of a packet from source to destination node is minimum. The complete algorithm, referred to as GLOBAL ROUTING hereafter, is discussed in chapter 2. It is the theoretical bound for the best possible routing and is used as a benchmark for comparison.

Such a *central observer* does not exist in any realistic communication system. The task of making routing decisions is therefore the responsibility of the individual nodes in the network. There are two possible ways of distributing this responsibility among the different nodes:

1. The first approach is that the **source node** computes the best route R to be traversed by the packet to reach its ultimate destination and attaches this computed route to the packet before it is sent out. Each intermediate node that receives this packet can deduce from R to which neighboring node this message should be forwarded. This approach is called SOURCE ROUTING and it assumes that every (source) node has complete information about the network topology. This assumption is not useful, because knowledge about the network topology alone is not enough. To make an optimal routing decision one has to also know the queue lengths of all the nodes in then network. Also in a dynamically changing network, some links or nodes might go

down and come up later, meaning that even the topology of the network is not fixed at all times. Moreover, each packet carries a lot of routing information (its complete route R) which creates a significant overhead. As a result, this approach is not very useful for adaptive routing in dynamically changing networks.

2. The second approach is that the **intermediate nodes** make local routing decisions. As a node receives a packet for some destination d it decides to *which neighbor this packet should be forwarded so that it reaches its destination as quickly as possible.* The destination index d is the only routing information that the packet carries. The overall route depends on the decisions of all the intermediate nodes. The following requirements have been identified for this approach (Tanenbaum 1989; Gouda 1998). Each node in the network needs to have:

   - for each of its neighbors, an estimate of how long it would take for the packet to reach its destination when sent via that neighbor;
   - a heuristic to make use of this information in making routing decisions;
   - a means of updating this routing information so that it changes with the change in the state of the network; and
   - a mechanism of propagating this information to other nodes in the network.

   This approach has lead to adaptive distance vector routing algorithms. DISTRIBUTED BELLMAN-FORD ROUTING (Bellman 1958), described in chapter 2, is the state of the art and most widely used and cited distance vector routing algorithm.

In the framework of the second approach, where all the nodes share the responsibility of making local routing decisions, the *routing problem* can be viewed as a complex optimization problem whereby each of the local routing decisions combine to yield a global routing policy. This policy is evaluated based on the *average packet delivery time* under the prevailing network and traffic conditions. The quality of the policy depends, in a rather complex manner, on all the routing decisions made by all the nodes. Due to the complexity

of this problem, a simplified version is usually considered. Instead of a globally optimal policy, one tries to find a collection of locally optimal ones:

> *When a node x receives a packet $P(s,d)$ destined to node d, what is the best neighbor y of x to which this packet should be forwarded so that it reaches its destination as quickly as possible?*

This problem is difficult for several reasons (as will be discussed in more detail later in this thesis):

1. Making such routing decisions at individual nodes requires a global view of the network which is not available; all decisions have to be made using local information available at the nodes only.

2. There is no training signal available for directly evaluating the individual routing decisions until the packets have finally reached their destination.

3. When a packet reaches its destination, such a training signal could be generated, but to make it available to the nodes that were responsible for routing the packet, the signal would have to travel to all these nodes thereby consuming a significant amount of network resources.

4. It is not known which particular decision in the entire sequence of routing decisions is to be given credit or blame and how much (the credit assignment problem).

These issues call for an approximate greedy solution to the problem where the routing policy adapts as routing takes place and overhead due to exploration is minimum.

1.3 Motivation for Adaptive Routing

As a solution to the routing problem, first consider the simplest possible routing algorithm, the Shortest Path Algorithm (Floyd 1962). This solution assumes that the network topology never changes, and that the best route from any source to any destination is the

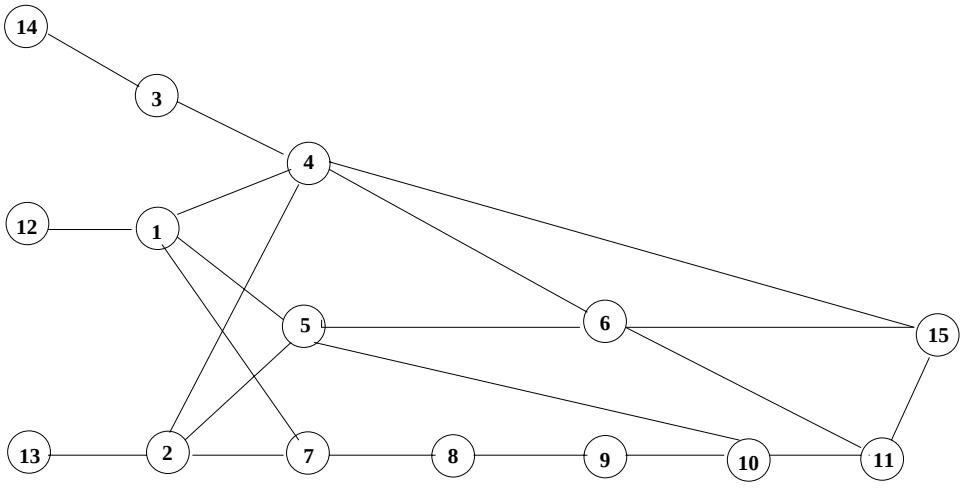

Figure 1.1: **15 node network** after (Choi and Yeung 1996). Nodes 12, 13 and 14 are source nodes, and node 15 is destination node.

shortest path in terms of the number of hops or length (l) of the route. The shortest path routing is the optimal routing policy when the network load is low, but as the load increases, the intermediate nodes that fall on popular routes get more packets than they can handle. As a result, the queue lengths of these intermediate nodes increase, and slowly the T_W component of T_D starts to dominate. This increase in queue lengths causes an increase in the average packet delivery time. Although the shortest path algorithm is simple and can be implemented in a source routing framework, it fails to handle the dynamics of a real communication network, as will be shown in section 2.1.

The main motivation for adaptive routing algorithms comes from the fact that *as the traffic builds up at popular nodes, the performance of the current routing policy starts to degenerate. Alternative routes, which may be longer in terms of the number of hops but lead to smaller delivery time, must then be learned through exploration.* As the network load levels, traffic patterns, and topology change, so should the routing policies. As an example, consider the network shown in figure 1.1 (Choi and Yeung 1996). If nodes 12, 13, and 14 are sending packets to node 15, then the routes ($12 \rightarrow 1 \rightarrow 4 \rightarrow 15$), ($13 \rightarrow 2 \rightarrow 4 \rightarrow 15$) and ($14 \rightarrow 3 \rightarrow 4 \rightarrow 15$) are the optimal routes for small loads. But as the load increases, node 4 starts getting more packets than it can handle and its queue length increases. Nodes

1 and 2 should then try sending packets to node 5. The new routes for packets from 12 and 13 to 15 should become ($12 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 15$) and ($13 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 15$). As traffic builds up at node 5, node 2 should start sending packets to node 7, and the overall routing policy for high loads should become ($12 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 15$), ($13 \rightarrow 2 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11 \rightarrow 15$) and ($14 \rightarrow 3 \rightarrow 4 \rightarrow 15$). When the load levels decrease again, the policy should revert back to the original one.

To implement such adaptive routing, (1) each node needs to maintain routing information which it can use for making routing decisions, and (2) there must be a mechanism for updating this information to reflect changes in the network state. Distributed Bellman-Ford is an example of adaptive routing and is discussed in detail in chapter 2. In this algorithm, each node in the network maintains two tables: the least COST to reach any destination node, and a routing table indicating to which neighbor should a packet be forwarded in order for it to reach the destination with minimum COST. These cost and routing tables are updated through periodic exchanges of cost tables among neighboring nodes. As shown in chapter 3, a version of Bellman-Ford, where COST is measured in terms of number of hops, converges to the shortest path policy at low loads. As the load increases, shortest path is no more the best policy, and another version of Bellman-Ford where COST is measured in terms of the packet delivery time, must be used. However, this version fails to learn a stable policy. Also, due to frequent exchanges of large cost tables among neighbors, there is a large exploration overhead.

These drawbacks of Bellman-Ford algorithm are addressed successfully by Q-Routing, as will be shown in chapter 4. In Q-Routing the cost tables are replaced by Q-tables, and the interpretation, exploration and updating mechanism of these Q-tables are modified to make use of the Q-learning framework. This thesis improves the Q-Routing algorithm further by improving its quality and quantity of exploration.

1.4 Contributions of This Work

The Q learning framework (Watkins and Dayan 1989) was used by Boyan and Littman (1994) to develop an adaptive routing algorithm called Q-Routing. Q-learning is well suited for adaptive routing as discussed above. The Q estimates are used to make decisions and these estimates are updated to reflect changes in the network. In Q-Routing, Q tables are maintained and updated based on the information coming from node x_{i+1} to node x_i when x_i sends a packet to x_{i+1}. In this thesis, two new components are added to Q-Routing;

1. CONFIDENCE-BASED Q-ROUTING (CQ-Routing), where the *quality* of exploration is improved by associating confidence values (between 0 - 1) with each of the Q-values in the network. These values represent how reliably the corresponding Q values represent the state of the network. The amount of adaptation for a Q-value, in other words, the learning rate, depends on the confidence values of the new and the old Q-values (whereas in Q-Routing a fixed learning rate is used for all updates). Section 5.1 describes this component of CDRQ-Routing in detail.

2. DUAL REINFORCEMENT Q-ROUTING (DRQ-Routing), where the *quantity* of exploration is increased by including backward exploration (in Q-Routing only forward exploration is used). As a result, with each packet hop, two Q-value updates take place, one due to forward exploration and the other due to backward exploration. Essentially, DRQ-Routing combines Q-ROUTING with *dual reinforcement learning*, which was first applied to a satellite communication problem (Goetz et al. 1996). Section 5.2 describes this component of CDRQ-Routing in detail.

As shown in figure 1.4, CDRQ-Routing combines the two components into a single adaptive routing algorithm. Experiments over several network topologies show that CQ-Routing and DRQ-ROUTING both outperform Q-Routing. The final algorithm, CDRQ-Routing is superior to both CQ-ROUTING and DRQ-Routing suggesting that both these features indeed optimize different parts of the problem.

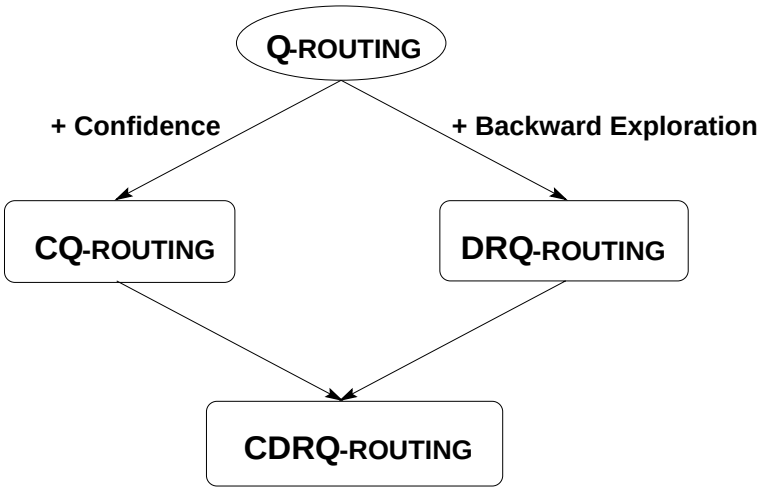

Figure 1.2: **Contributions of this Work**: Two features are added to Q-Routing, (1) confidence values, leading to CQ-Routing, and (2) backward exploration, leading to DRQ-Routing. These features are combined into the final algorithm called CDRQ-Routing.

The rest of the thesis is organized as follows. Chapter 2 describes the conventional non-adaptive shortest path, weighted shortest path and adaptive Bellman-Ford algorithms. Chapter 3 describes the Q-Routing algorithm and proves some important properties for the first time which are crucial to its performance. In chapter 4, the Q-Routing algorithm is evaluated against shortest path and Bellman-Ford routing algorithms. Chapter 5 describes the CDRQ-Routing with its two components, the CQ-Routing and the DRQ-Routing. Chapter 6 evaluates CDRQ-Routing comparing it with Q-Routing as well as its components for a number of tasks like learning an effective routing policy from scratch, adaptation to changes in traffic patterns and network topology etc. Chapter 7 gives a number of possible future directions and generalizations of CDRQ-Routing and its components. Chapter 8 concludes the thesis with main results.

Chapter 2

Conventional Routing Algorithms

In chapter 1, the distributed routing problem, where all intermediate nodes share the responsibility of making routing decisions, was formally stated. Features that make adaptive routing a difficult problem were highlighted and different conventional non-adaptive and adaptive routing algorithms were outlined. This chapter presents the details of the conventional routing algorithms. Section 2.1 describes the *non-adaptive* SHORTEST PATH ROUTING (Floyd 1962). Section 2.2 discusses the WEIGHTED SHORTEST PATH ROUTING (Dijkstra 1959) which is a generalization of shortest path routing and provides a benchmark for the algorithm developed in this thesis. It is a theoretical bound on the best possible routing one can do. Two versions of the state of the art distance vector *adaptive* routing algorithm BELLMAN-FORD (Bellman 1958; Bellman 1957; Ford and Fulkerson 1962) are discussed in section 2.3.

The key difference between non-adaptive and adaptive routing algorithms is that in the former, the routing policy of the network does not change with change in the state of the network. When a node receives a packet for a given destination, it *always* forwards that packet to the same neighbor. In adaptive routing, however, the routing decisions of a node for a particular destination might be different at different times. The adaptation is based on *exploration* where routing information at each node is spread to other nodes over the network as the packets are transmitted (see (Thrun 1992) for the role of exploration

in learning control). Individual nodes exploit this information by updating their routing policies.

2.1 Shortest Path Routing

In SHORTEST PATH ROUTING (Floyd 1962; Ravindra K. Ahuja and Tarjan 1988; Thomas H. Cormen and Rivest 1990), for any pair (s, d) of source and destination nodes, the route that takes the minimum number of hops is selected. Any packet from a given source to a given destination will always take the same route. Let $\mathbf{A}$ denote the $(n \times n)$ *adjacency matrix* ($[a_{ij}]$), where n is the number of nodes in the network, such that a_{ij} =1 if there is a link between nodes i and j, otherwise it is 0 (note that $\mathbf{A}$ is a symmetric matrix). The shortest path can be obtained from $\mathbf{A}$ using the sequence of matrices $\{\mathbf{A}^k{=}([a_{ij}^{(k)}])\}_{k=1}^{n}$, obtained by multiplying $\mathbf{A}$ with itself k times. The following property of $\mathbf{A}^k$ is used for this purpose:

> $a_{ij}^{(k)}$ *is the number of paths of length* k *from node* i *to node* j

Thus the length of the shortest path from node s to node d is given by:

$$l(s, d) = arg\min_{k}(a_{sd}^{(k)} \neq 0) \tag{2.1}$$

Shortest path routing can be implemented in the source routing framework as described in chapter 1, or, it could be implemented as local decisions at intermediate nodes. When a node x receives a packet $P(s, d)$ for some destination d, it chooses the neighbor y such that:

$$y = arg\min_{z \in \mathbf{N}(x)} l(z, d), \tag{2.2}$$

where $\mathbf{N}(x)$ is the set of all neighbors of node x. This routing policy is not very suitable for dynamically changing networks because:

- The network topology keeps changing as some nodes or links go down or come up (i.e. the adjacency matrix of the network changes).

- When the load is high, nodes falling on popular routes get flooded with packets. In other words, they receive more packets than they can process, thereby causing:
  - Delays in packet delivery times because of long queue waiting time (T_W).
  - Congestion at these nodes: Their packet queue buffers may fill up, causing them to drop packets.
- It makes suboptimal use of network resources: Most of the traffic is routed through small number of nodes while others are sitting idle.

2.2 Weighted Shortest Path Routing (GLOBAL ROUTING)

The theoretical upper bound on performance can be obtained if the entire state of the network is considered when making each routing decision. This is not a possible approach in practice but can be used as a benchmark to understand how well the other algorithms perform. The complete GLOBAL ROUTING algorithm due to Dijkstra (1959) is reviewed below.

The current state of the network is completely captured in the *Cost Adjacency Matrix* $\mathbf{C}$ $(= [c_{ij}])$ where:

- $c_{ii} = 0$ (no cost for sending a packet to itself).
- $c_{ij} = \infty$ if:
  - there is no link between node i and j,
  - link between node i and j is down, or
  - node i or node j is down.

  otherwise,
- $c_{ij} = q_i + \delta$ (i.e., the cost of sending a packet from node i to its neighbor j including both the queue waiting time and link transmission time.)

Let $S_h(i,j)$ denote the total cost of a path of h hops from node i to j, and $P_h(i,j)$ denote the path itself (sequence of nodes). Note that $S_h(i,j) = \infty$ and $P_h(i,j) = ()$ if there is no path of h hops from node i to j. Using this notation, the following algorithm can be used to find the weighted shortest path between every pair of nodes (i,j) in the network, where the weight between any pair of nodes i and j is given by c_{ij} of the cost matrix:

1. **Initialize:**
   - $S_1(i,j) = c_{ij}$ for all nodes i and j.
   - $P_1(i,j) = (i,j)$ if c_{ij} is finite, else $P_1(i,j) = ()$
   - hop index $h = 2$
2. while $h < N$ do steps 3 through 4
3. for each pair (i,j) of nodes, do the following:
   - Find the best index $\hat{k}$ such that:

$$\hat{k} = arg\min_k \{S_{\frac{h}{2}}(i,k) + S_{\frac{h}{2}}(k,j)\}. \tag{2.3}$$

   - Update $S_h(i,j)$ as follows:

$$S_h(i,j) = S_{\frac{h}{2}}(i,\hat{k}) + S_{\frac{h}{2}}(\hat{k},j). \tag{2.4}$$

   - Update path $P_h(i,j)$ as follows:
     - if $S_h(i,j) < S_{\frac{h}{2}}(i,j)$ then

$$P_h(i,j) = concat(P_{\frac{h}{2}}(i,\hat{k}), P_{\frac{h}{2}}(\hat{k},j)). \tag{2.5}$$

     - if $S_h(i,j) \geq S_{\frac{h}{2}}(i,j)$ then

$$P_h(i,j) = P_{\frac{h}{2}}(i,j). \tag{2.6}$$

4. $h = 2 * h$.

The CONCAT operation takes two sequences of nodes (two paths), say $(x_1, x_2, ..., x_m)$ and $(y_1, y_2, ..., y_{m'})$, and gives a longer sequence of nodes which is formed by concatenation of these two sequences:

$$concat((x_1, x_2, ..., x_m), (y_1, y_2, ..., y_{m'})) = (x_1, x_2, ..., x_m, y_2, ..., y_{m'}) \tag{2.7}$$

The *condition for concatenation* is that $x_m = y_1$. The time complexity of the weighted shortest path algorithm is $O(n^3 log_2 n)$, where n is the number of nodes in the network.

Each node in the network has the adjacency matrix **A** and using above algorithm, it calculates the shortest paths to every other node. When a node x has to send a packet destined for node d, it finds the best neighbor y to which to forward this packet as follows:

1. Compute the S_n and P_n matrices as shown above.
2. From P_n, obtain the best route $P_n(x, d)$ from node x to d.
3. The *second* node in the route $P_n(x, d)$ is the best neighbor to which the packet should be forwarded (the first node is x itself).

This three-step process is executed for every hop of every packet, each time using the entire cost adjacency matrix **C**. As mentioned before, the matrix **C** might change with change in network topology or load levels. Since, each node in the network is required to maintain the current state of the network, they should all have the same **C**. In other words, all nodes should have the global view of the network. Hence, in the remainder of the thesis, this algorithm is called GLOBAL ROUTING due to its global nature.

2.3 Distributed Bellman Ford Routing

It is not possible to maintain the current state of the network known to all nodes at all times. Instead a local view of the network can be maintained by each node and it can be updated as the network changes. One version of this approach is called the DISTANCE VECTOR ROUTING (Tanenbaum 1989) that is used in modern communication network

systems. Bellman-Ford routing (Bellman 1957; Bellman 1958; Ford and Fulkerson 1962; C. Cheng and Garcia-Luna-Aceves 1989; Rajagopalan and Faiman 1989) is one of the most commonly used and cited adaptive distance vector routing algorithms. In Bellman-Ford routing, each node x maintains two tables for storing its view of the network:

- A **cost table** $Cost_x(d)$:$\mathbf{V}\rightarrow$COST which contains the least COST incurred by a packet in going from node x to destination d ($Cost_x(x) = 0$).
- A **routing table** $Rtb_x(d) : \mathbf{V}\rightarrow N(x)$, where $N(x)$ is the set of neighbors of node x, which contains the best neighbor of x to send the packet for destination d ($Rtb_x(x) = x$).

$\mathbf{V}$ is the set of all nodes in the network. Two versions of Bellman Ford routing are considered in this thesis because there are two different ways of interpreting the COST of delivering a packet. In the first version, **BF1**, COST is measured in terms of the number of hops (cost table is referred to as $H_x(d)$), while in the second version, **BF2**, COST is measured in terms of the total packet delivery time (cost table is referred to as $T_x(d)$). The routing table Rtb is referred to as $R_x(d)$ for both **BF1** and **BF2**. Both the cost tables and the routing tables are updated as the state of the network changes. These updates take place through frequent exchanges of COST tables between neighbors. The rate of exchange is given by the parameter f, which is essentially the probability with which a node sends its COST table to its neighbor at each time step.

