Applying Reinforcement Learning to Packet Scheduling in Routers

Herman Ferra **Ken Lau** **Christopher Leckie** **Anderson Tang**
Telstra Research Laboratories
770 Blackburn Road, Clayton 3186
Australia
herman.ferra@team.telstra.com
ARC Special Research Centre for Ultra-Broadband Information Networks
Department of Computer Science and Software Engineering
The University of Melbourne, Parkville 3010 Australia
c.leckie@ee.mu.oz.au

Abstract

An important problem for the Internet is how to provide a guaranteed quality of service to users, in contrast to the current "best-effort" service. A key aspect of this problem is how routers should share network capacity between different classes of traffic. This decision needs to be made for each incoming packet, and is known as the *packet scheduling* problem. A major challenge in packet scheduling is that the behaviour of each traffic class may not be known in advance, and can vary dynamically. In this paper, we describe how we have modelled the packet scheduling problem as an application for reinforcement learning (RL). We demonstrate how our RL approach can learn scheduling policies that satisfy the quality of service requirements of multiple traffic classes under a variety of conditions. We also present an insight into the effectiveness of two different RL algorithms in this context. A major benefit of this approach is that we can help network providers deliver a guaranteed quality of service to customers without manual fine-tuning of the network routers.

Introduction

The current Internet Protocol (IP) provides a best-effort service, where no performance guarantees are given by the network. A major challenge in IP networks is how to support multiple classes of traffic with different quality of service (QoS) requirements (Zhang 1995). Services such as video, voice and data each have different service requirements in terms of packet loss, delay and delay variation. Several protocols have been proposed to support different classes of service, such as IntServ (Shenker et al. 1996) and DiffServ (Blake et al. 1998). For example, DiffServ aims to support a small number of traffic classes with different QoS requirements. A key question is how to allocate bandwidth to each class in order to satisfy their QoS requirements.

A common approach used in network routers is to provide separate queues for each class of traffic (Cisco 2002). A classifier is used to assign incoming packets to their appropriate queue. A scheduler then decides which queue to service next in order to manage the allocation of bandwidth to each traffic class. An open research problem is how to implement a suitable scheduling policy that satisfies the QoS constraints of each traffic class while still ensuring reasonable performance for best-effort traffic.

In this paper we present a system for packet scheduling that is based on reinforcement learning (Sutton and Barto 1998). In our approach, reinforcement learning (RL) is used to learn a scheduling policy in response to feedback from the network about the delay experienced by each traffic class. Key advantages of our approach are that our system does not require prior knowledge of the statistics of each traffic flow, and can adapt to changing traffic requirements and loads. In practice, this helps network providers to deliver a guaranteed QoS to customers, while maximising network utilisation and minimising the need for manual intervention.

We make three key contributions in this paper: (1) we present a model for using RL to address the problem of packet scheduling in routers with QoS requirements; (2) we demonstrate the advantages of RL in terms of convergence time in comparison to other scheduling schemes; and (3) we provide an insight into the relative merits of two alternative RL algorithms in the context of this application. We begin by describing the application of packet scheduling. We then describe our solution based on RL, and demonstrate its effectiveness in a range of simulated traffic conditions.

Previous Work

A wide variety of scheduling policies have been proposed for queue management (Zhang 1995). A simple approach is First-Come First-Served (FCFS), where all packets are serviced in the order they arrived. However, this approach ignores the priority of different classes of traffic. Two alternative policies that take priority into consideration are Earliest Deadline First (EDF) and Sequential Priority (SP). EDF selects the packet with the smallest time difference between its delay constraint and its accumulated delay. SP selects the highest priority queue that has packets waiting, i.e., a queue is serviced only if there are no packets waiting in any higher priority queue. A major drawback of EDF and SP is that best-effort traffic may be starved for resources because it is the lowest priority, and has no fixed deadline for service.

In order to prevent the lowest priority queue from being ignored, a technique called Weighted Fair Queuing (WFQ) has been proposed (Demers et al. 1989). A weight is assigned to each queue based on its priority, and the scheduler allocates service to each queue in proportion to its weight.

An important question in WFQ is how to determine the weight of each queue. Although it is possible to calculate weights that satisfy a worst-case delay bound, these weights result in a fixed allocation of network resources. Again, this starves resources from best-effort traffic when higher priority traffic classes are not utlising their allocated bandwidth.

