RLTE: Reinforcement Learning for Traffic-Engineering

Erik Einhorn and Andreas Mitschele-Thiel

Technical University Ilmenau,
Integrated Hardware and Software Systems Group,
98684 Ilmenau, Germany
{erik.einhorn,mitsch}@tu-ilmenau.de

Abstract. Quality of service (QoS) is gaining more and more importance in today's networks. We present a fully decentralized and self-organizing approach for QoS routing and Traffic Engineering in connection oriented networks, e.g. MPLS networks. Based on reinforcement learning the algorithm learns the optimal routing policy for incoming connection requests while minimizing the blocking probability. In contrast to other approaches our method does not rely on predefined paths or LSPs and is able to optimize the network utilization in the presence of multiple QoS restrictions like bandwidth and delay. Moreover, no additional signaling overhead is required. Using an adaptive neural vector quantization technique for clustering the state space a considerable speed-up of learning the routing policy can be achieved. In different experiments we are able to show that our approach performs better than classical approaches like Widest Shortest Path routing (WSP).

1 Introduction

Network traffic has become very versatile within the last few years. Each network application makes different demands on the underlying network infrastructure. Streaming Video on Demand (VoD) for instance requires a high bandwidth while for Voice over IP (VoIP) a small delay is more important. The ability to guarantee certain network parameters like bandwidth, delay, jitter, loss or availability usually is referred to as Quality of Service (QoS). However, most networks are still IP-based. Since IP is a connectionless protocol, IP packets do not use specific paths between two communicating endpoints. This results in unpredictable QoS in a best-effort network. In contrast, the connection oriented Multiprotocol Label Switching (MPLS) standard [1] allows a better control for traffic routing and Traffic Engineering [2]. Traffic Engineering decides how to map the traffic requirements to the physical network in order to optimize the whole network resource utilization [3].

However, the problem of optimal routing in the presence of multiple independent QoS requirements is known to be NP-hard [4]. Therefore, heuristic or approximation algorithms are applied to solve this problem. The most often used algorithm for routing LSPs is the Min-Hop-Algorithm [5]. From all possible paths

between a source and a destination of a connection that fulfill the desired QoS constraints, the one with the least number of links is chosen. This behavior often results in bottlenecks and consequently connection requests are rejected although other parts of the network still have enough resources available. The widest-shortest path routing (WSP) [6] tries to solve this problem by choosing the path with the largest residual bandwidth from all possible paths for a connection. Thus, it avoids the usage of heavily loaded links. One major drawback of this approach is the necessity for each node to have global knowledge about the current load situation of the network. It therefore imposes an additional signaling and information flooding overhead on the network. A more sophisticated technique is used for the Minimum Interference Routing Algorithms MIRA [4], LMIR [7] and DORA [8]. The main idea of these approaches is to route an incoming connection along a path that least interferes with other routes that may be crucial to satisfy possible future requests. For this purpose, MIRA manages a list of critical links and tries to preserve these links as long as alternative paths are available. As a consequence, some links will remain underutilized leading to a suboptimal usage of the network resources. Similar to WSP these approaches also require global knowledge about the network state and therefore increase the additional signaling overhead.

Apart from the aforementioned "classical" routing approaches a couple of alternative methods have been researched recently. Some of them use Ant Colony Optimization (ACO) for QoS routing in MPLS networks [9]. Other researchers have studied how reinforcement learning can be used to solve routing problems. In contrast to the classical routing protocols based on heuristics, where the routing decision is explicitly specified within the routing algorithm, routing approaches based on reinforcement learning are able to learn the routing on their own depending on a feedback given by the network. At first reinforcement learning was used for routing in IP networks. Boyan and Littman [10] use Q-Learning to learn an optimal routing policy that minimizes the delay for packet transmissions. In [11] the optimal policy is obtained using policy search via gradient ascent. In both publications the authors are able to show that their approach performs better than shortest path routing. In [3] a "Distributed Adaptive Path Selection Scheme" for MPLS networks (MAPS) is presented. This method uses reinforcement learning agents located at the networks edge routers. In comparison with Widest Shortest Path (WSP) MAPS significantly reduces the blocking probability without having global knowledge about the core network and hence without additional signaling traffic. However, the approach only focuses on the selection of predefined paths. Feasible paths must be established using a k-shortest path algorithm beforehand [3]. Therefore, the approach can not dynamically react to changes in the network topology or link failures. The same disadvantage applies for an approach that is described in [12] and that uses reinforcement learning to obtain a set of load-sharing factors for optimal load-sharing among different LSPs in MPLS networks.