2.3.1 Table Updates

Since cost is interpreted differently in **BF1** and **BF2**, the update rules for these versions are also different. In **BF1**, when node $y \in N(x)$ sends its cost table $H_y(*)$ to its neighboring node x the cost is updated by:

$$\forall d \in \mathbf{V}, H_x(d)^{upd} = \begin{cases} 0, & \text{if } x = d \\ 1, & \text{if } y = d \\ \min(n, 1 + H_y(d)), & \text{otherwise,} \end{cases} \tag{2.8}$$

where n is the total number of nodes in the network and d is a destination node. The routing table $R_x(*)$ of node x is updated as follows:

$$\forall d \in \mathbf{V}, R_x(d)^{upd} = \begin{cases} y & \text{if } H_x(d)^{upd} < H_x(d)^{old} \\ R_x(d)^{old} & \text{otherwise.} \end{cases} \tag{2.9}$$

The above update rules for cost and routing tables in **BF1** have been found to converge to the shortest path policy even if initially all table entries, other than those with base case values, are initialized randomly.

The corresponding updates in **BF2**, when node $y \in N(x)$ sends its cost table $T_y(*)$ to its neighboring node x, are given by:

$$\forall d \in \mathbf{V}, T_x(d)^{upd} = \begin{cases} 0, & \text{if } x = d \\ \delta, & \text{if } y = d \\ (1-\eta)T_x(d)^{old} + \eta(\delta + q_y + T_y(d)), & \text{otherwise,} \end{cases} \tag{2.10}$$

where δ is the transmission delay of the link between nodes x and y, q_y is the queue length of node y, and η is the learning rate. The routing table $R_x(*)$ of node x is updated similarly to 2.9 as:

$$\forall d \in \mathbf{V}, R_x(d)^{upd} = \begin{cases} y & \text{if } T_x(d)^{upd} < T_x(d)^{old} \\ R_x(d)^{old} & \text{otherwise} \end{cases} \tag{2.11}$$

2.3.2 Overhead analysis for Bellman-Ford algorithm

Storage and exploration of routing information (i.e. the cost tables) each have their own overheads. In both **BF1** and **BF2**, the size of the cost table and routing table in every node is $O(n)$. Hence total *storage overhead* for Bellman-Ford routing is $O(n^2)$. The more important overhead is the *exploration overhead*, which arises from sending cost tables between neighboring nodes, thereby consuming network resources. Each cost table is of size $O(n)$. If f is the probability of a node sending $O(n)$ entries of its table to its neighbor, then the total exploration overhead is $O(fBn^2)$ in each time step, where B is the average branching factor of the network. For large networks (high n), this is a huge overhead. Clearly there is a tradeoff between the speed at which Bellman-Ford converges to the effective routing

policy and the frequency of cost table exchanges. In chapter 4, Bellman-Ford is evaluated empirically at various load levels, with respect to its exploration overhead and speed, quality and stability of the final policy learned. It is compared with Q-Routing. It would be seen that **BF1** is suitable for low loads but fails as load increases, while **BF2** converges to inferior and unstable routing policies as compared to the Q-Routing algorithm discussed in chapter 3. Moreover, the exploration overhead in Bellman-Ford is found to be more than five to seven times that in Q-Routing for similar convergence properties.

2.4 Conclusion

Various conventional routing algorithms were described in this chapter. The two extremes of the routing algorithms, namely the shortest path routing and the weighted shortest path routing, were discussed. While shortest path routing is too naive and fails miserably when the load level increases to realistic values, weighted shortest path is impossible in practice. The most common adaptive routing algorithm currently is distributed Bellman-Ford routing, which is a version of distance vector routing. Two versions of this algorithm were described in this chapter with the update rules and overhead analysis.

The next chapter describes the Q-Routing algorithm, based on Q-learning framework in detail. Q-Routing forms the basis of the adaptive routing algorithm developed in this thesis. Both versions of Bellman-Ford algorithm are compared with Q-Routing in chapter 4 and Q-Routing is shown to be superior than Bellman-Ford with regard to speed, quality and stability of the policy learned and exploration overhead of adaptation.

Chapter 3

Q-Routing

In chapter 2, various conventional routing algorithms, both non-adaptive (shortest path routing and weighted shortest path) and adaptive (Bellman-Ford), were discussed in detail. In this chapter, the first attempt of applying Q learning to the task of adaptive network routing, called Q-ROUTING (Littman and Boyan 1993a; Littman and Boyan 1993b; Boyan and Littman 1994), is discussed in detail. The exploitation and exploration framework used in Q-ROUTING is the basis of the routing algorithm proposed in this thesis. After discussing the basic idea of Q-learning in section 3.1, the syntax and semantics of the routing information stored at each node as Q-tables is discussed in section 3.2. Means of exploitation and exploration of these Q-tables is discussed in section 3.3 and section 3.4 respectively. In section 3.5, the properties that form the basis of the forward exploration update rule are stated an proved. Section 3.6 summarizes the complete Q-ROUTING algorithm. Section 3.7 is devoted to the overhead analysis due to forward exploration in Q-Routing.

3.1 Q-Learning

There are two approaches to learning a controller for a given task. In *model-based* approach, the learning agent must first learn a model of the environment and use this knowledge to learn an effective control policy for the task, while in the *model-free* approach a controller

is learned directly from the actual outcomes (also known as *roll outs* or *actual returns*). Reinforcement learning is an example of the *model-based* approach which is used in this thesis for the task of adaptive network routing.

The main challenge with reinforcement learning is *temporal credit assignment*. Given a state of the system and the action taken in that state, how do we know if the action was good? One strategy is to wait until the "end" and reward the actions taken if the result was good and punish them otherwise. However, just one action does not solve the credit assignment problem and many actions are required. But, in on-going tasks like network routing, there is no "end" and the control policy needs to be learned as the system (communication network) is performing. Moreover, when the system is dynamic, the learning process should be continuous.

Temporal differences (Sutton 1988; Sutton 1996), a *model-based* approach is an example of reinforcement learning. The system is represented by a parametric model where learning entails tuning of the parameters to match the behavior of the actual system. The parameters are changed based on the immediate rewards from the current action taken and the estimated value of the next state in which the system goes as a result of this action. One temporal difference learning strategy is the Adaptive Heuristic Critic(AHC) (Sutton 1984; Barto 1992; A. G. Barto and Anderson 1983). It consists of two components, a critic (AHC) and a reinforcement-learning controller (RL). For every action taken by the RL in the current state, a primary reinforcement signal is generated by the environment based on how good the action was. The AHC converts the primary reinforcement signal into heuristic reinforcement signal which is used to change the parameters of the RL.

The work of the two components of adaptive heuristic critic can be accomplished by a single component in Watkins' Q-learning algorithm (Watkins and Dayan 1989). Q-learning is typically easier to implement. The states and the possible actions in a given state are discrete and finite in number. The model of the system is learned in terms of Q-values. Each Q-value is of the form $Q(s, a)$ representing the expected reinforcement of taking action a in state s. Thus in state s, if the Q-values are learned to model the system accurately,

the best action is the one with the highest Q-value in the vector $Q(s, *)$. The Q-values are learned using an update rule that makes use of the current reinforcement $R(s, a)$ computed by the environment and some function of Q-values of the state reached by taking action a in state s.

In general, Q-values can also be used to represent the characteristics of the system based on s and a instead of the expected reinforcement as mentioned above. The control action, therefore, could be a function of all the Q-values in the current state. In the Q-Routing algorithm, Q-learning is used to learn the task of finding an optimal routing policy given the current state of the network. The state of the network is learned in terms of Q-values distributed over all nodes in the network. In the next section an interpretation of these Q-values for a communication network system is given.

3.2 Routing Information at Each Node

In Q-Routing, Q-learning is used to first learn a representation of the state of the network in terms of Q-values and then these values are used to make control decisions. The task of Q-learning is to learn an optimal routing policy for the network. The state s in the optimization problem of network routing is represented by the Q-values in the entire network. Each node x in the network represents *its own view* of the state of the network through its Q-table $\mathbf{Q}_x$. Given this representation of the state, the action a at node x is to choose that neighbor y such that it takes minimum time for a packet destined for node d to reach its destination if sent via neighbor y.

In Q-Routing each node x maintains a table of Q-values $\mathbf{Q}_x(y, d)$, where $d \in \mathbf{V}$, the set of all nodes in the network, and $y \in N(x)$, the set of all neighbors of node x. According to Boyan and Littman (1994), the value $\mathbf{Q}_x(y, d)$ can be interpreted as

> "$\mathbf{Q}_x(y, d)$ is node x's best *estimated time* that a packet would take to reach its destination node d from node x when sent via its neighboring node y *excluding* any time that this packet would spend in node x's queue, and *including* the

total waiting time and transmission delays over the entire path that it would take starting from node y."

The base case values for this table are:

- $\mathbf{Q}_x(y, x) = \infty$ for all $y \in N(x)$, that is, if a packet is already at its destination node, it should not be sent out to any neighboring node.
- $\mathbf{Q}_x(y, y) = \delta$, in other words, a packet can reach its neighboring node in one hop.

In the steady state, when the Q-values in all the nodes represent the true state of network, the Q-values of neighboring nodes x and y should satisfy the following relationships:

- The GENERAL INEQUALITY:

$$\mathbf{Q}_x(y, d) \leq q_y + \delta + \mathbf{Q}_y(z, d) \quad \forall y \in N(x) \text{and} \quad \forall z \in N(y). \tag{3.1}$$

  This equation essentially states that if a packet destined for node d, currently at node x, is sent via x's neighbor y, then the maximum amount of time it will take to reach its destination is bound by the sum of three quantities: (1) the waiting time q_y in the packet queue of node y, (2) the transmission delay δ over the link from node x to y, and (3) the time $\mathbf{Q}_y(z, d)$ it would take for node y to send this packet to its destination via any of node y's neighbors (z).

- The OPTIMAL TRIANGULAR EQUALITY:

$$\mathbf{Q}_x(y, d) = q_y + \delta + \mathbf{Q}_y(\hat{z}, d) \tag{3.2}$$

  This equation is a special case of the above *general inequality* and it states that the minimum time taken to deliver a packet currently at node x and destined for node d and via any of neighbor $y \in N(x)$, is the sum of three components: (1) the time this packet spends in node y's queue, (2) the transmission delay δ between node x and y, and (3) the best time $\mathbf{Q}_y(\hat{z}, d)$ of node y for the destination d.

The *general inequality* is used later to formalize the notion of an admissible (or valid) Q-value update. The *triangular equality* is used to compute the estimated Q value for the update rules.

3.3 Exploiting the Routing Information

Once it is clear what the Q-values stored at each node stand for, it is easy to see how they can be used for making *locally greedy* routing decisions. When a node x receives a packet $P(s,d)$ destined for node d, node x looks at the vector $\mathbf{Q}_x(*,d)$ of Q-values and selects that neighboring node $\hat{y}$ for which the $\mathbf{Q}_x(\hat{y},d)$ value is minimum. This is called the MINIMUM SELECTOR RULE. This way, node x makes a locally greedy decision by sending the packet to that neighbor from which this packet would reach its destination as quickly as possible.

It is important to note, however, that these Q-values are not exact. They are just *estimates* and the routing decision based on these estimates does not necessarily give the best solution. The routing decisions are locally optimal only with respect to the estimates of the Q-values at these nodes and so is the overall routing policy that emerges from these local decisions. In other words the control action (the routing decision) is only as good as the model of the network represented by the Q-values in the network. The closer these estimates are to the actual values, the closer the routing decision is to the optimal routing decisions.

The following section shows how these Q-values are updated so that they adapt to the changing state of the network, in other words, learn a more accurate model of the network and gradually become good approximations of true values.

3.4 Forward Exploration of the Routing Information

In order to keep the Q-value estimates as close to the actual values as possible and to reflect the changes in the state of the network, the Q-value estimates need to be updated with minimum possible overhead. Boyan and Littman (1994) proposed the following update

mechanism, constituting the Q-Routing algorithm.

As soon as the node x sends a packet $P(s,d)$, destined for node d, to one of its neighboring nodes y, node y sends back to node x its best estimate $\mathbf{Q}_y(\hat{z},d)$ for the destination d:

$$\mathbf{Q}_y(\hat{z},d) = \min_{z \in N(y)} \mathbf{Q}_y(z,d). \tag{3.3}$$

This value essentially estimates the *remaining time* in the journey of packet $P(s,d)$. Upon receiving $\mathbf{Q}_y(\hat{z},d)$, node x computes the new estimate for $\mathbf{Q}_x(y,d)$ as follows:

$$\mathbf{Q}_x(y,d)^{est} = \mathbf{Q}_y(\hat{z},d) + q_y + \delta. \tag{3.4}$$

Note that this estimate is computed based on the *optimal triangular equality* (equation 3.2). Using the estimate $\mathbf{Q}_x(y,d)^{est}$, node x updates its $\mathbf{Q}_x(y,d)$ value as follows:

$$\mathbf{Q}_x(y,d)^{new} = \mathbf{Q}_x(y,d)^{old} + \eta_f(\mathbf{Q}_x(y,d)^{est} - \mathbf{Q}_x(y,d)^{old}), \tag{3.5}$$

where η_f is the learning rate. Substituting and expanding 3.5, the update rule is given by:

$$\mathbf{Q}_x(y,d) = \mathbf{Q}_x(y,d) + \eta_f(\overbrace{\mathbf{Q}_y(\hat{z},d) + q_y + \delta}^{new\ estimate} - \mathbf{Q}_x(y,d)). \tag{3.6}$$

When the learning rate η_f is set to 1, the update rule 3.6 reduces to the *optimal triangular equality* (equation 3.2). Since the value $\mathbf{Q}_y(\hat{z},d)$ and others from which it was derived (the $\mathbf{Q}_y(*,d)$ vector) were not accurate, the learning rate is set to some value less than 1 (e.g. 0.85), and equation 3.6 is an incremental approximation of the *optimal triangular equality*.

The exploration involved in updating the Q-value of the *sending* node x using the information obtained from the *receiving* node y, is referred to as **forward exploration** (figure 5.1). With every hop of the packet $P(s,d)$, one Q-value is updated. The properties of forward exploration are highlighted next, before the experimental results on the effectiveness of this idea are presented. In the DUAL REINFORCEMENT Q-ROUTING (Kumar and Miikkulainen 1997) algorithm discussed in chapter 5, the other possible direction of exploration, **backward exploration**, is also utilized.

3.5 Properties of Forward Exploration

Two aspects of a Q-value update rule (like that in equation 3.6) are characterized in this section. First, the update rule should be *admissible*. An update rule is said to be *admissible* if the updated Q-values obeys the *general inequality*, given that the old Q-values obeyed the *general inequality*. The admissibility of the update rule given by equation 3.6 is stated and proved as the first property (property 3.1) of forward exploration. Second, the update rule should asymptotically converge to the shortest path routing at low loads. The second property (property 3.2) of forward exploration guarantees the same. Although these results are original to this thesis, they form the basis of the update rules proposed by Boyan and Littman (1994). In fact these properties also form the basis of forward exploration update rule for the CDRQ-Routing and its versions CQ-ROUTING and DRQ-ROUTING described in the next chapter.

Property 3.1: *The update rule given by equation 3.6 guarantees that if the old value of* $\mathbf{Q}_y(x,d)$ *satisfies the general inequality (equation 3.1), then its updated value also satisfies the general inequality. i.e. update rule 3.6 is admissible.*

Proof: Let $\mathbf{Q}_x(y,d)'$ denote the updated value of $\mathbf{Q}_x(y,d)$ using equation 3.6. If the old value of $\mathbf{Q}_x(y,d)$ satisfies the *general inequality*, then for any neighbor z of node y and some non-negative $\theta(z)$,

$$\mathbf{Q}_x(y,d) = q_y + \delta + \mathbf{Q}_y(z,d) - \theta(z). \tag{3.7}$$

Also since $\mathbf{Q}_y(\hat{z},d)$ is the best estimate of node y, by definition this has to be the minimum of all other estimates of node y for the same destination d. Thus for all $z \in N(y)$ and some non-negative $\theta'(z,\hat{z})$,

$$\mathbf{Q}_y(\hat{z},d) = \mathbf{Q}_y(z,d) - \theta'(z,\hat{z}). \tag{3.8}$$

Substituting 3.7 and 3.8 in the update rule and simplifying,

$$\mathbf{Q}_x(y,d)' = q_y + \delta + \mathbf{Q}_y(z,d) - [(1-\eta_f)\theta(z) + \eta_f\theta'(z,\hat{z})], \tag{3.9}$$

which can be rewritten as:

$$\mathbf{Q}_x(y,d)' \leq q_y + \delta + \mathbf{Q}_y(z,d). \tag{3.10}$$

Hence the updated Q-value also satisfies the *general inequality.*

Property 3.2: *For low network loads, the routing policy learned by the Q-Routing update rule given in equation 3.6 asymptotically converges to the shortest path routing.*

Proof: Consider the complete route $(x_0(=s), x_1, ..., x_l(=d))$ of length l for a packet $P(s,d)$. The l Q-value updates associated with this packet are given by the following generic form $(i = 0...l-1)$:

$$\mathbf{Q}_{x_i}(x_{i+1}, x_l) = \mathbf{Q}_{x_i}(x_{i+1}, x_l) + \eta_f(\mathbf{Q}_{x_{i+1}}(x_{i+2}, x_l) + q_{x_{i+1}} + \delta - \mathbf{Q}_{x_i}(x_{i+1}, x_l)). \tag{3.11}$$

The best Q-value at node x_{i+1} for the remaining part of the journey to the destination node d is $\mathbf{Q}_{x_{i+1}}(x_{i+2}, d)$; that is why node x_{i+1} forwards the packet to node x_{i+2}. This Q-value is sent back to node x_i by node x_{i+1} for updating $\mathbf{Q}_{x_i}(x_{i+1}, d)$. The base case for forward exploration for this route, given by

$$\mathbf{Q}_{x_{l-1}}(x_l, x_l) = \delta, \tag{3.12}$$

follows directly from the optimal base cases of Q-values mentioned in section 3.2. For low loads, we can assume all q_x are negligible and the main component of packet delivery time comes from the transmission delay δ. The simplified update rule for any triplet (x_{i-1}, x_i, x_{i+1}) is given by:

$$\mathbf{Q}_{x_{i-1}}(x_i, x_l) = \mathbf{Q}_{x_{i-1}}(x_i, x_l) + \eta_f(\mathbf{Q}_{x_i}(x_{i+1}, x_l) + \delta - \mathbf{Q}_{x_{i-1}}(x_i, x_l)). \tag{3.13}$$

If $(x_0(=s), x_1, ..., x_l(=d))$ were the shortest route between nodes x_0 and x_l, then the Q-values at each of the intermediate nodes are given by:

$$\mathbf{Q}_{x_i}(x_{i+1}, x_l) = (l-i)\delta. \tag{3.14}$$

Equation 3.14 is proved by *induction* over $l - i - 1$ below:

Base case: $(i = l - 1)$ follows directly from equation 3.12 by substituting for i in 3.14.

Induction Hypothesis: Assume 3.14 for some i. Now substituting for $\mathbf{Q}_{x_i}(x_{i+1}, x_l)$ from 3.14 into 3.13 we get:

$$\mathbf{Q}_{x_{i-1}}(x_i, x_l) = \mathbf{Q}_{x_{i-1}}(x_i, x_l) + \eta_f((l - i + 1)\delta - \mathbf{Q}_{x_{i-1}}(x_i, x_l)). \tag{3.15}$$

Repeated updates using the update equation 3.15 will asymptotically yield

$$\mathbf{Q}_{x_{i-1}}(x_i, x_l) = (l - i + 1)\delta. \tag{3.16}$$

Equation 3.16 proves the induction hypothesis for $l - i$. Hence the property 3.2 holds for all $l - i - 1$.

3.6 Summary of the Q Routing Algorithm

The complete Q-Routing algorithm can be summarized in two steps. The $PacketReceive_y(x)$ step describes what node y does when it receives a packet from its neighboring node x, and the $PacketSend_x(P(s,d))$ step describes what node x does when it has to send a packet $P(s,d)$ for destination d. These two steps for Q-Routing are given in tables 3.1 and 3.2.

1	If (not EMPTY(*PacketQueue(x)*) go to step 2
2	$P(s,d) =$ *Dequeue* the first packet in the *PacketQueue*(x).
3	Compute best neighbor $\hat{y} = arg\min_{y \in N(x)} \mathbf{Q}_x(y,d)$
4	*ForwardPacket* $P(s,d)$ to neighbor $\hat{y}$.
5	Wait for $\hat{y}$'s estimate.
6	*ReceiveEstimate*$(\mathbf{Q}_{\hat{y}}(\hat{z},d) + q_{\hat{y}})$ from node $\hat{y}$.
7	*UpdateQvalue*$(\mathbf{Q}_x(y,d))$ as given in 3.6.
8	Get ready to send next packet (goto 1).