A key problem with these static approaches to packet scheduling is that the statistics of each traffic class are usually not known a priori, and may vary over time. Consequently, we require an approach that adaptively learns a suitable scheduling policy by adapting the weights assigned to each queue or traffic class.

Several adaptive approaches have recently been proposed for assigning weights to each traffic class (Anker et al. 2001, Hall and Mars 1998, Wang et al. 2001). In particular, Hall and Mars (1998) have proposed an adaptive approach to learning scheduling policies for queue management, where the statistics of the incoming traffic are unknown. They implement their scheduling policy as a stochastic learning automaton (SLA). The role of a SLA is to select from a given set of *actions*, where there is one action for each queue, and each action corresponds to selecting the next available packet from the associated queue. The learning automaton implements a stochastic scheduling *policy* by associating a probability with each action, so that actions are chosen at random according to their probabilities. When an action is selected and executed, the SLA receives a feedback signal in the form of a *reward* from the environment. This reward reflects whether the packet from the selected queue met the target delay requirements for that queue. In response to this reward, the SLA refines its control policy by updating the probabilities associated with each queue.

Hall and Mars demonstrated that their SLA was able to outperform the FCFS, EDF and SP scheduling policies. In particular, they found that the scheduling policy learned by the SLA was able to satisfy the delay constraints for two traffic classes, without starving a third best-effort class.

A limitation of the SLA approach is that the scheduling policy does not take into consideration the current state of the system when choosing an action. For example, the state of each queue may influence our choice of which queue to service next. In this paper, we propose an alternative approach to learning scheduling policies based on reinforcement learning (Sutton and Barto 1998). In contrast to SLA, reinforcement learning constructs a scheduling policy that takes the current state of the system into consideration when selecting an action. In this paper, we demonstrate that reinforcement learning is able to learn an efficient scheduling policy much faster than SLA. This means that by using reinforcement learning we can adapt our scheduling policy to changing traffic conditions much faster than if SLA is used.

Problem Definition

Our problem definition is based on the model of packet scheduling described by Hall and Mars (1998). We have used their model so that we can compare the performance of their SLA with our own proposal. Our aim is to schedule N classes of traffic, where each traffic class has its own queue

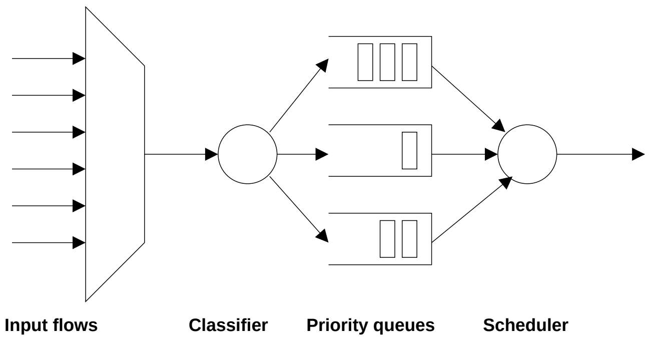

Figure 1: Packet scheduling for multiple traffic classes

$q_i, i = 1 \dots N$. Let q_N denote the queue for best-effort traffic, which has no predefined delay requirements. For each remaining queue $q_i, i = 1 \dots N-1$, there is a mean delay requirement R_i, which is the maximum acceptable mean queuing delay per packet for the traffic class assigned to q_i. Let M_i denote the measured mean queuing delay of packets in q_i over the last P packets. Our goal is to learn a scheduling policy that minimises M_N while ensuring that $M_i \leq R_i$ for $i = 1 \dots N-1$. In other words, we want to satisfy the QoS constraints for queues $q_1 \dots q_{N-1}$ while maximising the available bandwidth to the best-effort queue q_N.

In keeping with the model of Hall and Mars, all packets in our system have a constant fixed length. This is typical of the internal queues in routers that use a cell switching fabric. We can model this traffic using a discrete-time arrival process, where a fixed length timeslot is required to transmit a packet, and at most one packet can be serviced at each timeslot. The arrival of packets is described by a Bernoulli process, where the mean arrival rate l_i for q_i is represented by the probability of a packet arriving for q_i in any timeslot.

The role of the scheduler is to decide which queue should be serviced at each timeslot (see Figure 1). At each timeslot, the scheduler must select an action $a \in \{a_1, \dots, a_N\}$, where a_i is the action of choosing to service the packet at the head of queue q_i. The scheduler makes this selection by using a scheduling policy π, which is a function that maps the current state of the system s onto an action a. If the set of possible actions is denoted by A, and the set of possible system states is denoted by S, then $\pi : S \rightarrow A$.