In this paper we present a novel QoS routing algorithm based on reinforcement learning which can be used in MPLS networks or other connection oriented

networks that support QoS. In contrast to the approaches mentioned above our algorithm does not rely on predefined paths. Instead it learns feasible paths depending on the connection requests while minimizing the blocking probability. Our approach is distributed and does neither need any global knowledge about the network topology nor the current load situation of the whole network, instead local information is sufficient. Moreover, it does not only consider the bandwidth as one QoS parameter as most of the above approaches do, instead it also takes delay restrictions into account. Furthermore, our approach should attain the following goals that we consider important:

1. Compared to heuristic routing approaches like WSP that rely on global knowledge about the network, the approach should at least achieve a similar performance although it uses local information only.
2. The learning time - a major drawback of reinforcement learning approaches - must be reduced to a minimum and should scale well if the network size increases.
3. The approach must be able to react dynamically on changes in the network topology such as link and node failures.

The organization of this paper is as follows. In the next sections the QoS routing problem is defined. Thereafter our algorithm is described in detail and we present techniques to achieve the goals mentioned above. In section 4 we present the results of different simulations and experiments before we conclude the paper with a summary.

2 Problem Definition

We consider a network described by the quadruple $G = (N, L, B, D)$ consisting of a set of n nodes (routers) $N = \{1, \ldots, n\}$ and a set of m links (arcs) $L = \{1, \ldots, m\}$. Furthermore, the functions $B : L \mapsto \mathbb{R}$ and $D : L \mapsto \mathbb{R}$ assign a certain bandwidth $B(l)$ and some delay $D(l)$ to each link $l \in L$. In contrast to other approaches, neither bandwidth nor delay need to be integers. Furthermore, it is not necessary to distinguish between ingress, egress and core routers. In our approach each node $n \in N$ actually can be sending or receiving node. However, if the algorithm is used in MPLS networks, the differentiation between core and edge routers will be induced by the MPLS network.

Let $R = (r_0, \ldots, r_i, \ldots)$ be the (generally infinite) sequence of connection requests that arrive at the network, where each connection request or call $c_i = (d, \beta, \delta)$ sent from some source node $s \in N$ specifies the address of the destination node $d \in N$ a bandwidth demand $\beta \in \mathbb{R}$ and a maximum delay restriction $\delta \in \mathbb{R}$ for the desired connection. Since we do not rely on predefined paths or LSPs, the QoS routing algorithm has to find an appropriate path $p = (l_1, l_2, \ldots, l_k)$ of adjacent links that connects the ingress-egress pair (s, d) and that fulfills the desired QoS requirements, namely:

1. the required bandwidth demand:

$$\min_{i=1}^{k} \check{B}(l_i) \geq \beta$$

2. the maximal delay constraint:

$$\sum_{i=1}^{k} D(l_i) \leq \delta$$

where $\check{B}(l_i)$ denotes the residual bandwidth that is available for link l_i. If the path does not satisfy these constraints, the connection request is blocked and rejected. Otherwise the connection can be established and the required bandwidth is reserved. The resources remain reserved until the connection is released.

The concern of an optimal QoS routing approach now is to find an optimal routing policy that maximizes the number of accepted requests or in other words minimizes the blocking probability of the connection requests.

3 Reinforcement Learning for Traffic Engineering

Our reinforcement learning approach for finding such a routing policy is based on the SARSA-Learning algorithm [13], a variant of Q-Learning. Since SARSA-Learning and Q-Learning are quite common techniques for reinforcement learning they will not be described in detail here. Further information can be found in [13, 14] and [15].

Reinforcement learning (RL) is one type of Machine Learning, where a RL-agent learns how to map situations (states) to actions to maximize a numerical reward signal [15]. This mapping of states S to actions A is called *policy* $\pi : S \mapsto A$. In SARSA- and Q-Learning the policy can be determined by learning an *action-value function* $Q : S \times A \mapsto \mathbb{R}$. This function gives the expected reward $Q(s_t, a_t)$ for starting in state $s_t \in S$, taking action $a_t \in A$ and then following policy π thereafter. For choosing action a_t in state s_t the RL-agent receives a reward r_t and attains to state s_{t+1} where it again selects some action a_{t+1}. The SARSA-Learning rule will then update the Q-values as follows [15]:

$$Q(s_t, a_t) \leftarrow (1 - \beta)Q(s_t, a_t) + \beta\Big[r_t + \gamma Q(s_{t+1}, a_{t+1})\Big] \tag{1}$$

where the β notates the learning rate and γ the so-called discount rate.