Table 3.1: **PacketSend**$_x(P(s,d))$ at NODE x for Q-Routing (FE=Forward Exploration, MSR=Minimum selector rule)

1	Receive a packet $P(s,d)$, destined for node d from neighbor x
2	Calculate best estimate for node d; $\mathbf{Q}_y(\hat{z},d) = \min_{z \in N(y)} \mathbf{Q}_y(z,d)$.
3	Send $(\mathbf{Q}_y(\hat{z},d) + q_y)$ back to node x.
4	If $(d \equiv y)$ then $ConsumePacket_y(P(s,d))$ else goto 5.
5	If ($PacketQueue(y)$ is FULL) then $DropPacket(P(s,d))$ else goto 6.
6	$AppendPacketToPacketQueue_y(P(s,d))$
7	Get ready for receiving next packet (goto 1).

Table 3.2: **PacketReceive**$_y(x)$ at NODE y for Q-Routing

3.7 Overhead Analysis of Q-Routing

The term overhead refers to the time taken to execute steps in the routing algorithm that would be either completely missing or executed in constant time in the non-adaptive shortest path algorithm. The overhead incurred by an adaptive routing algorithm for exploitation and exploration should be carefully analyzed in order to evaluate how feasible it is. In this section, overhead due to *forward exploration and exploitation* in Q-Routing are analyzed in detail for the first time. From the summary of the complete algorithm in the previous section, there are four distinct overheads associated with *each hop* of every packet routed in the network:

1. **Decision Overhead** (t_d) is defined as the time that the sending node x takes to decide what its best neighbor is:

$$t_d(x) = O(|N(x)|), \tag{3.17}$$

   which is essentially the time taken to find the minimum in a vector $\mathbf{Q}_x(*,d)$.

2. **Estimate Computation Overhead** (t_c) is defined as the time taken by the receiving node y in computing the estimate that it sends back when it receives a packet from the sender node:

$$t_c(y) = O(|N(y)|), \tag{3.18}$$

   which is essentially the time taken to find the minimum in the vector $\mathbf{Q}_y(*,d)$.

3. **Estimate Transmission Overhead** (t_r) is defined as the time taken by this estimate to travel from the receiver node x to the sender node y. Let $\mathbf{e}$ be the size of the *estimate packet* and $\mathbf{p}$ be the size of *data packets*. Now making use of the fact that the transmission delay over the link is proportional to the length of the packet being transmitted, and using δ as the transmission delay of the *data packet*, the estimate transmission overhead is given by:

$$t_r(x, y) = \delta(\frac{\mathbf{e}}{\mathbf{p}}). \tag{3.19}$$

4. **Q-value Update Overhead** (t_u) is defined as the time node x takes to update its Q-value $\mathbf{Q}_x(y, d)$ once the sender node receives the appropriate estimate from the receiving node. This is essentially an $O(1)$ time operation.

These overheads are generally minor; the only significant one is the *estimate transmission overhead* (t_r). According to equation 3.19, the overhead per packet hop is only $\mathbf{e}/\mathbf{p}$, which is less than 0.1% of the transmission delay of a packet. This is because the *estimate packet* contains nothing more than a real number, while the *data packets* could be as big as 1-10 KB. Hence,the additional overhead due to forward exploration is insignificantly small, while the gains due to the adaptability are very rewarding, as will be shown in the next chapter.

3.8 Conclusion

In this chapter, the basic framework of Q-Routing was reviewed. Two properties of *forward exploration* update rule (equation 3.6), namely (1) admissibility of the update rule (invariance of Q-values with respect to the *general inequality*) and (2) its convergence to shortest path policy at low load level, were identified and proved for the first time. These properties form the basis of Q-Routing. Overhead due to forward exploration is analyzed.

In the next chapter, Q-Routing is evaluated empirically for its ability to adapt under various load levels and initial conditions. Comparison of Q-Routing with non-adaptive

shortest path routing and with the state- of-the-art Bellman-Ford routing are used to highlight the performance of Q-Routing which forms the basis of CDRQ-Routing.

Chapter 4

Evaluating Q-Routing

The Q-Routing algorithm is evaluated in this chapter. It is compared with both the non-adaptive shortest path routing (Section 4.3) and the adaptive Bellman-Ford routing (Section 4.4). Terminology used in the rest of the thesis for experiments is defined in section 4.1 and the experimental setup for all experiments in this chapter is given in section 4.2.

4.1 Definitions

In this section, terms used in the experiments in this thesis are defined. The three main network properties considered include the network load levels, traffic patterns, and the network topology. Definitions of these properties is followed by the definitions of routing policy and different measures of performance, including speed and quality of adaptation, the average packet delivery time, and the number packets in the network.

1. **Network Load Level** is defined as the average number of packets introduced in the network per unit time. For simulation purposes, *time* is to be interpreted as *simulation time* (discrete time steps synchronized for all nodes in the network). Three ranges of network load levels are identified: *low load*, *medium load*, and *high load*. At *low loads*, exploration is very low and the amount of information per packet hop significantly affects the rate of learning. At *medium loads*, the exploration is directly related to

the number of packets in the network. Medium load level represents the average load levels in a realistic communication network. Although the amount of exploration is high at *high loads*, there are a large number of packets in the network, and it is actually more difficult to learn an effective routing policy. When a node's buffer gets filled up, additional incoming packets are dropped leading to loss of information, which is called congestion. In this thesis, however, infinite packet buffers are used, and finite packet buffers and congestion control are discussed in chapter 7 as part of the future work.

2. **Traffic Pattern** is defined as the probability distribution $p(s,d)$ of source node s sending a packet to node d. This distribution is normalized such that:

$$\sum_{x \in \mathbf{V}} p(s,x) = 1 \quad \forall s \in \mathbf{V}, \tag{4.1}$$

where $\mathbf{V}$ is the set of all nodes in the network. The value of $p(s,s)$ is set to 0 for all $s \in \mathbf{V}$. A *uniform traffic pattern* is one in which the probability $p(s,d)$ is $\frac{1}{n-1}$ where n is the number of nodes in the network.

3. **Network Topology** is made up of the nodes and links in the network. The topology changes when a link goes down or comes up. The following Q-value updates are used to model the going down of the link between nodes x and y:

$$Q_x(y,d) = \infty \quad \forall d \in \mathbf{V}, \tag{4.2}$$

and

$$Q_y(x,d) = \infty \quad \forall d \in \mathbf{V}. \tag{4.3}$$

4. A **Routing Policy** is characterized by the Q tables in the entire network. Changes in these Q-values by exploration leads to changes in the routing policy of the network. The algorithm is said to have *converged* to a routing policy when changes in Q-values are too small to affect any routing decisions. An indirect way of testing whether the routing policy has converged is to examine the *average packet delivery time* or the *number of packets in the network* as routing takes place. When average packet

delivery time or number of nodes in the network stabilize or converge to a value and stay there for a long time, we can say that the routing policy has converged.

5. The performances of an adaptive routing algorithm can be measured in two ways:

   (a) **Speed of Adaptation** is the time it takes for the algorithm to *converge* to an effective routing policy starting from a random policy. It depends mainly on the *amount of exploration* taking place in the network.

   (b) **Quality of Adaptation** is the quality of the final routing policy. This is again measured in terms of the average packet delivery time and the number of packets in the network. Quality of adaptation depends mainly on how accurate the updated Q value is as compared to the old Q value. Hence quality of exploration affects the quality of adaptation.

6. **Average Packet Delivery Time**: The main performance metric for routing algorithms is based on the *delivery time* of packets, which is defined as the (simulation) time interval between the introduction of a packet in the network at its source node and its removal from the network when it has reached its destination node. The *average packet delivery time*, computed at regular intervals, is the average over all the packets arriving at their destinations during the interval. This measure is used to monitor the network performance *while* learning is taking place. Average packet delivery time *after* learning has settled measures the quality of the final routing policy.

7. **Number of Packets in the Network**: A related performance metric for routing algorithms is the number of packets in the network also referred to as the amount of traffic in the network. An effective routing policy tries to keep the traffic level as low as possible. A fixed number of packets are introduced per time step at a given load level. Packets are removed from the network in two possible ways; either they reach their destination or the packets are dropped on the way due to congestion. Let $n_g(t)$, $n_r(t)$ and $n_d(t)$ denote the total number of packets generated, received and dropped at time t, respectively. Then the number of packets in the network $n_p(T)$ at the current

time T is given by:

$$n_p(T) = \sum_{t=0}^{T} (n_g(t) - n_r(t) - n_d(t)). \tag{4.4}$$

where time $t = 0$ denotes the beginning of the simulation. In this thesis, infinite packet buffers are used therefore no packets are dropped (i.e. $n_d(t) = 0\ \forall t$). The problem of congestion is dealt with in chapter 7 as part of the future work.

In the rest of this chapter Q-Routing is compared with the non-adaptive shortest path routing and the adaptive state-of-the-art distance vector routing algorithm namely distributed Bellman-Ford routing (both versions **BF1** and **BF2**). Experimental setup of these comparisons is discussed first, followed by the two sets of experimental results.

4.2 Experimental Setup

The network topology used in these experiments is the 6×6 irregular grid shown in figure 4.1 due to Boyan and Littman (1994). In this network, there are two possible ways of routing packets between the left cluster (nodes 1 through 18) and the right cluster (nodes 19 through 36): the route including nodes 12 and 25 (R_1) and the route including nodes 18 and 19 (R_2). For every pair of source and destination nodes in different clusters, either of the two routes, R_1 or R_2 can be chosen. Convergence to effective routing policies, starting from either random or shortest path policies, is investigated below. There are two sets of experiments to be performed:

The learning rate of 0.85 in Q-Routing is found to give best results. Learning rate of 0.95 was found to give the best results for the Bellman-Ford routing (equation 2.10). Packets destined to random nodes are periodically introduced into this network at random nodes. *LoadLevel* is the probability of generating a packet by a node at any simulation time step. Therefore, the average number of packets introduced in the network in any simulation time step is $n \times LoadLevel$ where n is the number of nodes in the network. Multiple packets at a node are stored in its *unbounded* FIFO queue. In one time step, each node removes the packet in front of its queue, determines the destination and uses its routing decision maker

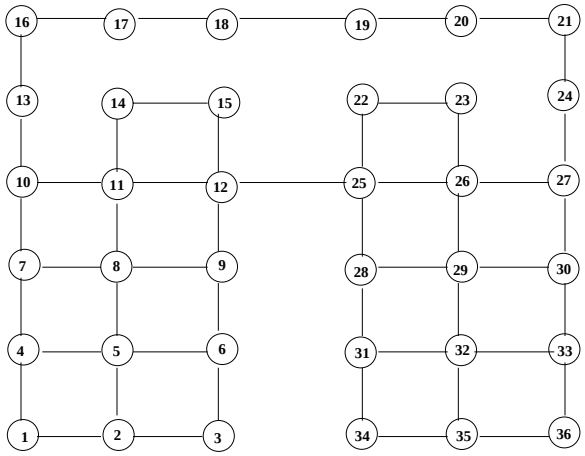

Figure 4.1: **The 6x6 Irregular Grid** (adapted from Boyan and Littman (1994)). The left cluster comprises of nodes 1 through 18 and the right cluster of nodes 19 through 36. The two alternative routes for traffic between these clusters are the route including the link between nodes 12 and 25 (route R_1) and the route involving the link between nodes 18 and 19 (route R_2). R_1 becomes a bottleneck with increasing loads, and the adaptive routing algorithm needs to learn to make use of R_2.

(Q-tables in case of Q-Routing and routing tables in case of Bellman-Ford routing) to send the packet to one of its neighboring nodes. When a node receives a packet, it either removes the packet from the network or appends it at the end of its queue, depending on whether or not this node is the destination node of the packet. Learning of routing information takes place as follows. In case of Q-Routing, when a node receives a packet from its neighbor, it sends back an estimate of the packet's remaining travel time. In Bellman-Ford routing, the cost tables are exchanged between neighboring nodes in each simulation time step with a probability f.

Two experiments are discussed in this work. The first compares three routing algorithms, namely (1) SHORTEST PATH ROUTING, (2) Q-Routing with *random* start (Q-ROUTING(R)), and (3) Q-Routing with a *shortest path start* (Q-ROUTING(S)). These experiments are intended to demonstrate the learning behavior of Q-Routing under different starting conditions and how they compare with the shortest path routing. The second experiment compares the conventional Bellman-Ford routing with Q-Routing, for their speed and quality of learning.

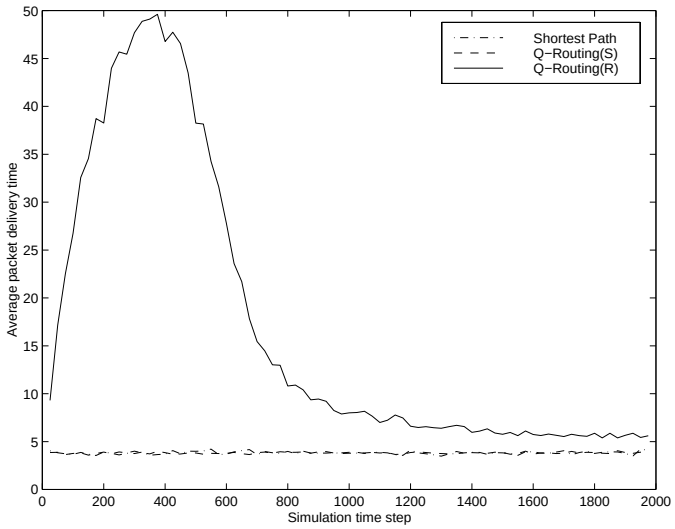

Figure 4.2: **Average packet delivery time at a low load.** While the difference between shortest path and Q-ROUTING(S) is not statistically significant, the difference between Q-ROUTING(R) and Q-ROUTING(S) is statistically significant over all time steps.

4.3 Q-Routing vs. Shortest Path Routing

Figures 4.2 and 4.3 show *Average Packet Delivery Times* at low (1.25 packets per simulation step) and medium (2.5 packets per simulation step) loads for grid topology. Averages over 50 test runs are shown at all loads. Statistical significance is calculated to 99% confidence using standard t-test (W. H. Press 1995).

At *low loads* (figure 4.2), shortest path routing is the best routing algorithm. The average packet delivery time remains at the lowest level for shortest path throughout the simulation. The Q-Routing algorithm, on the other hand, shows different trends depending on whether it starts with random initialization (Q-ROUTING(R)) or the shortest path initialization (Q-ROUTING(S)). In Q-ROUTING(S), as a result of small amount of learning at low loads, the average packet delivery time shows a small increase in the beginning and then settles back to the lowest value. However, the increase is not statistically different from shortest path and is not very obvious in figure 4.2. The reason for this very small amount of initial learning phase is that the Q-values are supposed to be *proportional to* the shortest

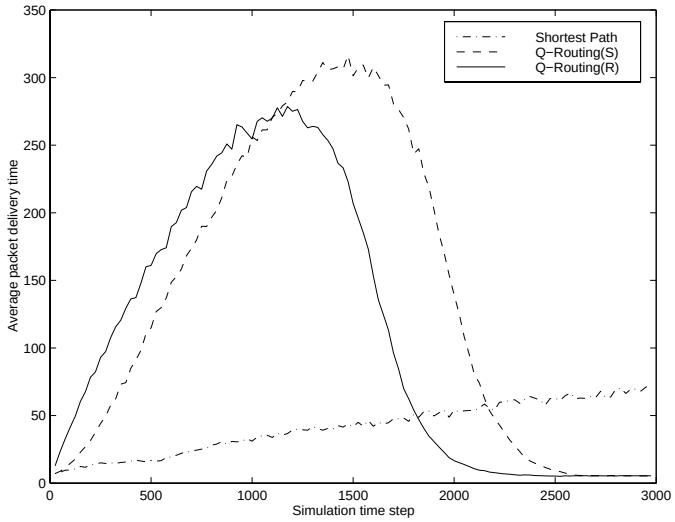

Figure 4.3: **Average packet delivery time at a medium load.** Difference between Q-ROUTING(S) and Q-ROUTING(R) is statistically significant till first 2500 time steps but becomes insignificant after that as they both converge to the same routing policy. Shortest path shows clear trends of congestion.

distance and not necessarily equal. The proportionality constant depends on the load itself and needs to be learned. In Q-ROUTING(R), there is an initial phase of learning in which the average packet delivery time increases. Initially the delivery time is low because statistics for packets that take a long time to reach their destinations are not yet available. As these statistics start pouring in, the average packet delivery time starts increasing. Eventually, after the learning is complete (1000 time steps), the average packet delivery time settles to a low value. Q-ROUTING(R) fails to learn the shortest path routing policy exactly. The difference between Q-ROUTING(R) and shortest path algorithms is statistically significant throughout the simulation time while the difference between Q-ROUTING(S) and shortest path is not significant at any time.

At *medium loads* (figure 4.3) the shortest path routing breaks down and the average packet delivery time grows linearly as the simulation time progresses. This is because the packet queues at popular nodes 12 and 25 increase without bound. A lot of queue waiting time is incurred by packets going through these nodes.

It is interesting to note that at medium loads, Q-ROUTING(S) learns *slower* than

Q-ROUTING(R). The reason is that Q-ROUTING(S) has to first *unlearn* its shortest path routing policy before it can learn the effective policy, which is different from the shortest path. Q-ROUTING(R), on the other hand, takes less time to learn the same policy because it has no initial bias towards any policy and has therefore nothing to *unlearn*. After an initial phase of learning, both Q-Routing algorithms settle down to the qualitatively similar stable routing policy. The difference between Q-ROUTING(R) and Q-ROUTING(S) is statistically significant only during the learning phase till 2500 steps after which they both converge to same policy.

The results in this section are slightly different from those obtained by Boyan and Littman (1994). This is because Boyan and Littman imposed a limit of 1000 on the total number of packets in the network. No such limit exists in our simulations. In the rest of the thesis, only Q-Routing(R) is used for any further comparisons as we are interested in the learning of routing policies from random policy.

4.4 Q-Routing vs. Bellman-Ford Routing

The state of the art distance vector routing algorithm, Bellman-Ford, is compared with Q-Routing in four aspects:

1. *Speed of learning*, which is the time taken for the average packet delivery time of the routing policies to settle down to a stable value.

2. *Quality of the routing policy learned*, which is measured in terms of the average packet delivery time after the learning has settled down.

3. *Stability of the final policy learned*, which is measured in terms of the perturbations in the average packet delivery time; the more perturbations, less stable the policy is.

4. *Exploration overhead*, which is measured in terms of the average number of Routing Information Units (RIU's) sent between neighboring nodes in one time step. The RIU for Q-Routing is a Q-value and RIU for Bellman-Ford routing is one cost table entry.

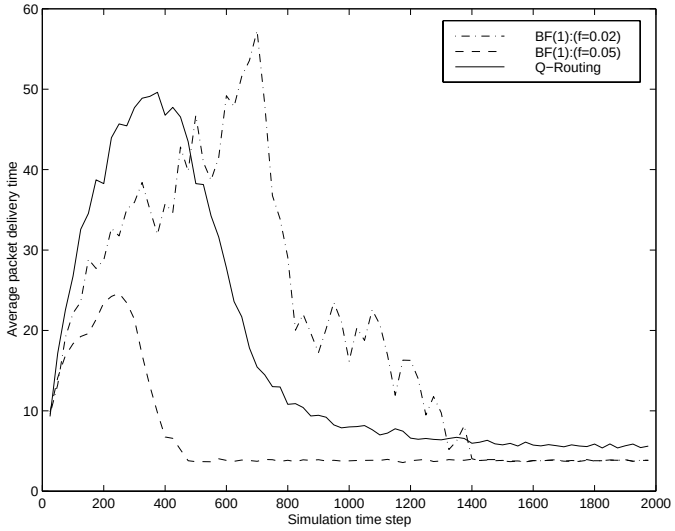

Figure 4.4: **Q-Routing vs Bellman-Ford(1) at low loads**: Both Q-Routing and **BF1** learn the shortest path routing at a low load level of 1.25 pkt/step. Overhead in Q-Routing is only 23 RIU's as compared to 86 RIU's in **BF1** (when $f = 0.02$) for similar speed of learning. The speed of learning for **BF1** can be doubled (for $f = 0.05$) resulting in a 10 fold increase in exploration overhead (212 RIU's).

Again, the results shown are averages over 50 test runs and statistical significance is computed with 99% confidence.

At *low loads*, both Q-routing and **BF1** are initialized randomly and are then allowed to learn. Figure 4.4 shows the average packet delivery time at a low load (1.25 pkt/step) as the learning takes place. Q-Routing takes around 800 time steps to converge. The exploration overhead is very low (just 23 RIU's), and the final policy is very effective (i.e., close to the shortest path routing) and it is highly stable.

