Routing Using Safe Reinforcement Learning

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

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

Abstract

The ever increasing number of connected devices has lead to a metoric rise in the amount data to be processed. This has caused computation to be moved to the edge of the cloud increasing the importance of efficiency in the whole of cloud. The use of this fog computing for time-critical control applications is on the rise and requires robust guarantees on transmission times of the packets in the network while reducing total transmission times of the various packets.

We consider networks in which the transmission times that may vary due to mobility of devices, congestion and similar artifacts. We assume knowledge of the worst case tranmssion times over each link and evaluate the typical tranmssion times through exploration. We present the use of reinforcement learning to find optimal paths through the network while never violating preset deadlines. We show that with appropriate domain knowledge, using popular reinforcement learning techniques is a promising prospect even in time-critical applications.

2012 ACM Subject Classification Computing methodologies → Reinforcement learning; Networks → Packet scheduling

Keywords and phrases Real time routing, safe exploration, safe reinforcement learning, time-critical systems, dynamic routing

Digital Object Identifier 10.4230/OASIcs.Fog-IoT.2020.6

Funding The authors are members of the LCCC Linnaeus Center and the ELLIIT Strategic Research Area at Lund University. This work was supported by the Swedish Research Council through the project "Feedback Computing", VR 621-2014-6256.

1 Introduction

Consider a network of devices in a smart factory. Many of the devices are mobile and communicate with each other on a regular basis. As their proximity to the other devices change, the communcation delays experienced by the device also change. Using static routing for such time-critical communications leads to pessimistic delay bounds and underutilization of network infrastructure.

Recent work [2] proposes an alternate model for representing delays in such time-critical networks. Each link in a network has delays that can be characterised by a conservative upper bound on the delay and the typical delay on the link. This dual representation of delay allows for capturing the communication behavior of different types of devices.

For example, a communication link between two stationary devices can be said to have equal typical and worst case delays. A device moving on a constant path near another stationary device can be represented using a truncated normal distribution. Adaptive routing techniques are capable of achieving smaller typical delays in such scenarios compared to static routing.

2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020).
Editors: Anton Cervin and Yang Yang; Article No. 6; pp. 6:1–6:8
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

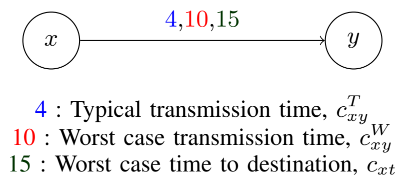

Figure 1 Each link with attributes.

The adaptive routing technique described from [2] uses both delay information (typical and worst case) to construct routing tables. Routing is then accomplished using the tables which consider typical delays to be deterministic. This is however not the case as described above.

We propose using Reinforcement Learning (RL) [8, 12] for routing packets. RL is a model-free machine learning algorithm that has found prominence in the field of AI given its light computation and promising results [5, 9]. RL agents learn by exploring the environment around them and then obtaining a reward at the end of one iteration denoted one episode.

RL has been proven to be very powerful but it has some inherit drawbacks when considering its application to time-critical control applications. RL requires running a large number of epsiodes for an agent to learn. This leads to the possibility of deadline violations during exploration. Another drawback is the large state-space used for learning in classical RL methods that leads to complications in storage and search.

In this paper, we augment classical reinforcement learning with safe exploration to perform *safe* reinforcement learning. We use a simple (Dijkstras[4]) algorithm to perform safe exploration and then use the obtained information for safe learning. Using methodology described in Section 4 we show that safety can be guaranteed during the exploration phase. Using safe RL also restricts the state-space reducing its size. Our decentralised algorithm allows each agent/node in the network to make independent decisions further reducing the state space. *Safe* reinforcement learning explores the environment to dynamically sample typical transmission times and then reduce delays for future packet transmissions. Our Decentralised approach allows each node to make independent and safe routing decisions irrespective of future delays that might be experienced by the packet.

2 System Architecture

Consider a network of nodes, where each link $e : (x \rightarrow y)$ between node x and y is described by delays as shown in Figure 1.

- **Worst case delay (c_{xy}^W):** The delay that can be guaranteed by the network over each link. This is never violated even under maximum load.
- **Typical delay (c_{xy}^T):** The delay that is encountered when transmitting over the link and varies for each packet. We assume this information to be hidden from the algorithm and evaluated by sampling the environment.
- **Worst case delay to destination (c_{xt}):** The delay that can be guaranteed from node x to destination t. Obatined after the pre-processing described in Section 4.1.

A network of devices and communication links can be simplified as a Directed Acyclic Graph as shown in Figure 2. The nodes denote the different devices in the network and the links denote the connections between the different devices. For simplicity we only assume one-way communication and consider a scenario of transmitting a packet from an edge device i to a server, t at a location far away from it.