The second component of the scheduler is a reward function $r : S \times A \rightarrow \mathbb{R}$. When an action $a \in A$ is executed in state $s \in S$, the scheduler receives a reward $r(s, a)$ from the system. This reward provides feedback about the immediate value of executing the action a.

Our goal is to learn an optimal scheduling policy π^* by iteratively refining an initial policy π^0. Each time we use our current policy π to select a scheduling action a in state s, we observe the immediate reward $r(s, a)$, and use this reward as feedback to update our current scheduling policy $\pi \rightarrow \pi'$. This approach is known as reinforcement learning, which has been applied to a variety of scheduling and control tasks (see Sutton and Barto 1998). In the next section, we describe our method for using reinforcement learning to optimise the scheduling policy of our queue management system.

Our Reinforcement Learning Approach

There are three key components to our application of using reinforcement learning to learn scheduling policies for queue management. First, we require a representation of the state s of our system, which reflects the state of the traffic in our queues. Second, we require a suitable reward function $r(s, a)$, which reflects the immediate value of our scheduling actions. Finally, we require a learning algorithm to refine our policy function $\pi(s)$ based on the feedback provided by our reward function. Let us now describe our solution for each of these components of our system.

State Representation

The reason for introducing the system state into the policy function is so that the scheduler can learn how to act in different situations. This is in contrast to the approach of using a SLA, which uses a single state in its policy function, i.e., the scheduling policy does not depend on the state of the queues. By introducing a more sophisticated state representation we can potentially gain greater control, albeit at the risk of greater complexity. However, we need to ensure that the state representation is not too complex, otherwise there may be too many parameters to be tuned, which may slow the convergence rate of the algorithm.

Our aim is to use different scheduling policies depending on which queues are not meeting their delay requirements. We represent the state of the system by a set of $N-1$ binary variables $\{s_1, \ldots, s_{N-1}\}$, where each variable s_i indicates whether traffic in the corresponding queue q_i is meeting its mean delay requirement R_i,

$$s_i = \begin{cases} 0, & M_i \leq R_i, \\ 1, & M_i > R_i. \end{cases}$$

Note that there is no variable corresponding to the best-effort queue q_N, since there is no mean delay requirement for that queue. For example, the state $\{0, 0, \ldots, 0\}$ represents that all queues have satisfied their mean delay constraint, while $\{1, 0, \ldots, 0\}$ represents that the mean delay requirements are being satisfied for all queues except q_1. Thus, if there are N queues in the system including one best-effort queue, then there are 2^{N-1} possible states. In practice, the number of traffic classes is normally small, e.g., four classes in Cisco routers with priority queuing (Cisco 2002), in which case the number of states is acceptable.

Reward Function

The role of the reward function is to provide feedback to the reinforcement learning algorithm about the effect of a scheduling action. Based on this feedback, the learning algorithm can decide how to update the current scheduling policy. Our aim is to provide a positive reward when packets are serviced within their delay requirement, and a negative reward when they are late. We also want to provide a positive reward when the system moves to a better state, i.e., when the measured mean delay for a queue falls below the required mean delay. Thus our reward function r comprises a time reward component r_{time} and a state reward component r_{state}, where $r = r_{time} + r_{state}$.

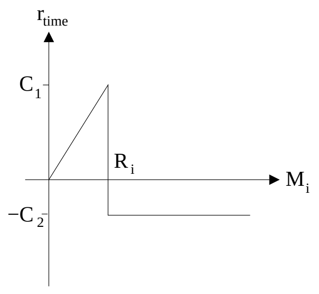

Figure 2: Time reward function

Every time a scheduling action is executed, the time reward $r_{time,i}$ for each queue q_i is calculated in terms of the mean delay requirement R_i and the measured mean delay M_i

$$r_{time,i} = \begin{cases} \frac{C_1 M_i}{R_i} & \text{if } M_i < R_i \\ C_1 & \text{if } M_i = R_i \\ -C_2 & \text{if } M_i > R_i. \end{cases}$$

As shown in Figure 2, the time reward is positive when $M_i \leq R_i$, and negative when $M_i > R_i$. It is maximised when the mean delay requirement is just satisfied. There is a diminishing reward as M_i approaches zero, since any reduction in M_i below R_i is wasting bandwidth that could be allocated to other queues. In general, it is possible to change the form of the reward function depending on the type of QoS requirements that need to be satisfied.

The total time reward is a weighted sum of the rewards for each queue

$$r_{time} = \sum_{i=1}^{N-1} w_i \, r_{time,i}$$

where the weights w_i depend on which queue was serviced by the last scheduling action. In practice, we have found that suitable weights are $w_i = 0.3$ if q_i was the queue serviced by the last action, otherwise $w_i = 1.0$. These weights discourage the scheduler from servicing queues with satisfactory performance if there are other queues experiencing unsatisfactory performance. Although the choice of weight values is not critical, we found that we can signficantly improve the convergence rate of our system by using a non-zero weight for queues that were not serviced by the last action.

The state reward is positive if a scheduling action causes the system to move to a better state s' compared to the previous state s. State s' is considered to be better than s if it has more queues whose mean delay requirements are being met, e.g., $s' = \{0, 0, \ldots, 0\}$ is better than $s = \{1, 0, \ldots, 0\}$. Thus

$$r_{state} = \begin{cases} C_3 & \text{if } s' \text{ is better than } s \\ 0 & \text{otherwise.} \end{cases}$$

Learning Algorithm

Our approach to learning a scheduling policy $\pi(s)$ is based on the standard reinforcement learning approach of Q-learning (Watkins 1989). Let us begin by describing the general goal of learning in this context. Consider the system at time t in state s_t. The scheduler selects an action a_t and in turn receives a reward r_{t+1}. As a result of this action and the arrival of new packets, the system moves to a new state s_{t+1}. From this new state the scheduler then selects another action a_{t+1} according to its current policy π and consequently another reward r_{t+2} is received. This process continues and we can think of a trial resulting in a particular sequence of future rewards $(r_{t+1}, r_{t+2}, ...)$ as having been generated by the scheduler in following its policy at time t.

How good was the selection of the action a_t at time t? Had the scheduler selected a different action, say a'_t, a different sequence of rewards $(r'_{t+1}, r'_{t+2}, ...)$ would most likely have been generated. This is where the notion of *return* becomes important. We define the return for being in state s_t and taking action a_t as:

$$R^{\pi}(s_t, a_t) = \sum_{l=t+1}^{\infty} \gamma^{l-t-1} r_l$$

where $0 \leq \gamma < 1$ is a constant known as the *discount factor*. For bounded values of r the terms in the summation above diminish with increasing value of l. This is in line with the notion that the more remote in time the reward is, the less credit for that reward should be given to the choice of action a_t. The superscript π indicates that the scheduler follows the policy at every time step $\geq t+1$.

In a sense $R^{\pi}(s, a)$ represents the total discounted future reward received for picking action a in state s. Using $R^{\pi}(s, a)$ as a basis we can evaluate the selection of one action over another from a given state s. We say that for policy π, choosing action a is better than action a' in state s if $R^{\pi}(s, a) > R^{\pi}(s, a')$. Since the arrival of packets is in fact probabilistic and π may also select future actions according to some probabilistic process, $R^{\pi}(s, a)$ is a random variable and it makes more sense to evaluate actions based on the expected value of $R^{\pi}(s, a)$. Let $Q^{\pi}(s, a)$, be defined as the *expected return* starting in state s and selecting action a following policy π thereafter, i.e.,

$$Q^{\pi}(s, a) = E\big[R^{\pi}(s, a)\big].$$

In reinforcement learning the goal of the learner is to maximise the expected return from any state s.

We use the Q-learning approach to reinforcement learning (Watkins 1989) to learn a scheduling policy $\pi(s)$ by updating the matrix $Q(s, a)$. Since our problem involves a finite number of states, the scheduler can maintain a matrix $Q(s, a)$, which estimates the expected return received for choosing action a in state s. For a given state s, the action with the maximum value can be chosen based on the corresponding entries in Q. Initially, the Q matrix contains random values. The values in the Q matrix for each state and action pair are updated using the formula below, which combines the tasks of policy evaluation and policy improvement to evolve the current policy π to the optimum policy π^*. After an action a_t is executed in state s_t, the measured reward is used to update the value in $Q(s_t, a_t)$ using the formula:

$$Q(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \, max_{a'} \, Q(s_{t+1}, a')$$

where s_{t+1} is the state after action a_t was executed in state s_t, and γ is the discount factor.

When choosing an action using the Q matrix, we need to make a trade-off between exploration of new policies and exploitation of existing policies. When Q represents the optimum policy, then we should exploit this policy when choosing the next scheduling action. However, in a dynamic environment we need to explore the state space so that we can learn the value of each action. We have investigated two common approaches to balancing exploration and exploitation, namely, *batch updating* and *ϵ-greedy* search.

Batch Updating: In the batch updating procedure, we derive a scheduling policy from the current value of the Q matrix, and then update Q off-line using the rewards from actions chosen by the current scheduling policy. Our scheduling policy in state s is to choose an action at random according to a probability distribution $P(a_i|s)$, where we assign a probability to each action using a Boltzmann-like distribution based on the current Q matrix, i.e.,

$$P(a_i|s) = \frac{k^{Q(s,a_i)}}{\sum_j k^{Q(s,a_j)}},$$

where $k > 0$ is a constant. After a batch size of T actions, we update the probabilities in the scheduling policy based on the new Q matrix. This approach gives the scheduler a chance to evaluate the current policy before updating it.

ϵ-greedy Search: In the ϵ-greedy updating procedure, the Q matrix is updated on-line every time a scheduling action is executed. In this case, the scheduling policy in a given state s is usually the action a with the largest $Q(s, a)$ value. However, with probability ϵ the scheduler may choose at random from the remaining actions a' where $Q(s, a') < Q(s, a)$, in order to explore the value of alternative actions.

Evaluation

In order to evaluate our reinforcement learning (RL) system we have used the stochastic learning automaton (SLA) of Hall and Mars as a basis for comparison. In particular, our aim is to investigate the following issues: (1) how does RL perform compared to the SLA in terms of packet delay and convergence rate; (2) how adaptive is RL to changes in arrival rates and delay requirements; (3) how does the choice of learning scheme (batch or ϵ-greedy) affect performance?

Our initial test conditions are based on the simulation described by Hall and Mars (1998), which involves a three queue system whose arrival statistics and mean delay requirements are listed in Table 1. We have reproduced the experiments of Hall and Mars as a benchmark. The measured mean delay using the SLA is shown in Figure 3, which matches the results quoted in (Hall and Mars 1998). Note that Hall and Mars found that the SLA outperformed common static policies, such as FCFS, EDF and SP.

We then applied our RL system under the same simulation conditions, using the parameters shown in Table 2. The measured mean delay for the batch algorithm is shown in Figure 4, and for the ϵ-greedy algorithm in Figure 5. Both RL algorithms satisfied the mean delay requirements for queues 1 and 2, while achieving similar performance to the SLA for the best effort traffic in queue 3. However, both RL schemes

Queue	Arrival Rate [packets / timeslot]	Mean Delay Requirement [timeslots]
1	0.15	10
2	0.25	4
3	0.25	best-effort

Table 1: Test conditions for 3 traffic classes

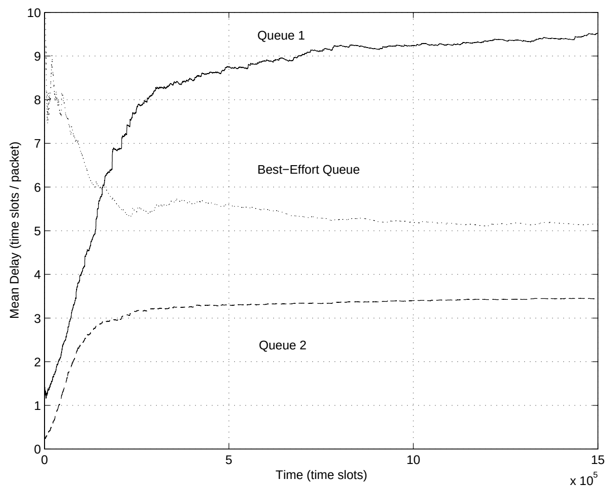

Figure 3: Mean delay using a stochastic learning automaton for 3 traffic classes

had a significant advantage over the SLA in terms of convergence time. The convergence time for the SLA was at least 1.0×10^6 timeslots. In contrast, the batch RL approach required only 1.5×10^4 timeslots, and the ϵ-greedy RL approach required approximately 1.0×10^4 timeslots. This represents a 60 times and 100 times reduction in convergence time, respectively, compared to the SLA.

We evaluated how adaptive our RL schemes were by simulating three classes of traffic whose arrival rates and delay requirements change at a particular time. Table 3 describes the change in traffic conditions that occurred in our simulation after 100,000 timeslots. Figures 6 and 7 show the measured mean delay of each queue under these conditions for batch RL and ϵ-greedy RL, respectively. In each case the RL system was able to adapt and converge to the new delay requirements under the new traffic levels.

This experiment highlights a key difference between the batch and ϵ-greedy RL schemes. In the batch RL scheme, the scheduling policy is updated for both queues in parallel,

Parameter	Batch RL	ϵ-greedy
Discount factor γ	0.5	0.5
Reward constant C_1	50	50
Penalty constant C_2	20	20
State reward constant C_3	50	50
Others	$T = 1000$ $k = 3$	$\epsilon = 0.2$

Table 2: Parameters used in reinforcement learning

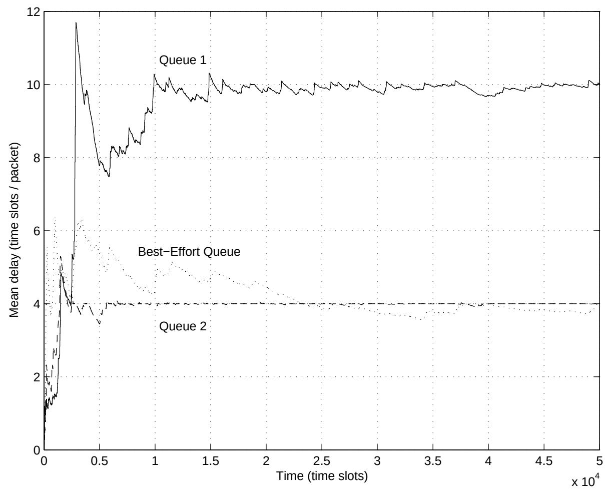

Figure 4: Mean delay using batch reinforcement learning for 3 traffic classes

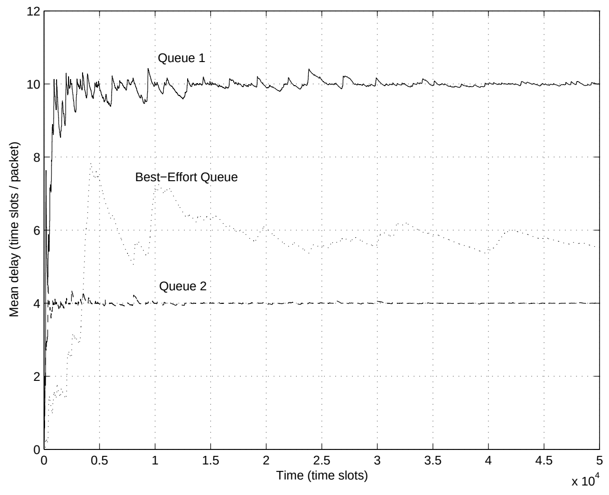

Figure 5: Mean delay using ϵ-greedy reinforcement learning for 3 traffic classes

Queue	Arrival Rate [packets / timeslot]	Mean Delay Requirement [timeslots]
1	0.30 → 0.20	5 → 2
2	0.25 → 0.20	5 → 3
3	0.40 → 0.55	best-effort

Table 3: Test conditions for 3 traffic classes with a step change at 100,000 timeslots

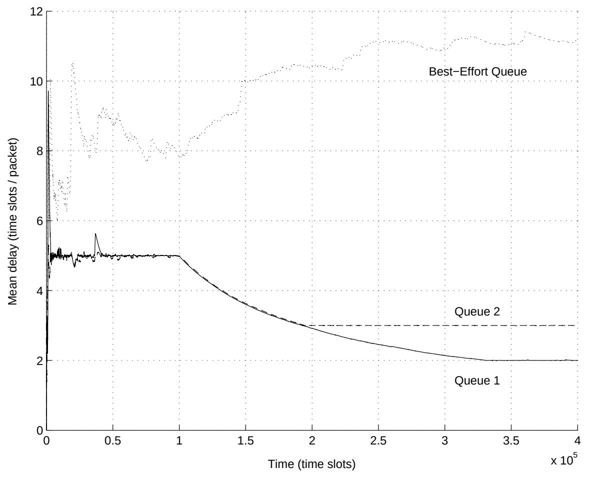

Figure 6: Mean delay using batch reinforcement learning for 3 traffic classes with a step change at 100,000 timeslots