In Bellman-Ford routing, the rate of exchange of cost tables between neighboring nodes, largely determine the speed of learning and the amount of exploration. Therefore the performance of **BF1** with two values of f are shown in figure 4.5. For $f = 0.02$, **BF1** learns the optimal policy almost as fast as Q-Routing, but the exploration overhead (86 RIU) is almost 4 times that of Q-Routing. For $f = 0.05$, **BF1** learns the optimal policy almost twice as fast as Q-Routing but the exploration overhead is very high (212 RIU), almost 10 times that of Q-Routing. Hence, as mentioned in section 2.3.2, one major drawback of

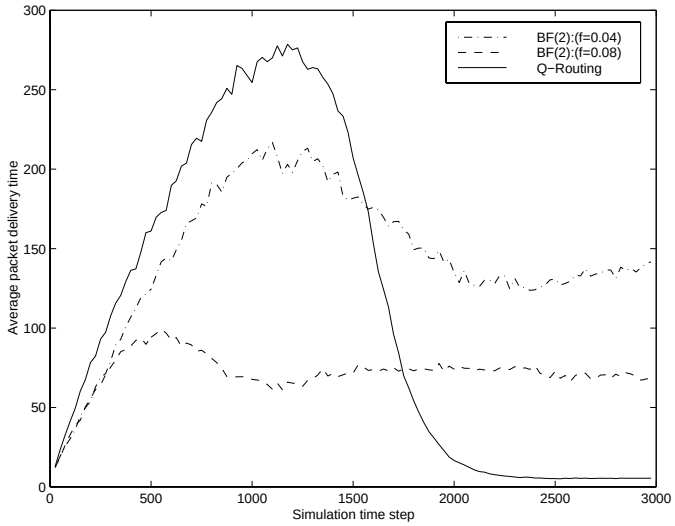

Figure 4.5: **Q-Routing vs Bellman-Ford(2) at medium loads**: Q-Routing learns an effective policy while **BF2** is not able to learn a stable policy. All the differencese are statistically significant.

Bellman-Ford routing is the exorbitant exploration overhead compared to Q-Routing. The final policies learned by **BF1** (both at $f = 0.02$ and $f = 0.05$) are optimal and stable.

At *medium loads* shortest path is not the best policy (as was seen in figure 4.3). Therefore, **BF1**, which converges to shortest path, shows a similar increasing trend as the shortest path algorithm at medium load (figures 4.3). Therefore, the **BF2** version of Bellman-Ford routing, where cost T represents the total delivery time (rather than number of hops), is compared with Q-Routing. Figure 4.5 shows the average packet delivery time at medium load (2.5 pkt/step) as the learning takes place. Q-Routing learns an effective routing policy by around the 2250^{th} time step and the overhead of exploration incurred is only 23 RIU's (Load level does not affect the exploration overhead).

Again two different values of f are used for **BF2**. For $f = 0.04$, the speed of learning was found to be almost the same as that of Q-Routing, but the exploration overhead (RIU = 152) is almost seven times that of Q-Routing. Choosing $f = 0.08$, the learning speed is further increased but again the overhead (RIU = 294) is more than 12 times that of Q-Routing. Moreover, Q-Routing learns a much superior and stable routing policy than **BF2**

for all values of f tested (only two of which are shown in figure 4.5). It is worthwhile to mention that (as pointed out by Boyan and Littman (1994)) Q-Routing fails to revert back to the original routing once the network conditions (topology/traffic condition/load) are reverted back to what they were before. Bellman-Ford routing does not have this problem because exploration does not depend on the routing tables of various nodes while in Q-Routing, it is the packet movement which decides which Q-values should be updated. As a conclusion, there are two main problems with Bellman-Ford. First, its exploration overhead is very high; second, it is not able to learn an effective stable policy at medium and high loads. However, Bellman-Ford has the advantage over Q-Routing that it can revert back to the original routing policy.

4.5 Conclusions

This chapter evaluated Q-Routing on two sets of experiments. In the first set of experiments the ability of Q-Routing to learn an effective routing policy in terms of average packet delivery time was demonstrated over different load conditions. The conclusion is that Q-Routing is capable of adapting to low and medium load conditions but takes a long time to converge at high load conditions. As we will see later, the policy learned at high load is not as good as it could be.

In another set of experiments Q-Routing was compared with the state-of-the-art adaptive distance vector routing, the Distributed Bellman-Ford algorithm. At low loads the best version of Bellman-Ford (learns the shortest path policy) as will Q-Routing. At medium loads, the more general version of Bellman-Ford must be used, but even then it learns an inferior path compared to Q-Routing. Exploration overhead in Bellman-Ford is 7-10 times that of Q-Routing for same speed of learning.

As seen from experiments in section 4.4, higher amount of exploration does increase the speed of learning. There is room for improving the quantity of exploration in Q-Routing by making use of the backward direction of exploration. Moreover, the quality of exploration in Q-Routing is not as good as it could be because the same learning rate

(0.85) is used for updating Q-values without any regard as to how closely the estimated Q-value represents the state of the network. The main contribution of this work is to extend the quality and quantity of exploration of Q-Routing to yield a superior routing algorithm with regard to the speed of adaptation, the quality of final routing policy learned, ability of the routing policy to handle higher load levels and finally their ability to adapt quickly to the changes in network topology and traffic patterns. The resulting algorithm called Confidence based dual reinforcement Q-Routing (CDRQ-Routing) is developed in the next chapter. Since CDRQ-Routing is superior than Q-Routing, Bellman-Ford is not compared with CDRQ-Routing, only Q-Routing is compared with CDRQ-Routing in chapter 6.

Chapter 5

Confidence-Based Dual Reinforcement Q-Routing

In chapter 3, the Q-learning framework was applied to the problem of adaptive network routing, yielding the Q-Routing algorithm (Boyan and Littman 1994). The ability of Q-Routing to learn an effective routing policy starting from a random policy was also demonstrated.

In this chapter, a new routing algorithm based on Q-Routing is developed. The *quality* of exploration is improved by attaching confidence measures to each of the Q-values in the network. These confidence values (C-values) are used in determining the learning rate for the Q-values. Moreover, these C-values are themselves updated in order to reflect how closely the corresponding Q-values represent the current state of the network. The *quantity* of exploration is also increased by including *backward exploration* to the algorithm. Backward exploration doubles the exploration in terms of the number of Q-value updates per packet hop. The combination of these two enhancements constitutes the Confidence-based Dual-reinforcement Q-routing.

Section 5.1 introduces Confidence-based Q-routing (CQ-routing), which consists of Q-Routing together with the confidence values (see figure 1.4). Section 5.2 presents the Dual reinforcement Q-routing, which consists of Q-Routing together with the backward exploration. In section 5.3 these two extensions are combined into Confidence-

5.1 Confidence-based Q-routing

The quality of the routing policy depends mainly on how closely the Q-values in the network represent its current state. Therefore, they must be continuously updated. Depending on the network load and traffic patterns, however, some of the Q-values may not get updated for a long time. Decisions based on such unreliable Q-values are unreliable.

In order to quantify the amount of reliability in the Q-values, *confidence measures* are introduced in Q-Routing in this section. For every Q-value in the network, there is a corresponding confidence value (C-value) between 0 and 1. The interpretation of these C-values is discussed formally in section 5.1.1. Essentially, a low C-value implies that we have low confidence in the corresponding Q-value because it has not been updated for a long time. When a Q-value with low confidence is to be updated, it is advisable to update it more; in other words, the learning rate for this Q-value should be high. Similarly, if the confidence in the new estimate of a Q-value is high then also the learning rate should be high. Section 5.1.2 describes the use of these confidence values in deciding the learning rate. These confidence values are themselves updated so that they decay exponentially with every time step if the corresponding Q-value is not updated. On the other hand, if the Q-value is updated in the last time step then the corresponding C-value should also be updated. Thus every Q value update is associated with a corresponding C-value update. Section 5.1.3 presents the update rules for confidence values. Since the learning rate depends on the reliability (C-value) of the Q-value being updated and that of the estimated Q-value, the *quality of exploration* is improved by updating these Q-values more.

5.1.1 Confidence Measure

Each Q-value $\mathbf{Q}_x(y, d)$ in the network is associated with a measure of confidence $\mathbf{C}_x(y, d)$, which is a real number between 0 and 1. A value of 1 means that there is full confidence in the corresponding Q-value and that this Q-value reflects the current state of the network

(completely reliable). In other words, this Q-value has recently been updated. A value of 0, on the other hand, means that the corresponding Q-value is random and does not necessarily reflect anything about the current state of the network (completely unreliable). In other words, this Q-value has never been updated. With this interpretation, the base case C-values follow directly from the base case Q-values, that is, $\mathbf{Q}_x(y,y) = \delta$ and $\mathbf{C}_x(y,y) = 1$. In other words, there is full confidence in all Q-values of the form $\mathbf{Q}_x(y,y)$. Because all these Q-values are constant, the corresponding C-values are also held constant.

5.1.2 Using Confidence Values

There is no way of telling how reliable a Q-value is, in the standard Q-Routing, and, the learning rate is constant throughout the learning. In CQ-Routing, however, the confidence values reflect the reliability of Q values, and the learning rate depends on the confidence of the Q-value being updated and its new estimate. In particular, when node x sends a packet $P(s,d)$ to its neighbor y, it gets back not only the best estimate of node y for the remaining part of $P(s,d)$'s journey, namely $\mathbf{Q}_y(\hat{z},d)$, but also the confidence value $\mathbf{C}_y(\hat{z},d)$ associated with this Q-value. When node x updates its $\mathbf{Q}_x(y,d)$ value, it first computes the learning rate η_f which depends on both $\mathbf{C}_x(y,d)$ and $\mathbf{C}_y(\hat{z},d)$. The update rule 3.6 is replaced by:

$$\mathbf{Q}_x(y,d) = \mathbf{Q}_x(y,d) + \eta_f(\mathbf{C}_x(y,d), \mathbf{C}_y(\hat{z},d))(\overbrace{\mathbf{Q}_y(\hat{z},d) + q_y + \delta}^{new\ estimate} - \mathbf{Q}_x(y,d)) \tag{5.1}$$

The *learning rate function*, $\eta_f(\mathbf{C}_{old}, \mathbf{C}_{est})$ is chosen based on the following rule:

Rule 5.1: *Learning rate should be* **high** *if either:*

- *Confidence in the* **old Q-value** *is* **low** *or*
- *Confidence in the* **new Q-value** *is* **high**.

A simple and effective learning rate function is given by:

$$\eta_f(\mathbf{C}_{old}, \mathbf{C}_{new}) = \max(\mathbf{C}_{new}, 1 - \mathbf{C}_{old}) \tag{5.2}$$

Since all the C-values are between 0 and 1, the learning rate is also between 0 and 1.

5.1.3 Updating the Confidence Values

The confidence values (except for the base cases) keep changing with time, in order to reflect the reliability of the corresponding Q-values. Depending on whether a Q-value was updated in the last time step or not, different update rules for the C-value apply:

> **Rule 5.2(a)**: *Every C-value except the base cases decays with time if their corresponding Q-values are not updated in the last time step.*

$$\mathbf{C}_x(y,d) = \lambda \mathbf{C}_x(y,d), \tag{5.3}$$

where $\lambda \in (0,1)$ is the decay constant.

> **Rule 5.2(b)**: *If a Q-value is 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:*

$$\mathbf{C}_x(y,d) = \mathbf{C}_x(y,d) + \eta_f(\mathbf{C}_x(y,d), \mathbf{C}_y(\hat{z},d))(\mathbf{C}_y(\hat{z},d) - \mathbf{C}_x(y,d)) \tag{5.4}$$

The C-value update equation 5.4 uses the old C-value of the Q-value being updated and the C-value of the Q estimate coming from neighboring node to first compute the learning rate. This learning rate is used in the Q-value update rule. Same learning rate is then used to update the C-value. If the learning rate is high, the confidence in the updated Q-value is closer to that of the estimated Q-value's confidence.

5.1.4 Overhead Analysis of CQ-Routing

Three additional overheads are associated with the confidence values in CQ-Routing are:

1. **Confidence value transmission overhead** (t_{cx}): After node x sends a packet $P(s,d)$ for destination d to node y, node y sends back to node x not only the estimate $Q_y(\hat{z},d)+q_y$, but also the confidence value $C_y(\hat{z},d)$. If $\mathbf{e}$ is the size of the estimate and $\mathbf{c}$ is the size of the confidence value then the estimate-confidence value transmission overhead is given by:

$$t_{cx} = \delta \left(\frac{\mathbf{e}+\mathbf{c}}{\mathbf{p}} \right). \tag{5.5}$$

2. **Learning rate computation overhead** (t_{lr}) is defined as the time taken by node x to compute the learning rate once the confidence value of the Q-value being updated and that of the estimate is known. Learning rate can be obtained from these two confidence values in $O(1)$ time using equation 5.2.

3. **Confidence value update overhead** ($t_{cu}(x)$) is the total time taken to update all the confidence values at node x. With every time step, all confidence values are updated irrespective of whether the corresponding Q value is updated or not. Either equation 5.3 or equation 5.4 is applied to each of the confidence value $C_x(*,*)$ at node x. Each update itself is $O(1)$. If $\mathrm{N}(x)$ is the number of neighbors of node x and n is the number of nodes in the network then the number of C-values updated in each time step at node x is $(n - |\mathrm{N}(x)|)|\mathrm{N}(x)|$ (the base case C-values are not updated at all). Thus the overhead due to confidence value updates at node x is given by:

$$t_{cu}(x) = O(n|N(x)|). \tag{5.6}$$

These overheads are not significant compared to the gain in performance as shown in chapter 6.

5.2 Dual Reinforcement Q-routing

In this section, DUAL-REINFORCEMENT Q-ROUTING (DRQ-ROUTING) based on the general idea of *Dual Reinforcement Learning* (Goetz, Kumar & Miikkulainen 1995), is discussed in detail. This algorithm is obtained by adding the other direction of exploration, *backward exploration*, into Q-Routing. After introducing the basic idea of dual reinforcement learning, its use in DUAL-REINFORCEMENT Q-ROUTING is presented in this section.

5.2.1 Dual Reinforcement Learning

The basic element in reinforcement learning is the *reinforcement signal*. As mentioned in section 1.2 there is no direct reinforcement signal available for the routing decisions

taken at individual nodes. Such training signal can be generated at the destination node but to make it available to all the nodes in the path, significant network resources will be consumed. Due to this limitation, a straightforward reinforcement learning cannot be applied to the routing problem. Goetz, Kumar, and Miikkulainen (1995) developed the *Dual Reinforcement Learning* algorithm for such situations. Instead of trying to use the final reinforcement signal, an indirect reinforcement signal is extracted from the incoming information and is used to update the local decision maker.

Goetz et.al. applied dual reinforcement learning to on-line adaptation of channel predistorters in satellite communication. Consider two distant places A and B connected by a satellite communication link. Both have a sender and a receiver unit. The sender units S(A) and S(B) at A and B have predistorters P(A) and P(B). The idea of a predistorter is to distort the signal before sending it over a communication channel, so that when the signal is received on the other side, the distortions due to the channel are cancelled out. When A sends a signal s to B, the signal is first predistorted by P(A). When B receives this signal, a training (reinforcement) signal $r(P(A), s)$, estimating how well the predistorter P(A) has done its job on signal s, can be computed at B. In order to update the predistorter P(A), $r(P(A), s)$ should be made available at A, but sending it from B to A over the satellite communication link would incur a large overhead and therefore is not feasible. Both P(A) and P(B) are trying learn the same control action for the same environment, that is, the atmosphere. Hence starting from the same P(A) and P(B), it is safe to assume that P(B) would have generated the same reinforcement for the signal s, that is, $r(P(A), s)$=$r(P(B), s)$. Making use of this symmetric property, P(B) can be updated using s and $r(P(A), s)$.

The problem in network routing is similar to that in satellite communication. The training signal for any of the local routing decisions is not available directly. In fact a communication network is a more complex system than the two node satellite communication system discussed above, since (1) there are more than two nodes, and (2) the environment (i.e., the communication network) is not symmetric in the two directions across a link. The

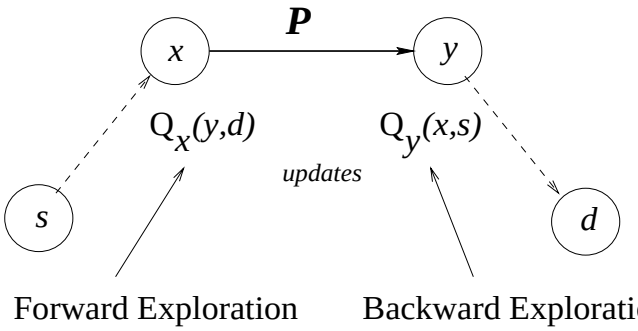

Figure 5.1: **Forward and backward exploration**: Q-value $\mathbf{Q}_x(y,d)$ of sending node x is updated in forward exploration, while Q-value $\mathbf{Q}_y(x,s)$ of receiving node y is updated in backward exploration when a packet $P(s,d)$ (originated at node s and destined to node d) hops from x to y.

dual reinforcement learning idea can be used in the network routing problem as follows. When a node x sends a packet to neighboring node y, some additional routing information can be sent along with the packet. This information can be used to update node y's decisions in the direction opposite to the direction of the packet. This update adds *backward exploration* to Q-Routing.

5.2.2 Backward Exploration

The Q-Routing algorithm makes use of forward exploration in updating the Q-values in the network. It may be recalled that in forward exploration, the Q-value of the sending node is updated based on the information coming from the receiving node (section 3.4). DRQ-Routing makes use of *backward exploration* as well (see figure 5.1). When a node x sends a packet $P(s,d)$ to one of its neighbors, y, the packet can take along information about the Q-values of node x. When node y receives this packet, it can use this information in updating its Q-values pertaining to the neighbor x. Figure 5.1 summarizes both the forward and backward exploration in each packet hop. Later when node y has to make a decision, it can use the updated Q values for x. The only overhead is a slight increase in the size of the packets.

Let s denote the source node of packet $P(s,d)$, which is currently at node x. This packet carries to node y the estimated time $\mathbf{Q}_x(\hat{z},s)$ it takes for a packet destined to node

s to travel from node x:

$$\mathbf{Q}_x(\hat{z}, s) = \min_{z \in N(x)} \mathbf{Q}_x(z, s) \tag{5.7}$$

This value is essentially an estimate of the minimum time it would take for the packet to reach back to its source s from node x. Upon receiving this estimate, node y computes the new estimate for $\mathbf{Q}_y(x, s)$ as follows:

$$\mathbf{Q}_y(x, s)^{est} = \mathbf{Q}_x(\hat{z}, s) + q_x + \delta \tag{5.8}$$

Note that this estimate is based on the *optimal triangular equality* (equation 3.2). Node y then updates its $\mathbf{Q}_y(x, s)$ value as follows:

$$\mathbf{Q}_y(x, s)^{new} = \mathbf{Q}_y(x, s)^{old} + \eta_b(\mathbf{Q}_y(x, s)^{est} - \mathbf{Q}_y(x, s)^{old}) \tag{5.9}$$

where η_b is the learning rate for *backward exploration.* Substituting and expanding 5.9, the update rule is given by:

$$\mathbf{Q}_y(x, s) = \mathbf{Q}_y(x, s) + \eta_b(\overbrace{\mathbf{Q}_x(\hat{z}, s) + q_x + \delta}^{new\ estimate} - \mathbf{Q}_y(x, s)) \tag{5.10}$$

The *general inequality* (equation 3.1) can be restated for backward exploration as:

$$\mathbf{Q}_y(x, s) \leq q_x + \delta + \mathbf{Q}_x(z, s) \quad \forall y \in N(x) \quad \text{and} \quad \forall z \in \mathbf{N}(x). \tag{5.11}$$

for all $y \in \mathrm{N}(x)$ and for all $z \in \mathbf{N}(x)$. Similarly, the *optimal triangular equality* (equation 3.2) for backward exploration is:

$$\mathbf{Q}_y(x, s) = q_x + \delta + \mathbf{Q}_x(\hat{z}, s) \tag{5.12}$$

5.2.3 Properties of Backward Exploration

Three properties of backward exploration, original to this thesis, are stated and proved in this section. The first property shows that the update rule given by equation 5.10 is admissible (section 3.5). In other words, the *general inequality* (equation 5.11) is invariant in the backward exploration update. The second property shows that in this update, Q

values asymptotically converge to the shortest path routing at low loads. These properties follow from the update rules proposed in the previous section. The third property argues that the quality of exploration in backward exploration is better than that in forward exploration.