For selecting an action the *softmax action selection* can be used. It is based on the Boltzmann distribution and chooses a certain action $a \in A$ in state s with the following probability:

$$P(a) = \frac{\exp\left(\frac{Q(s,a)}{T}\right)}{\sum_{b \in A} \exp\left(\frac{Q(s,b)}{T}\right)} \tag{2}$$

where the parameter T is called the *temperature*. High temperatures cause the actions to be all (nearly) equi-probable [15]. For low temperatures the softmax action selection converges to a greedy action selection that chooses the action with the highest Q-value. In our experiments a temperature between 0.1 and 0.25 gives the best results.

Overview

In our approach we use one RL-agent $\mathfrak{A}$ that is distributed over the network. Each node (router) $i \in N$ of the network contains one part $\mathfrak{A}_i$ of the RL-agent that is responsible for learning just one part $\pi_i : S_i \mapsto A_i$ of the whole policy π. Without loss of generality we assume that adjacent links for each node i are renumbered in ascending order from 1 to m_i. Moreover, we assume that the partial agent $\mathfrak{A}_i$ at each node is able to measure the residual bandwidth $\check{B}(j)$ and the delay $D(j)$ of each adjacent link $j \in (1, \ldots, m_i)$.

For signaling the connection requests and for establishing the paths we use a simple protocol similar to RSVP-TE [16]. The protocol uses a few messages only, that are described afterwards. For each incoming connection request a PathResv-message is sent hop-by-hop to the destination node. Analogous to [10] the partial RL-agent $\mathfrak{A}_i$ at each node $i \in N$ has to select one of its outgoing links for forwarding the PathResv-message to the next hop (see Fig. 1a). If the PathResv-message successfully arrives at its targeted node, a ResvAcc-message is sent back to the sender of the connection request. This ResvAcc-message takes the reverse path as the corresponding PathResv-message and reserves the required resources for the connection. Additionally, this ResvAcc-message contains a positive reward, that reinforces the action each partial agent has selected (see Fig. 1b). If one of the demanded QoS parameters could not be satisfied at some chosen link, a ResvReject-message is sent back to the sender taking the reverse path of the PathResv-message. In contrast to the ResvAcc-message the ResvReject-message contains a negative reward. By including the reward into the necessary signaling messages any additional signaling overhead is avoided.

In a real world implementation the reward can be easily included as additional object into the signaling messages of the RSVP-TE protocol.

Detailed Algorithm

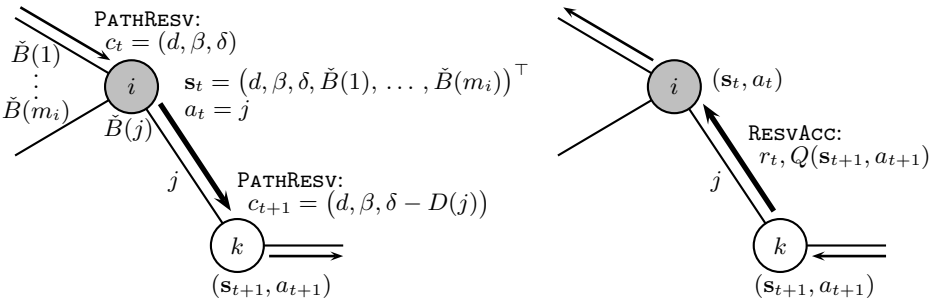
As stated above, for each incoming PathResv-message containing a connection request $c_t = (d, \beta, \delta)$ the agent $\mathfrak{A}_i$ chooses one of its outgoing links as action $a_t \in A_i$ using the softmax action selection. Since the links are numbered in ascending order, the discrete action space can be described by $A_i = \{1, \ldots, m_i\}$. The action selection depends on the current state of the agent, which is determined by taking the destination address d, the bandwidth requirement β and the delay restriction δ of the connection request $c_t = (d, \beta, \delta)$ into account. Additionally, the residual bandwidths $\check{B}(1), \ldots, \check{B}(m_i)$ of all of adjacent links are measured. Using this information a state vector $\mathbf{s}_t \in S_i$ is formed:

$$\mathbf{s}_t = \Big(d, \beta, \delta, \check{B}(1), \ldots, \check{B}(m_i)\Big)^\top \tag{3}$$

Please note, that no global knowledge about the network state is required here at all.

Each partial RL-agent keeps track of its chosen action and the state which led to that action. This allows the agent to associate the correct state-action

pairs to the delayed reward that arrives later with the corresponding ResvAcc-message or ResvReject-message. However, although the RL-agent attains a new state with each arriving connection request, it does not need to maintain all of these states. In practice the state and the chosen action can be recovered when the ResvAcc-message or ResvReject-message returns to the agent and do not need to be stored inside of each agent.



