Adaptive Routing with Guaranteed Delay Bounds using Safe Reinforcement Learning

Gautham Nayak Seetanadi
Department of Automatic Control
Lund University
Lund, Sweden
gautham@control.lth.se

Karl-Erik Årzén
Department of Automatic Control
Lund University
Lund, Sweden
karlerik@control.lth.se

Martina Maggio
Department of Automatic Control
Lund University
Lund, Sweden
martina@control.lth.se

ABSTRACT

Time-critical networks require strict delay bounds on the transmission time of packets from source to destination. Routes for transmissions are usually statically determined, using knowledge about worst-case transmission times between nodes. This is generally a conservative method, that guarantees transmission times but does not provide any optimization for the typical case. In real networks, the typical delays vary from those considered during static route planning. The challenge in such a scenario is to minimize the total delay from a source to a destination node, while adhering to the timing constraints. For known typical and worst-case delays, an algorithm was presented to (statically) determine the policy to be followed during the packet transmission in terms of edge choices.

In this paper we relax the assumption of knowing the typical delay, and we assume only worst-case bounds are available. We present a reinforcement learning solution to obtain optimal routing paths from a source to a destination when the typical transmission time is stochastic and unknown. Our reinforcement learning policy is based on the observation of the state-space during each packet transmission and on adaptation for future packets to congestion and unpredictable circumstances in the network. We ensure that our policy only makes safe routing decisions, thus never violating pre-determined timing constraints. We conduct experiments to evaluate the routing in a congested network and in a network where the typical delays have a large variance. Finally, we analyze the application of the algorithm to large randomly generated networks. [1] [2]

ACM Reference Format:
Gautham Nayak Seetanadi, Karl-Erik Årzén, and Martina Maggio. 2020. Adaptive Routing with Guaranteed Delay Bounds using Safe Reinforcement Learning . In *28th International Conference on Real-Time Networks and Systems (RTNS 2020), June 9–10, 2020, Paris, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3394810.3394815

[1] https://github.com/AdaptiveRouting-using-RL/AdaptiveRoutingUsingRL
[2] https://www.youtube.com/channel/UCE6pdp_0ciXB-fKcH22Mbyg/

RTNS 2020, June 9–10, 2020, Paris, France

ACM ISBN 978-1-4503-7593-1/20/06…$15.00
https://doi.org/10.1145/3394810.3394815

1 INTRODUCTION

This paper describes an application of reinforcement learning to the problem of routing in networks where each edge can be represented by a very conservative upper bound on the delay to traverse it, but the typical delay experienced when traversing edges can differ from its upper bound.

Context. A recent paper [3] introduced the problem of determining routes in graphs in which the delays across edges are characterized by both conservative upper bounds and typical values. The problem solved in the paper is to determine a route that from an initial source node i traverses edges of the graph and reaches a destination node t minimizing the delay under typical circumstances and preserving guarantees of not exceeding a total budget for the transmission, denoted with D_F. Compared to *static* routing, in which a decision on the entire route that the transmitted packet should follow, is taken prior to setting out from the source, the paper introduces *adaptive* routing, in which the decision on which edge to traverse next in the graph is taken online, based on information about the elapsed transmission time. Adaptive routing is in general capable of achieving smaller typical delays compared to static routing. This is because adaptive routing uses knowledge of the delay that was experienced across already traversed edges to aid future decisions.

The adaptive routing technique presented in [3] is based on the construction of tables that at each vertex determine which outgoing edge should be taken for intervals of experienced delays. In the presented solution, these tables are built once and then the complexity of selecting the outgoing edge from the current vertex only depends on the number of rows that each of these tables has, which in turn depends on the topology of the network and on the number of potential time intervals.

In our work, we use *safe* reinforcement learning to combine optimal path finding with safe state-space exploration. This reduces the size of the tables stored at each vertex. The de-centralized approach allows each vertex to make routing decisions irrespective of the future delays while respecting end-to-end delay constraints.

Notation. In the remainder of this paper, we use the following notation. We consider a graph G, where the vertices represent nodes and the edges represent a direct path between two nodes. We denote with $\mathcal{V}$ the set of vertices, and with $\mathcal{E}$ the set of edges of G. We use the notation $e : (x \rightarrow y)$ to indicate edge e, connecting node x with node y. Each edge is characterized by two numbers, a worst-case transmission time c^W_{xy} and a typical experienced delay c^T_{xy}, dropping the arrow for simplicity. We use c_{xyt} to denote the

minimum worst case delay from the node x to the destination t when choosing edge $(x \rightarrow y)$.

Problem Statement. Our problem is to determine a policy to route messages from a source vertex $i \in \mathcal{V}$ to a destination vertex $t \in \mathcal{V}$, to minimize (typical) transmission time for the message, and ensure that the total delay experienced by the message, τ does not exceed a specified final deadline value, D_F, $\tau \leq D_F$. We aim at solving this problem in a de-centralised way despite changes in the typical experienced delays that can happen during run-time.

Contribution. In this paper, we argue that estimates of typical delays are unlikely to be available prior to exploration of the network behavior and these typical delays are in general time-varying. To address this problem in adaptive network, we develop an algorithm based on *reinforcement learning* (RL) to automatically (and safely) perform adaptive routing. The complexity of computing the outgoing edge for each vertex only depends on the number of outgoing edges from the vertex. In general, this does not exceed the complexity of the adaptive routing technique proposed in [3], where tables could have repeated entries (the same outgoing edge can be selected for different intervals of experienced delay).

Our proposal automatically adjusts to events happening in the network (like a link suddenly becoming less reliable or congested). In this paper we show that the policy identified by the reinforcement learning algorithm exposes *stochastic convergence* to the optimal policy identified by the algorithm presented in [3] when typical delays are experienced. We also show that when typical delays are represented by probability distributions the policy learns to take the variance of the delays into account. Finally we show that our algorithm works to minimize transmission times even in large networks.

Compared to [3], we show that our work has low computational and storage complexity while improving network adaptation to time-varying typical delays. Compared to classical reinforcement learning (RL), our work guarantees safety bounds. We show that with proper domain knowledge, powerful reinforcement learning techniques can be used for time critical real-time applications.

Paper Organization. The remainder of this paper is organized as follows. In Section 2 we present our algorithm, explaining the necessary background on reinforcement learning techniques and our choices. Section 3 discusses an experimental evaluation showing the shortcomings of previous contributions when typical delays are not fixed and how our algorithm overcomes them. We also show that the identified policy in our case stochastically converges to the optimal routing technique proposed in [3]. Section 4 discusses related work and Section 5 concludes the paper.

2 ALGORITHM

As in previous work [3], we distinguish between a *pre-processing* phase and a *run-time* phase. The pre-processing phase is used for the definition of the safe bounds, i.e., to ensure that information about the worst-case paths is propagated to each node. The run-time phase is used for the execution of the reinforcement learning algorithm [16]. During run-time the system explores safe routes in order to build and update a policy that determines the best action to take depending on the currently experienced packet delay.

2.1 Pre-processing phase

The aim of the pre-processing phase is to determine the possible worst-case delays experienced by a packet in the network. More precisely, given an edge $e : (x \rightarrow y)$, we need to compute a safe bound for the worst-case delay to the destination t experienced by a packet that exits x via e. This can be done simply by determining the minimum worst-case delay to the destination from vertex y, c_{yt}, and then adding the worst-case delay of the edge e, c^W_{xy} . Assuming that the worst-case delays are positive numbers, we can simply use Dijkstra's shortest path algorithm [6, 12] to determine these values. This gives us the delay bound for which transmission can be guaranteed for each edge.