Property 5.1: *The update rule (equation 5.10) guarantees that if the old value of* $\mathbf{Q}_x(y,s)$ *satisfies the general inequality (equation 5.11), then its updated value also satisfies the general inequality. i.e. update rule 5.10 is admissible.*

Proof: Let $\mathbf{Q}_y(x,s)'$ denote the value of $\mathbf{Q}_y(x,s)$ updated using equation 5.10. If the old value of $\mathbf{Q}_y(x,s)$ satisfies the *general inequality*, then for any neighbor z of node y and some non-negative $\theta(z)$,

$$\mathbf{Q}_y(x,s) = q_x + \delta + \mathbf{Q}_x(z,s) - \theta(z). \tag{5.13}$$

Also since $\mathbf{Q}_y(\hat{z},d)$ is the best estimate of node y, by definition this has to be the minimum of all other estimates of node y for the same destination d. Thus for all $z \in \mathrm{N}(x)$,

$$\mathbf{Q}_y(\hat{z},d) = \mathbf{Q}_x(z,s) - \theta'(z,\hat{z}) \tag{5.14}$$

for some non-negative $\theta'(z,\hat{z})$. Substituting 5.13 and 5.14 in the update rule 5.10 and simplifying,

$$\mathbf{Q}_y(x,s)' = q_x + \delta + \mathbf{Q}_x(z,s) - [(1-\eta_b)\theta(z) + \eta_b\theta'(z,\hat{z})], \tag{5.15}$$

which can be rewritten as:

$$\mathbf{Q}_y(x,s)' \leq q_x + \delta + \mathbf{Q}_x(z,s). \tag{5.16}$$

Hence the updated Q-value also satisfies the *general inequality.*

Property 5.2: *For low network loads, the routing policy learned by the DRQ-Routing update rule (equation 5.10) asymptotically converges to the shortest path routing.*

Proof: Consider the complete route $(x_0(=s), x_1, ..., x_l(=d))$ of length l for a packet $P(s,d)$. The l Q-value updates due to backward exploration (equation 5.10) along this route are given by the following generic form ($i = 0...l-1$):

$$\mathbf{Q}_{x_{i+1}}(x_i, x_0) = \mathbf{Q}_{x_{i+1}}(x_i, x_0) + \eta_b(\mathbf{Q}_{x_i}(x_{i-1}, x_0) + q_{x_i} + \delta - \mathbf{Q}_{x_{i+1}}(x_i, x_0)). \quad (5.17)$$

Equation 5.17 follows from the fact that if $(x_0, x_1, ..., x_l)$ is the shortest route between x_0 and x_l, then the best estimate of node x_i to the node x_0 would be $\mathbf{Q}_{x_i}(x_{i-1}, x_0)$, and it is this estimate that goes along with the packet from node x_i to node x_{i+1}, where it is used to update the estimate $\mathbf{Q}_{x_{i+1}}(x_i, x_0)$.

The base case for backward exploration follows from the boundary conditions in section 3.1:

$$\mathbf{Q}_{x_1}(x_0, x_0) = \delta. \quad (5.18)$$

Like in the proof of property 3.2, assume that for low loads, all q_{x_i} are small compared to δ. Using these simplifications, equation 5.17 can be rewritten as:

$$\mathbf{Q}_{x_{i+1}}(x_i, x_0) = \mathbf{Q}_{x_{i+1}}(x_i, x_0) + \eta_b(\mathbf{Q}_{x_i}(x_{i-1}, x_0) + \delta - \mathbf{Q}_{x_{i+1}}(x_i, x_0)). \quad (5.19)$$

If this were the shortest route between nodes x_0 and x_l then the following would hold for the Q-values at each of the intermediate nodes:

$$\mathbf{Q}_{x_i}(x_{i-1}, x_0) = i\delta. \quad (5.20)$$

Equation 5.20 is proved by *induction* over i:

Base case: $(i = 1)$ follows directly from equation 5.18 by substituting for i in 5.20.

Induction Hypothesis: Assume 5.20 for some i. Substituting $\mathbf{Q}_{x_i}(x_{i-1}, x_0)$ from 5.20 into 5.19 we get:

$$\mathbf{Q}_{x_{i+1}}(x_i, x_0) = \mathbf{Q}_{x_{i+1}}(x_i, x_0) + \eta_b((i+1)\delta - \mathbf{Q}_{x_{i+1}}(x_i, x_0)). \quad (5.21)$$

Repeated updates using the update equation 5.21 will eventually lead to:

$$\mathbf{Q}_{x_{i+1}}(x_i, x_0) = (i+1)\delta. \tag{5.22}$$

Equation 5.22 proves the induction hypothesis for $i+1$. Hence the property 5.2 holds for all i.

Property 5.3: *Q-value updates in backward exploration are more accurate than Q-value updates in forward exploration.*

Proof: A Q-value update is more accurate if the estimate coming from the neighboring node for the update has been updated more recently. Consider a route $(x_0, x_1, ..., x_l)$ of a packet from source node x_0 to destination node x_l. For this route, forward exploration update of the Q-value $\mathbf{Q}_{x_i}(x_{i+1}, x_l)$ takes place **before** the update of the Q-value $\mathbf{Q}_{x_{i+1}}(x_{i+2}, x_l)$. In other words the Q-value used for updating $\mathbf{Q}_{x_i}(x_{i+1}, x_l)$ is itself updated **afterwards**, and hence is less accurate. In backward exploration, however, the Q-value $\mathbf{Q}_{x_{i+1}}(x_i, x_0)$ is updated **after** the update of the Q value $\mathbf{Q}_{x_i}(x_{i-1}, x_0)$. Therefore, the Q-value used for updating $\mathbf{Q}_{x_{i+1}}(x_i, x_0)$ is itself updated **before** and is more accurate.

5.2.4 Overhead Analysis of DRQ Routing

The overheads discussed in chapter 3 for *forward exploration* hold for DRQ-Routing too. An additional overhead stems from the increased size of the packet being transmitted at each hop. Using the same notation as before, if $\mathbf{p}$ is the size of the original packet and $\mathbf{e}$ is the additional size of the estimate value that is appended to this packet, the total packet size forwarded in each hop is ($\mathbf{p}$+$\mathbf{e}$). The time taken to transmit this longer packet also increases. If δ is the transmission delay of a packet of size $\mathbf{p}$, and since this time is proportional to the length of the packet, the time taken to transmit a new packet augmented with the estimate is given by:

$$t_x = \delta(1 + \frac{\mathbf{e}}{\mathbf{p}}). \tag{5.23}$$

The percentage increase in time is therefore only $\mathbf{e/p}$ which is less than 0.1% (see also section 3.6). Hence the total significant overhead in DRQ-Routing is only $2\mathbf{e/p}$. As will be shown in chapter 6, this overhead is acceptable, considering the gains in speed and quality of adaptation due to backward exploration.

5.3 Confidence-based Dual Reinforcement Q-routing

In the last two sections, two ways of improving the basic Q-Routing were proposed. In CQ-Routing, the *quality of exploration* was improved by learning faster when the Q-values represent the current state of the network more closely. In DRQ-Routing, on the other hand, the quantity of exploration was improved by adding another direction of exploration to Q-Routing. In this chapter, these two features are combined into CDRQ-Routing. Experiments in chapter 6 show that CDRQ-Routing is superior than both CQ-Routing and DRQ-Routing independently.

5.3.1 Combining CQ and DRQ Routing

CDRQ-Routing combines the features of both CQ-ROUTING and DRQ-Routing. Thus, with each hop of a packet $P(s,d)$ from node x to node y, the Q and C-values of both nodes x and y are updated in the forward and backward exploration, respectively. Following is the summary of the four updates associated with a single hop of a packet from node x to node y in CDRQ-Routing. The confidence update rules and variable learning rate based on old and new C-values for backward exploration are given below:

1. **Q-value update of sender node**(x) (FORWARD EXPLORATION):

$$\mathbf{Q}_x(y,d) = \mathbf{Q}_x(y,d) + \eta_f(\mathbf{C}_x(y,d), \mathbf{C}_y(\hat{z},d))(\overbrace{\mathbf{Q}_y(\hat{z},d) + q_y + \delta}^{new\ estimate} - \mathbf{Q}_x(y,d)), \quad (5.24)$$

2. **C-value** update of **sender node**(x) (FORWARD EXPLORATION):

$$\mathbf{C}_x(y,d) = \mathbf{C}_x(y,d) + \eta_f(\mathbf{C}_x(y,d), \mathbf{C}_y(\hat{z},d))(\mathbf{C}_y(\hat{z},d) - \mathbf{C}_x(y,d)), \quad (5.25)$$

3. **Q-value** update of **receiver node**(y) (BACKWARD EXPLORATION):

$$\mathbf{Q}_y(x,s) = \mathbf{Q}_y(x,s) + \eta_b(\mathbf{C}_y(x,s), \mathbf{C}_x(\hat{z},s))(\overbrace{\mathbf{Q}_x(\hat{z},s) + q_x + \delta}^{new\ estimate} - \mathbf{Q}_y(x,s)), \quad (5.26)$$

4. **C-value** updates of **receiver node**(y) (BACKWARD EXPLORATION):

$$\mathbf{C}_y(x,s) = \mathbf{C}_y(x,s) + \eta_b(\mathbf{C}_y(x,s), \mathbf{C}_x(\hat{z},s))(\mathbf{C}_x(\hat{z},s) - \mathbf{C}_y(x,s)), \quad (5.27)$$

5.3.2 Summary of the CDRQ Routing Algorithm

Like Q-Routing, the complete CDRQ-Routing algorithm can be summarized in terms of two steps: The $PacketReceive_y(x)$ step describes what the node y does when it receives a packet from one of its neighboring nodes, x and the $PacketSend_x$ step describes what node x does when it has to send a packet. These two steps for CDRQ-Routing are given in tables 5.1 and 5.2:

1	If (not EMPTY(*PacketQueue(x)*) go to step 2
2	$P(s,d)$ = *Dequeue* first packet from *PacketQueue*(x)
3	Compute best estimate $\mathbf{Q}_x(\hat{z},d) = \min_{z \in N(x)} \mathbf{Q}_x(z,s)$.
4	Append $(\mathbf{Q}_x(\hat{z},s)+q_x)$ and $\mathbf{C}_x(\hat{z},s)$ to the packet $P(s,d)$.
5	Compute best neighbor $\hat{y} = arg\min_{y \in N(x)} \mathbf{Q}_x(y,d)$.
6	*ForwardPacket* $P(s,d)$ to neighbor $\hat{y}$.
7	Wait for $\hat{y}$'s Q estimate and C-value.
8	*ReceiveEstimate*$(\mathbf{Q}_{\hat{y}}(\hat{z},d) + q_{\hat{y}})$ and $\mathbf{C}_{\hat{y}}(\hat{z},d)$ from node $\hat{y}$.
9	Compute the learning rate $\eta_f(\mathbf{C}_x(y,d))$,$\mathbf{C}_{\hat{y}}(\hat{z},d))$ using 5.2
10	*UpdateQvalue*$(\mathbf{Q}_x(y,d))$ as given in 5.24.
11	*UpdateCvalue*$(\mathbf{C}_x(y,d))$ as given in 4.4.
12	Update all other C-values using 5.3
13	Get ready to send next packet (goto 1).

Table 5.1: **PacketSend**$_x(P(s,d))$ at NODE x for CDRQ-Routing

1	Receive a packet $P(s,d)$ from neighbor x.
2	Extract the estimate ($\mathbf{Q}_x(\hat{z},d)+q_x$) and $\mathbf{C}_x(\hat{z},s)$ from packet $P(s,d)$.
3	Compute learning rate $\eta_b(\mathbf{C}_x(\hat{z},s)),\mathbf{C}_y(x,s))$ using 5.2
4	$UpdateQvalue(\mathbf{Q}_y(x,s))$ value using equation 5.26.
4	$UpdateCvalue(\mathbf{C}_y(x,s))$ value using equation 4.27.
5	Calculate best estimate for node d, $\mathbf{Q}_y(\hat{z},d) = \min_{z \in N(y)} \mathbf{Q}_y(z,d)$.
6	Send ($\mathbf{Q}_y(\hat{z},d) + q_y$) and $\mathbf{C}_y(\hat{z},d)$ back to node x.
7	If ($d \equiv y$) then $ConsumePacket_y(P)$ else goto 8
8	If ($PacketQueuey$) is FULL) then $DropPacket(P(s,d))$ else goto 9
9	$AppendPacketToPacketQueue_y(P(s,d))$.
10	Get ready to receive next packet.

Table 5.2: **PacketReceive**$_y(x)$ at NODE y for CDRQ-Routing

5.4 Conclusion

Exploration is essential to adaptive routing. Two ways of improving the exploration mechanism of Q-Routing were discussed in detail in this chapter. These lead to two intermediate adaptive routing algorithms, namely CQ-ROUTING, which improves the quality of exploration by maintaining C values for every Q-value in the network and using them to compute the learning rate for the update rules, and DRQ-Routing, which improves the quantity of exploration through an additional backward exploration, leading to two Q-value updates per packet hop as against only one in Q-Routing. These two features were combined into a single adaptive routing algorithm, in CDRQ-Routing.

Experimental evaluation of these algorithms on two different network topologies is provided in the next chapter. Comparison of speed and quality of adaptation and ability to adapt to changing traffic patterns and network topologies for Q-Routing, CQ-Routing, DRQ-Routing and CDRQ-Routing shows that the quality and quantity of exploration does improve the overall learning and adapting capabilities of an adaptive routing algorithm.

Chapter 6

Evaluating CDRQ-Routing

In the previous chapter, the CDRQ-ROUTING algorithm was introduced. Extending the basic Q-Routing, the complete algorithm has two components, the confidence part (CQ-Routing) and the backward exploration part (DRQ-Routing). In this chapter, Q-Routing is compared with CDRQ-ROUTING and its two components CQ-Routing and DRQ-Routing over two network topologies.

Section 6.1 describes the experimental setup. This section is followed by four comparative results: learning at constant load levels in section 6.2, adaptation to variable traffic conditions in section 6.3, adaptation to changing network topology in section 6.4, and sustaining high load levels in section 6.5.

6.1 Experimental Setup

The Q tables were initialized with small random Q values, except for the base cases (section 3.2). In CQ-ROUTING and CDRQ-ROUTING, the C values were all initialized to 0 except for the base cases (section 5.1.1). Learning rate for forward exploration η_f, learning rate for backward exploration η_b, and the C value decay constant λ are summarized in table 6.1. Performance of the algorithms was found to be the best with these parameters.

Note that $\eta_b > \eta_f$ in DRQ-Routing. The reason for this difference follows from prop-

Algorithm	η_f	η_b	λ
Q-Routing	0.85	-	-
CQ-Routing	-	-	0.95
DRQ-Routing	0.85	0.95	-
CDRQ-Routing	-	-	0.90

Table 6.1: **Parameters for the Algorithms**. In Q-Routing and DRQ-ROUTING, the learning rates are constant and there are neither C values, nor decay constants. In CQ-ROUTING and CDRQ-Routing, the learning rates for both the forward and backward exploration depend on the C values hence there are no explicit learning rates.

erty 5.1, which states that Q value updates in backward exploration are more accurate than those in forward exploration. The performance of DRQ-Routing was also experimentally found to be better when $\eta_b > \eta_f$. Also, note that the decay constant in CDRQ-Routing is smaller than that in CQ-Routing. This is because two Q values are updated per packet hop in CDRQ-Routing, instead of only one in CQ-Routing, and therefore the Q values in CDRQ-Routing tend to be more reliable than those in CQ-Routing. With a smaller decay constant (i.e. with a faster decay), higher penalty for reliability is incurred in CDRQ-Routing. If some Q value is not updated for a long time, the confidence in that Q value will decay faster and when it is updated later on the learning rate will be higher (rule 5.1, equation 5.2). As a result, the quality of exploration is balanced with the quantity of exploration to keep the Q values as reliable as possible.

6.2 Learning at Constant Loads

In the first set of experiments, the load level was maintained constant throughout the simulation. Results on two network topologies, namely the 36 node irregular 6×6 grid (figure 3.1) and a 128 node 7-D hypercube are presented. The speed and quality of adaptation at three load levels, low, medium and high, were compared. The typical load level values for the two topologies are given in table 6.2.

The adaptive routing algorithms were found to have relatively similar performance and learning behaviors (in average packet delivery time) within a given range. The per-

Topology	# nodes	Low Load	Medium Load	High Load
6 × 6 irregular grid	36	0 - 1.75	1.75 - 2.50	2.50 - more
7-D hypercube	128	0 - 5	5 - 8	8 - more

Table 6.2: **Load level ranges** The number stands for number of packets introduced in the network per time step. The learning behavior of adaptive routing algorithms remains roughly similar within a given load range (low, medium or high), but if the load changes from one range to another, the behavior can change quite dramatically. In real life communication networks, the load is usually in the medium range, and occasionally changes to low or high levels.

formance and learning behavior is significantly different from one load level to another for some routing algorithms. For example, at low load, shortest path performs very well but at medium and high load levels, it breaks down completely (figures 4.2 and 4.4). As another example, learning behavior of Q-Routing shows different behavior at medium and high load levels (figure 6.3 and 6.5).

Three representative load levels, one in each of the three ranges, were used in the experiments. For the grid topology, they were 1.25 pkts/step, 2.5 pkts/step and 3.5 pkts/step. For 7-D hypercube topology, they were 3 pkts/step, 7 pkts/step and 10 pkts/step. The load level ranges depend on the topology, more specifically on the average branching factor and number of nodes in the network. The 36 node 6 × 6 grid and the 128 node 7-D hypercube have different topologies, hence their load ranges are also different. These ranges were decided after a number of experiments.

The learning behavior was observed in terms of average packet delivery time and number of packets in the network during learning. The packet delivery times of all packets reaching their destination in a window of 25 time steps were averaged. Similarly, the number of packets in the network were averaged for every successive window of 25 time steps. Results averaged over 50 simulation runs, each starting with random initializations of Q values for both network topologies are reported in figures 6.1 through 6.12. Statistical significance is computed at 99% confidence using the standard student's t-test (W. H. Press 1995).

At **low load levels**, the average packet delivery time for both the grid (figure 6.1)

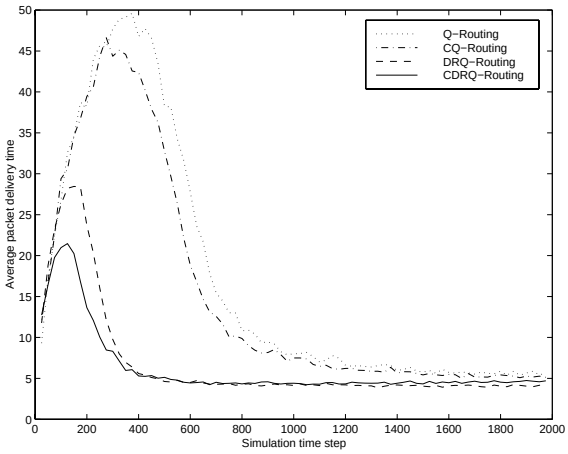

Figure 6.1: **Average packet delivery time at a low load level for the grid topology**: Difference between Q-Routing and CQ-Routing is statistically significant between 300 to 800 time steps and difference beteween CDRQ-Routing/DRQ-Routing and Q-Routing/CQ-Routing is significant from 50 to 1200 time steps.

and the hypercube (figure 6.3) and the number of packets in the network for the grid (figure 6.2) and the hypercube (figure 6.4) shows that DRQ-Routing learns an effective routing policy much faster than CQ-Routing. There is not much gain in the speed of learning from DRQ-Routing to CDRQ-Routing. Reason for this trend is that at low loads, what matters the most is the amount of exploration and the algorithm that allows more exploration per packet hop will learn faster. Adding confidence values to DRQ-Routing, leading to CDRQ-Routing, does not help much. CDRQ-Routing learns more than 3 times as fast as Q-Routing for both topologies.

At **medium load levels**, the average packet delivery times (figure 6.5 for the grid and 6.7 for the hypercube) and the number of packets in the network (figure 6.6 for the grid and 6.8 for the hypercube) show that both DRQ-Routing performs slightly better than CQ-Routing. CDRQ-Routing combines the benefits of both the C values and the backward exploration, thereby achieving increased performance over Q-routing, DRQ-Routing and CQ-Routing. This result is significant because it highlights the contribution of both the *quality* and *quantity* of exploration in learning. They contribute in two different ways to increase the performance of CDRQ-Routing.

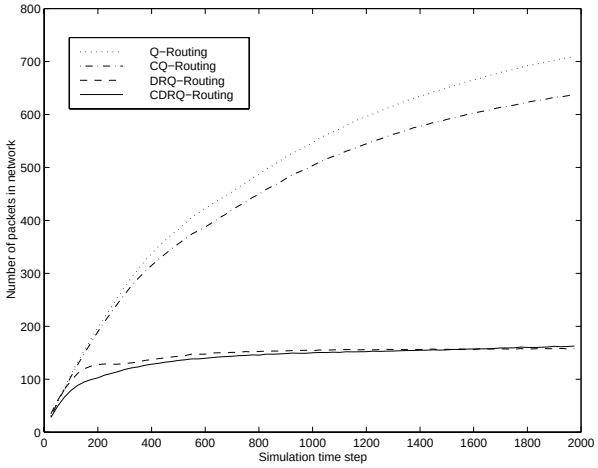

Figure 6.2: **Number of packets in the network for the grid topology at a low load level**: Difference between Q-Routing and CQ-Routing is statistically significant after 300 times steps. Difference between CDRQ-Routing/DRQ-Routing and Q-Routing/CQ-Routing is significant after 175 time step.

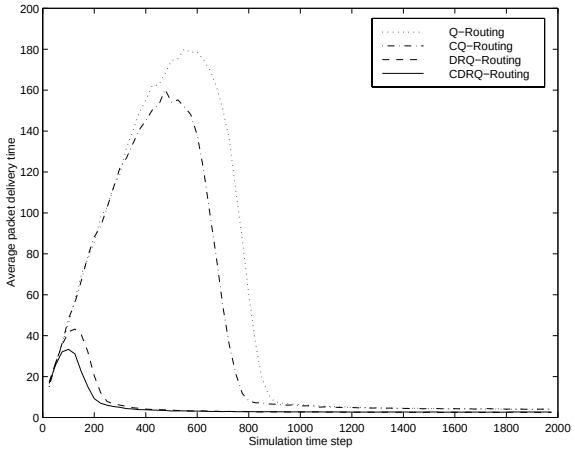

Figure 6.3: **Average packet delivery time for 7-D hypercube at low load**: Difference between Q-Routing and CQ-Routing is statistically significant between 450 to 850 times steps. Difference between CDRQ-Routing/DRQ-Routing and Q-Routing/CQ-Routing is significant between 175 to 800 time steps.