(a) PathResv-message (b) ResvAcc-message

Fig. 1. a: The PathResv-message containing the connection request c_t is sent hop-by-hop towards the destination node. Depending on the current state $\mathbf{s}_t$ each partial agent selects one link as action a_t for forwarding the message. This will bring the distributed agent into the state $\mathbf{s}_{t+1}$. Each state is observed by only taking local information of the concerning partial agent into account. **b:** The ResvAcc-message takes the reverse path as the PathResv-message and carries the reinforcement r_t and the Q-value $Q(\mathbf{s}_{t+1}, a_{t+1})$ of each subsequent node.

Corresponding to the selected action a_t the PathResv-message is forwarded along the chosen link j to the next node $k \in N$ and its partial agent $\mathfrak{A}_k$. Thereby, the delay restriction δ of the PathResv-message is decreased by the delay $D(j)$ of the chosen link. The connection request that arrives at node k can then be described by $c_{t+1} = \big(d, \beta, \delta - D(j)\big)$. It will bring the distributed agent $\mathfrak{A}$ into a new state $\mathbf{s}_{t+1}$ that is observed by the partial agent $\mathfrak{A}_k$ according to equation 3 again using local information only. Once more an action a_{t+1} is chosen for forwarding the message (see Fig. 1a).

Using this forwarding mechanism the PathResv-message will finally arrive at its destination node d. As described earlier a ResvAcc-message containing a positive reward r will be sent back to the sender taking the reverse path as the corresponding PathResv-message.

The PathResv-message is also used by each partial agent to propagate its Q-value $Q(\mathbf{s}, a)$ of the observed state s and the taken action a back to the previous node on the path. Therefore, each partial agent that was involved in forwarding the PathResv-message will receive a corresponding ResvAcc-message containing a reward r_t and the Q-value $Q(\mathbf{s}_{t+1}, a_{t+1})$ of its subsequent node (see Fig. 1b).

Together with its own state $\mathbf{s}_t$ and its chosen action a_t each agent is able to adapt its own Q-values $Q(\mathbf{s}_t, a_t)$ according to the SARSA-learning rule using equation 1. The same mechanism is used for RESVREJECT-messages, that are sent if one of the QoS restriction is not satisfied during the path selection. The only difference is that RESVREJECT-messages contain a negative reward. If the RESVREJECT-message finally arrives at the sending node it depends on the upper layer protocols if the request is sent again or definitely rejected.

At the beginning all Q-values are initialized uniformly and each partial RL agent begins its learning *tabula rasa*. Hence, for the first arising connection requests the routing in the network will be random. After some time of exploration and learning the agents will develop a feasible routing policy and perform better with each new request. In section 4 we show that the required time for learning is acceptable in comparison to classical routing approaches.

State Space Clustering

Most related RL-based algorithms for routing [3][10] use a table where the Q-values of each state-action pair are stored. However, since our state space is continuous we cannot use such a table based Q-Learning approach here. Additionally, as seen in equation 3 each state consists of $3 + m$ elements, where m is the number of outgoing links. Hence, the dimension of the state space can become very high depending on the valency of the nodes. This would lead to a slow convergence while learning the optimal routing policy. Therefore, we have to apply some kind of state space clustering.

Similar to [17] we use a variant of Growing Neural Gas (GNG) [18] as adaptive neural vector quantization technique for optimal clustering of the continuous state space. The neurons of the GNG store the Q-values for the actions of the action space A and they are associated to reference vectors in the state space, which can be regarded as positions of the corresponding neurons. Depending on its position $\mathbf{w}_n \in S$ each neuron n is responsible for a certain (voronoi) region in the state space S.

To obtain the Q-value $Q(\mathbf{s}, a)$ for a given state-action pair $(\mathbf{s}, a)$ the best-matching neuron of the GNG is chosen, i.e. the neuron whose reference vector $\mathbf{w}_n$ has the smallest Euclidean distance to the state $\mathbf{s}$. Finally, the desired Q-value stored in the best-matching neuron can be obtained as seen in Fig. 2.

Similar to [19] we insert new neurons if the distance between the best-matching neuron and the state exceeds a certain threshold. While learning we do not only adapt the Q-values of the best-matching neuron according to the learning rule in equation 1 but also the Q-values of the topologically neighboring neurons, since they represent similar states. Neurons close to the best-matching neuron are adapted more (with a higher learning rate) than neurons in larger distances. This method is elaborately described in [17] and increases the speed of learning dramatically as shown in section 4.