Figure 1 shows an example[3] of a network and the corresponding generated state space $\mathcal{S}$. On the top left side, we show the network and the delays over each edge. The typical delays, c^T_{xy} are denoted in blue and the the worst-case delays, c^W_{xy} are shown in red. Below that we show the network computed by the application of the Dijkstra's shortest path algorithm for each edge. The worst-case delay to the destination t over each edge, c_{xyt} is shown in green. On the right side, we show the state space. For clarity, the figure does not display the edges between nodes. The state space is explored in detail in the following subsection.

We argue that this pre-processing phase is necessary regardless of the choice of run-time algorithm, to determine safe bounds for the network and avoid choosing an edge that may lead to violation of the worst-case delay constraint.

2.2 Run-time phase

During the run-time phase, a policy to select a path to send a packet from origin i to destination t is determined. This is accomplished using reinforcement learning. In particular, we model our problem using a Markov Decision Process (MDP).

An MDP is a 4-tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ is a set of finite states, $\mathcal{A}$ is a set of actions, $P : (s, a, s') \rightarrow \{p \in \mathbb{R} \mid 0 \leq p \leq 1\}$ is a function that encodes the probability of transitioning from state s to state s' as a result of an action a, and $R : (s, a, s') \rightarrow \mathbb{N}$ is a function that encodes the reward received when the choice of action a determines a transition from state s to state s'. We use actions to encode the selection of an outgoing edge from a vertex.

State Space. In our state, we encode both the current vertex we are in and the elapsed time from the beginning of the packet transmission (in time units). The set of possible states $\mathcal{S}$ is then the Cartesian product between the set of vertices $\mathcal{V}$ and the set of natural numbers that are less than the deadline D. Denoting with N the set of integer values that satisfy the deadline constraint, i.e., $N = \{n \in \mathbb{N} \mid 0 \leq n \leq D\}$ then the set of possible states is

$$\mathcal{S} = \mathcal{V} \times N = \{(v, n) \mid v \in \mathcal{V} \wedge n \in N\}.$$

We use the compact notation v_n to denote the state (v, n).

In the graph from figure 1, the initial source node i is connected to the destination node t with an edge having typical transmission time 12 (in blue) and worst-case transmission time 25 (in red). Alternatively, node i is connected to node x with a typical transmission

[3]The example is based on the presentation of [3], where it was used to illustrate the problem.

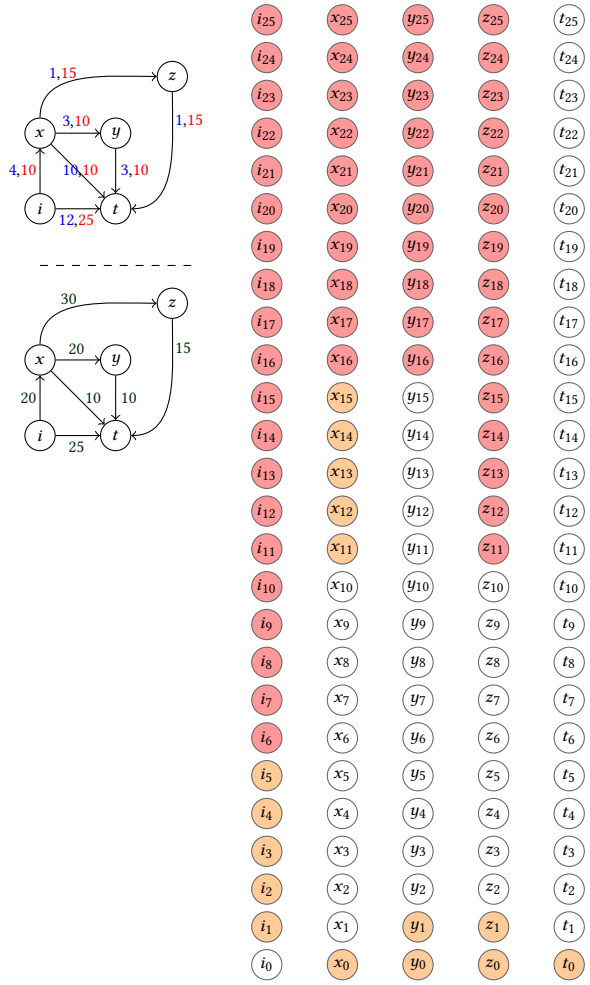


Figure 1: Example of graph and corresponding state space for the reinforcement learning problem formulation.

time of 4 time units and a maximum delay of 10 time units. The maximum admissible delay D_F is 25 time units.

On top of the state space, we want to enforce the constraint that we only explore safe paths, i.e., paths that cannot lead to a violation of the worst-case transmission time requirement D_F. For example, in node x choosing the edge $(x \rightarrow t)$ leads us to the destination node in 10 time units in the worst case, choosing the edge $(x \rightarrow y)$ leads us to the destination in 20 time units in the worst case, and choosing $(x \rightarrow z)$ has a worst case delay of 30 time units. Therefore, it is not safe in the example to be in state x_{20}, as there is no path to the destination that allows us to *certainly* reach t in less than D_F time units. We mark the unsafe states in red.

Finally, not all the nodes of the state space $\mathcal{S}$ are reachable. For example, node t_0 is clearly not reachable as there is no way to cross the path from the source node to the destination node in zero time. However, node t_1 is potentially reachable from the source when the edge $(i \rightarrow t)$ is chosen, if the experienced delay is one time unit. Furthermore, we assume that the packets are not stalled, which means for example that node i_1 is not reachable. We mark the unreachable nodes with orange in Figure 1.

There are still some edges that we should eliminate among the choices. For example, if we are in state x_{10}, with $D_F = 25$, we need to impede the choice of the edge $(x \rightarrow y)$, as the worst case delay for the single possible path is higher than the 15 time units that are left. We do this by limiting the actions that the policy can choose in each state.

Reinforcement Learning. Once we limit the set of possible actions to ensure we will meet the worst case deadline, we use a reinforcement learning policy to decide which outgoing edge – action $a \in A$ – to take from a state $s = v_n$. Generally speaking, decisions based on reinforcement learning are constructed using feedback to *reward* successful actions taken to explore the state space [16]. This decision making process, or *policy*, leads to a formal definition of the value of being in the current state and how to select the action that is taken in each state. The goal of this decision making process is to take optimal actions so as to maximize the reward obtained. Using the reinforcement learning terminology, we denote the transmission of a message from source to destination with the word *episode*. We also define the reward received in each state as the total amount of time saved while traversing the path to the destination. This reward is calculated at the end of each episode and then propagated to the other nodes.

TD Learning. A popular reinforcement algorithm is the Temporal-Difference (TD) learning [16] algorithm. TD learning gained popularity in TD-Gammon, a program that learned to play backgammon at the level of expert human players [18].

It is a model-free learning method which learns by sampling the environment and determines an estimate of how good it is for the algorithm to perform an action from a state, i.e., a value function taking action a in state s, usually identified as $Q(s, a)$. This learning method is coupled with an exploration algorithm, which in our case is the ε-greedy algorithm. Similar to Monte-Carlo (MC) methods, TD learning samples the environment and learns from it, by updating the value function. While MC methods update the value $Q(s)$ of each state s in the current episode at the conclusion of the episode, TD learning adjusts its estimate to match later predictions about the future before the final outcome is known.

Using MC would be impractical for our algorithm as it would generate a lot of messages in the network to transmit reward information at the conclusion of the message transmission. On the contrary, the one step TD algorithm allows us to make routing decision and value iteration based only on $Q(s, a)$ and $Q(s', a')$, i.e., the Q-value for the current and next state. The next state Q-value can be appended to the acknowledgement of the reception of a message, therefore making this choice practical from an implementation standpoint. The value update policy is computed as

$$Q(s, a) = Q(s, a) + \alpha \cdot (\mathcal{R} + \max(\gamma\, Q(s', a')) - Q(s, a)) \quad (1)$$

where s and s' are the current and next states respectively, a' is the action with the highest Q-value for state s', α is the learning rate with which we overwrite old information with new, $\mathcal{R}$ is the reward obtained during the transition from state s to state s', and γ is the discount factor that captures uncertainty of $Q(s', a')$.

Exploration Policy. ε-greedy exploration ensures that the system chooses the edge that was identified as best for the transmission of most packets while at the same time exploring other edges in search of a path that has a higher reward. The ε-greedy policy chooses one action from the vector of feasible actions $\mathcal{A}$. Specifically this action a is either the one that has the maximum value of $Q(s, a)$ or a random action that explores the state space. The policy explores the state space with a probability ε and take the action with the maximum estimated reward with a probability of $(1-\varepsilon)$. In principle, ε is chosen to be a small number, such that most of the time the policy exploits knowledge about the current state. The continuous exploration of the state space, ensures (in a probabilistic sense) the detection of paths with higher rewards and resilience to changes.

In order to guarantee that the total transmission time never violates the set deadline D_F, we modify the ε-greedy policy to perform *safe* reinforcement learning. Safe reinforcement learning is the process of learning actions that maximize the rewards while ensuring reasonable system performance and/or respect safety constraint [9]. Safe reinforcement learning can be broadly divided into two categories:

- *Modified Optimization Criterion*: In this case, the concept of risk is introduced into the optimization process. The policy is determined with explicit knowledge about the risk involved in taking specific actions.
- *Safe Exploration Process*: In this second case, the exploration process is modified to avoid exploratory actions that could have harmful consequences. This is the alternative taken in this work, where we impede certain actions from being taken.

In previous work, this knowledge has been used mostly in scenarios where a human teacher demonstrates a safe path [7, 15]. In this paper, the modified ε-greedy policy ensures a safe exploration process through incorporating external knowledge obtained in the pre-processing phase. The exploration policy then allows for small explorations that deviate from the optimal policy.

A popular variation of this exploration algorithm is the decaying ε-greedy. The decaying algorithm reduces ϵ, thus minimizing the exploration during routing of later packets. This reduces the delay over the network as most of the exploration is required during the transmission of the first few packets. Once the value of most of state action pairs $Q(s, a)$ of the MDP are known, the need for exploration reduces. Tuning the decay function according to the size of the network is considered out of scope of this paper.

The paper investigates both exploration policies in Section 3. We use the decaying ε-greedy policy in Experiment 3.1 as a static network is considered. In the following experiments, we use the non-decay ε-greedy due to probabilistic distributions of delays and dynamic networks with congestion.

2.3 Algorithm

Algorithms 1, 2 and 3 show the pseudo-code of our algorithm. Algorithm 1 is executed globally to obtain worst-case transmission times c_{it} during network initialization and reconfiguration(node addition or removal). Algorithm 2 executed at the node can either be run globally or at a local level depending upon the network configuration. Monte-Carlo methods require knowledge of the entire state-space and thus are practical for small networks. Algorithm 2 describes one step TD-learning making routing decision and value iteration based only on $Q(s, a)$ and $Q(s', a')$, i.e., the Q-value for the current and next state allowing for distributed decision making.

Algorithm 1 Pre-Processing:

```
for each node u do
    for each edge (u → v) do
        // Delay bounds as described in Section 2.1
        c_uvt = c^W_uv + min(c_v_t)
        // Initialise the Q values to 0
        Q(u, v) = 0
```

Consider the routing problem from source i to destination t. As described above, the algorithm is split into pre-processing and runtime phases. The pre-processing phase calculates the shortest path to the destination using Dijkstra's algorithm [6]. The minimum worst case transmission time c_{it} to the destination t over the edge $(i \rightarrow x)$ is obtained by adding the worst case transmission time over the edge c^W_{ix} and the minimum worst case transmission time c_{x_t} from v to destination t over all outgoing edges of x. The values of all state-action pairs are initialized to 0.

Algorithm 2 is run at each node u when a packet arrives. If u = source node, the deadline $D_u = D_F$ is set to the final deadline. For the other nodes, the deadline is determined online. For each edge $(u \rightarrow v)$ from u, if $c_{uvt} > D_u$, the edge is infeasible and $P(u|v) = 0$. This ensures that deadlines are never violated. From the feasible edges ($\mathcal{F}$), the edge that corresponds to the action with maximum value, v such that $Q(u, v*) = \max(Q(u, v \in \mathcal{F}))$ has the highest probability $P(u|v*) = (1 - \epsilon)$. The remaining feasible edges are assigned $P(u|v \setminus \{v*\}) = \epsilon/(\text{size}(\mathcal{F} - 1))$. The next node is decided according to the probability P. The actual transmission time over the edge is δ_{ux}. This is subtracted from the deadline for the node D_u to obtain D_x, the deadline for the next node x. No reward is awarded in the middle of the episode and the value of the state action pair, $Q(u, v)$ is calculated using (1).

The reward R after each edge traversal is obtained as shown in Algorithm 3. If $v \neq t$, the destination is not reached and the episode continues by taking the node v as the current node. If $v = t$, the episode is complete. The reward, R is calculated as $R = D_F - \delta_{it}$ where D_F is the pre-determined deadline and δ_{it} is the total transmission delay over the whole path from source i to destination t.

The back-propagation of calculated reward is inherent in the value iteration as shown in Equation 1. As seen, the value iteration is dependent on the maximum Q-value of the next node. This can be sent to the sending node after each successful packet transmission depending on the protocol used. For example, If Transmission control protocol (TCP) is used, the acknowledgement messages could be extended to include the Q-value. In our implementation, we simply transmit the value back to the origin node.

2.4 An Example Episode

Using the same example as above with deadline $D_F = 25$, we begin the episode at node i. The feasible available edges are $[(i \rightarrow x), (i \rightarrow t)]$ as they both satisfy the constraint $c_{it} \leq D_i$ where D_i

Algorithm 2 Node Logic (u)

for Every packet **do**
  if u = source node i **then**
    $D_u = D_F$ // Initialise the deadline
    $\delta_{it} = 0$ // Initialise total delay for packet = 0
  for each edge $(u \rightarrow v)$ **do**
    if $c_{uvt} > D_u$ **then** // Edge is infeasible
      $P(u|v) = 0$
    else if $Q(u, v) = max(Q(u, a \in A))$ **then**
      $P(u|v) = (1 - \epsilon)$
    else
      $P(u|v) = \epsilon/(size(\mathcal{F} - 1))$
  Choose edge $(u \rightarrow v)$ with P
  Observe δ_{uv}
  δ_{it} += δ_{uv}
  $D_v = D_u - \delta_{uv}$
  R = Environment Reward Function(v, δ_{it})
  $Q(u, v)$ = Value iteration from Equation (1)
  if $v = t$ **then**
    DONE

Algorithm 3 Environment Reward Function(v, δ_{it})

Assigns the reward at the end of transmission
if $v = t$ **then**
  $R = D_F - \delta_{it}$
else
  $R = 0$

is the deadline at node i. As this is the first episode, there exists no information of the value of the two edges. Both edges have the same probability and one of them is chosen at random. If the edge chosen is $(i \rightarrow x)$ and the observed transmission time is $\delta_{ix} = 4$ then the new deadline D_x is $D_i - \delta_{ix} = 21$. The feasible edges from node x are $[(x \rightarrow y), (x \rightarrow t)]$. The value of the state action pair $Q(i, x)$ is calculated using Equation (1). As the value at the next node x is 0 due to no prior information, another random selection of the edge is made. If $(x \rightarrow t)$ is traversed with $\delta_{xt} = 8$, the destination node is reached and the episode is terminated. The reward for the state is calculated as $R = D_F - \delta_{it}$ where δ_{it} is the total transmission time of the packet. During the following transmissions, the algorithm makes use of this state-action value function to make a more informed decision. As we use ϵ-greedy algorithm for exploration, all state-action pairs are eventually explored and their value is calculated. This combination of TD learning and safe exploration leads to small transmission times while respecting time constraints.

3 EVALUATION

In this section, we will discuss the behavior and performance of the algorithm presented in Section 2, by applying it to the example shown in Figure 1 in different network conditions and to large-scale networks.

We build the network using Python and the NetworkX [10] package. NetworkX allows us to build Directed Acyclic Graphs (DAGs) and to provide information about nodes and edges of those DAGs. In building the example of Figure 1, we annotate the graph using for each edge $(x \rightarrow y)$ the worst case delay as a weight. We also include the typical delay c^T_{xy} for each edge as an annotation in the graph, but we hide this information from the algorithm and only use it to determine the edge traversing time. The pre-processing phase calculates the worst case delay from each node to the destination. This initial phase is executed once during the creation of the network. Once the network is created, our reinforcement algorithm is executed for every packet.

The experiment presented in Section 3.1 shows a comparison between the results obtained with our algorithm and the optimal choices. It highlights that the transmission times experienced using our algorithm converge to the optimal numbers obtained with the technique presented in [3].

However, the algorithm presented in [3] does not adapt to online changes. We show how our algorithm behaves in this case in the experiment presented in Section 3.2. In this case, a link gets congested, and its traversal time then becomes equal to the worst case. We show how our algorithm is able to dynamically adapt and converge to a new optimal policy.

However, the real contribution of this paper is shown when edge traversal times are unknown and vary over time. We show this in the experiments presented in Section 3.3. For the rest of the experiments, we assume that the edges traversal times behave according to a given probability distribution. This is similar to a real network where the delay for a packet is varied at every time instance. In particular, we investigate both a truncated normal distribution and a uniform distribution. With truncated normal distribution we mean a probability distribution that is a normal distribution, but is cut at zero (to avoid negative traversal times) and at the worst case traversal times (to ensure that the constraint is satisfied).

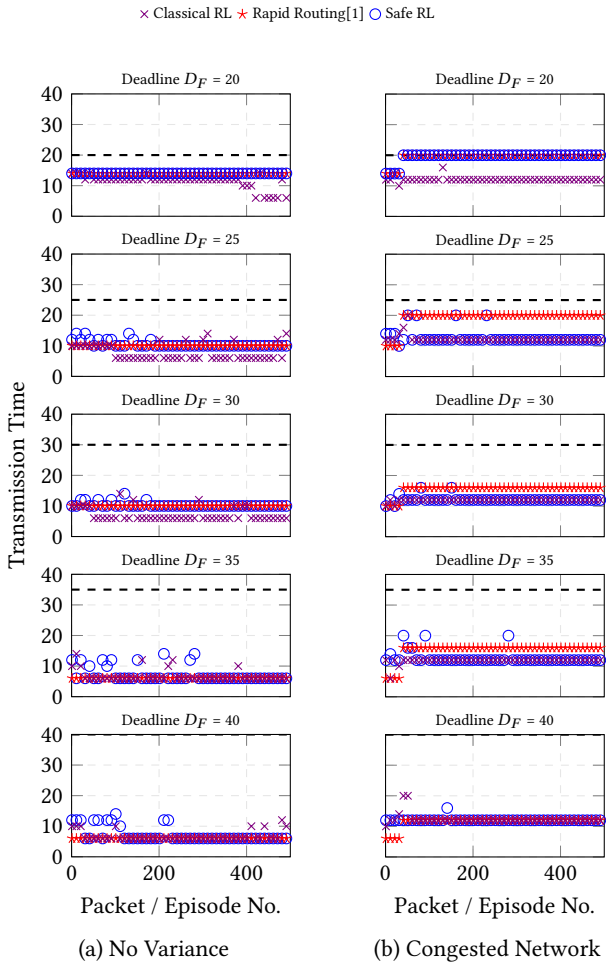
Finally the experiment presented in Section 3.5 shows the scalability of our algorithm and how it behaves when applied to an networks with an increased number of nodes.

3.1 Experiment 1: Convergence to the Optimal Route

To check that our reinforcement algorithm identifies the optimal route from the source to the destination node, we compare the length of optimal path determined by the RL algorithm after 1000 episodes, with the traversal time of the optimal path computed using the algorithm from [3]. Both algorithms are applied to the network shown in Figure 1.

The delays for traversing an edge are set to the typical delays, thus giving us the possibility to verify convergence in nominal conditions. For the RL, the delay to traverse an edge becomes available only after the destination node of the edge is reached.

Table 1 shows the results we obtained for different values of D_F. When $D_F = 15$, it is impossible to find a solution that guarantees the worst case transmission delay, and both our algorithm and the optimal one presented in [3] are able to identify that using Dijkstra's algorithm on the worst case transmission times. For the other deadlines, we show the delays obtained with the optimal algorithm and the average delay obtained in 1000 transmissions using our



(a) No Variance (b) Congested Network

Figure 2: Smoothed Total Delay for Experiments 3.1 and 3.2.

algorithm. In all cases, the delays experienced by packets using algorithm are very close to the optimal delays. The slight variations are due to the exploration that is built in our algorithm, and specifically to the exploring nature of the ϵ-greedy policy. For all the values of D_F and in all episodes, the deadline is never violated.

Figure 2(a) shows the evolution of the transmission delays as the number of packets sent increases. Rapid routing[3] and safe RL converge to the same path and experience the similar transmission times. Classical RL is consistently able to have low transmission times at the expense of taking unsafe paths. This causes deadline violations in networks with varying transmission times are seen in section 3.4.

Except for $D_F = 20$, in all the other plots there is an initial exploration phase, in which the safe RL algorithm is exploring alternative routes, to find the optimal path. There is no exploration when $D = 20$ as the only feasible path in the network is $(i \rightarrow x \rightarrow t)$. Therefore the total path delay is always equal to 14. As mentioned before, the algorithm explores new routes with a probability that decays over time, which shows how the algorithm settles for a given route in the static case.

Table 1: Optimal Path for Different Deadlines

D_F	Optimal Path	Delays [3]	Average Delays (1000 episodes)
15	Infeasible	-	-
20	{i,x,t}	14	14
25	{i,x,y,t}	10	10.24
30	{i,x,y,t}	10	10.22
35	{i,x,z,t}	6	6.64
40	{i,x,z,t}	6	6.55

3.2 Experiment 2: Adaptation to Edge Congestion

In Experiment 2 we analyze the capability of the algorithm to adapt to new circumstances. In this specific case, we see how the algorithm reacts to the congestion of one of the edges. In the network we used before, we artificially introduce congestion on the edge $(i \rightarrow x)$. The delay δ_{ix} to traverse this edge increases from 4 to 10 time units, after the transmission of 40 packets. For the rest of the experiment, this edge remains congested.

Figure 2(b) shows the transmission times for different deadlines D_F. Due to the congestion, there is a need to adapt.

Both classical and safe RL adapt to the congestion and take a different path to the destination. For $D_F = 20$, classical RL again chooses a path that violates safety guarantees. Rapid Routing is at a disadvantage as the routing tables generated in the pre-processing stage are not sufficient to ensure adaptation. The delay increases (see the values the algorithm converges to compared to the values shown in Figure 2) and the optimal path changes. The congested edge is the first edge traversed for all previously determined optimal paths, therefore the algorithm has to determine if there is a better path using exploration. Eventually, the rewards are propagated and the algorithm adapts. In all cases except for $D_F = 20$, the algorithm converges to the path $(i \rightarrow t)$ with a total delay of 12. In the case of $D_F = 20$, the only feasible path is $(i \rightarrow x \rightarrow t)$ and the total transmission time on the path increases from 14 to 20 due to the experienced congestion.

The ability to adapt to current conditions is one of the motivations for using our algorithm rather than a static adaptive choice.

3.3 Experiment 3: Probabilistic Traversal Times

In this section, we discuss the convergence of our algorithm when the delay times for transmitting over the edges $(x \rightarrow y)$ are random variables δ_{xy} drawn from probability distributions. In this experiment we model the typical delay times using truncated normal and uniform distributions. Figure 3 shows the total path delays when the traversal time for an edge is distributed as a truncated normal distribution with mean value c^T_{xy} and different variance values. Each column in the figure shows a different value of D_F and each row a different variance, from 1 to 5. We denote the used probability distribution as truncated normal, because we cut the probability distribution below 0 and above the worst case c^W_{xy} to ensure that the traversal times respect the constraints of our problem $(0 \leq \delta_{xy} \leq C^W_{xy})$.

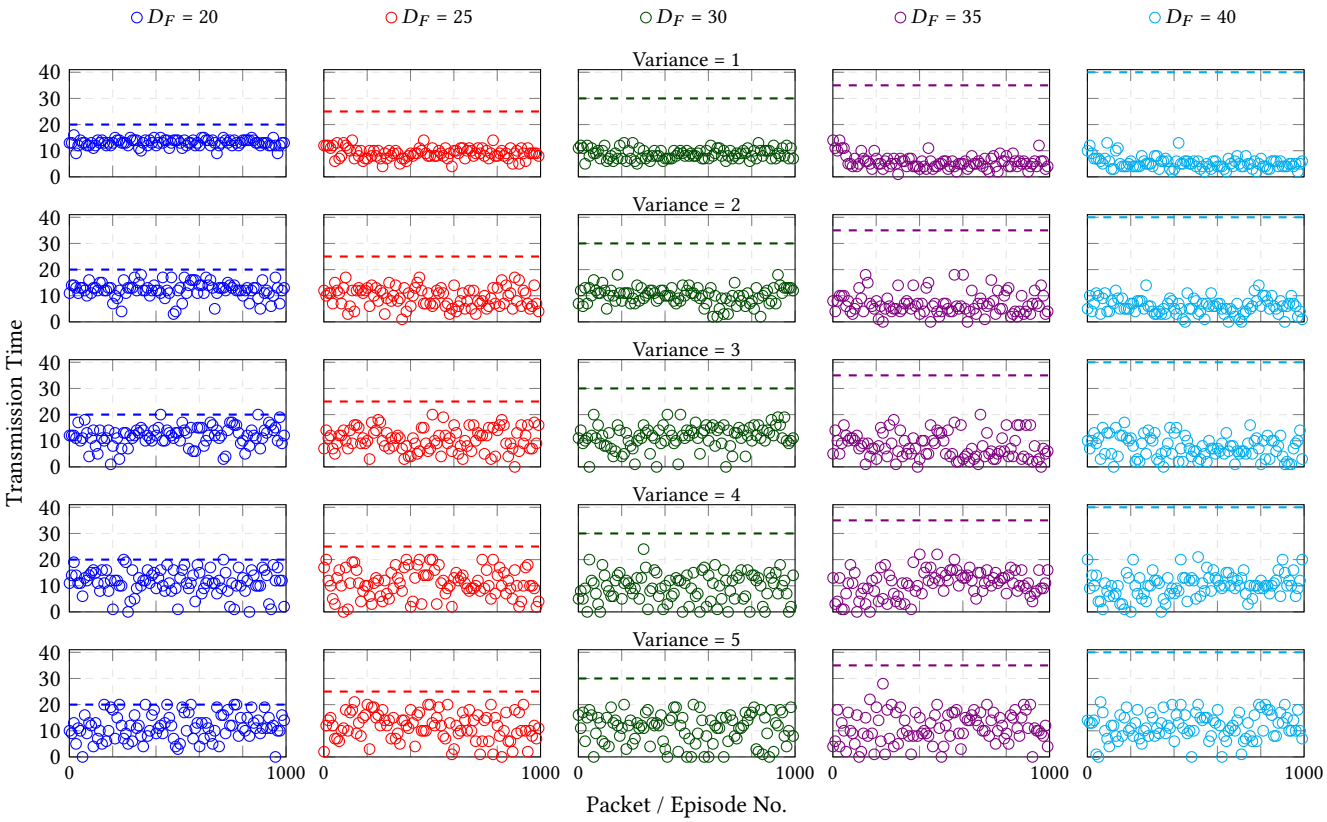


Figure 3: Transmission Time with Truncated Normally Distributed Edge Traversing Times.

Table 2: Average Delays(1000 Episodes) in Experiments 3.1, 3.2, and 3.3

D_F	No Variance	Congested Network	Truncated Normal Distribution	Uniform Distribution around c^T	Uniform Distribution $[0, c^W]$
20	14.00	19.76	[12.80, 12.26, 12.02]	[13, 12.6, 12.49]	9.14
25	10.30	12.42	[9.15, 10.20 ,10.63]	[13.01, 12.69, 12.49]	9.84
30	10.27	12.17	[8.92, 10.01, 10.23]	[8.89, 9.04, 10.62]	10.28
35	06.94	12.42	[5.81, 7.15, 8.60]	[5.51, 5.95, 6.95]	9.98
40	06.84	12.32	[6.08, 6.945, 8.556]	[5.59, 5.81, 6.42]	10.12

As the variance increases, the total delay also increases on average, as expected. One take away message is the deadline D_F is never violated, showing that the algorithm behaves correctly. Also, when the variance increases, different paths are explored as the Q values for the nodes tend to be closer to one another – rather than experiencing one best path, the variance blurs the differences between the paths and makes it more important and rewarding to distribute packets on different edges. In particular, looking at the case with $D_F = 35$, the case with variance 1 and 2 converges (primarily) to one specific path. The case with variance 3 and 4 explore different paths and reach different conclusions on the optimality of the chosen policy. The case with variance 5 tends to converge to a different optimal policy. This seems to suggest that the presence of high variance is another reason to use an adaptive policy rather than an optimal pre-determined choice.

Similarly Figure 4 shows the results we obtain when δ_{xy} is drawn from a uniform distribution. Specifically, for each edge $(x \rightarrow y)$, $(0 \leq \delta_{xy} \leq c^W_{xy})$ is enforced and the extremes of the uniform distribution are chosen to be centered about the typical delay value with a varying interval length. Similar to the truncated normal distribution case, the network adapts and our algorithm ensures that the deadlines are never violated. Compared to the truncated normally distributed case, the uniform distribution seems to have a better effect on finding optimal routes and sticking to these routes.

3.4 Experiment 4: Uniformly Distributed Worst Case Traversal Times

We perform an experiment with a uniform distribution in which we select the extremes of the distribution as 0 and the worst case

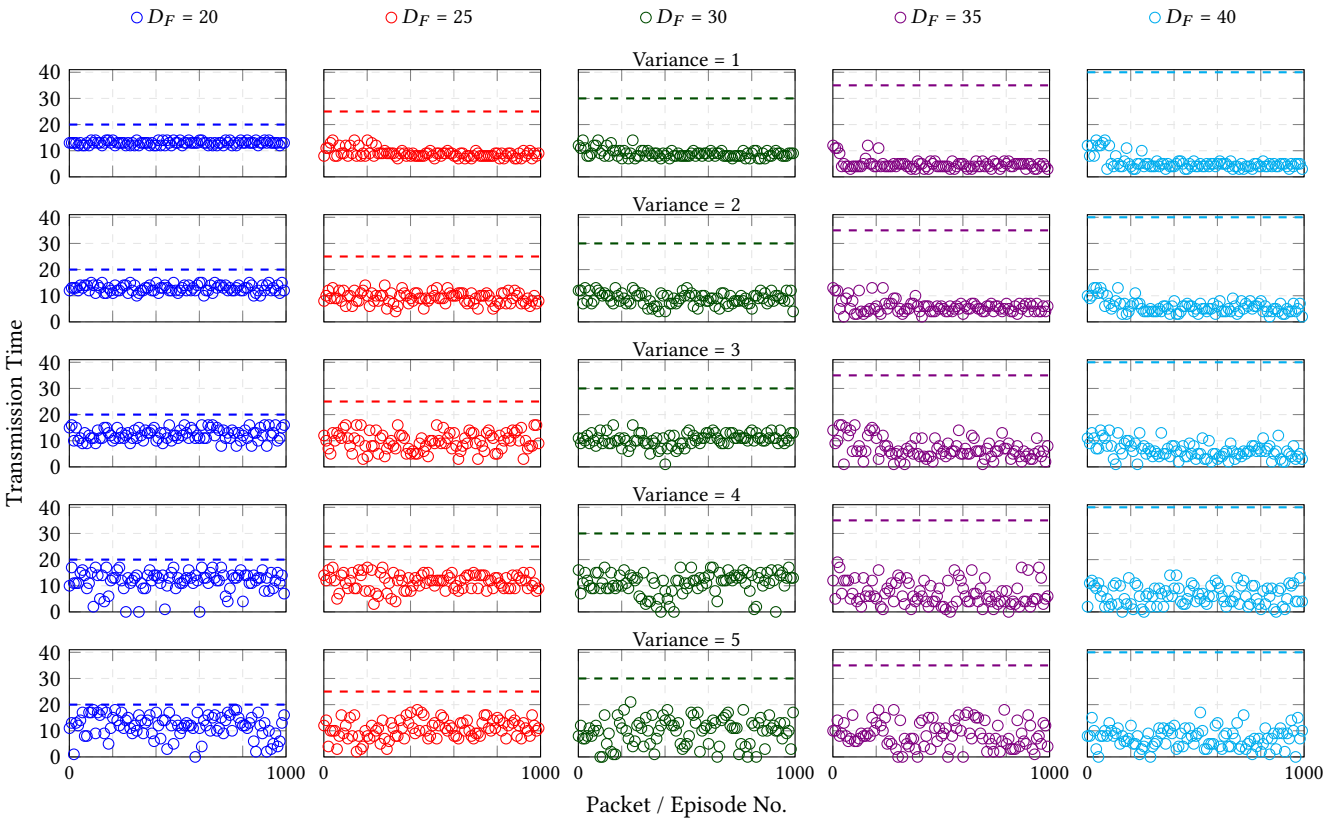


Figure 4: Transmission Time with Uniformly Distributed Edge Traversing Times.

transmission delay (completely disregarding the information about the typical transmission time). In this case, the interval length creates much more variation in the typical traversal times. The results for this run are shown in Figure 5.

We compare the transmission times obtained using our safe reinforcement learning approach to the ones using classical RL and the rapid routing algorithm from [3]. Classical RL algorithms are only concerned with maximizing the obtained reward, thus lead to deadline violations as seen in the figure. The violations occur due to the exploration of unsafe edges. The algorithm from[3] does not violate deadlines but the typical transmission times are higher compared to our safe reinforcement learning approach. This is because the routing tables are built only once during network creation. The routing is not adaptive to the changing environment and routes messages according to the pre-built routing tables. One way of compensating would be to rebuild the routing tables for [3] for every packet. This would be highly compute intensive and impractical as seen in experiment 3.5

Our algorithm is shown to work in meeting the deadlines D_F and also discovering new potential paths as the traversal times over the initial links in the path vary. Table 2 summarizes the delays experienced in the network for the three algorithms.

3.5 Experiment 5: Large Networks

In this last set of experiments we investigate the scalability of our algorithm. We create random networks, with a large number of nodes n. We ensure the network includes one edge $(0 \rightarrow 1)$ from node 0, the initial node, and one edge $(n-2 \rightarrow n-1)$ that reaches the final node $n-1$. We randomize all the other edges present in the network. We only add edges from a node with a lower index node to a higher one while ensuring no loops are created in the networks. Generally the networks generated consist of nodes with a large number of outgoing edges.

For every edge in the network, $(x \rightarrow y)$, we randomly extract a value for the typical delay $c^T_{xy} \in (0, 10]$ and for the worst case delay $c^W_{xy} \in [10, 30]$. We ensure that there exists a path from every node x to the destination node n. Networks that do not satisfy the condition are discarded.

The pre-processing phase is applied as described in Section 2 to all networks to get the shortest guaranteed total delay to the destination node n. The deadline for the randomly generated networks is chosen to be the $1.2 \cdot c_{it}$ from the initial node i to the destination node t.

Figure 6 shows the average delay time and the deadline set for increasing number of nodes in the randomly generated network. We send 1000 packets over the network from source to destination and record the total delay. The algorithm works well finding

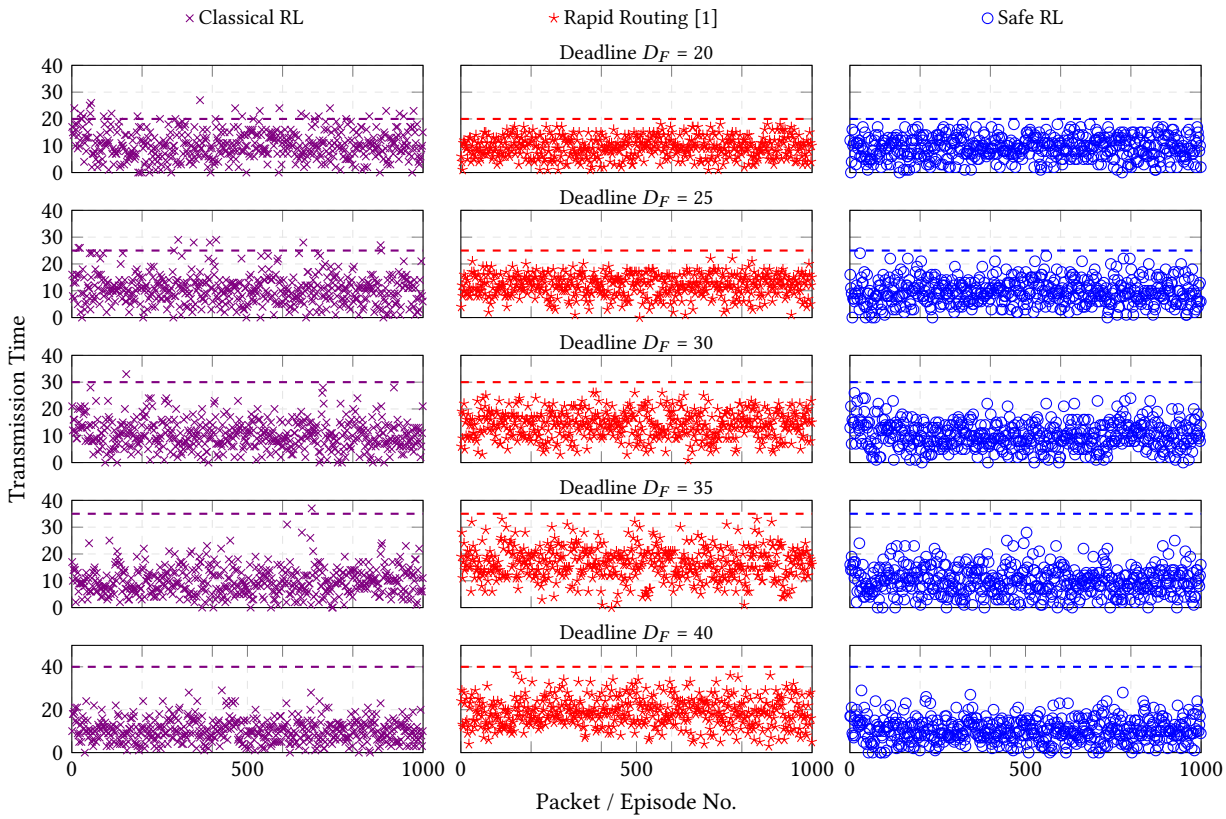


Figure 5: Transmission Time with Uniformly Distributed Edge Traversing Times for Large Intervals.

Table 3: Average Delays(1000 Episodes) in Section 3.4

D_F	Classical RL	Rapid Routing	Safe RL
20	10.32	9.609	9.135
25	10.19	11.98	9.841
30	10.10	14.10	10.283
35	10.23	16.43	9.99
40	9.98	18.593	10.193

paths to the destination minimizing total delays while ensuring no deadline violations, even for large networks. Classical RL has higher transmission times generally because of the large number of outgoing edges from each node and deadlines are violated during exploration. Rapid routing [3] has low transmission times due to the routing tables built in the pre-processing stage. However this method has very high computational complexity and the routing tables have to be recalculated for every change in δ.

Figure 7 shows the computational time for the transmission of 1000 packets over the networks. We compare the performance of classical RL and rapid routing algorithm from [3] with our safe reinforcement learning. Classical RL is the least computationally complex as it has no pre-processing stage. Our safe learning approach is a magnitude less computationally intense compared to rapid routing. In safe RL the pre-processing has to be only run once during network creation. In case of node addition, the recalculation is minimal. Comparatively, rapid routing has large routing tables have to be constructed and evaluated as described in [3].

4 RELATED WORK

In this section, we discuss prior research related to our work.

The problem of optimal paths in a stochastic network has been studied previously in [14], [11], and [4]. These prior works consider the stochastic and dynamic shortest distance problems in a weighted network and minimize a cost function to obtain the optimal path. The weights in this case can be considered to be equal to the delays in the network. Our work builds on these by guaranteeing an upper bound on the delay while minimizing the weight. We also assume no prior knowledge on the distributions and also take into account the dynamic nature of real networks.

Networking systems are of great interest for the application of machine learning algorithms [13]. Actor-critic method [20] and value function methods based on gain scheduling method [5] have investing in depth on using modern computing advances for better adaptive routing. Similar to these works, we also use reinforcement learning but perform safe exploration to respect delay constraints.

Using external knowledge for safe reinforcement learning is a popular method as it enhances learning by using prior information (derived from human supervisor or otherwise) and prohibits exploration of unsafe paths. This can be broadly distinguished into

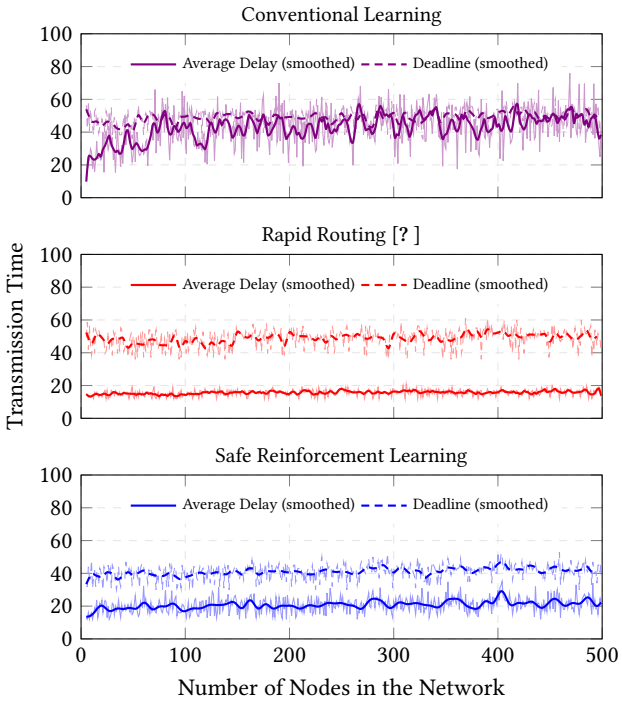


Figure 6: Delays for increasing nodes in network.

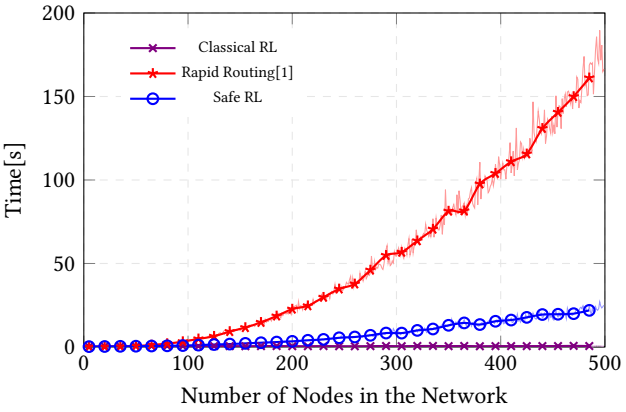


Figure 7: Computational times for increasing the number of nodes in the network.

three methods [9]. *Providing initial knowledge* can mitigate exploration problems using bootstrapping as shown in [7]. A similar methodology of restricting exploration space is to have a *finite set of demonstrations* to discover the state space. These methods have been applied mainly to physical systems as shown in [17] and [2]. Finally *Providing advice* uses a teacher to assist during the learning process. This can either be a teacher offering advice when the learner considers necessary as shown in [1] and [8]. Similarly, the teacher can offer advice whenever it deems necessary as shown in [21], [19]. These methods are not very effective in our problem as we consider a decentralized approach with each node making independent decisions.

5 CONCLUSION

In this paper we have applied reinforcement learning to the problem of routing over real-time networks. To guarantee packet transmission within a pre-determined timing constraint, we augment the exploration of the state-space to only explore paths that are safe. This allows for small delays over the network. The constant evaluation of the state-space and exploration of new paths make the algorithm resilient to changes in the network due to, e.g., congestion, link failures. This also allows us to route packets over new paths which might not be realized to be safe in the case of static routing. The decentralized approach used here allows each node to make routing decisions based only on the current observed packet delay and the value function of the next node reducing the amount of information needed at each node.

We verified the stochastic convergence of the algorithm to the optimal path and performed experiments to verify the resilience of the reinforcement learning algorithm. Compared to classical RL we show that our algorithm is robust and never violates set deadlines. While compared to previous research, our algorithm is shown to be more adaptive to transmission time variations while having reduced computational complexity.