At **high load levels** the average packet delivery time (figure 6.9) and the number of packets in the network (figure 6.10) for the grid topology show that while Q-Routing converges to a qualitatively poor routing policy, CQ-Routing shows a significant improvement in quality of routing policy. DRQ-Routing and CDRQ-Routing converge to qualitatively similar policies which are significantly better than that to which CQ-Routing converged to.

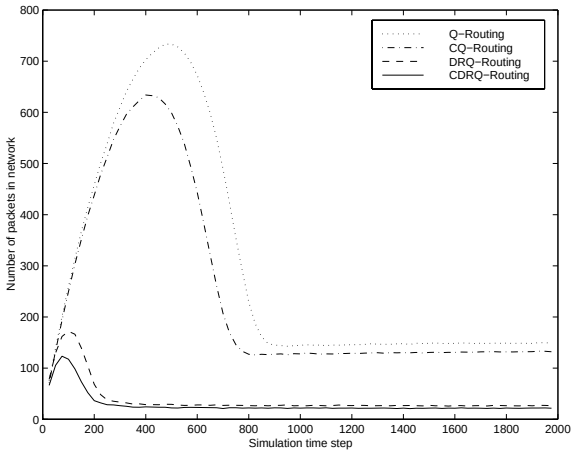

Figure 6.4: **Number of packets for 7-D hypercube at low load**: Difference between Q-Routing and CQ-Routing is statistically significant between 300 to 800 times steps. Difference between CDRQ-Routing/DRQ-Routing and Q-Routing/CQ-Routing is significant after 150 time step.

The speed of convergence of CDRQ-Routing is slightly better than that of DRQ-Routing. The result again signifies the independent and complementary contributions of features from CQ-Routing and DRQ-Routing in improving adaptation speed of CDRQ-Routing high loads. In case of hypercube topology, Q-Routing learns qualitatively similar routing policy as CDRQ-Routing but learning is twice as fast in the later.

The learning behavior for Q-Routing and CDRQ-Routing (and its two versions) is similar in both topologies at low and medium load levels. However, at high loads Q-Routing fails to converge to a policy qualitatively as good as CDRQ-Routing for grid topology but this trend is not reflected in the hypercube topology. The reason being that the hypercube is a very symmetric network with not much alternatives to choose from while in grid topology, there could be multiple routing policies due to no symmetry in the topology. Hence, the performance improvements might vary with topology but in general CDRQ-Routing was found in all cases to improve significantly in speed of learning and in some topologies even in quality of the policy learned. Only grid topology is used in the next two sets of experiments, evaluating and comparing the performance of adaptive routing algorithms for adaptation to changes in traffic pattern and network topology.

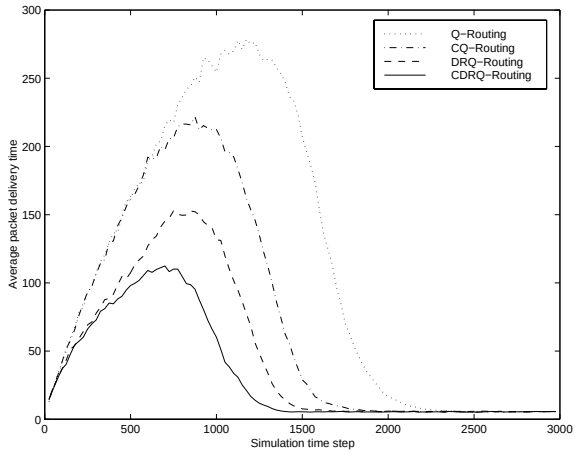

Figure 6.5: **Average packet delivery time for the grid topology at a medium load level**: Difference between Q-Routing and CQ-Routing is statistically significant between 900 to 1900 times steps. Difference between DRQ-Routing and CDRQ-Routing is significant between 750 to 1300 time steps. Difference between CDRQ-Routing and Q-Routing is significant between 300 to 1900 time steps.

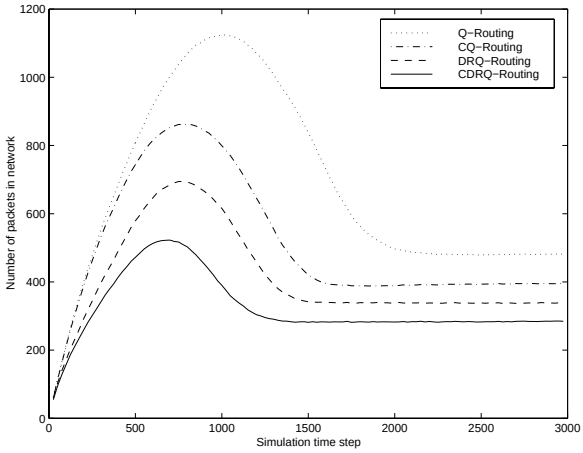

Figure 6.6: **Number of packets in the network for the grid topology at a medium load level**: Differences between all pairs of algorithms are significant after 500 time step and remain so for ever.

6.3 Adaptation to Varying Traffic Conditions

In the second set of experiments, Q-Routing, CQ-Routing, DRQ-Routing and CDRQ-Routing, are compared with respect to their ability to adapt to variations in traffic patterns in the grid topology. All algorithms were first allowed to learn an effective routing policy at a

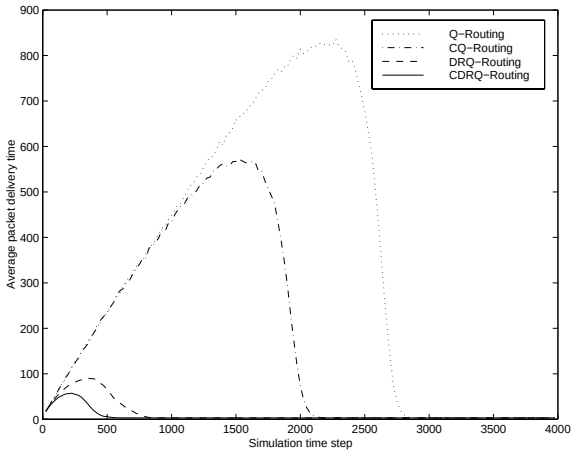

Figure 6.7: **Average packet delivery time for 7-D hypercube at medium load**: Difference between Q-Routing and CQ-Routing is statistically significant between 1500 to 2750 times steps. Difference between DRQ-Routing and CDRQ-Routing is significant between 200 to 600 time steps. Difference between CDRQ-Routing and Q-Routing is significant between 200 to 2750 time steps.

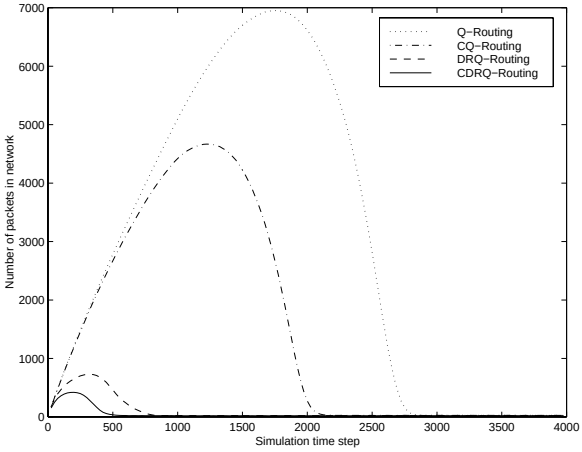

Figure 6.8: **Number of packets for 7-D hypercube at medium load**: Trends are very similar to the ones in figure 6.7. The significance limits are also same as in figure 6.7.

load level of 2 pkts/step and a uniform traffic pattern, where the probability that each node generates a packet to any other node is equal. After convergence (in 2000 simulation steps), the traffic pattern was changed so that the probability of generating a packet destined to a node *across* the cluster becomes 10 times higher than *within* the cluster (figure 3.1). Only single run is shown in this case to depict the variability during adaptation process. Results

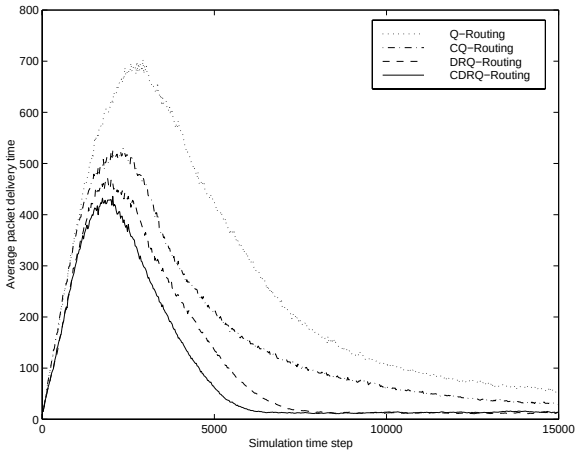

Figure 6.9: **Average packet delivery time for grid topology at high load**: Difference between Q-Routing and CQ-Routing is statistically significant after 1000 time steps. Difference between CDRQ-Routing and Q-Routing is significant between after 1000 time steps.

from 50 test runs were used to compute statistical significance and are reported below.

The average packet delivery time for each algorithms is shown in figure 6.13. All four algorithms converge to an effective routing policy for the initial traffic condition, by time step 2000. As the traffic pattern changes at 2000, the old routing policy is no longer effective and all algorithms start updating their Q values to adapt to the change in the pattern. CQ-Routing converges to the effective routing policy faster than the Q-Routing, DRQ-Routing faster than CQ-ROUTING, and CDRQ-Routing is faster than all others. The differences between CDRQ-Routing, CQ-Routing and DRQ-Routing is not statistically significant, but there is a significant improvement over Q-Routing between 2600 to 2650 time steps. In fact, CDRQ-Routing adapts to change in traffic pattern 50 time steps faster than Q-Routing on an average. This result shows that adding confidence values and backward exploration speeds up the adaptation to changes in traffic patterns.

6.4 Adaptation to Changing Network Topology

The third set of experiments compared the routing algorithms' ability to adapt to the changes in network topology. A link was added between nodes 3 and 34 in the 6×6 grid

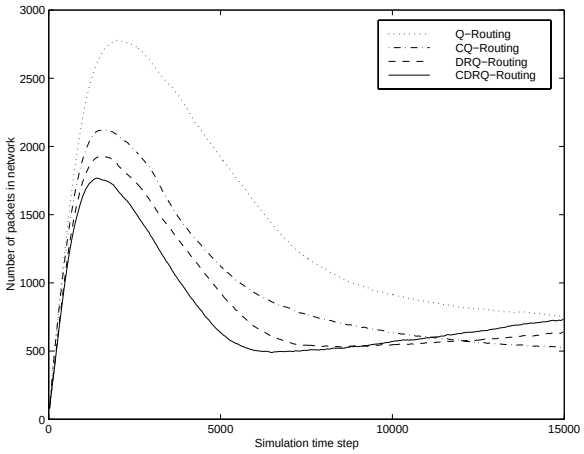

Figure 6.10: **Number of packets in the network for grid topology at high load**: Differences between all pairs are significant during the learning phase from 1000 to 5500 time steps after which differences between CQ-Routing, DRQ-Routing and CDRQ-Routing are insignificant, while Q-Routing is significantly different from CDRQ-Routing.

topology (figure 6.14) for these experiments. The routing algorithms were first allowed to learn an effective routing policy for the new network at a load level of 2.0 pkts/step until they converged (in 2000 time steps). At time step 2000, the link between node 12 and 25 was removed (figure 6.15). That is, the Q tables of node 12 and 25 were updated such that $Q_{12}(25, *)$ and $Q_{25}(12, *)$ were all set to *Infinite Cost* and the corresponding routing tables were also updated accordingly. The C values were not changed. Only single run is shown in figure 6.16 to depict the variations in adaptation process. However, statistical significance is computed over 50 runs and is given below.

Figure 6.16 shows the average packet delivery time between time steps 1800 and 3200 for the four algorithms. As soon as the link between nodes 12 and 25 went down at time 2000, the average packet delivery time of all these algorithms started increasing. They all tried to adapt to the change in network topology. DRQ-Routing learns a better routing policy slightly faster than CQ-Routing, while CDRQ-Routing learns even better and faster than both CQ-Routing and DRQ-Routing. The difference between the performance of CQ-Routing, DRQ-Routing and CDRQ-Routing is not statistically significant while the improvement from Q-Routing is significant with 99% confidence in the interval 2400 through

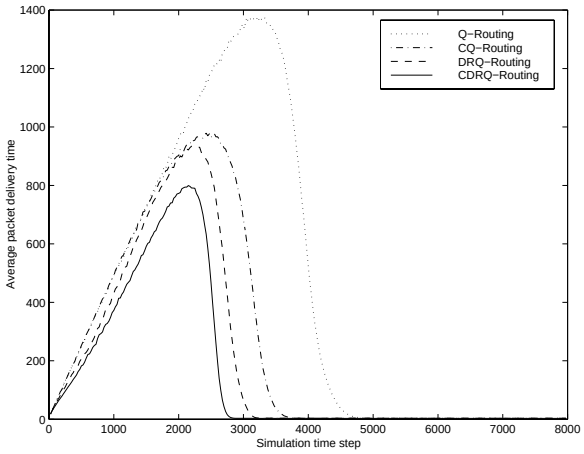

Figure 6.11: **Average packet delivery time for 7-D hypercube at high load**: All differences are significant between 2400 to 300 time steps. Q-Routing is significantly different from CDRQ-Routing till 4700 time steps.

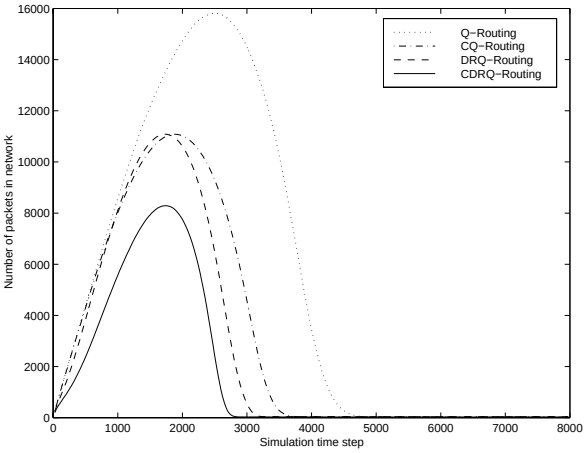

Figure 6.12: **Number of packets for 7-D hypercube at high load**: The trend is similar to that in figure 6.11 both for range of significant differences and quality of final policy learned.

3200 (and beyond) time steps.

Adaptation behavior of CQ-Routing and DRQ-Routing in figure 6.16 shows that the contribution of backward exploration is more significant than that of confidence values in adapting to the changes in network topology. The significant improvement in CDRQ-Routing over Q-Routing is also evident. Q-Routing fails to learn an effective routing policy

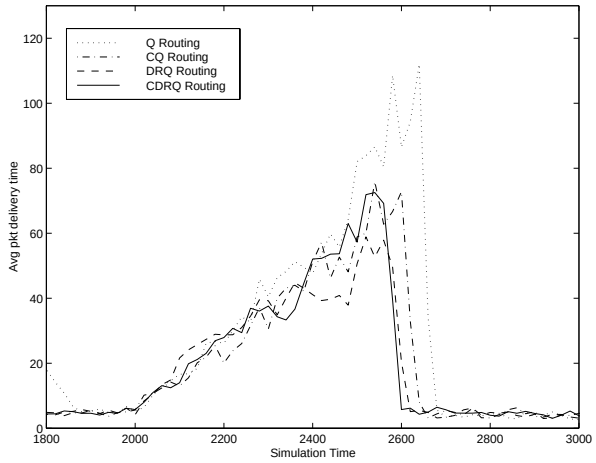

Figure 6.13: **Adaptation to a change the in traffic pattern**. Single run is shown. Note the variations in the adaptation process. CDRQ-Routing adapts around 50 time steps faster than Q-Routing. Difference between Q-Routing and CDRQ-Routing is statistically significant between 2600-2650 time steps. CQ-Routing and DRQ-Routing do not have a significant difference from CDRQ-Routing.

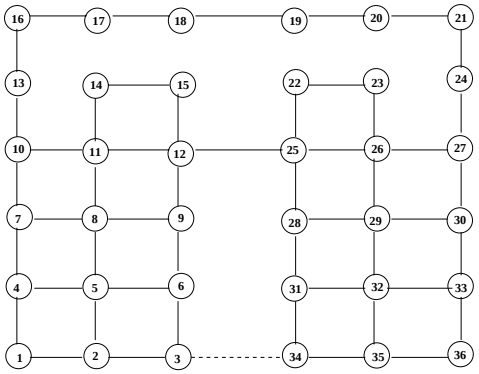

Figure 6.14: Grid topology before link 12 and 25 went down. This topology is different from the one in figure 3.1, it has an additional link between nodes 3 and 34

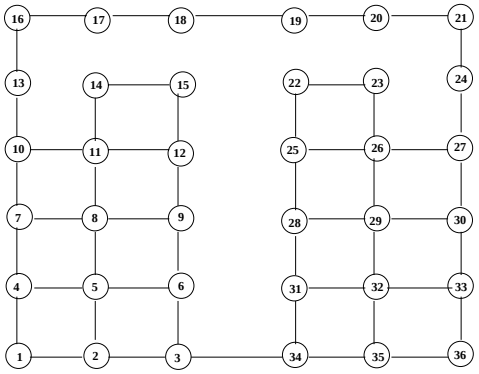

Figure 6.15: Grid topology after the link between nodes 12 and 25 has been removed, by fixing vectors $Q_{12}(25,*)$ and $Q_{25}(12,*)$ to $InfiniteCost$.

for the changed topology even in 1000 steps (till step 3200), while CDRQ-Routing has almost settled to an effective routing policy at that time.

Boyan and Littman (1994) observed that Q-Routing fails to revert to the original routing policy once the network topology is restored. This problem persists in CDRQ-Routing also. However, Bellman-Ford routing algorithm does not suffer from this problem and can adapt to changes in network topology very effectively. In section 7.4, one possible

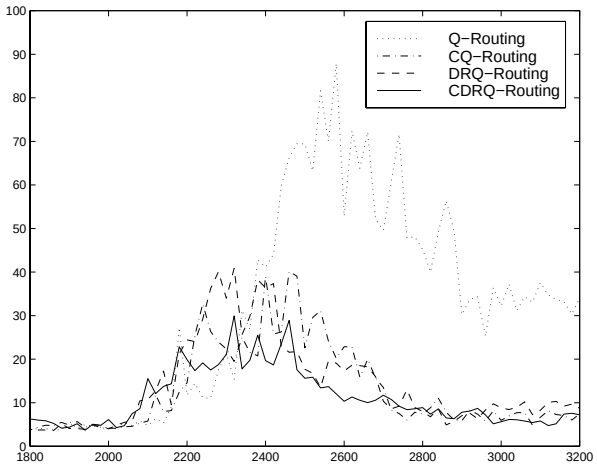

Figure 6.16: **Adaptation to change a in the network topology**. Q-Routing does not converge to an effective policy while CQ-Routing, DRQ-Routing and CDRQ-Routing all converge to an effective policy very fast.

way of solving this problem is presented.

6.5 Load level Sustenance

In addition to evaluating how fast the routing algorithms learn, it is important to evaluate how good the final policy is. For this reason, the average packet delivery time, after steady state was reached, was measured for different load levels. These results indicate how much load the final routing policy can sustain.

Figure 6.17 shows the relative performance of six routing algorithms on the 6×6 gird topology (figure 3.1). The results were averaged over 20 simulations. The first algorithm is the non-adaptive shortest path routing algorithm discussed in chapter 2. The second is the GLOBAL routing algorithm, where a central observer makes routing decisions using all the routing information available at all nodes in the network. This algorithm is a bound on how well any routing algorithm can perform. In addition to these two extremes, Q-Routing, CQ-Routing, DRQ-Routing andCDRQ-routing were also evaluated.