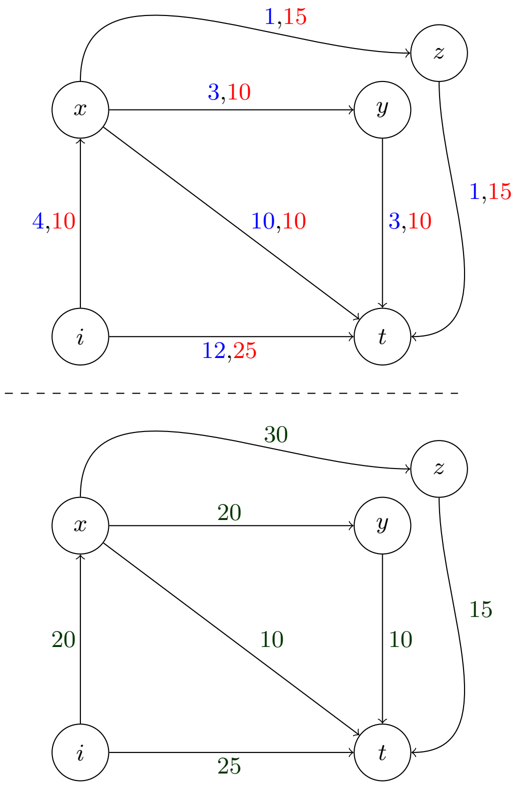

Figure 2 Example of graph and corresponding state space for the reinforcement learning problem formulation.

As seen from the graph, many paths exist from the source i to t destination that can be utilised depending upon the deadline D_F of the packet.

The values of c^T_{xy} and c^W_{xy} are shown in blue and red respectively for each link $e(x \rightarrow y)$. We also show the value of c_{xt} in green obtained after the pre-processing stage described in the following section.

3 Reinforcement Learning

Reinforcement Learning is the area of machine learning dealing with teaching agents to learn by performing actions to maximise a reward obtained [8] RL generally learns the environment by performing actions (safe actions) and evaluating the reward obtained at the end of the episode. We use Temporal-Difference (TD) methods for estimating state values and discover optimal paths for packet transmission.

We model our problem of transmitting packets from source i to destination t as a Markov Decision Process (MDP) as is the standard in RL. 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.

3.1 TD Learning

TD learning [8] is a popular reinforcement learning algorithm that gained popularity due to it expert level in playing backgammon [9]. This model-free learning uses both state s and action a information to perform actions from the state given by the Q-value, $Q(s, a)$. TD learning is only a method to evaluate the value of being in the particular state. It is generally coupled with an exploration policy to form the strategy for an agent. We use a special TD learning called one step TD learning that allows for decentralised learning and allows for each node to make independent routing decisions. The value update policy is given by

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

3.2 Exploration Policy

ϵ-greedy exploration algorithm ensures that the optimal edge is chosen for most of the packet transmissions while at the same time other edges are explored in search of a path with higher reward. The chosen action $a \in A$ is either one that has the max value V or is a random action that explores the state space. The policy explores the state space with a probability ϵ and the most optimal action is taken with the probability $(1 - \epsilon)$. Generally the value of ϵ is small such that the algorithm exploits the obtained knowledge for most of the packet transmissions. To ensure that the deadline D_F is never violated, we modify the exploration phase to ensure safety and perform *safe* reinforcement learning.

4 Algorithm

We split our algorithm into two distinct phases. A pre-processing phase tha gives us the initial safe bounds required to perform safe exploraion. A run-time phase then routes packets through the network.

At each node, the algorithm explores feasible paths. During the inital transmissions the typical tranmssion times are evaluated after packet transmission. During the following transmissions, the path with the least delay is chosen more frequently while also exploring new feasible paths for lower delays. All transmissions using our algorithm are guaranteed to never violate any deadlines as we use safe exploration.

4.1 Pre-processing Phase

The pre-processing phase determines the safe bound for the worst case delay to destination t from every edge $e : (x \to y)$ in the network. The algorithm used by our algorithm is very similar to the one in [2]. This is crucial to ensure that there are no deadline violations during exploration in the run-time phase and is necessary irrespective of the run-time algorithm used. Dijkstras shortest path algorithm [7, 4] is used to obtain these values as shown in Algorithm 1.

4.2 Run-time Phase

The run-time algorithm is run at each node on the arrival of a packet. It determines $e : (x \to y)$ the edge on which the packet is transmitted from the node x to node y. Then the node y executes the run-time algorithm till the packet reaches the destination.

Algorithm 1 Pre-Processing.

```
for each node u do
    for each edge (u → v) do
        // Delay bounds as described in Section 4.1
        c_uv = c^W_uv + min(c_vt)
        // Initialise the Q values to 0
        Q(u, v) = 0
```

Algorithm 2 Node Logic (u).

```
for Every packet do
    if u = source node i then
        D_u = D_F // Initialise the deadline
        δ_it = 0 // Initialise total delay for packet = 0
    for each edge (u → v) do
        if c_uv > D_u then // Edge is infeasible
            P(u|v) = 0
        else if Q(u, v) = max(Q(u, a ∈ A)) then
            P(u|v) = (1 − ϵ)
        else
            P(u|v) = ϵ/(size(F − 1))
    Choose edge (u → v) with P
    Observe δ_uv
    δ_it += δ_uv
    D_v = D_u − δ_uv
    R = Environment Reward Function(v, δ_it)
    Q(u, v) = Value iteration from Equation (1)
    if v = t then
        DONE
```

The edge chosen can be one of two actions:

- **Exploitation action**: An action that chooses the path with the least transmission time out of all known feasible paths. If no information is known on all the edges, then an edge is chosen at rondom.
- **Exploration action**: An action where a sub-optimal node is chosen to transmit the packet. This action uses the knowledge about c_{xy} obtained during the pre-prcocesing phase to ensure that the exploration is safe. This action ensure that the algorithm is dynamic by ensuring that if there exists a path with lower transmission delay, it will be explored and chosen more during future transmissions. Exploration also optimises for a previously congested edge that could be decongested at a later time.

Algorithm 2 shows the pseudo code for the run-time phase. The computation is computationally light and can be run on mobile IoT devices.

4.3 Environment Reward

The reward $\mathcal{R}$ is awarded as shown in Algorithm 3. After each traversal of the edge, the actual time taken δ is recorded and added to the total time traversed for the packet, $\delta_{it}+=\delta$. The reward is then awarded at the end of each episode and it is equal to the amount of time saved for the packet, $R = D_F - \delta_{it}$.

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
  $R = 0$

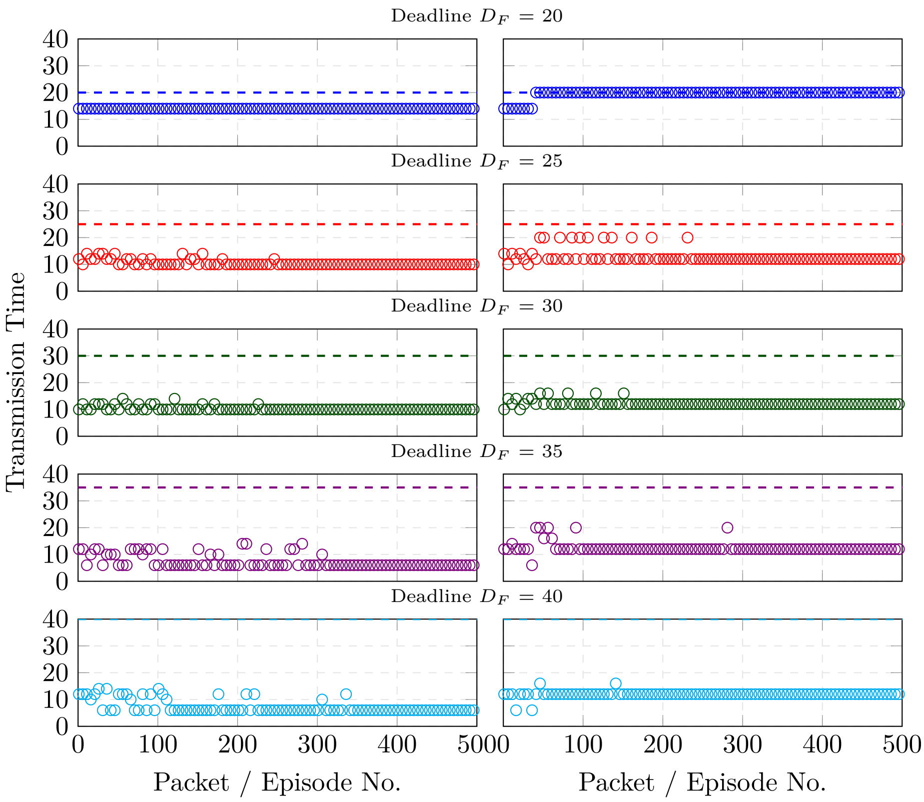

Figure 3 Smoothed Total Delay for Experiment with (a) Constant delays and (b) Congestion at packet 40.

5 Evaluation

In this section, we will evaluate the performance of our algorithm. We apply it to the network shown in Figure 2. The network is built using Python and the NetworkX package [6] package. The package allows us to build Directed Acyclic Graphs (DAGs) with custom delays. Each link $e : (x \rightarrow y)$ in the network has the constant worst case link delay c^W_{xy} visible to the algorithm but the value of c^T_{xy} although present is not visible to our algorithm. The pre-processing algorithm and calculates the value of c_{xt}. This is done only once initially and then Algorithms 2 and 3 are run for every packet that is transmitted and records the actual transmission time δ_{it}.

Figure 3 shows the total transmission times when the actual transmission times δ and typical transmission times c^T are equal. We route 500 packets through the network for deadine $D_F \in (20, 25, 30, 35, 40)$. For $D_F = 20$, the only safe path is $(i \rightarrow x \rightarrow t)$ and so has a constant δ_{it} for all packets. For the remaning deadlines, the transmission times vary as new paths are taken during exploration. The deadlines are never violated for any packets irrespective of the deadline. Table 1 shows the optimal paths and the average transmission times compared to the algorithm from [2].

Figure 3 shows the capability of our algorithm in adapting to congestions in the network. Congestion is added on the link $(i \to x)$ after the transmission of 40 packets. The tranmission time over the edge increases from 4 to 10 time units and is kept congested for the rest of the packet transmissions. The algorithm adapts to the congestion by exploring other paths that might now have lower total transmission times δ_{it}. In all cases other than $D_F = 20$, the algorithm converges to the path $(i \to t)$ with $\delta_{it} = 12$. When $D_F = 20$, $(i \to x \to t)$ is the only feasible path.

6 Practical Considerations

In this section we will discuss some of the practical aspects when implementing the algorithms described in Section 4.

6.1 Compuational Overhead

The computational compexity of running our algorithm mainly arises in the pre-processing stage. This complexity is dependent on the number of nodes in the network. Dijkstras algorithm has been widely studied and have efficient implementations that reduce computation. The pre-processing has to be run only once for all networks given that there are no structural changes.

6.2 Multiple sources

The presence of multiple sources and thus multiple packets on the same link can be seen as an increase in the typical delays on the link. This holds true given that the worst case delay c_{xy}^W over each link is properly determined and guaranteed.

6.3 Network changes

- **Node Addition:** During the addition of a new node the pre-processing stage has to be run in a constrained space. The propagation of new information to the preceeding nodes is only necessary if it affects the value of c_{xt} over the affected links. The size of the network affected has to be investigated furthur.
- **Node Deletion:** In the event of node deletion during the presence of a packet at the deleted node, the packet is lost and leads to deadline violation. However no further packages will be transmitted over the link as the reward $\mathcal{R}$ is 0. Similar to the case of node addition, the pre-processing algorithm requires furthur investigations.

Table 1 Optimal Path for Different Deadlines.

D_F	Optimal Path	Delays [2]	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

7 Conclusion and Future Work

In this paper we use *safe* reinforcement learning to routing networks with variable transmission times. A once used pre-processing algorithm is used to determine safe bounds. Then a safe reinforcement learning algorithm uses this domain knowledge to route packets in minimal time with deadline guarantees. We have considered only two scenrios in this paper but we believe that the algorithm will be able to adapt with highly variable transmission times and network failures. The use of low complexity RL algorithm makes it suitable for use on small, mobile platforms.

Although we show stochastic convergence in our results with no deadline violations, our current work lacks **formal guarantees**. Recent work has been published trying to address analytical safety guarantees of safe reinforcement learning algorithms [10, 11]. In [10], the authors perform safe Bayesian optimization with assumptions on Lipschitz continuity of function. While [10] estimates the safety of only one function, our algorithm is dependent on the continuity of multiple functions and requires more investigation.

The network implementation and evaluation using NetworkX in this paper have shown that using *safe* RL is a promising technique. An extension of this work would be implementation on a network emulator. Using network emulators (for example CORE [1], Mininet [3]) would allow us to evaluate the performance of our algorithm on a full internet protocal stack. Using an emulator allows for implementation of multiple flows between multiple sources and destinations.

References

1. Jeff Ahrenholz. Comparison of core network emulation platforms. In *2010-Milcom 2010 Military Communications Conference*, pages 166–171. IEEE, 2010.
2. Sanjoy Baruah. Rapid routing with guaranteed delay bounds. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 13–22, December 2018.
3. Rogério Leão Santos De Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. Using mininet for emulation and prototyping software-defined networks. In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6. Ieee, 2014.
4. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959. doi:10.1007/BF01386390.
5. Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. An investigation of model-free planning. *arXiv preprint*, 2019. arXiv:1901.03559.
6. Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, Pasadena, CA USA, 2008.
7. Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer Publishing Company, Incorporated, 1 edition, 2008.
8. Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An Introduction*. Adaptive computation and machine learning. MIT Press, 2018.
9. Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995. doi:10.1145/203330.203343.
10. Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration for interactive machine learning. In *Proc. Neural Information Processing Systems (NeurIPS)*, December 2019.
11. Kim P Wabersich and Melanie N Zeilinger. Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning. *arXiv preprint*, 2018. arXiv:1812.05506.
12. Marco Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12:3, 2012.