In our approach each partial RL-agent at each node uses its own GNG to cluster its state space. However, the destination address of the connection request that is specified in each state is not yet included into the state space clustering.

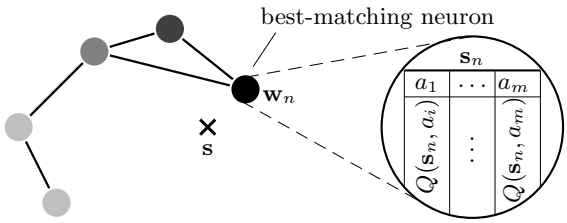


Fig. 2. The state space is clustered using a GNG *(left)*. For obtaining a certain Q-value $Q(\mathbf{s}, a)$ the best-matching neuron for the state $\mathbf{s}$ is chosen. It contains a table with the corresponding Q-values for each action *(right)*. The different colors indicate, that the Q-values of neurons close to the state $\mathbf{s}$ are adapted more than the values of neurons in larger distances during a learning step.

Hence, our next step will be to perform the clustering over the whole state space comprising the destination addresses. To do so a topological addressing scheme and an appropriate metric must be chosen. If two nodes in the network were separated by one or a few hops only, the metric should yield a small distance for the addresses of both nodes. If - on the other hand - both nodes were separated by many hops, the metric should yield a big distance for the addresses. Geographic addressing [20, 21] and the Euclidean distance e.g. satisfy these requirements but do not support mobility. Therefore, we have to research other suitable addressing schemes first.

Convergence of the Approach

It has been shown that Q-Learning will converge to an optimal policy if certain conditions are met. One requirement is that the whole decision process must be Markovian and fully observable. However, we apply SARSA-Learning although we are dealing with a Markovian decision process that is only partial observable (POMDP). Additionally, our state space is not discrete. Therefore, the convergence to the optimal policy can not be guaranteed. Nevertheless, in section 4 we show that our approach at least converges to a routing policy that performs better than classical approaches.

Analysis of the Runtime and Learning Complexity

One advantage of our reinforcement learning routing approach is that it is computationally inexpensive and does not require extensive router hardware. The whole algorithm is distributed over all routers of the network. For a connection request each router merely has to look up some Q-values, select an action and adapt the Q-values afterwards. The adaption according to the learning rule is cheap. The look-up of the Q-values is more expensive since it includes finding the best-matching neuron in the GNG which usually is done by a *nearest neighbor search*. For a very high dimensional state space an *approximate nearest neighbor*

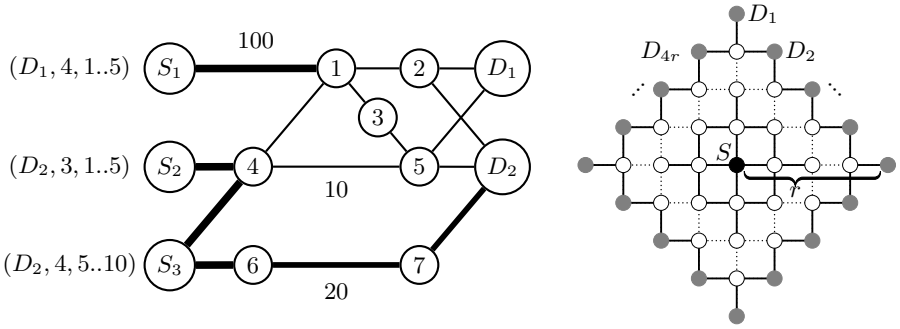
search based on *locality sensitive hashing* is suitable. Using this method a query time of $O(dN^{O(1)})$ [22] can be achieved, where N is the number of neurons in the GNG and $d = 3 + m$ is the dimension of the state space of a router with m outgoing links as described earlier. Obviously, the runtime does neither depend on the number of routers nor the amount of links, therefore the runtime complexity of each routing decision is $O(1)$ in terms of the network size.

Unfortunately, the biggest problem of reinforcement learning based approaches is the required time for learning a feasible routing policy if they are applied to a completely new network without any prior knowledge. As stated earlier the routing will be random during the bootstrapping at the beginning and many requests will not reach their destination, which results in a high blocking rate. Depending on the network layout the number of possible paths between a source-destination pair that have to be learned by each RL-agent usually increases with the square of the distance between both nodes. Therefore, the learning complexity and time for bootstrapping increases according to $O(r^2)$ where r is the average number of hops between all source-destination pairs. However, in practice the size of networks increases incrementally by adding few new nodes or links only and hence the learning complexity will be smaller.