The non-adaptive shortest path routing is best at low load levels (0-1.5 pkts/step), but as the load increases to medium ranges (1.5-2.5 pkts/step), the nodes on popular routes

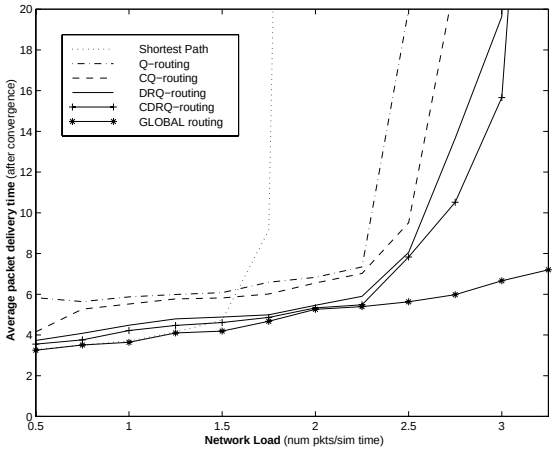

Figure 6.17: **Average Packet Delivery time after convergence**. The quality of the final routing policy is directly related to the amount of load level it can sustain. CDRQ-Routing is much superior than Q-Routing in its ability to sustain high load levels. The difference between the two is statistically significant over all load levels with 99% confidence.

start flooding and waiting time increases, degenerating the performance. Q-ROUTING, CQ-Routing, DRQ-Routing and CDRQ-Routing at low load levels perform very close to the GLOBAL. Among these, CDRQ-Routing and DRQ-Routing perform significantly better than Q-Routing and CQ-Routing. This is because the quantity of exploration is more important than quality at low load levels. At medium load levels, Q-Routing leads to flooding of packets in the network, and the average packet delivery time increases quickly. CQ-Routing, DRQ-Routing and CDRQ-Routing can sustain these load levels, with CDRQ-Routing outperforming the other two. At high load levels (2.5 and higher pkts/step), the CQ-Routing breaks down rather quickly at 2.5 pkts/step, while DRQ-Routing sustains up to 2.75 pkts/step and the combination of the two, the CDRQ-Routing, up to 3 pkts/step before breaking down.

The difference between shortest path, DRQ-Routing, CDRQ-Routing and GLOBAL routing is not statistically significant in low load range (1.25 pkts/sim). Thereafter, shortest path becomes significantly poor while DRQ-Routing and CDRQ-Routing continue to perform close to GLOBAL routing till the load of 2.25 pkt/sim. Difference between Q-Routing and CQ-Routing is significant only at load level 0.5 pkts/sim and after 2.25 pkts/sim. Be-

tween this interval, the difference is not significant. Difference between CDRQ-routing and Q-Routing is statistically significant over all load levels with a 99% confidence.

This result is a clear indication that both quality and quantity of exploration contribute to the final routing policy, quantity being more significant than quality. In the Q learning framework, CDRQ-Routing is the best adaptive routing algorithm currently known. The GLOBAL routing results show that there is still a lot of room between the theoretical upper bound and practical best that has been reached in this work.

6.6 Conclusion

CDRQ-Routing and its components were empirically evaluated in this chapter. They were compared in (1) their ability to learn an effective routing policy at constant load levels starting from a random policy, both in terms of the speed of adaptation and quality of the final policy learned, (2) their ability to adapt to changing routing traffic patterns once they have learned an effective routing policy, (3) their ability to adapt to changes in network topology once they have learned an effective routing policy and, (4) the quality of the routing policy learned in terms for various load levels. The following are some of the conclusions that can be drawn from these experiments.

- The additional direction of exploration in DRQ-Routing increases the speed of adaptation by a factor of almost 3 at low load conditions, and there is significant improvement at medium and high loads as well. This improvement is mainly due to higher amount of exploration due to extra update per packet hop. At low loads, there is a small number of packet hops, and the number of updates per packet hop makes a large difference in the quantity of exploration. Even at medium and high load levels, the higher exploration leads to faster adaptation.

- Incorporating the confidence values and thereby improving the quality of exploration also leads to significant improvements in the speed and quality of learning at various load levels. Although the improvement in not as large as that of DRQ-Routing.

- Combination of the two features into one algorithm, CDRQ-ROUTING, yields an adaptive routing algorithm that is superior to DRQ-Routing and CQ-Routing as can be seen clearly from figures 6.3 and 6.5 and 6.9. This shows that the role of improving the quality and quantity of exploration are complementary and contribute differently to the improvement of the final algorithm.

- Finally, an overall comparison of these algorithms together with the non-adaptive extremes leads to the following ordering:

  - GLOBAL (theoretical upper bound)
  - CDRQ-Routing (best)
  - DRQ-Routing
  - CQ-Routing
  - Q-routing
  - Bellman-Ford
  - Shortest Path Routing (worst)

CDRQ-Routing is an improvement over the conventional adaptive routing algorithms such as **BF** in a number of ways. First, CDRQ-Routing tries to optimize a more realistic criteria, the average packet delivery time, while the conventional adaptive routing algorithms (as discussed in chapter 2) try to learn the shortest path. Moreover, the amount of exploration overhead in CDRQ-Routing is significantly smaller than that in **BF** which exchanges complete cost tables between neighboring nodes. CDRQ-Routing strikes a balance between the amount of overhead incurred and speed of adaptation whether it is from random policy at fixed loads or to change in traffic pattern or change in network topology.

Chapter 7

Discussion and Future Directions

CDRQ-Routing with higher quantity of exploration and better quality of exploration than Q-Routing was developed in this work. It was evaluated and compared to Q-Routing and to the two main components of CDRQ-Routing, namely CQ-routing and DRQ-Routing, in three aspects: the ability to learn an effective routing policy starting from a random policy, the ability to adapt to changes in traffic patterns, and the ability to adapt to changes in network topology. In this chapter, possible future directions of this thesis and the generalization of the Q learning framework are discussed.

In this thesis, all the transmission links were assumed to incur the same amount of transmission delay δ per packet. Also, all the routers (nodes) were assumed to have equal processing speeds and hence the waiting time in any node's queue was proportional to its queue length, and the constant of proportionality was the same for all nodes. These assumptions are realistic in homogeneous networks like LANs, but in a heterogeneous network as complex as the Internet, they are not valid. Different transmission delays and router speeds introduce enormous complexity to the routing problem. Formal description of such heterogeneous networks and the generalization of the forward and backward exploration Q value update rules (equation 3.4 and 5.6) to such networks is given in section 7.2. Q value update rules for CDRQ-Routing are also derived in that section.

In this thesis, all packet buffers were assumed to be infinite. This assumption is

realistic for normal traffic conditions because the routers in the real communication networks have a lot of packet buffer space. When load levels are very high, or the routing policy is very poor, or a large number of nodes and/or links are down, it is possible that queues become flooded with more packets than they can buffer. In such cases, additional incoming packets are dropped leading to congestion. An adaptive routing algorithm should be able to "sense" such congestion and adapt effective to avoid information loss. In section 7.3, congestion control in adaptive routing algorithms is addressed. A measure called *congestion risk* that characterizes the risk of congestion in taking a particular routing decision is formally defined. The Q value update rules for heterogeneous networks given in sections 7.2 are extended to incorporate congestion control by making use of congestion risk. Using these update rules, CDRQ-Routing can be used for adaptive routing and congestion control at the same time.

Uncertainty in how well a Q value represents the current state of the network is characterized by its confidence value. CQ-ROUTING is one way of using these confidence values. Another way is by defining a *probability distribution function* for each Q value. An adaptive routing algorithm that makes use of these distributions, referred to as PROBABILISTIC CONFIDENCE BASED Q-ROUTING (PRCQ ROUTING), is discussed in detail in section 7.4. An algorithm similar to CDRQ-Routing where Q values are generated according to a probability distribution is outlined.

Choi and Yeung (1996) introduced a memory-based PREDICTIVE Q ROUTING (PQ-ROUTING) algorithm where first order information in terms of the *rate of change* of Q values is also maintained for each Q value. These two pieces of information, the Q value at some time before the current time and its rate of change, help predict the actual Q value at the current time. Only forward direction of exploration was used in PQ-ROUTING. An obvious extension to PQ-Routing would be to use the backward exploration as well. The addition of backward exploration to PQ-ROUTING yields the DUAL REINFORCEMENT PREDICTIVE Q ROUTING (DRPQ-ROUTING) discussed in detail in section 7.5. Further extension of DRPQ-ROUTING by adding confidence values for each Q value will yield a predictive version of CDRQ-Routing.

7.1 Generalizations of the Q Learning Framework

The network routing problem is one of many domains where a complex system comprising of distributed components tries to optimize a global performance criteria. Examples of such complex systems range from ant colonies to artificial life scenarios with multiple agents, from road traffic to network routing and congestion control to scheduling and synchronization in distributed systems. The behavior of the overall system depends in a rather complex manner on the behaviors of the individual components, and it is not easy to model the overall system behavior. Behavior of the individual components depend only on local information available to them. The overall objective function is divided into local greedy objective functions, which the components try to learn to optimize. The local information is shared with other components through exploration, which incurs a non-zero exploration cost proportional to the amount of information shared and the frequency with which it is exchanged.

The adaptive network routing problem addressed in this thesis has all these characteristics. Q learning framework was used to develop routing algorithms that learn to optimize a global criterion by making use of only local information. Some of the possible generalizations of the Q learning framework that would allow applying it to other domains are:

- **Discrete vs. Continuous**: In the network routing problem, the decision of choosing one out of a finite set of neighbors constitutes search in a discrete space. Using tables for such discrete systems to store information is a reasonable approach. But in general, the space of possible options might be continuous. For example in robot control, the control vector could be a vector of real numbers. In such cases, instead of maintaining tables and updating them, parametric functions would be more useful. These functions can be learned and adapted by changing the parameters. A neural network is a general parametric function that can be used in continuous domains. The parameters are the weights in the links. Error back propagation can be used to update these parameters, equivalent to Q value updates in CDRQ-Routing. Boyan

and Littman observed that using neural networks instead of Q tables for network routing lead to poor performance (Boyan and Littman 1994). Thus if the search space is discrete, the tables approach is useful and neural network should be used only for continuous spaces.

- **Heterogeneous systems**: The different components of a system might have different capacities. For example in communication network systems, the speeds of links between nodes or the processing speeds of routers could be different. These differences lead to more complex behaviors. Detailed discussion for such generalizations for network routing problem is given in section 7.2.

- **Multiple objective functions**: In general, there could be more than one objective function that the system is trying to optimize. In the network routing problem, for example, there could be costs associated with using different links. In this case there are then the two competing objective functions: (1) average packet delivery time, and (2) average transmission cost over links. A weighted combination of these objective functions can be defined to give the final overall objective function. Network routing generalized for costs of using links is discussed in section 7.2.

- **Predictive capabilities**: The CDRQ-Routing uses Q values to make routing decisions. These Q values are not always accurate. They reflected the state of the network when they were updated, but at the current time, the state of the network might be different. If the network had some prediction capabilities it would be possible to predict the current Q values from the Q values in the table. Choi and Yeung (1996)'s PQ-Routing discussed in section 7.5.1. is an example of this generalization. Ideas from PQ-Routing are used to add predictive capabilities in CDRQ-Routing in section 7.5.2. The rate of change of Q values (i.e. the recovery rate) is an example of *first order characteristic* of the system. In general, higher order characteristics can be maintained if the system is of higher order. For example a second order parameter extending PQ-Routing could be the rate of change of the recovery rate.

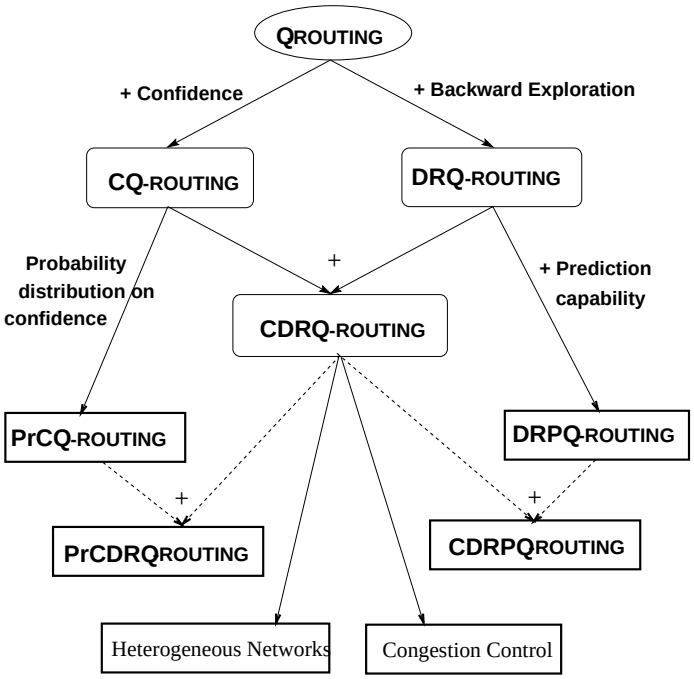

Figure 7.1: **Summary of Future Directions**: Blocks with bold face boundaries constitute future research. CDRQ-Routing can be generalized for heterogeneous networks (section 7.2) and congestion control (section 7.3). Probabilistic Q routing (PrCQrouting) (section 7.4) is an extension of CQ-Routing. Dual Reinforcement Predictive Q Routing (DRPQ-Routing) (section 7.5) is an extension of DQR-routing. Both PrCQrouting and DRPQ-Routing can be extended to CDRQ-Routing (shown by the dotted lines) into Probabilistic CDRQ-Routing (PrCDRQ-Routing) or Confidence-dual Reinforcement Predictive Q-Routing (CDRPQ-Routing).

When applied to adaptive network routing, these generalizations lead to several possible directions of future work as will be discussed briefly in subsequent sections (figure 7.1).

7.2 Adaptive Routing in Heterogeneous Networks

In this thesis so far all the nodes have been assumed to have equal speed of processing packets and all links equal transmission delays (section 1.2). This is a valid assumption for most LANs where routers of almost equal capacities and similar links between routers are used, but in large scale networks like the Internet there is a lot of heterogeneity. A network can be heterogeneous in a number of ways. Three of the most important ways are defined

below:

1. **Speed of routers (nodes)**: Different nodes process packets with different speeds. If β_x is the time taken by node x to process a packet, the definition of the total *waiting time in the intermediate queues* for a packet traversing the route $(x_0, x_1, ..., x_l)$, given in equation 1.1, generalizes to:

$$T'_W = \sum_{i=1}^{l-1} \beta_{x_i} q_{x_i}. \tag{7.1}$$

   More specifically, sending a packet through node x_i will add a queue waiting time of $\beta_{x_i} q_{x_i}$ to its total delivery time. Thus an effective routing policy should be biased towards sending packets via faster nodes.

2. **Speed of links**: The links between nodes are not always equal. For example, fiber optic links are faster than Ethernet cables which in turn are faster than telephone line connections. More specifically, let δ_{xy} be the time taken to transmit a packet from a neighboring node x to y. Then the total *transmission delay over the links* for a packet traversing the route $(x_0, x_1, ..., x_l)$, given in equation 1.2, generalizes to:

$$T'_X = \sum_{i=0}^{l-1} \delta_{x_i x_{i+1}}. \tag{7.2}$$

   Thus sending a packet through node x_i to x_{i+1} will add a transmission delay of $\delta_{x_i x_{i+1}}$ to the total delivery time of the packet from source to destination node, and hence an effective routing policy should be biased towards sending packets over that part of the network that has faster links.

3. **Cost of using links**: The cost of using a link was not considered in this thesis at all, but in general, the costs of alternative links might differ. For example, satellite links are more costly than fiber optic connections, which in turn are more costly than phone lines. The cost and speed tradeoff of these links redefines the overall objective function that the routing policy needs to optimize: *What is the best routing policy which sends the packets in the least amount of time and incurring the least amount of*

cost? More specifically, let γ_{xy} be the cost per packet of the link from x to y, and let w_d be the weight of **total packet delivery time** ($T'_W + T'_X$), and w_c be the weight of the **total transmission cost** of sending a packet over a route $\mathbf{x} = (x_0, x_1, ..., x_l)$. Then the combined delay/cost objective function may be defined as:

$$J(\mathbf{x}) = w_d \left(\sum_{i=1}^{l-1} \beta_{x_i} q_{x_i} + \sum_{i=0}^{l-1} \delta_{x_i x_{i+1}} \right) + w_c \sum_{i=0}^{l-1} \gamma_{x_i x_{i+1}} \tag{7.3}$$

As a result of incorporating these costs in the objective function, both the interpretation of the Q values and the update rules for the CDRQ-ROUTING algorithm (equations 4.22 to 4.25) will change. The new interpretation of the $Q_x(y,d)$ is now the *total value of sending the packet from node x to destination d via node x's neighbor y, considering both the total packet delivery time and the total transmission cost.* The estimated Q values in the update rules for forward exploration (equations 3.4) and backward exploration (equation 4.6) now become:

- **Q value estimate of sender node**(x) (FORWARD EXPLORATION):

$$\mathbf{Q}_x(y,d)^{est} = \mathbf{Q}_y(\hat{z},d) + w_d(\beta_y q_y + \delta_{y\hat{z}}) + w_c \gamma_{y\hat{z}}, \tag{7.4}$$

- **Q value estimate of receiver node**(y) (BACKWARD EXPLORATION):

$$\mathbf{Q}_y(x,s)^{est} = \mathbf{Q}_x(\hat{z},s) + w_d(\beta_x q_x + \delta_{x\hat{z}}) + w_c \gamma_{x\hat{z}}. \tag{7.5}$$

Different settings of the weight parameters w_c and w_d will yield different objective functions and hence different routing policies. By setting $w_c = 0$, $\beta_x = 1 \; \forall \; x \in \mathbf{V}$, $w_d = 1$ and all $\delta_{xy} = \delta$ for all neighbors x and y in equations 7.4 and 7.5, one gets the equations 3.4 and 4.6 for Q value estimates for forward and backward exploration. Similarly, by choosing different values of parameters w_c and w_d and for a given network (i.e., given the parameters β_x, δ_{xy} and γ_{xy}), different types of optimization functions (equation 7.3) and correspondingly different update rules can be defined.

7.3 Finite Packet Buffers and Congestion Control

In this thesis so far the packet buffers have been assumed to be infinite so that each node can accommodate all packets coming into it. Because of the increasing size of packet buffers in the existing routers and the decreasing cost of memory, this is a realistic assumption for most network systems where the load is significantly lower than the capacity of the routers. However, there are cases where buffer size is a limitation. In these cases, packet buffers are finite and when they are full, additional packets have no space to go to and they are dropped, leading to congestion. The task of a routing algorithm is to route packets not only in a manner that they reach the destination as soon as possible, but also to make sure that they do not cause congestion at any nodes.

Consider a node x with a finite buffer size B_x. For generality assume that different nodes have different buffer sizes. If the queue length of this node is $q_x \leq B_x$, then one can associate a *congestion risk* with this node of the form $g(\frac{q_x}{B_x})$, which essentially is the risk or cost of dropping the packet if it is sent through this node. This cost is a monotonically increasing function of the ratio $\frac{q_x}{B_x}$. Some of the suggested base case values for this function are: $g(0) = 0$ and $g(1) = \infty$. An example of such a function is:

$$g_\theta(\alpha) = \frac{\alpha}{(1-\alpha)^\theta}, \tag{7.6}$$

where θ is a parameter that controls how fast the function asymptotes to ∞ as α tends to 1. If w_g is the weight of the congestion cost and this cost is to be incorporated into interpretation and update rules for Q values, then the overall update rules become:

- **Q value estimate of sender node**(x) (FORWARD EXPLORATION):

$$\mathbf{Q}_x(y,d)^{est} = \mathbf{Q}_y(\hat{z},d) + w_d(\beta_y q_y + \delta_{y\hat{z}}) + w_c\gamma_{y\hat{z}} + w_g g_\theta(\frac{q_y}{B_y}), \tag{7.7}$$

- **Q value estimate of receiver node**(y) (BACKWARD EXPLORATION):

$$\mathbf{Q}_y(x,s)^{est} = \mathbf{Q}_x(\hat{z},s) + w_d(\beta_x q_x + \delta_{x\hat{z}}) + w_c\gamma_{x\hat{z}} + w_g g_\theta(\frac{q_x}{B_x}), \tag{7.8}$$

Proper adjustments of the weight parameters w_c, w_d, and w_g, together with the proper choice of the θ parameter for the congestion cost function g, will yield different objective functions. The basic intuition is that if the packet buffer of a node x is about to get filled, its $\frac{q_x}{B_x}$ ratio will be close to 1 and the g function (congestion risk) will be high using backward exploration. A packet P going from node x to node y takes this information along with it and a packet P' coming into x from a neighbor y' also propagates this information to node y' using forward exploration. Hence slowly other nodes in the network get the information that node x is heavily congested and stop sending packets via node x. As a result, the queue length of node x will become small again. Backward exploration due to the packets going out of node x will propagate the information about the decreased congestion risk at node x, allowing other nodes to resume sending packets to x.

CDRQ-Routing can be extended to incorporate the generalized Q estimates given in equations 7.7 and 7.8, so that it can learn an effective routing policy that gives the best performance depending on the weights w_c and w_c in a heterogeneous network, and at the same time performs congestion control depending on the weight w_g.

7.4 Probabilistic Confidence-based Q-Routing

In CQ-Routing, confidence values are used to quantify the uncertainty in corresponding Q values. These confidence values only help deciding how much a Q value should be changed. They do not affect routing decisions directly. That is, the uncertainty is used only in exploration of Q values and not in exploitation.

In this section an extension of CQ-Routing called PROBABILISTIC CONFIDENCE BASED Q ROUTING (PRCQ-ROUTING) is introduced. This extension can also be incorporated into CDRQ-Routing. In PRCQ-ROUTING, the confidence values are used not only for exploration as in CQ-Routing, but also in making routing decisions. Instead of treating Q values as parameters that represent state of the network they are treated as *random variables with Gaussian probability distribution.* Such a probabilistic interpretation of Q values is another way of introducing uncertainties. In CDRQ-Routing, in making a rout-

ing decision for a packet destined to node d, node x picks the neighbor y for which the Q value $Q_x(y,d)$ in the vector $Q_x(*,d)$ is minimum (section 3.3). In PRCQ-ROUTING, instead of comparing the Q values directly, a vector of Q values $\mathbf{Q}(d)$ is generated using the probability distribution associated with the vectors $Q_x(*,d)$ and $C_x(*,d)$. The decision of which neighbor to choose is based on the vector $\mathbf{Q}(d)$ of random Q values. This way the confidence values are used in making actual routing decisions. Section 7.4.1 describes the probability function, and 7.4.2 the complete algorithm of picking the neighbor, given the Q value distributions.