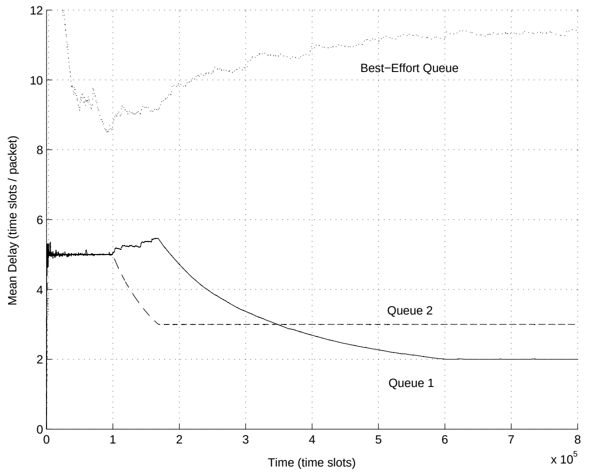

Figure 7: Mean delay using ϵ-greedy reinforcement learning for 3 traffic classes with a step change at 100,000 timeslots

and thus the mean delay improves in both queues simultaneously. This is because the batch RL scheme updates the whole Q matrix after many scheduling decisions have been made. In contrast, the ϵ-greedy RL scheme first improves the performance of queue 2, and then improves the performance of queue 1. This is because the ϵ-greedy scheme normally picks the action with the highest Q value at each step. Consequently, just one Q value will be updated after performing a particular action. This demonstrates that the batch RL scheme has an important advantage over the ϵ-greedy scheme in terms of its speed of adaptation.

We have tested the scalability of our RL schemes as we increased the complexity to four or five traffic classes. In each case, we were able to satisfy the requirements of each queue using both the batch updating and the ϵ-greedy schemes.

Conclusion and Further Work

The main contributions in this paper are: (1) a model for using RL for packet scheduling in routers; (2) insight into the effectiveness of two RL algorithms (batch and ϵ-greedy) to this task; and (3) a demonstration of the effectiveness of RL using a variety of simulations.

Both RL schemes are able to satisfy QoS requirements for multiple traffic classes, without starving resources from best-effort traffic. Furthermore, our RL schemes can adapt to changing traffic statistics and QoS requirements. In terms of convergence rate, both RL schemes outperformed stochastic learning automata. A key difference between the two RL schemes is the speed with which they can adapt to changing traffic statistics and requirements. From this perspective, batch RL outperforms ϵ-greedy RL, because batch RL updates the scheduling policy for all queues simultaneously, whereas ϵ-greedy RL updates the policy for each queue in a sequential manner.

Our approach to packet scheduling raises several topics for further research. An interesting research problem is to model other types of QoS constraints in addition to mean delay. It is also worth investigating the effect of different traffic arrival models on the performance of our RL schemes. Finally, we can also consider alternative models for representing the state and reward in our system.

Acknowledgements

This work was supported by the Australian Research Council. The permission of the Chief Technology Officer, Technology Strategy and Research, Telstra Corporation, to publish this paper is gratefully acknowledged.

References

T. Anker, R. Cohen, D. Dolev and Y. Singer 2001. Probabilistic Fair Queuing. In *IEEE 2001 Workshop on High Performance Switching and Routing,* pp. 397-401.

S. Blake et al. 1998. An Architecture for Differentiated Services. Internet RFC 2475, December 1998.

Cisco 2002. Quality of Service Networking. www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm

A. Demers, S. Keshav and S. Shenker 1989. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of SIGCOMM'89,* pp. 1-12.

J. Hall and P. Mars 1998. Satisfying QoS with a Learning Based Scheduling Algorithm. In *6th International Workshop on Quality of Service,* pp. 171-176.

S. Shenker, C. Partridge and R. Guerin 1996. Specification of Guaranteed Quality of Service. Internet RFC 2212.

R. Sutton and A. Barto 1998. Reinforcement Learning: An Introduction. MIT Press, 1998.

H.Wang, C. Shen and K. Shin 2001. Adaptive-Weighted Packet Scheduling for Premium Service. In *IEEE Int Conf on Communications (ICC 2001),* pp. 1846-1850.

C. Watkins 1989. Learning from Delayed Rewards. Ph.D. dissertation, King's College, Cambridge, United Kingdom.

H. Zhang 1995. Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks. In *Proceedings of the IEEE,* 83, October 1995, pp. 1374-1396.