4 Results

We have implemented our QoS-routing approach and simulated the packet flow, the routing and the resource reservation using a discrete event network simulator. In our tests we have compared our RLTE approach to the often used Widest-Shortest-Path routing (WSP). As mentioned in section 1 WSP chooses the path with the largest residual bandwidth from all possible shortest paths between a source and a destination. Additionally, WSP needs information about the current load situation of the network. If a high refresh rate for updating this information is chosen, WSP will perform better but imposes a larger signaling overhead and vice versa. In the following results we have used different refresh intervals for WSP.

Since bandwidth and delay are inherently guaranteed by the routing approach, the blocking probability, i.e. the ratio of blocked and requested connections within a certain period of time, remains the most important measure of the routing performance. In the first test we have compared the blocking probability of our approach with WSP. We have used the topology shown in Fig.3a. It consists of three source nodes $S_1, \ldots, S_3$ and two destination nodes D_1, D_2. The bandwidth of the outgoing links of the source nodes is set to 100 in order to avoid bottlenecks here. The bandwidth for the two links between the nodes $6, 7$ and D_2 is set to 20, the bandwidth of the remaining links is set to 10. The delay of all links is 1. Each source node continuously sends connection requests to the destination nodes. The connection requests are shown left to each source node. S_1 for example requests connections to node D_1 with an allowed delay of 4 and a randomly chosen bandwidth between 1 and 5. The arrival and the holding time of new connections is exponentially distributed using an arrival rate of $\lambda = 1$



(a) topology for the first test (b) regular meshed topology

Fig. 3. Different topologies that were used in our experiments. The radius r of the right topology can be changed to simulate different network sizes.

and an average holding time of $\mu = 1$, i.e. on average each source node requests one connection within one time unit and a duration of one time unit.

In Fig.4 our RLTE approach is compared to WSP. The blue graph marked with +'s shows the blocking probability for RLTE with table based SARSA-learning and the red graph marked with diamonds shows RLTE with state space clustering using GNGs. In both cases the RL-agents start without any knowledge about the network and cause high blocking rates at the beginning. It is obvious that the clustering dramatically speeds up the learning and reduces the blocking probability much faster. Hence, a smaller number of learning steps and therefore less connection requests are necessary to achieve a certain blocking probability. After 100 requests and learning steps our proposed RLTE(GNG) approach performs better than WSP(1) with a refresh interval of 1 time unit (dashed green graph). Remember that the average inter arrival time of new requests also is 1 time unit, hence for WSP(1) the refreshs occur as often as connection requests and would impose a lot of signaling overhead in practice. After 4000 learning steps our approach even performs better than WSP(∞) with an infinite refresh rate, where each WSP router has access to the current global network state at any time (solid green graph). Of course an infinite refresh rate for WSP is not possible in practice. After 10000 requests we simulate a link failure between node 6 and 7. Hence the connections must be rerouted. Right after the link failure the blocking probability of our RLTE approach increases slightly more than WSP(∞) but stays below WSP(1). A few requests later our approach has adapted its routing policy and again performs better than WSP(∞). These tests reveal that our approach can achieve the goals mentioned in the introduction to this paper and performs better than WSP although it uses local information only. Moreover in comparison to WSP the learning time of RLTE is not a problem. 100 request for bootstrapping are negligible and right after the link failure RLTE still performs better than WSP(1).

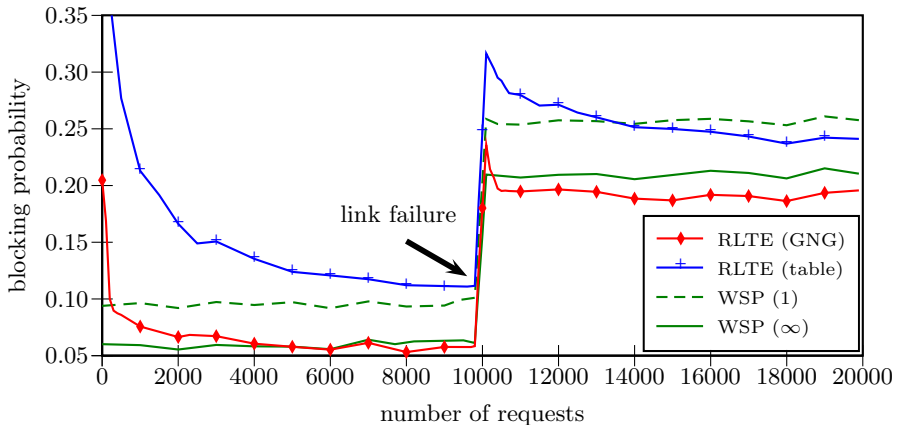


Fig. 4. Comparison of the blocking probability of our RLTE approach with WSP. State space clustering using GNGs speeds up the learning and our approach even performs better than WSP with an unlimited refresh rate. It is also able to handle link failures autonomously.

To ascertain the impact of the network size on the required learning time during the bootstrapping phase we use the meshed network topology shown in Fig.3b. The radius r of the network graph can be varied in order to change the number of nodes and links in the network. Both increase with the square of the radius. At the center of the network we placed a source node S that continuously sends connection requests to a randomly chosen destination node D_i at the periphery of the network. The bandwidth of all links is 10.0. It is the advantage of the chosen network topology that all shortest paths between the source node and the destination nodes have the same length r. Therefore, at least r routing decisions have to be made in order to establish a connection between the source S and one destination D_i. For this network we use an arrival rate of $\lambda = 1$ and an average holding time of $\mu = 2$. As QoS parameters for each connection the allowed delay is set to $r + 2$ to allow slightly longer paths than the shortest ones and the required bandwidth is again chosen randomly between 1.0 and 5.0.

In Fig.5 the number of learning epochs and requests, that are needed to achieve a blocking probability below a certain value, is plotted against the radius of the network. As reference we use the blocking probability that WSP(1) and WSP(10) with a refresh time of 1 and 10 respectively produce in the network with the given size. The solid red graph for example shows, that after 200 learning epochs RLTE(GNG) starts to perform better than WSP with a refresh time of 10 for a network with the radius $r = 4$. Again RLTE with state space clustering outperforms its table based variant. As expected at the end of the previous section the learning time increases with the square of the network size. To reduce the learning complexity we have tried a different way for initializing

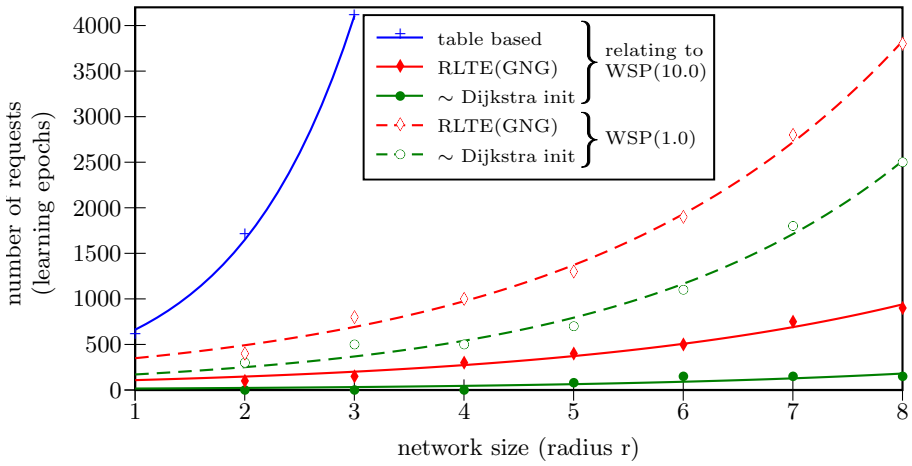


Fig. 5. Influence of the network size on the necessary time for learning. The graphs show the required learning epochs to achieve at least the same blocking probability as WSP with an refresh interval of 1 (dashed lines) and 10 (solid lines). Again state space clustering reduces the learning time (red graphs). Initialization of the Q-values using the Dijkstra algorithm leads to a further reduction (green graphs).

the Q-values. Instead of initializing the RL-agents tabula rasa, we provide them with rudimental information about the network topology. At the beginning of each simulation we apply the Dijkstra algorithm to compute the distance to each destination node and initialize the Q-values of each agent depending on these distances. The two green graphs marked with circles show that this significantly reduces the necessary time for learning.

5 Conclusion and Future Work

In this paper we have presented a novel distributed and self-organized QoS routing approach that is based on reinforcement learning. In contrast to other reinforcement learning approaches our algorithm combines optimal path planing and path selection and does not depend on predefined paths.

We have shown that our approach performs better than WSP routing although it uses local information only and therefore does not impose any additional signaling overhead. Since we use a constant learning rate to achieve life long learning our algorithm is able to react to link failures. While learning an alternative optimal routing policy a differentiation between local and global repair is no longer necessary. For the first time we have applied state space clustering in a routing approach based on reinforcement learning. We have shown that the state space clustering dramatically reduces the necessary time for learning the routing policy to an acceptable level. This is essential if reinforcement learning

approaches shall be used for routing in "real life" networks. Moreover, we have shown that the time for learning can be reduced if the RL agents are initialized using basic network knowledge obtained by applying the Dijkstra algorithm.

In the future the learning time can be further decreased if a topological addressing scheme is used and the destination addresses are included in the clustering as described in section 3. Then each agent will automatically cluster the destination addresses and build its own optimal subnets. In contrast to most other approaches our algorithm already takes two QoS constraints into account. However, it can easily be extended to handle much more parameters just by expanding the state vector. In addition to the continuous state space a continuous action space can be used. This will allow the agents to learn an optimal load sharing policy. Thus, routing approaches based on reinforcement learning have a high potential and provide many more possibilities that are worth to be investigated in future research.

References

1. Rosen, E., Viswanathan, A., Callon, R.: Multiprotocol Label Switching Architecture. IETF RFC 3031 (2001)
2. Evans, J., Filsfils, C.: Deploying IP and MPLS QoS for Multiservice Networks: Theory and Practice. Morgan Kaufmann, ISBN 0-12-370549-5 (2007)
3. Liu, Y., Tham, C., Hui, T.: MAPS: A Localized and Distributed Adaptive Path Selection Scheme in MPLS Networks. In: Proc. of IEEE Workshop on High Performance Switching and Routing (HPSR). (2003)
4. Kodialam, M.S., Lakshman, T.V.: Minimum Interference Routing with Applications to MPLS Traffic Engineering. In: INFOCOM (2). (2000) 884–893
5. Awduche, D., Malcolm, J., Agogbua, J., ODell, M., Mcmanus, J.: Requirements for Traffic Engineering Over MPLS. IETF RFC 2702 (1999)
6. Guerin, R., Williams, D., Orda, A.: QoS Routing Mechanisms and OSPF Extensions. In: Proc. of Globecom. (1997)
7. Figueiredo, G., da Fonseca, N., J.A.S.Monteiro: A minimum interference routing algorithm. In: Proc. of the IEEE Int. Conf. on Communications. Volume 4. (2004)
8. Boutaba, R., Szeto, W., Iraqi, Y.: DORA: Efficient Routing for MPLS Traffic Engineering. Journal of Network and Systems Management, Special Issue on Internet Traffic Engineering and Management **10** (2002) 309–325
9. Carrillo, L., Marzo, J., Vil, P., Fbrega, L., Guadall, C.: A Quality of Service Routing Scheme for Packet Switched Networks based on Ant Colony Behavior. In: Proc. of the Int. Symposium on Performance Evaluation of Computer and Telecommunication Systems. (2004) 637–641
10. Boyan, J.A., Littman, M.L.: Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In: Advances in Neural Information Processing Systems. Volume 6., Morgan Kaufmann Publishers, Inc. (1994) 671–678
11. Peshkin, L., Savova, V.: Reinforcement learning for adaptive routing. In: Proc. of the International Joint Conference on Neural Networks (IJCNN). (2002)
12. Heidari, F., Mannor, S., Mason, L.: Reinforcement learning-based load shared sequential routing. In: Proc. of the IFIP Networking. (2007)
13. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department (1994)

14. Watkins, C.: Learning from Delayed Rewards. PhD thesis, King's College, Cambridge University, UK. (1989)
15. Sutton, R., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
16. Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., Swallow, G.: RSVP-TE: Extensions to RSVP for LSP Tunnels. IETF RFC 3209 (2001)
17. Gross, H.M., V.Stephan, M.Krabbes: A Neural Field Approach to Topological Reinforcement Learning in Continous Action Spaces. (In: Proc. 1998 IEEE World Congress on Computational Intelligence WCCI'98) 1992–1997
18. Fritzke, B.: A Growing Neural Gas Network Learns Topologies. In Tesauro, G., Touretzky, D.S., Leen, T.K., eds.: Advances in Neural Information Processing Systems 7. MIT Press, Cambridge MA (1995) 625–632
19. Marsland, S., Shapiro, J., Nehmzow, U.: A self-organising network that grows when required. Neural Networks **15** (2002) 1041–1058
20. Watteyne, T., Auge-Blum, I., Dohler, M., Barthel, D.: Geographic Forwarding in Wireless Sensor Networks with Loose Position-Awareness. In: Personal, Indoor and Mobile Radio Communications, PIMRC. (2007) 1–5
21. Navas, J.C., Imielinski, T.: GeoCast – Geographic Addressing and Routing. In: Mobile Computing and Networking. (1997) 66–76
22. Andoni, A., Indyk, P.: Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on (2006) 459–468