REFERENCES

[1] J. A. Clouse and P. E. Utgoff. A teaching method for reinforcement learning. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 92–110, 12 1992.
[2] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *Int. J. Rob. Res.*, 29(13):1608–1639, Nov. 2010.
[3] S. Baruah. Rapid routing with guaranteed delay bounds. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 13–22, December 2018.
[4] D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16(3):580–595, Aug. 1991.
[5] J. Carlström. Decomposition of reinforcement learning for admission control of self-similar call arrival processes. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, pages 989–995. MIT Press, 2000.
[6] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec 1959.
[7] K. Driessens and S. Džeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, Dec 2004.
[8] J. Garcia and F. Fernandez. Safe exploration of state and action spaces in reinforcement learning. *CoRR*, abs/1402.0560, 2014.
[9] J. García and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal on Machine Learning Research*, 16(1):1437–1480, Jan. 2015.
[10] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.
[11] R. P. Loui. Optimal paths in graphs with stochastic or multidimensional weights. *Commun. ACM*, 26(9):670–676, Sept. 1983.
[12] K. Mehlhorn and P. Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer Publishing Company, Incorporated, 1 edition, 2008.
[13] L. Peshkin and V. Savova. Reinforcement learning for adaptive routing. *CoRR*, abs/cs/0703138, 2007.
[14] G. H. Polychronopoulos. *Stochastic and Dynamic Shortest Distance Problems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1992.
[15] W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 903–910, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
[16] R. S. Sutton and A. G. Barto. *Reinforcement learning: An Introduction*. Adaptive computation and machine learning. MIT Press, 2018.
[17] J. Tang, A. Singh, N. Goehausen, and P. Abbeel. Parameterized maneuver learning for autonomous helicopter flight. In *2010 IEEE International Conference on Robotics and Automation*, pages 1142–1148, May 2010.

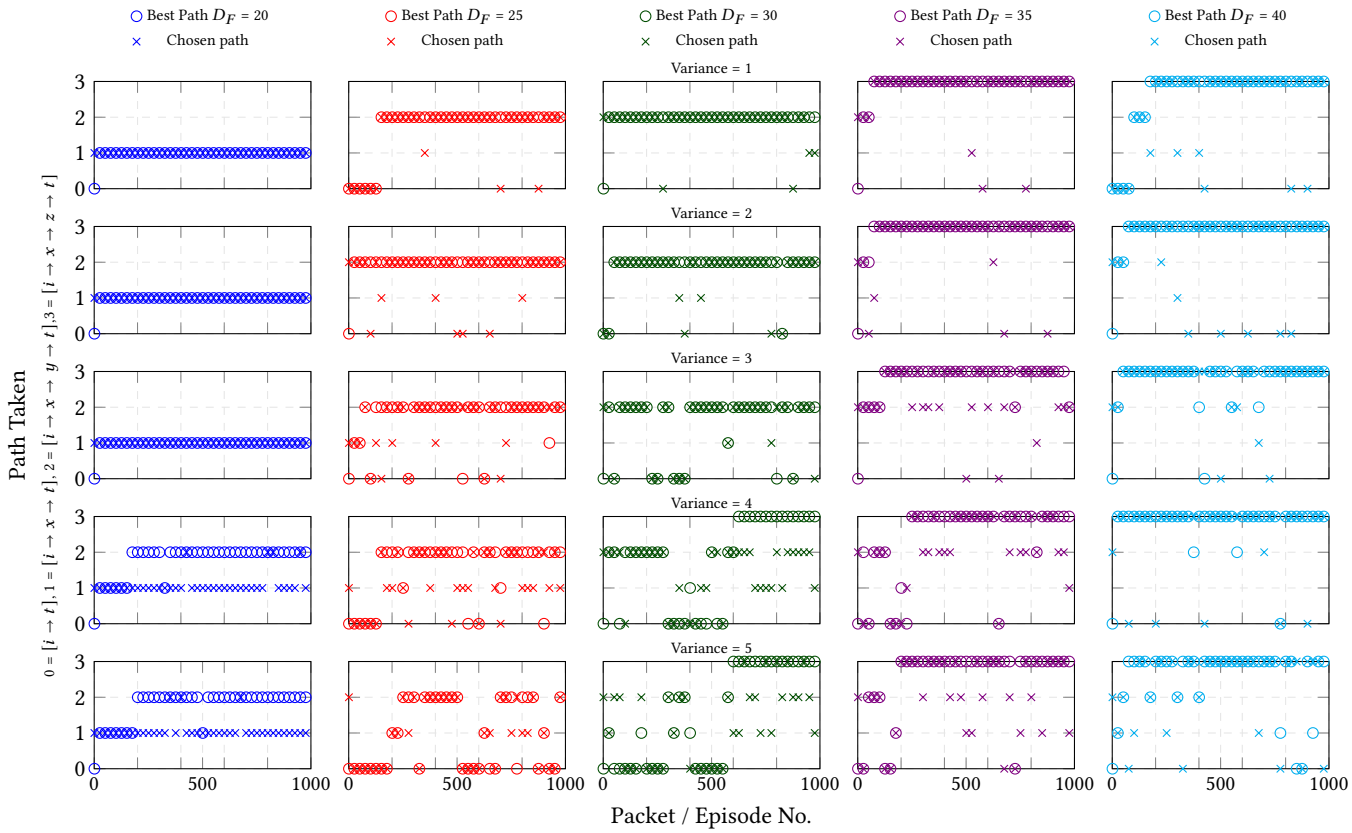


Figure 8: Best Path and Chosen Path with Uniformly Distributed Edge Traversing Times.

[18] G. Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, Mar. 1995.

[19] A. L. Thomaz and C. Breazeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 1000–1005. AAAI Press, 2006.

[20] H. Tong and T. X. Brown. Reinforcement learning for call admission control and routing under quality of service constraints in multimedia networks. *Machine Learning*, 49(2):111–139, Nov 2002.

[21] P. Q. Vidal, R. I. Rodríguez, M. Ángel Rodríguez González, and C. V. Regueiro. Learning on real robots from experience and simple user feedback. *Journal of Physical Agents*, 7(1):57–65, 2013.

A ROUTING PATH ANALYSIS

This appendix contains more experiments that analyse the paths taken by the algorithm.

Figure 8 shows the best path determined by the algorithm and the path taken during the transmission of packets through the network for different deadlines and for increasing uniform variance.

The first column shows the best and chosen paths when $D_F = 20$. For low variances, the only possible path is $\{i, x, t\}$. This ensures that the deadline is not violated. For higher variances in the transmission times, new paths are available for transmission of the packets. For Variance = 4, new path $\{i, x, y, t\}$ is feasible and is explored around packet number 200. The algorithm determines that this is the best path ($\{c^T_{xy} + c^T_{yt}\} \lessapprox c^T_{xt}$) for transmission. Even though the algorithm discovers a better path, it is not always feasible due to the variance in the transmission times. The path $(x \rightarrow y)$ is only feasible if $(i \rightarrow t)$ is traversed with $\delta_{it} = 0$[4], as this guarantees that the deadline will not be violated if $(x \rightarrow y)$ is traversed ($c_{xt} = 20$). A similar phenomenon is observed when $D_F = 30$, enabling the traversal of the path $\{i, x, z, t\}$ leading to small transmission times for most packets.

In all cases, higher variance increases the uncertainty in path selection while making it possible to explore new paths that could potentially lead to shorter delays. This highlights the need for a dynamic routing strategy in real networks.

B ALGORITHM COMPARISONS

Table 4 shows general characteristics of safe RL compared to the previous works. Comparing the three algorithms shows some similarities but also the major areas in which the algirthms differ.

[4] $\delta = 0$, is not feasible in real networks. We enable 0 transmission times here for ease of explanation

Table 4: Algorithm Properties Comparison

	Classical RL	Rapid Routing[1]	Safe RL
Safety Guarantees	No	Yes	Yes
Pre-Processing Stage	No	Need to be run for every δ change	Run only during structural changes
Exploration	ϵ-greedy	-	Safe ϵ-greedy
Routing	Based on probabilities	Routing table dependent	Based on probabilities
Storage at each node	One Q-value for each outgoing edge	Static tables	One Q-value for each outgoing edge