7.4.1 Gaussian Distribution of Q values

In order to incorporate uncertainty, Q values are treated as random variables with a Gaussian probability distribution. The Q value in the Q table is the *mean* of the Gaussian, denoted by $\hat{Q}$, and the corresponding confidence value is used to compute the variance. Let $\sigma(C)^2$ be the variance function for the confidence C. The base case values are $\sigma(1)^2 = 0$ and $\sigma(0)^2 = \infty$. A variance of 0 represents a complete certainty in the Q value (corresponding to a confidence value of 1) while a variance of ∞ represents a complete uncertainty in the Q value (corresponding to a confidence value of 0). An example of such a variance function is:

$$\sigma_a(C) = \frac{1}{C^a} - 1, \tag{7.9}$$

where parameter a controls the shape of the function (see figure 7.2).

The amount of uncertainty in the Q value is directly related to the variance of its Gaussian distribution. If the system is very dynamic or prone to sudden changes, even a slight decrease in confidence should incur high uncertainty, in other words, the parameter a should be high.

Using the above interpretations for the Q and C value, the Gaussian distribution of Q is given by:

$$Prob(Q) = \frac{1}{\sqrt{2\pi}\sigma(C)} e^{-\frac{1}{2}\left(\frac{\hat{Q}-Q}{\sigma(C)}\right)^2}. \tag{7.10}$$

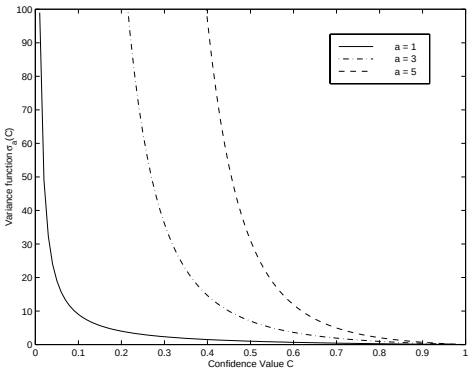

Figure 7.2: **Variance function** $\sigma_a(C)$ for PROBABILISTIC CONFIDENCE BASED Q ROUTING. The variance, $\sigma_a(C)$ decreases as confidence (C) increases (equation 7.9). For high values of parameter a, the increase is very sharp.

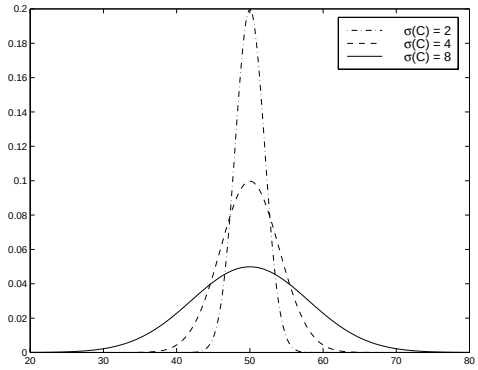

Figure 7.3: **Gaussian distribution for Q values** with mean Q value = 50 and three different variances. The distribution is centered around the mean value 50 and the spread of the distribution depends on how much is the variance $\sigma(C)^2$.

Figure 7.3 shows the Gaussian distribution for three different values of $\sigma(C)$. A Gaussian is centered around the mean. For low variance (high confidence) the distribution peaks around the mean sharply and for high variance (low confidence) the distribution is spread out. The basic idea for using Gaussian distribution is as follows. When the confidence in certain Q value is high, the probability that the Q value in the table is the true estimate is also high and thus we should pick a random Q value close to that in the table. In other words, the probability of choosing the Q values close to the mean should be high and decrease sharply for Q values away from the mean. Since for high confidence the distribution peaks sharply around the mean, choice of a Q value according to this distribution will be close to the mean. For a low confidence, chances are that the actual Q value is away from the one in the Q table. That is, the probability of actual Q value being away from the mean is not very low but decreases slowly as one moves away from the mean. The Gaussian distribution shown in figure 7.3 has the property that the probability decreases slowly as one moves away from the mean.

Thus, the confidence is used to in quantifying the uncertainty in Q values. The more uncertainty there is, the higher is the probability that the actual Q value is away from the

one in the Q table.

7.4.2 Using probabilistic Q-values

At the time of making a routing decision for a packet destined to node d, node x computes the probable Q values for each of the neighboring nodes. In particular, instead of using the Q values in the Q table directly to compute the best neighbor, a vector $\mathbf{Q}(d)$ of random Q values is computed using the above probability distributions. Let $\hat{Q}_x(y,d)$ and $C_x(y,d)$ be the entries in the tables then the random Q value for neighbor y in the vector $\mathbf{Q}(d)$ that is $Q_x(y,d)$ is drawn from the distribution:

$$Prob(Q_x(y,d)) = \frac{1}{\sqrt{2\pi}\sigma(C_x(y,d))} e^{-\frac{1}{2}\left(\frac{\hat{Q}_x(y,d)-Q_x(y,d)}{\sigma(C_x(y,d))}\right)^2} \tag{7.11}$$

These random Q values in the vector $\mathbf{Q}(d)$ are used to find the best neighbor $\hat{y}$ for which $Q_x(y,d)$ is minimum.

The main advantage of using random Q values in increased adaptation is as follows. Choi and Yeung (1996) pointed out that "Q-Routing suffers from the so-called hysteresis problem, in that it fails to adapt to the optimal (shortest) path when the network load is lowered. Once a longer path is selected due to increase in network load, a minimum selector is no longer able to notice the subsequent decrease in traffic along the shortest path." Moreover, Boyan and Littman (1994) pointed out that "Q-Routing fails to adapt to the original routing policy once the network topology is restored". All these observations indicate that using random decisions at times would be useful in reverting back to the original routing policy when the network state (load level, topology or traffic pattern) is restored. PrCQ-Routing is doing exactly that. When a Q value is not updated for a long time, confidence in that value goes down and probability of choosing a Q value away from the one in the table increases. Hence with a non-zero probability, those routes will be explored by PrCQ-Routing that would not have been explored by Q-Routing and CDRQ-Routing due to the minimum selector rule (Section 3.3).

The issue of exploration versus exploitation is important. If a lot of random packets are sent then the performance will go down but adaptation will be good and vice versa.

Hence a proper choice of parameter a is important in PrCQ-Routing. If a high value for a is chosen, then even for a small decrease in confidence, the probability distribution of Q value will become smooth and spread out and will increase chances of making random decisions. Therefore, PrCQ-Routing could be useful for reverting back to the original policy when network state is restored.

7.4.3 Updating C and Q values

C values can be updated as specified in equations 5.23 and 5.25. Updating Q values requires the best Q estimate for the remaining path in the forward exploration (equation 5.24) or the covered path in the backward exploration (equation 5.26). These Q estimates are also computed in a probabilistic manner. Thus the Q values in equations 5.24 and 5.26 are replaced by the probable Q values obtained by Gaussian distribution similar to that in equation 7.3.

In summary, PrCQ-Routing extends CQ-Routing so that the confidence values are not only used in exploration but also play crucial role in the exploitation or actual decision making. Due to its probabilistic nature, it can be useful in overcoming the problem of Q-Routing and CDRQ-Routing not being able to revert to original policies once the network state is restored.

7.5 Dual Reinforcement Predictive Q-Routing

As pointed out before, Q values are not accurate and decisions based on inaccurate Q values may not be optimal. CQ-Routing and its extension, PrCQ routing try to quantify these uncertainties in terms of confidence values and probability distributions. Another way of dealing with uncertainties is to introduce prediction capabilities that make use of the *rate of change* of Q values to *predict* the correct Q value at the current time step. Choi and Yeung (1996) proposed a memory-based reinforcement learning approach to adaptive routing called Predictive Q-routing (PQ-Routing). PQ-Routing is discussed in detail in section 7.5.1. It makes use of only forward exploration and hence can be extended by

adding the backward exploration in it. This extension leads to the DUAL REINFORCEMENT PREDICTIVE Q-ROUTING discussed in section 7.5.2

7.5.1 Predictive Q-Routing

PQ-Routing is an extension of Q-Routing that maintains and uses a recovery rate of Q values and the best estimated delivery time to make its routing decisions. The recovery rate is used to predict the Q value at the current time step. The routing information at each node (node x) in PQ-ROUTING comprises of four routing tables:

- $Q_x(y, d)$ - the estimated delivery time from node x to node d via a neighboring node y (same as in Q-Routing);
- $B_x(y, d)$ - the best estimated delivery time from node x to node d via a neighboring node y so far, that is the minimum value of $Q_x(y, d)$ since the routing started;
- $R_x(y, d)$ - the recovery rate for a path from node x to node d via a neighboring node y; and
- $U_x(y, d)$ - the last update time for a path from node x to node d via a neighboring node y.

After a packet arrives from node y to node x, the following table updates take place in node x:

1. Compute $\Delta Q = (\delta + q_y + \min_z\ Q_y(z, d)) - Q_x(y, d)$.
2. Update Q: $Q_x(y, d) \leftarrow Q_x(y, d) + \eta_Q^{(f)}\ \Delta\ Q$.
3. Update B: $B_x(y, d) = \min(B_x(y, d),\ Q_x(y, d))$.
4. Update R:
   if $(\Delta Q < 0)$ then
   - $\Delta R \leftarrow \Delta Q$ / (current time - $U_x(y, d)$)

- $R_x(y,d) \leftarrow R_x(y,d) + \eta_R^{(f)}\ \Delta R$.

else if $(\Delta Q > 0)$ then $R_x(y,d) \leftarrow \lambda_R^{(f)}\ R_x(y,d)$.

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

The three learning parameters in PQ-Routing are (1) $\eta_Q^{(f)}$, the learning rate for Q value update in *forward exploration*, (2) $\eta_R^{(f)}$, the learning rate for R value update in forward exploration, and (3) $\lambda_R^{(f)}$, the decay constant in R value update.

When a packet destined to node d comes to the node x, the best neighbor y is decided as follows:

$$\hat{y} = arg \min_{y \in N(x)} \{\max\{Q_x(y,d)^{pred}, B_x(y,d)\}\}, \tag{7.12}$$

where,

$$Q_x(y,d)^{pred} = Q_x(y,d) + (CurrentTime - U_x(y,d)).R_x(y,d). \tag{7.13}$$

That is, first $Q_x(y,d)^{pred}$ is computed based on the Q value in the Q table $Q_x(y,d)$, the recovery rate $R_x(y,d)$ and the time elapsed since the Q value was last updated $(CurrentTime - U_x(y,d))$ using 7.13. If $Q_x(y,d)^{pred}$ is less than the best estimate $B_x(y,d)$ so far, then $B_x(y,d)$ itself is used. Otherwise, $Q_x(y,d)^{pred}$ is used in making the routing decision.

PQ-Routing, like Q-Routing, makes use of only one direction of exploration and does not use any measure of confidence to decide the learning rates. Confidence measure can also be incorporated into PQ-Routing but since the uncertainty in Q values is already taken care of by the recovery rate and prediction mechanism, confidence values are not expected to improve the performance significantly. Next subsection extends PQ-Routing to incorporate backward exploration into PQ-Routing to yield DUAL REINFORCEMENT PREDICTIVE Q-ROUTING. Since both prediction mechanism in PQ-Routing (Choi and Yeung 1996) and backward exploration in DRQ-Routing (Kumar and Miikkulainen 1997) have been shown to improve Q-Routing separately, bringing them together into one algorithm is expected to further improve Q-Routing.

7.5.2 Backward Exploration in PQ-Routing

PQ-Routing can be extended to include backward exploration. All the updates in PQ-Routing that are done in the tables at node x when it sends a packet for destination d to one of its neighbor y can be done in the opposite direction also that is, in node y when it receives this packet from its neighbor x. The following are the update rules for such backward exploration. As before, s denotes the source node for this packet.

1. Compute $\Delta Q = (\delta + q_x + \min_z Q_x(z,s))$ - $Q_x(y,s)$.

2. Update Q: $Q_y(x,s) \leftarrow Q_y(x,s) + \eta_Q^{(b)} \Delta Q$.

3. Update B: $B_y(x,s) = \min(B_y(x,s), Q_y(x,s))$.

4. Update R:
   if $(\Delta Q < 0)$ then

   - $\Delta R \leftarrow \Delta Q$ / (current time - $U_y(x,s)$)
   - $R_y(x,s) \leftarrow R_y(x,s) + \eta_R^{(b)} \Delta R$.

   else if $(\Delta Q > 0)$ then $R_y(x,s) \leftarrow \lambda_R^{(b)} R_y(x,s)$.

5. $U_y(x,s) \leftarrow$ current time.

The dual parameters are (1) $\eta_Q^{(b)}$, the learning rate for Q value update in *backward exploration*, (2) $\eta_R^{(b)}$, the learning rate for R value update in backward exploration, and (3) $\lambda_R^{(b)}$, the decay constant in R value update. This way two sets of updates per packet hop will occur and it is expected that faster adaptability than PQ-Routing will be obtained. As seen from experiments in chapter 5, adding backward exploration to Q routing increases the quantity of exploration and thus increases the speed and quality of adaptation. Similar increase in the speed and quality of PQ-Routing is expected in DRPQ-ROUTING due to backward exploration.

7.6 Conclusion

Figure 7.1 summarizes the five main future directions proposed in this chapter: (1) Generalization of CDRQ-Routing to heterogeneous networks, where processing speed of nodes and transmission delay and transmission cost of links are incorporated in the interpretation and update rules of Q values. (2) Generalization to finite buffer sizes, where congestion risk measure for each node is used for adaptive congestion control. (3) An alternative means of incorporating uncertainties in CQ-Routing by treating Q values as random variables with Gaussian distributions is proposed. (4) An extension of DRQ-Routing by adding prediction mechanism or recovery rate from PQ-Routing to it is proposed. While (1) and (2) should allow the CDRQ-Routing to be applied to more realistic and a wider variety of networks, further experimentation is required to quantify the effects of (3) and (4) and whether they lead to algorithms with superior performance.

Chapter 8

Conclusion

In this thesis, a new adaptive network routing algorithm called CONFIDENCE BASED DUAL REINFORCEMENT Q-ROUTING for on-line adaptation of routing policies in dynamic communication networks was developed and evaluated. It is first shown that Q-Routing, upon which CDRQ-Routing is based, is superior to the state-of-the-art distance vector Bellman-Ford routing in terms of speed, quality and overhead requirements. Then CDRQ-Routing is shown to be superior to Q-Routing in two distinct ways:

1. *The quality of exploration* in CDRQ-Routing is better than in Q-Routing. This result due to confidence values associated with each of the Q values in the network. The confidence values characterize how reliable the corresponding Q values are. The C values are themselves updated as routing takes place. They are used in determining the learning rate so that it is higher if the new estimate of Q value is more reliable (high confidence) or old Q value is less reliable (low confidence). When Q-Routing is extended with confidence values, its ability to learn effective routing policies from random policies and to adapt to changing traffic patterns and topology improves significantly.

2. *The quantity of exploration* in CDRQ-Routing is twice as large as in Q-Routing. This is due to backward exploration, which is additional to the regular forward exploration

in Q-Routing. As a result, two Q values are updated per packet hop in CDRQ-Routing as against just one in Q-ROUTING. When backward exploration is added to Q-Routing, its ability to learn effective routing policies from random policies and to adapt to changing traffic patterns and topology improves significantly.

The contributions of these extensions were demonstrated experimentally over different network topologies. For the task of learning an effective policy starting from a random policy, CDRQ-Routing learns the shortest path routing at low load levels almost three times as fast as Q-Routing. At medium load levels, CDRQ-Routing learns an effective routing policy almost twice as fast as Q-Routing. At high load levels, CDRQ-routing converges to a much superior routing policy in a significantly shorter time than Q-Routing.

In adapting to changing network topology, where a link goes down, and to changing traffic patterns after initial learning has settled down, CDRQ-Routing outperforms Q-routing in terms of speed of the adaptation. Finally, CDRQ-Routing is found to sustain much higher network loads than Q-Routing.

In addition to the empirical results demonstrating the superior performance of CDRQ-Routing, illuminating theoretical properties of forward and backward exploration were identified and proved in this work. Rules for choosing the learning rate function based on confidence values and for confidence value update were also proposed.

Careful analysis of exploration overhead leads to the conclusion that the cost of exploration in both forward and backward exploration is less than 0.2% of the overall traffic, which is insignificant compared to the increase in performance.

With superior exploration mechanisms and low overhead, CDRQ-Routing is both effective and practical adaptive routing algorithm. It can adapt effectively to changes in traffic patterns, topology and load levels. The network routing algorithms used currently in the Internet are either very simplistic in nature, or are variants of the distance vector routing algorithms tested in chapter 4. All the currently used algorithms essentially use shortest path routing, which is a suboptimal routing policy at the load levels in the Internet. CDRQ-Routing provides a superior alternative to these algorithms.

Bibliography

A. G. Barto, R. S. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846.

Barto, A. G. (1992). Reinforcement learning and adaptive critic methods. In White, D. A., and Sofge, D. A., editors, *Handbook of Intelligent Control*, 469–491. New York: Van Nostrand-Reinhold.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bellman, R. E. (1958). On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90.

Boyan, J. A., and Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. In Cowan, J., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*. Cambridge, MA: MIT Press.

C. Cheng, R. Ripley, S. K., and Garcia-Luna-Aceves, J. (1989). A loop-free extended bellman-ford routing protocol without boincing effect. In *ACM Sigcomm '89 Symposium*.

Choi, S. P. M., and Yeung, D.-Y. (1996). Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*, 945–951. Cambridge, MA: MIT Press.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numer. Math.*, 1:269 – 271.

Floyd, R. W. (1962). Algorithm 97 (shortest path). In *Communications of the ACM*, vol. 5(6).

Ford, L. R., and Fulkerson, D. R. (1962). *Flows in Networks.* Princeton, NJ: Princeton University Press.

Goetz, P., Kumar, S., and Miikkulainen, R. (1996). On-line adaptation of a signal predistorter through dual reinforcement learning. In *Machine Learning: Proceedings of the 13th Annual Conference (Bari, Italy).*

Gouda, M. (1998). *Elements of Network Protocols.* New York: John Wiley & Sons.

Huitema, C. (1995). *Routing in The Internet.* Prentice Hall PTR.

Kumar, S., and Miikkulainen, R. (1997). Dual reinforcement Q-routing: An on-line adaptive routing algorithm. In *Proceedings of the Artificial Neural Networks in Engineering Conference (St. Loius, USA).*

Littman, M., and Boyan, J. (1993a). A distributed reinforcement learning scheme for network routing. Technical Report CMU-CS-93-165, Computer Science Department, Carnegie Mellon Universit y, Pittsburgh, PA.

Littman, M., and Boyan, J. A. (1993b). A distributed reinforcement learning scheme for network routing. In *Proceedings of the First Internation Workshop on Applications of Neural Networks to Telecommunications*, 45–51. Hillside, New Jersy: Erlbaum.

Rajagopalan, B., and Faiman, M. (1989). A new responsive distributed shortest-path routing algorithm. In *ACM Sigcomm '89 Symposium.*

Ravindra K. Ahuja, Kurt Mehlhorn, J. B. O., and Tarjan, R. E. (1988). Fast algorithms for shortest path problem. Technical Report 193, MIT-Operations research center.

Sutton, R. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44.

Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*. Cambridge, MA: MIT - Press.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massechusetts, Amherst, MA.

Tanenbaum, A. (1989). *Computer Networks*. Prentice Hall. Second edition.

Thomas H. Cormen, C. E. L., and Rivest, R. L. (1990). *Introduction to Algorithms*. Cambridge, MA: MIT Press.

Thrun, S. B. (1992). The role of exploration in learning control. In White, D. A., and Sofge, D. A., editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. New York: Van Nostrand Reinhold.

W. H. Press, S. A. Teukolsky, W. T. V. . B. P. F. (1995). *Numerical Recipies in C*. Cambridge, UK: Cambridge University Press.

Watkins, C. J. C. H., and Dayan, P. (1989). Q-learning. *Machine Learning*, 8:279–292.

Vita

Shailesh Kumar was born in Meerut, India on January 27, 1974. He did his schooling from Delhi Public School Hardwar and Jagdishpur. He obtained his Bachelor of Technology degree in Computer Science and Engineering from the Institute of Technology, Banaras Hindu University, Varanasi, India, in May 1995. He joined the University of Texas at Austin in the Fall of 1995. In Fall 96, he joined the Laboratory of Artificial Neural Systems in the Department of Electrical and Computer Engineering, University of Texas at Austin.

Permanent Address: Shailesh Kumar
2808, Whitis Ave., #101
Austin, TX-78705
USA
skumar@cs.utexas.edu

This thesis was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this thesis were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin.