TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektrotechnik und Informationstechnik Institut für Nachrichtentechnik

Deutsche Telekom Lehrstuhl für Kommunikationsnetze

Diploma Thesis

Investigation of Reinforcement Learning Strategies for Routing in Software-Defined Networks

Peter Sossalla

Born on: 23rd February 1993 in Dresden
Matriculation number: 3873149
Matriculation year: 2012

to achieve the academic degree

Diplomingenieur (Dipl.-Ing.)

Supervisor
Dipl.-Ing. Justus Rischke

Supervising professor
Prof. Dr.-Ing. Dr. h.c. Frank H. P. Fitzek

Submitted on: 14th October 2019

Statement of authorship

I hereby certify that I have authored this Diploma Thesis entitled *Investigation of Reinforcement Learning Strategies for Routing in Software-Defined Networks* independently and without undue assistance from third parties. No other than the resources and references indicated in this thesis have been used. I have marked both literal and accordingly adopted quotations as such. During the preparation of this thesis I was only supported by the following persons:

Dipl.-Ing. Justus Rischke
Dipl.-Ing. Sandra Zimmermann
Marek Sobe

Additional persons were not involved in the intellectual preparation of the present thesis. I am aware that violations of this declaration may lead to subsequent withdrawal of the degree.

Dresden, 14th October 2019

Peter Sossalla

Abstract

In recent years, the number of Internet users and their bandwidth requirements have steadily increased. During peak periods, high bandwidths often lead to service failures because the communication connections are overloaded. Large Internet companies such as Google and Facebook are therefore using Software-Defined networking to cleverly distribute loads in the network to minimize downtime. Recent research has tried to determine and optimize this load distribution with Machine Learning (ML). In this thesis, an approach based on Reinforcement Learning, a discipline of ML, is used, as it does not require any previously determined data, but can learn to route advantageously through interaction with the network.

The heterogeneous and distributed structure of the Internet makes the application of ML complicated. Thanks to central control and extended measurement capabilities, SDN can simplify this. Current approaches use the features of RL and SDN not for direct routing but to determine graph weights. Based on these graphs, the data streams are then routed or split using well-known algorithms such as Shortest Path First.

Instead, this work pursues an approach in which an RL framework can directly influence the network through routing decisions to optimize routing based on connection latencies. This makes it possible to create a system that optimizes itself without the need for a model. For this purpose, an application was developed, which was implemented in an SDN controller. It can measure current latencies and independently perform routing decisions. Substantial measurements were performed in a realistic emulated environment to compare the implemented RL approach with state of the art routing. In addition to various RL parameters, dynamic data stream behavior and several topologies were investigated. The results show the advantage of using the RL solution over shortest path algorithms to ensure low latency in the network and to prevent congestion.

Zusammenfassung

In den letzten Jahren hat sich die Anzahl der Internetnutzer und deren Anforderungen an Bandbreite stetig erhöht. Hohe Bandbreiten führen zu Stoßzeiten häufig zu Ausfällen der bereitgestellten Dienste, da die Kommunikationsverbindungen überlastet sind. Große Internetunternehmen wie Google und Facebook versuchen deswegen mittels Software-Defined Networking Lasten im Netzwerk intelligent zu verteilen um so Ausfälle zu minimieren. Aktuelle Forschung hat versucht mit Machine Learning (ML) diese Lastverteilung zu bestimmen und zu optimieren. In dieser Arbeit wird ein Ansatz basierend auf Reinforcement Learning, einer Disziplin von ML, eingesetzt, da es keine vorher ermittelten Daten benötigt, sondern durch Interaktion mit dem Netzwerk selbständig lernen kann vorteilhaft zu routen.

Die heterogene und verteilte Struktur des Internets macht die Anwendung von ML kompliziert. Dank zentraler Kontrolle und erweiterter Messfähigkeiten, kann SDN diese jedoch vereinfachen. Derzeitige Ansätze nutzen die Möglichkeiten von RL und SDN nicht zum direkten Routen sondern um Graphgewichte zu ermitteln. Anhand dieser Graphen werden die Datenströme dann mit altbekannten Algorithmen wie Shortest Path First geroutet oder aufgeteilt.

In dieser Arbeit wird stattdessen ein Ansatz verfolgt, bei dem ein RL Framework direkt durch Routing Entscheidungen auf das Netzwerk einwirken kann, um das Routing anhand der Verbindungslatenzen zu optimieren. Dadurch ist es möglich, ein System zu schaffen welches sich selbstständig optimiert ohne ein Modell zu benötigen. Hierfür wurde eine Applikation entwickelt, welche in einen SDN Controller implementiert wurde. Diese kann aktuelle Latenzen messen sowie selbstständig Routing Entscheidungen durchführen. Es wurden Messungen in einer realistischen emulierten Umgebung durchgeführt, um den implementierten RL Ansatz mit State of the Art Routing zu vergleichen. Neben verschiedenen RL Parametern wurden dabei dynamisches Datenstromverhalten und verschiedene Topologien untersucht. Die Ergebnisse zeigen den Vorteil der Nutzung der RL Lösung gegenüber Shortest Path Algorithmen, um eine geringe Latenz im Netzwerk zu gewährleisten und Congestion zu verhindern.

Contents

List of Figures

Symbols

Symbol	Description
$G(\mathcal{N}, \mathcal{L})$	Software-Defined Network with nodes $\mathcal{N}$ and links $\mathcal{L}$
$\mathcal{L}$	Set of links l
$l_{1,2}$	Link between node 1 and node 2
$C(l)$	Capacity of link l
$\mathcal{N}$	Set of nodes
$\mathcal{F}$	Set of flows f
f_{h_s,h_d}	Flow from source host h_s to destination host h_d
b^f	Traffic rate of flow f
$b^f(l)$	Traffic rate in link l caused by flow f
$w(f)$	Cost of flow f
W	Total cost of all flows
$\delta_{f,l}$	Flow-link indicator; 1 if link l is used by flow f
$\mathcal{P}(f)$	Set of possible paths for flow f
$p(f)_s$	Path of flow f in state s
t	Time step
$\mathcal{S}$	State space
S_t	State at time step t
s'	Next state
$\mathcal{A}$	Action space
$\mathcal{A}(s)$	Action space in state s
A_t	Action chosen at time step t
$\mathcal{R}$	Reward space
R_t	Reward at time step t
r	Reward
$p(s', r\|s, a)$	Probability of getting reward r in state s' after selecting action a in state s
G_t	Future reward at time step t
E	Terminal state
γ	Discount rate
α	Learning rate
π	Policy
$\pi(a\|s)$	Stochastic policy; probability of taking action a in state s
$v(s)$	Value of s; expected future reward in state s
v_π	Expected reward by following policy π

$q_\pi(s, a)$	Action-value; expected reward of taking action a in state s following policy π
π_*	Optimal policy
$v_*(s), q_*(s, a)$	Optimal state-value function, optimal action-value function
$V(s)$	Estimate of value function $v(s)$
$Q(s, a)$	Estimate of action-value function q(s,a)
$\mathfrak{Q}$	Set of Q-table entries/ Q-values
ε	Exploration probability for ε - greedy
τ	Temperature; exploration factor for softmax
b^+	Bonus; UCB
c	Exploration factor for UCB
$N(s, a)$	Times action a was chosen in state s
$N(s)$	Total number of visits of state s
Sw	Switch
Co	Controller
L_T	Total packet travel time
$L_{Sw1-Sw2}$	One-way delay from $Sw1$ to $Sw2$
RTT	Round-trip time
$d(f)$	Delay of flow f
$\overline{d}$	Arithmetic mean of all flow delays
$\overline{d}(t)$	Measured mean latency at time step t
t_c	Convergence time step
d_l	Latency of link l
D	Quadratic mean of all flow delays
k_{UDP}	Packet size
T_{wait}	Waiting time for static state
b_{diff}	Bandwidth difference between link capacity $C(l)$ and flow traffic rate b^f
K	Queue length
r_{empty}	Queue emptying rate
T_{empty}	Queue emptying duration
$p(dropped)$	Packet drop rate
T_{meas}	Total measuring time
T_{delay}	Congestion delay
LL	Load Level

Acronyms

ARP Address Resolution Protocol
AS Autonomous System
CSP Constrained Shortest Path
DC Data Center
DFS Depth-first search
DRL Deep Reinforcement Learning
IGP Interior Gateway Protocol
IP Internet Protocol
ISP Internet Service Provider
LLDP Link Layer Discovery Protocol
MAC Media Access Control
MCSP Multi-Constrained Shortest Path
MDP Markov Decision Process
ML Machine Learning
NE Network Elements
NFV Network Functions Virtualization
OSPF Open Shortest Path First
OVS Open vSwitch
QoS Quality of service
RL Reinforcement Learning
SDN Software-Defined Networking
SLA Service Level Agreement
SPF Shortest Path First
TBF Token Bucket Filter
TD Temporal Difference

TE Traffic Engineering

UCB Upper Confidence Bound

WAN Wide Area Network

1 Introduction

Motivation

An explosive growth in network size and a rising number of users with increasing traffic demands lead to an extension of the network infrastructure of Internet Service Provider (ISP). This is necessary to meet the Quality of service (QoS) requirements in an environment of increasing demand and strong dynamics due to unpredictable user behavior. QoS refers to the description and measurement of the network quality by quantitative parameters such as latency, packet losses, bandwidth and jitter. Additionally, the provider needs to comply with the Service Level Agreement (SLA), an arrangement between the service provider and its costumers that defines the assured service availability and performance. In packet-switched networks, congestion can be a reason for not meeting these requirements. Congestion occurs in network links if the demanded traffic rate is higher than its capacity and results in a increased latency and packet losses. In order to comply with the obligations defined in a SLA, the network is *overprovisioned* [2][3], meaning the allocation of resources to a greater extent than necessary, often as high as two or three times the typical demands. This results in a greater resilience, but the additional switches and fiber optic connections mean higher acquisition and operating costs for the network provider. A cheaper alternative would managing the network more efficiently and reactive by balancing the load. In packet-switched networks, the optimization can take place via a beneficial selection of paths for traffic flows between given source-destination pairs. The further optimization ability is provided by Software-Defined Networking (SDN) through the centralization of control and monitoring capabilities. Another advantage of SDN is the possible softwarization, which allows the usage cheaper and easily replaceable off-the-shelf devices instead of expensive, application-specific and proprietary systems that are provided by a limited amount of vendors. In addition, it would also be possible to develop and operate network management software in-house instead of relying on vendor solutions [4].

Additionally to ISPs, large corporations such as Google or Facebook, whose contents account for a significant part of the data transferred over the Internet, have encountered the benefits of network optimization to increase the efficiency of their inter-Data Center (DC) networks using solutions based on SDN [5][6]. The common performance objectives are minimizing the end-to-end latency, preventing congestion and achieving a higher resource utilization, so overprovisioning can be omitted. Over the years, a large set of network modelling and optimization strategies have been developed [7]. Following the characteristics of network optimization, it is only possible to optimize what is modellable. Routing while

complying with the QoS principles often involves complex multi-constrained optimization problems, such as integer or mixed-integer programs, which scale polynomial due to their NP-hardness [8]. Another technique are models based on Queuing Theory [9], which often rely on assumptions such as traffic generation based on a poisson distribution what can differ from networks in reality [10]. Additionally, these models usually scale poorly because network problems involving multi-hop routing would result in a complicated multi-point to multi-point queuing network. A solution can provide Machine Learning (ML), which has got much attention in research and industry through its application to a wide range of complicated problems. An interesting area of Machine Learning is Reinforcement Learning (RL), a paradigm that allows systems to learn how to behave beneficial by interacting with the environment. Therefore RL can address complex optimization tasks even without a model and only relies on a feedbacks for its actions. The usage of a controller in an SDN network makes it possible to interact directly with the network by customizing forwarding decisions in switches and collect the resulting network state with its metrics such as the latency, which can serve as the feedback.

As a consequence of using RL, it is possible to optimize to different objectives in a network without relying on complex, knowledge- and labour-intensive modelling.

Task

To investigate the application of Reinforcement Learning for routing, a framework should be designed and implemented in the scope of this work. The RL framework is obliged to select routes beneficially to optimize for a performance objective, in this work the latency of all flows in the network. The necessary components for observing the network metrics such as the link latency have to be implemented. It is supposed that the controller is able to route at run-time and thereby decrease the overall latency. The evaluation should take place in a heterogeneous network with varying traffic and different end-to-end connections. Also the dynamic behaviour during the adaptation process ought to be observed and evaluated. A proper test network with the possibility to emulate non-stationary traffic should be provided.

Structure

Chapter 2 gives an introduction of the topics of this thesis. Firstly a look is taken at Software-Defined Networking, especially its application for routing and traffic engineering problems. Then Routing and Traffic Engineering paradigms are presented with an overview of the currently used technologies. This is followed by a discourse of Reinforcement Learning and its use cases in RL networks. In chapter 3, the implementation, including the components of the controller and the individual measurement setup, is described. Chapter 4 contains the measurements of the different RL strategies and measurement setups. The resulting findings are evaluated and it is shown how the RL framework can benefit to optimize the latency or network utilization.

2 Related Work & Background

2.1 Software-Defined Networks

"Software-Defined Networking (SDN) refers to a new approach for network programmability, that is, the capacity to initialize, control, change, and manage network behavior dynamically via open interfaces" [11]. The higher programmability is achieved by dividing the network into two different layers. the *control plane* and the *forwarding plane*. The forwarding plane consists of Network Elements (NE) (i.e. switches), which handle the packets via actions like forwarding or dropping. Figure 2.1 illustrates the concept of SDN. The actions of the switches depend on the instructions received from the control plane, which consists of one or more *controller* entities that manage the routes (e.g. flows) in which the packets are forwarded. These forwarding rules are passed to the switches. The packets are identified by specific characteristics including MAC or IP addresses, ports or Ethernet types and are matched to the actions through a *Flow Table*. The controller can be realized by several frameworks, e.g. NOX[1], Floodlight[2] or Beacon[3]. For this thesis, the ryu Controller [4] was selected, because ryu combines an easy and agile development via python with a comprehensive documentation. As forwarding devices , switches that support SDN are required. Therefore Open vSwitch (OVS), a software switch introduced in [12], was chosen. OVS is a high performance multilayer software switch which delivers the benefits of vendor independence and a high flexibility. Additionally, OVS supports *OpenFlow* [13], a protocol developed by Stanford university back in 2008, which serves as an interface between switches and controller. OpenFlow have gathered broad interest since Google announced the usage of OpenFlow for the optimization of B4 [5], an internal Wide Area Network (WAN) network which connects their data centers around the globe.

[1] Nox Controller, `github.com/noxrepo/nox`
[2] Floodlight Controller, `projectfloodlight.org`
[3] Beacon Controller, `openflow.stanford.edu/display/Beacon`
[4] ryu controller, `osrg.github.io/ryu/`

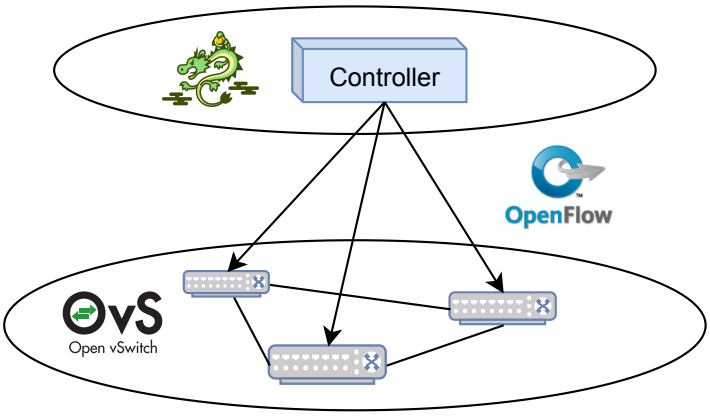


Figure 2.1: Overview of the SDN structure containing a data plane with several OVS switches and a control plane with a ryu controller.

2.1.1 Advantages of SDN

Today, many features such as security, routing and energy management are delivered by a small amount of companies in a proprietary manner, highly priced and complicated to integrate. These features are often integrated as part of their whole solution package and a dynamic composition of different vendors is not easily achievable. Some parts are even offered as additional hardware, making it difficult to integrate, adapt or extend the individual parts as a complete system. SDN solves the problem of integrability and missing flexibility by its high programmability. Additional features can be easily implemented into the controller and can use the obtained global knowledge about the current network metrics. Beside WAN networks, SDN is also applied in campus networks, an internal network of a companies or institutions that manage their networks by themselves [13]. These networks are often affected by a diverse and fluently changing user behaviour in terms of working location and strongly varying traffic due to bandwidth intensive services such as video conferences and data distribution within the company. Predefined or static network deployments can struggle to handle bursting traffic. Therefore it is challenging to meet the QoS requirements of latency and jitter sensitive traffic by relying only on the best-effort model. Software-Defined Networks can help with their flexibility to shift traffic and relieve occurring bottlenecks. For companies, Security and User Identification is an important issue as well. SDN enables the usage of Network Functions Virtualization (NFV) [14], an emerging technology that allows to implement network functions such as domain named services or security options like firewalls. NFV leverages the flexibility of a deployed network by allowing to deploy network functions fast, location independent and tailored to the current needs. In combination with NFV, SDN helps in data centers to deliver higher network utilization, better resilience against link failures, and a dynamic adaption on fluctuating user demands. For every application area, SDN helps to reduce complexity and raise programmmibility. Eventually, SDN allows a significantly faster innovation and serves as an enabler for more efficiency in modern networks. The advantages of SDN are clear and should be applied for a reliable and effective routing. mi

2.2 Routing

In simple terms, network routing breaks down to the ability of determining a path to send a piece of information from point A to point B in an electronic communication network.

Therefore an appropriate path, beneficially as efficient and quick as possible, needs to be determined. In a communication network, the quality of a path depends on several factors like the reliability, on the provisioned bandwidth and the end-to-end delay (i.e. latency). The delay in a network is composed of the propagation, serialization, queuing and the processing delay. Propagation delay is the time the signal needs to be transmitted from one point to the another in a link. It depends on the length of the wire between the two endpoints and the medium (commonly copper or fiber). Serialization delay describes the time it takes for an interface to push all bits into the wire depending on the datarate and the packet's frame size. The processing delay can be by processes such as bit error checking or forwarding decisions in the switches. The queuing delay defines the waiting time of the packets in a queue until being transmitted if the capacity of a link is exceeded. In this work, the focus lies on the propagation as well as the queuing delay, because the propagation delay represents the physical distance between switches and the queuing delay is the affected one if congestion occurs, i.e. if the links are overloaded due to improper routing decisions. In a multi-hop network, these factors depend on quality of the intermediate forwarding devices and the links in between them. The forwarding devices should deliver efficient and reliable switching. A quantitative measure of the link quality can be the latency, available bandwidth or transmission errors. Responsible for discovering possible paths by communicating with neighbors is the *routing protocol*. Additionally, routing protocols are responsible for communicating congestion in the network. Congestion would result in a higher end-to-end delay or limit the reachability of network nodes. If an efficient path has been discovered by the protocol, the next possible hop is saved in a table, called the *routing table*. The determination of the best next hop depends on the so called *routing algorithm*. When the best path is chosen, the next hop is saved in the *forwarding table*.

2.2.1 Intradomain vs. Interdomain

As the internet grows continuously, it is divided into different parts, called Autonomous Systems (ASs) [15], which are controlled by a single entity or organization, commonly by an Internet Service Provider. An AS can consist of one or several networks, but all of these subnetworks share an affiliated routing logic and collective routing policies. Each AS has to be uniquely identifiable, achieved by the so called autonomous system number and is issued globally by a central authority. To route inside of an AS, a type of routing protocols named Interior Gateway Protocol (IGP) , is used. The routing of packets between different ASs is addressed by the Border Gateway Protocol (BGP). Used for the exchange of routing and reachability information, it serves as the connection between the networks of different ISP and allows the functionality of the internet as it is known today. Additionally, it offers stability by rerouting over another AS in case of failures. Within the scope of this work, only the routing within one network and hence falling under IGP, is considered. IGP can be divided into two types, the link-state and the distance-vector routing protocols.

2.2.2 Link-state & Distance-vector routing

To identify efficient paths in a network, it is necessary to update the routing tables appropriately. There are two ways the tables are updated, depending on how the information of the network is shared within the forwarding elements. One way is to share the complete knowledge about the network with each other. Taking a routing decision based on the whole information of the topology, which can be symbolized as a weighted graph, is called *link-state routing*. Another way is to share their routing tables, only with the neighbors. This normally takes place in periodic updates and changes in the network (e.g. link failures) are advertised

sequentially across the entire network. However, the main difference is the point of view. In case of distance-vector routing the path calculation is based on the neighbors point of view, while in link-state routing the whole topology is taking into account. In SDN, the controller can receive knowledge about the network topology and state by the switches. As a result, the controller can construct a weighted graph and perform algorithms to determine the best possible path.

2.2.3 OSPF

As specified in [11], the control plane is usually in charge of topology discovery, route selection and mechanisms in case of link failures. The common routing protocol in IP networks is Open Shortest Path First [16]. This static routing scheme is based on shortest path algorithms, commonly Dijkstra [17], which are performed on a graph based on the topology information. OSPF has an extension for QoS [18], which dynamically changes the weights depending on measured traffic. However it is often not implemented in real networks because of two major reasons: Firstly, the changing of the weights of the links can lead to many routing updates that can influence negatively the traffic in other parts of the network. Secondly because of the awareness of routing loops, meaning that packets get routed in between two or more routers because a delayed update of their routing and as a result of their forwarding table, which can occur before the convergence of the routing protocol [19]. As SPF routing based only on hop count or latency ignores the influence of increasing levels of traffic in the links, which can cause congestion.The incompleteness of algorithms based on SPF leads to another interesting and popular topic of the network-world called Traffic Engineering.

2.2.4 Traffic Engineering

To overcome the shortages of SPF routing algorithms, it is necessary to include traffic constellations into the selection of paths. Traffic Engineering (TE) has the goal to route traffic by balancing several objectives:

1. Maximization of throughput
2. Spread the link utilization fairly across the network
3. Ensuring safe operations by adding awareness of link failures or changing traffic patterns
4. Minimizing the latency of individual or all connections

TE is a critical topic because it allows to route more traffic in a significant scale than using simpler approaches like routing all traffic based on shortest paths [20]. As it is an important topic in the network community, a broad range of solutions on this optimization problem based on different backgrounds were developed [21]. Some selected approaches are described later in section 2.5.2.

2.2.5 Routing in SDN

In terms of SDN, to gain knowledge about the network topology, the controller uses the Link Layer Discovery Protocol (LLDP) [22][23]. During the link discovery, the controller sends a LLDP packet out to one switch. The moment the switch receives the packet, he sends it out through all his ports. When the other switches receive the LLDP packet, they do not have

a corresponding rule in their flow table for this packet type. As a result, the switches send a message containing the packet to the controller. From the information contained in the packets, the controller can derive the connections between the switches and thereby the topology. This results in an unweighted graph on which the controller can perform shortest path algorithms to find the way with the smallest amount of hops If the controller has the capacities to measure network metrics such as latency or the current load over the links, it is possible to construct a weighted graph which can be used for the path calculation. With a smart selection of metrics as costs for the graph, the chosen path can comply with Quality of service (QoS) requirements. Commonly, the weights are assigned statically and updated hence the paths are recalculated only if a topology occurs. In SDN, this can occur because of distributed controller entities, which might have different information bases. Another reason could be a high end-to-end delay between the controller and the switches that lead to a delayed update of the forwarding table.

2.3 Problem formulation

Let $G(\mathcal{N}, \mathcal{L})$ be a network with a set of *links* $\mathcal{L}$ connecting the nodes $\mathcal{N}$, in case of this work network forwarding devices (i.e. switches). The capacity of a link is denoted by $C(l)$. In this thesis, the focus lies on *unicast* communication, meaning the data is transmitted only from one sender to another receiver at a time. Each of these transmissions of sequential packets is called a flow f. The communication from the source host h_s to the destination host h_d will be defined as f_{h_s,h_d}. How the traffic rate is defined depends on type of user demand that can be distinguished by the transport layer protocols TCP and UDP. In UDP a fixed traffic rate is demanded depending on the application. For TCP, the rate for a connection is rising until it reaches the maximum level, the network can deliver [24]. In the scope of this work, the focus lies on fixed demanded traffic rates b^f for each flow and a flow cannot be splitted over different paths. The traffic rate in link l caused by flow f is defined as $b^f(l)$. So the problem can be defined as a *multi-commodity flow problem* with *non-splittable flows* [25, p. 145 - 147]. Each flow has a cost $w(f)$, that can serve as representation of different functions like delay, jitter, reliability or probability as congestion. To define it as the Routing Problem, the cost W of all flows $\mathcal{F}$ should be minimized:

$$\textit{minimize } W = \sum_{f \in \mathcal{F}} w(f) \tag{2.3.1}$$

Different constraints can be added, such as the capacity $C(l)$ for each link. It depends on the selected path, if a flow does influence a link l with its traffic rate b^f. To take that into consideration, an indicator $\delta_{f,l}$ is introduced which is set to 1 if a flow uses the link and to 0 otherwise. As it would cause congestion, it should not be exceeded:

$$\sum_{f \in \mathcal{F}} \delta_{f,l} r^f(l) \leq C(l), \quad \forall l \in \mathcal{L} \tag{2.3.2}$$

Solving this problem means to minimize a cost function, often the end-to-end delay, while respecting the physical capabilities of each link. Another optimization objective can be the maximization of the network utilization. For network utility maximization, the links should be chosen to achieve the maximum possible throughput between the hosts. As mentioned before, the user demand does not have to be fixed. For this case, the optimization objective lies on finding a path constellation which maximizes the traffic rate of the flows.

2.4 Reinforcement Learning

2.4.1 Machine Learning

Machine learning is categorized as a subfield of Artificial Intelligence and provides methods to give systems the ability to learn and improve from experience [26, p. 2-3]. The methods can be categorize by the source the system takes to learn, whether it is data, instructions or the direct interaction with the environment.

Unsupervised Learning describes the search for structure in unclassified and unlabeled data. The regular goal is to detect characteristics in the data, like clusters or anomalies. A typical application is e-commerce where clustering algorithms are used to create a costumer specific recommendation system [27].

In *Supervised Learning*, the training is performed with labeled data. The system learns this structured data, known as training data. It then is validated by its performance to categorize new data, called test data. A typical example is object classification in images [28].

In this thesis, Reinforcement Learning (RL) is applied, which gives the opportunity to learn directly by interacting with the environment. The learning entity, called *agent*, receives an immediate signal for each of his actions and the resulting state transition. The signal is a measure of the goodness of his decision, called the *reward*. One of the characteristics of RL is the objective of the agent to maximize the cumulative reward and not only the immediate one, meaning the agent has the ability to learn how to act for achieving a long-term goal. The main difference between Supervised and Reinforcement Learning is that in Supervised Learning the teaching takes place by example, given by provided data, and in Reinforcement Learning by experience gained by interaction with the environment.

2.4.2 Markov Decision Processes

To provide a mathematical framework for the sequential decision making, finite *Markov Decision Processes* (MDPs) are used. They make it possible to consider not only immediate rewards, but also future states and rewards for the action selection. The interaction between the agent and environment takes place in discrete time steps $t = 0, 1, 2...$. A visualization of the learning process is shown in figure 2.2. Every time step t, a model of the environment, state $S_t \in \mathcal{S}$ is given to the agent. According to the given information, the agent selects an action $A_t \in \mathcal{A}(s)$. As a result of the chosen action, the agent receives a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ in the following time step. In MDPs, the *Markov Property* applies, which defines the characteristic that the transition probabilities and rewards only depend on the current state and there is no dependency to the history of states that have been visited before.

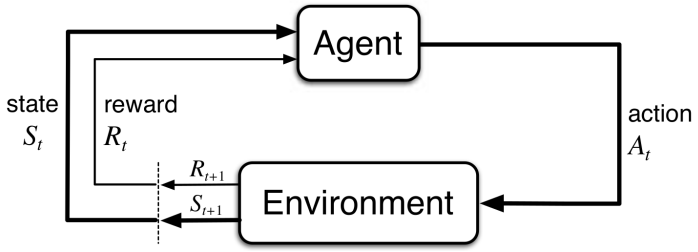


Figure 2.2: Interaction between the agent and environment [1].

All sets of actions $\mathcal{A}$, rewards $\mathcal{R}$ and states $\mathcal{S}$ contain a finite amount of elements. Due to the Markov Property, the probability distribution of the following state $s' \in \mathcal{S}$ and of the resulting reward $r \in \mathcal{R}$ are only dependent on the action and the previous state.

The dynamics of a Markov Decision Process are reflected by an ordinary deterministic four-argument function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ called the *dynamics function*. $p(s', r|s, a)$ defines the probability distribution for each possible state transition to s' in s taking a while receiving reward r. As a conditional probability distribution over each choice of state s and a, it follows:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \tag{2.4.1}$$

If the reward is not taken into account, p results to be a *state-transition probability* of three arguments $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. One of the reasons of using the finite MDP as framework for Reinforcement Learning is the probabilistic behaviour of state transitions due to previous actions and state. It can be beneficial as an abstraction of an objective-directed learning by interacting with the environment.

Goals and Episodes

Basically, the goal of the agent is to maximize the total amount of the rewards he receives by the environment over the time steps, specified by the expected return:

$$G_t = R_{t+1} + R_{t+2} + ... + R_E. \tag{2.4.2}$$

E defines the terminal state of the agent-environment interaction. A terminal state is the end of each iteration over the environment , called *episode*. One of the first applications of Reinforcement Learning was the studies by Arthur L. Samuel, a pioneer in the field of Artificial Intelligence. He let two RL agents compete against each other in the game of checkers [29].

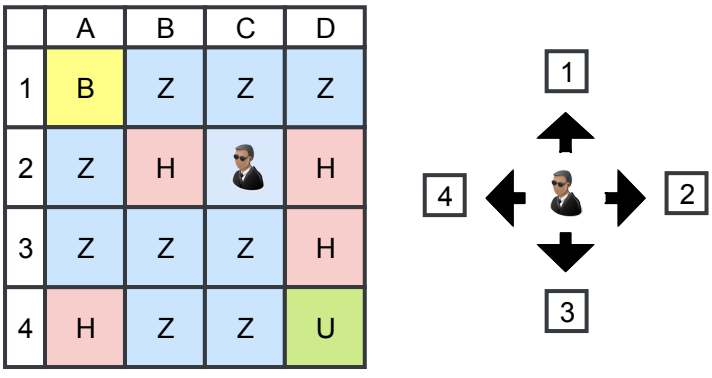


Figure 2.3: Representation of the frozen lake environment and the possible actions of the agent.

To describe the elements of RL properly, an example is introduced. The environment, as shown in figure 2.3 is a frozen lake which is broken down into a grid world. The world consists of 16 different states with different types: a starting state (B), frozen surfaces (Z), holes (H) and a goal state (U). The objective of the player (i.e. agent) is to find a way from the starting state B to the goal state U. He can choose from four different actions $\mathcal{A}$ that are numbered in the negative mathematical way (clockwise): moving to the north a_1, east a_2, south a_3 or west a_4. If the agent is on the border and chooses a direction to an undefined state, he remains in his current state. In this setup, the terminal states are the goal U and the holes

H. For moving into a hole, the agent gets the reward $r_H = -1$, but in the case he reaches the goal he earns the reward $r_U = 5$. As the agent wants to maximize his gained reward, he learns how to act in the environment. In this case, he learns to avoid falling into the holes H and finding a way to the goal U.

Discounting

The expected return G is calculated by equation 2.4.2 by the value of the further states the agent will visit. The agent does currently have no interest in reaching the goal fast because the calculation does not take the amount of future steps into account . This can lead the agent to stay in the world infinitely. To prevent that, *discounting* is introduced. The parameter γ defines the discount rate and specifies how much future rewards contribute to G.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \gamma^{E-t-1} R_E = \sum_{k=1}^{E-t} \gamma^{k-1} R_{k+t} \tag{2.4.3}$$

A reward received after k time steps will be valued γ^{k-1} of the amount if it would have been gained currently.

If $\gamma \in [0, 1)$, it is called *discount factor*, meaning that expected rewards in future states G_{t+1} are valued less than they would be valued in the next state R_{t+1}. Thus it is guaranteed that the sum of rewards is always finite. Through discounting it is possible to influence the agent's behaviour over several time steps. In the presented example, the agent would prefer to choose a route that leads him directly to his destination without taking detours.

Value Functions and Policies

To determine, how beneficial it is to be in a certain state, value functions $v(s)$ are used. Over time, the agent learns how to behave in an environment, so taking a specific action a in a state. This behaviour is summarized in a policy π, which the agent follows. A policy provides the agent with rules to decide for an action in a state, in other words π defines the agent's behaviour and provides a mapping of the actions of the actions a to a state s [30, p.521]. If the agent follows the rules provided by policy π at a certain time step t, then the *stochastic policy* $\pi(a|s)$ is the probability of taking action $A_t = a$ in $S_t = s$. It specifies a probability distribution across all $a \in \mathcal{A}(s)$ for every $s \in \mathcal{S}$. The value v_π is defined as the future rewards that can be expected from acting according to a policy π. Different methods of Reinforcement Learning specify the way the policy changes by gaining experience. The value v_π of acting according to a policy π in state s is defined by it's expected value:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]. \tag{2.4.4}$$

To also give value to an action, $q_\pi(s, a)$ is introduced:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]. \tag{2.4.5}$$

It can be viewed as a measure of quality (in terms of the reward expectation) of taking a specific action in a state s while following policy π.

The expected return G_t from 2.4.4 can also be expressed by:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma(R_{t+3} + \gamma R_{t+4} + \gamma^2 R_{t+5} + ...)) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.4.6}$$

Following for the value function:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \tag{2.4.7}$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \Big[r + \gamma \, \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \Big]$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2.4.8}$$

Equation 2.4.8, called *Bellman equation*, links the value of the current state to its following states. Starting from state s, the agent can select from the actions defined by policy π. Based on the chosen action, it can end up in one of the successor state based on probability p and collects the associated reward. The *Bellman equation* for v_π 2.4.8 indicates that the discounted value of the following state (including all the following expected rewards) have to be equal to the starting state. As mentioned before, the goal of the agent is to find the policy π^* with the highest expected reward.

Optimal policies and value functions

In finite MDPs exists at least one policy whose expected return is the highest in comparison to others, $v_\pi(s) \geq v_{\neg\pi}(s)$ for all $s \in \mathcal{S}$, called *optimal policy* π_*. The state-value connected to the policy is defined as *optimal state-value function*:

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.4.9}$$

Acting in accordance to the best policy additionally provides the *optimal action-value function* q_*.

$$q_*(s, a) = \max_\pi q_\pi(s, a) \tag{2.4.10}$$

$$= \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \tag{2.4.11}$$

By solving the Bellman optimality equation, it is possible to find a π_*. Taking the best action according to the determined best policy deduce via equation 2.4.11 the *Bellman optimally equation* for v_*

$$v_*(s) = \max_a q_{\pi_*}(s, a)$$

$$= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

$$= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \tag{2.4.12}$$

and q_*

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a]$$

$$= \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{2.4.13}$$

As a reminder, the goal is to obtain the optimal policy π_* by finding the optimal value functions v_* and q_*. The optimal policy is found, when the Bellman optimality equations 2.4.12 and 2.4.13 are satisfied.

Dynamic Programming

To calculate the value functions v_* and q_*, algorithms are provided by *dynamic programming*. For solving a particular problem by dynamic programming, it is necessary to solve different parts of the problem (sub-problems) and combine their solutions to attain an overall solution. Recursive algorithms also describe algorithms that seek a solution to a problem by solving several subproblems. If these subproblems have more subproblems and they overlap, the difference between recursive and dynamic programming based algorithms becomes clear. Dynamic programming stores the calculated solutions of the subsubproblems and can use them if the same problems arise again. This prevents recomputing if already calculated problems occur again. The approach seeks to solve each "non-overlapping" sub-problem only once by storing the calculated solution. Typically, dynamic programming is applied to *optimization problems*. It is possible that the problem has more than one possible solution, but each solution is attributed a value and the goal is to find the maximum or minimum, called *optimal solution* [31]. These technique is applied in RL by improving approximations of v_* and q_* by updating rules.

Model-Based Learning

It is necessary to estimate the optimal values v_* and q_* that lead to optimal policy π_*. In model-based learning, the dynamics $p(s', r|s, a)$ of the environment are known completely. That simplifies the solving for the optimal state value function 2.4.12 through the optimal action-value 2.4.13. The values are determined by iteration and successive updating an estimation $V(s)$ of the state value $v(s)$ until $V(s)$ converges, meaning the computed values do not change anymore. This process is called *policy evaluation*, first an arbitrary policy is selected and over time the estimate $V(s)$ probably converges to the true state-value function $v(s)$ [32]. For solving MDPs by using the bootstrapping method from dynamic programming, two types of algorithms for the iteration are introduced, the *value iteration* and the *policy iteration*. The policy iteration algorithm is defined by two steps that are executed alternately. First, the policy is evaluated by policy evaluation to approximate the state value V^{π} for the current policy π. In the second step, which is called *policy improvement*, the policy is updated towards an improvement meaning that the agent will choose better actions over time. It stays in contrast to the value iteration, in which the Bellman Equation 2.4.8 is iteratively applied to update $V(s)$ until the estimation is considered to be converged.

Model-Free Learning

As shown in the previous section, the optimal policy can be determined by using dynamic programming. In realistic use cases, the model of the environments dynamics with its probability functions $p(s', r|s, a)$ is not always known. The knowledge can be gained by sampling over the environment and use the obtained experience to learn optimal policies and value functions. The value function 2.4.8 needs to be updated by sampling over the environment. A mathematical framework provide the so called *Monte Carlo methods* by Sutton and Barto [1, p. 91] which allows to approximate the optimal behaviour by sampling over episodes. Over time, the estimation converges to the true state value function $v(s)$. One drawback of Monte-Carlo methods is that it is necessary to wait until an episode finishes for updating the value $V(s)$. Another approach to update the value is *bootstrapping*, which describes the process of continuously updating the state value estimation while iterating across the states in an episode:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{2.4.14}$$

α is defined as the *learning rate*, a quantitative measure of how strong the discrepancy between current value and the new guess is weighted for the update. The learning rate $0 < \alpha < 1$ has an influence on the convergence speed. A small size results in slow convergence while choosing it higher means that the algorithm always replaces the current estimate with a new guess [1, p. 122-123]. As a result of bootstrapping, the agent only needs to wait until the next time step $t + 1$ and has not to wait until the end of the episode for updating the value $V(S_t)$. The idea to use an estimated value is called Temporal Difference (TD) learning. As defined in equation 2.4.2, the expected return is defined as the expected value over the following states. Using only the estimation $V(S_{t+1})$ of one step ahead for the update of $V(S_t)$, as described in equation 2.4.15, is called *one step TD* or TD(0).

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.4.15}$$

Within the scope of this work, the network should be optimized with an efficient, data-driven and model-free approach. Therefore, algorithms which are based on TD-learning are now discussed in more detail.

2.4.3 On- and off-Policy Methods

In comparison to learning the state-value function like *TD*(0) as shown in equation 2.4.15, another option is to update the estimate $Q(S_t, A_t)$ of the action-value function $q(s, a)$ to obtain a policy.

It can be distinguished between two types of methods. *On-Policy* methods estimate the value of a policy and use it for control at the same time. In contrast, *Off-policy* methods follow a behaviour policy and learn how to improve the target policy. In other words, when using an On-Policy method, the Q-value is computed by a certain policy and is followed by the agent. For an Off-policy method, the estimated action value $Q(s, a)$ is computed in accordance to a different policy, so an no-optimal policy while learning about optimal action-value function q_*.

SARSA

The On-policy method SARSA tries to estimate the action-value function $q_\pi(s, a)$ which is performed by the agent following policy π including the exploration steps. The term SARSA is derived from the quintuple of events $< S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1} >$ describing the transition from one state-action pair to the next. In equation 2.4.16, the state-action pairs are updated by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{2.4.16}$$

It describes the experience the agent gains when he was in state S_t taking action A_t what results in being in state S_{t+1}, receiving reward R_{t+1} and continues with action A_{t+1}. The new gained experience leads to an update of value $Q(S_t, A_t)$ of $\alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})]$.

Q-Learning

In 1989, Watkins developed the off-policy TD control algorithm Q-learning [33][34]:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.4.17}$$

It is an Off-policy algorithm, so $Q(s, a)$, the learned action-value function, approximates directly the optimal action-value function q_* independently of a policy. The part $\max_a Q(S_{t+1}, a)$

shows the property of an Off-policy algorithm that the Q-value $Q(s, a)$ is computed according to a different policy, in the case of Q-learning a greedy one by taking the maximum action-value of the following state $\max_a Q(S_{t+1}, a)$ as an approximation for its value. This simplifies the application and analysis of the algorithm because the value is estimated by the most promising action and the actions are selected by the behaviour policy.

Comparison

Both algorithms are very similar, as shown in the algorithms 1 and 2. The main difference between SARSA and Q-learning is how the expected value of a next state is determined.

Algorithm 1 SARSA

Initialize $Q(s, a)$ with zeros
for each episode **do**
  Initialize s
  Choose a from s using policy π
  repeatfor each step of an episode:
    Take action a
    Observe r, s'
    Choose a' from s' using policy π
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'; a \leftarrow a'$
  until s is terminal
end for

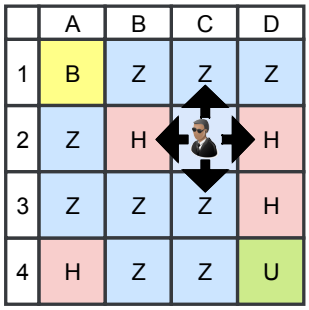
Algorithm 2 Q-Learning

Initialize $Q(s, a)$ with zeros
for each episode **do**
  Initialize s
  Choose a from s using policy π
  repeatfor each step of an episode:
    Take action a
    Observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
  until s is terminal
end for

Using the Q-learning algorithm, it is assumed that the value of the next state can be approximated by its highest Q-value. In comparison to just selecting the highest $Q(s, a)$, SARSA considers the Q-value selected by the policy π, the agent follows.

Tabular Learning

The algorithms, the agent can use to learn about the quality of respective actions in certain states were described before. To enable him, to learn from his past actions, the Q-values for each action in all possible states should be saved. In the case of discrete actions or states, the values can be saved in a table. The corresponding table for the previously introduced example in section 2.4.2 is shown in figure 2.4.



$Q(s,a)$	a_1	a_2	a_3	a_4
s_{A1}	$Q(s_{A1}, a_1)$	$Q(s_{A1}, a_2)$	$Q(s_{A1}, a_3)$	$Q(s_{A1}, a_4)$
⋮			⋮	
s_{C2}	$Q(s_{C2}, a_1)$	$Q(s_{C2}, a_2)$	$Q(s_{C2}, a_3)$	$Q(s_{C2}, a_4)$
⋮			⋮	
s_{D4}	$Q(s_{D4}, a_1)$	$Q(s_{D4}, a_2)$	$Q(s_{D4}, a_3)$	$Q(s_{D4}, a_4)$

Figure 2.4: Representation of table entries for the frozen-lake example.

The number of elements $|\mathcal{Q}|$ in the table depends on the number of states $|\mathcal{S}|$ and the pos-

sible actions $|\mathcal{A}|$.

$$|\mathcal{Q}| = |\mathcal{S}| \cdot |\mathcal{A}| \tag{2.4.18}$$

Saving his experience gives the agent the possibility to learn from its actions. Nevertheless, the agent should be able to explore the environment, but at the same time exploiting the knowledge he already gathered.

2.4.4 Exploration vs. Exploitation

As there is no specification in the algorithms what the agent should do, the possible actions are categorized into two types [35, p. 472-473], *exploration* and *exploitation*. The agent can exploit his knowledge by choosing the actions that are estimated to maximize $Q(s, a)$ and thereby G_t. Another important part is to maintain sufficient *exploration* in order to construct a more accurate estimate of the optimal q-function q_*. It is desirable to force an exploratory character because of two reasons. Firstly, it should be possible to explore all possible states, to find the global maximum and to not converge into a local minimum. Secondly, in a dynamic environment, the system should react to changes in an efficient manner. So the a continuous exploration of different states in the environment should be provided. But at the same time, a strong exploration can lead the agent performing actions that result in a lower reward. A measure for the expected decrease of the reward due to not acting optimal is given by the *regret* [36]. It defines the difference between the gained reward from the agent's policy to the highest reward that can be earned by behaving optimally. The exploration can take place by using different strategies. A good overview and comparison in a dynamic environment is given in [37]. In this thesis, the focus lies on the three commonly used exploration strategies.

ε-greedy

The first approach is *ε-greedy* search. Greedy is used in computer science to describe algorithms that take the choice that looks optimal at the moment [31, p. 415]. The decision is based on the hope that the locally best choice leads to a global maximum, but can lead to misconceptions. So it is useful to implement randomness to preserve the exploratory character. The probability ε determines whether a random action is selected. With a probability of $1 - \varepsilon$, the best action in terms of the highest expected value v_*, is chosen.

$$\pi(s) = \begin{cases} \arg\max_{a \in \mathcal{A}(s)} Q(s, a), & \text{with probability } 1 - \varepsilon \\ \text{select random action}, & \text{otherwise} \end{cases} \tag{2.4.19}$$

Choosing the highest valued action, previously referred to as greedy, is called *exploitation*. Often it is not desired to explore infinitely what can be achieved by a decreasing ε over time, called *annealing*.

Softmax Function

The second widely used exploration strategy is the *softmax* method, which is also known as Boltzmann distribution. It converts the values into probabilities for each action of the states and samples over the results. The softmax function, shown in equation 2.4.20, can

be controlled via a parameter called *temperature* τ.

$$p(a|s) = \frac{\exp(Q(s,a)/\tau)}{\sum_{b \in \mathcal{A}(s)} \exp(Q(s,b)/\tau)} \tag{2.4.20}$$

A high τ leads to a more explorative behaviour and a low τ favours the actions with higher values. As mentioned before, it is often desired to decrease exploration over time. This can be achieved by continuously reducing the value of τ, as mentioned before this process is called annealing and allows to move smoothly from exploration to exploitation [30, p. 525-526].

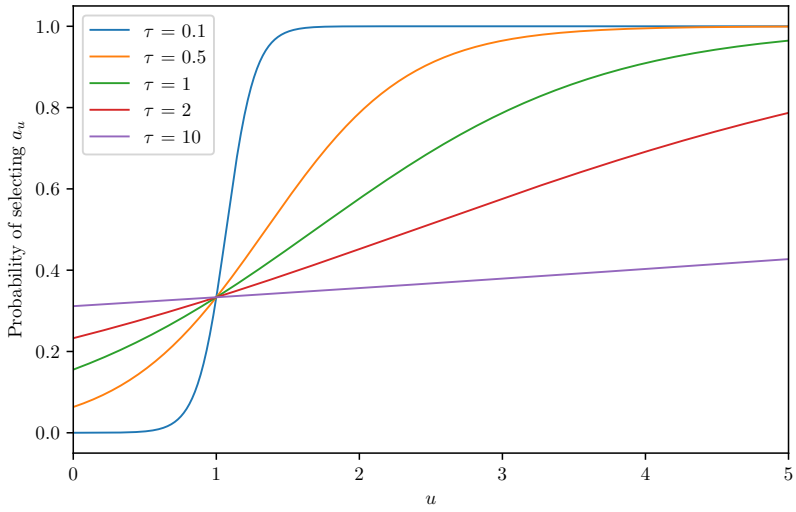


Figure 2.5: Influence of different values for τ on the probabilities.

In figure 2.5, an example for the probabilities that were calculated from the action-values $Q(s,a)$ with different temperatures τ is shown. For the action-values, an array $[u, 1, 1]$ is used with varying one value u in a range of $u \in [0, 5]$. It shows that the probabilities of taking action a_u rises with a higher action-value $u = Q(s, a_u)$. How greedy the action is chosen depends on the temperature τ and $\tau = 10$ would lead for instance to a relatively strong exploratory character, while $\tau = 0.1$ results in an exploitative character.

Upper Confidence Bound

An enforced exploration is desirable to discover and evaluate all possible states. Therefore, actions with uncertain outcome should be prioritized simultaneously to choosing the ones with the a greater expected value. Upper Confidence Bound (UCB) is based on this principle, which is called *optimism in face of uncertainty* [1, p. 35-36]. To do so, the number of times an action was selected is included. Thereby, the actions are additionally valued by their potential to be beneficial. The action selection relies on the Q-value and a bonus b^+, also called *variance*, which is a measure of the uncertainty of $Q(s,a)$. It is defined as a relation between the natural logarithm of the total number of visits $N(s)$ and how many times $N(s,a)$ action a was chosen in state s. If an action a in a state s has never been chosen, meaning

$N(s,a) = 0$, then this action is selected immediately.

$$\begin{aligned} A(s) &= \arg\max \left(Q(s,a) + cb^{+} \right) \\ &= \arg\max \left(Q(s,a) + c\sqrt{\frac{\ln N(s)}{N(s,a)}} \right) \end{aligned} \tag{2.4.21}$$

The degree of exploration can be adjusted with the help of parameter $c > 0$. With a higher value of c, the bonus b^{+} is given more leverage what results in a more exploitative character.

2.4.5 Summary & Comparison

Above, the fundamentals of RL were explained, including model-based and model-free learning and exploration methods. In this section, components which are necessary for a model free RL will be applied to the frozen-lake example described before.

Exploration Methods

The previously described methods that enable the agent to explore the environment autonomously can be adjusted by different parameters. For the ε-greedy method (2.4.4), the ε is the probability to take a random action instead of the most promising (i.e. with the highest Q-value $Q(s,a)$). In the case of softmax funtion (2.4.4), which converts the Q-values to a probability distribution, the temperature τ controls the tendency to exploration. As described in 2.4.4, the bonus b^{+} in Upper Confidence Bound method depends on the number $N(s,a)$ how often an action has been chosen in relation to the total number of state visits $N(s)$. For UCB, the parameter c regulates the degree of exploration. In figures 5.9, 5.10 and 5.11 in the appendix, the resulting reward over episodes using different exploration strategies is shown. The learning takes place over 50000 episodes and the average of 200 iterations is plotted. The lower plot shows the average reward until 1000 episodes. When the agent starts to learn, the average reward is negative, because he falls into holes while exploring. What can additionally be seen for all exploration strategies is that a stronger exploration leads faster to finding the goal state which returns a reward of 5. In the upper plots, the average reward over a longer learning time of 50000 is shown. In figure 5.9, the plot for ε-greedy, can be seen that on the long run, a strong exploration can lead to less reward due to the incentive to discover more, what can lead to falling into the holes. This shows that exploring and exploiting is a trade-off situation and the parameter τ, ε or c needs to be selected carefully to achieve a balanced problem solving. Over time, the agent learns a policy, so how to act properly. The $Q(s,a)$ give the agent a value for each of the possible actions. In figure 2.6 on the left, the resulting $Q(s,a)$ for the frozen lake environment are shown after 10^5 episodes as an average over 100 iterations following the ε-greedy strategy with an $\varepsilon = 0.1$. On the right side, the rewards related to each state and the way the agent would take following the policy $\pi(s) = \arg\max_a Q(s,a)$ with the are demonstrated. The example shows three ways that have the same length, so the agent learns a policy π in which all paths are treated equally.

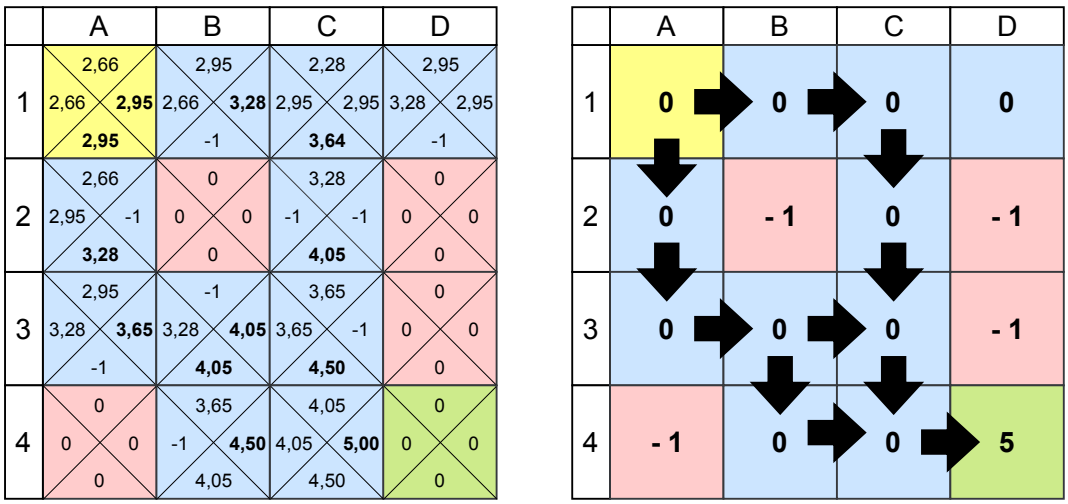


Figure 2.6: On the left: Resulting Q-values after $5 \cdot 10^4$ episodes, on the right: reward for each area and the resulting optimal paths.

It shows how the expected value is reflected in the Q-values. The reason that the values that hold the agent on the optimal path are not the same as the resulting reward is because the agent follows the ε-greedy policy. To give an example how the expected $Q(s, a)$ can be calculated by hand, the state s_{B4} and action a_3 is chosen. It is assumed that the Q-values for the next state are already calculated and $Q(s_{B4}, a_2) = 0$ in this time step. To ease this example calculation, inappropriate values for the discount factor, $\gamma = 1.0$ and for the learning rate $\alpha = 1.0$, are chosen. As explained in 2.4.2, a $\gamma < 1$ should be chosen for guaranteeing convergence.

$$\begin{aligned} Q(s_{B4}, a_2) &= Q(s_{B4}, a_2) + \alpha \cdot (r + \gamma \cdot (1 - \varepsilon) \cdot \arg\max_a(Q(s_{D4}, a)) - Q(s_{B4}, a_2)) \\ &= 0 + 1.0(0 + 1.0(1 - 0.1) \cdot 5.0 - 0) = 4.5 \end{aligned}$$

With a learning rate of $\alpha < 1$ or $\gamma < 1$, the Q-value converges with the times the action is taken to the calculated value.
This simple example shows the potential of RL to solve optimization problems and deliver a framework to learn to predict the consequences of certain actions in a state. In this thesis, these properties should be applied to find an efficient routing policy in Software-Defined networks.

2.5 Related Work

In the scope of this thesis, a data driven approach for routing based on RL is developed. Combining machine learning methods with routing and traffic engineering were early in the focus of research, beginning in the 1990s by Boyan and Littmann with their *Q-routing* [38] algorithm. At this time, no central observer existed in a communication system[39]. This changed with the introduction of the concept of Software-Defined Networking. For an SDN controller it is possible to monitor the metrics of the network and directly influence the network by modifying the flow table. Due to the larger amount of information available, it is possible to perform a more precise optimization according to different objectives. On the one hand, this led to approaches that were applied to different models and solution methods. On the other hand, the focus is on novel approaches based on machine learning

methods such as supervised or reinforcement learning. First, the different current methods for traffic engineering and routing are briefly described. If existing, they are followed by their implementation into SDN networks. Finally, approaches based on machine learning are presented. For routing, as described in 2.2.2, it is necessary to distinguish based on the information base. As a reminder, in a network, if a forwarding device gets the view over the network only based on its neighbors and it performs it routing decisions on that, we refer to distance-vector algorithms. If the routing is based on the whole topology, which can be converted into a graph, it is called link-state routing. The focus lies clearly on approaches and protocols that are classifiable as link-state routing, due to the centralized authority and view of the SDN controller. There are different approaches to solve the routing problem formulated in 2.3. How the routing is performed depends also if it is performed when a new flow joins the network, so if a connection is built between hosts. That happens if a user to server communication or even a server to server communication is demanded. Therefore this type is called *on-demand routing* in this work. Another type is to determine the optimal path configurations before or after the routes have been established, referred to as *pre- and postcomputed*. At last an overview of applications of machine learning, especially Reinforcement Learning, in SDN is given.

2.5.1 On-demand

When a new route (i.e. flow) needs to be created through the network, the requirement is that the route is calculated as fast as possible. Because of this, the common routing protocols OSPF [16] and IS-IS rely on fast path search algorithms like Dijkstra [40]. If the route is only calculated based on the Hop-Count, the algorithm ignores fully the conditions in the link and metrics such as the delay or the available bandwidth. As it is possible that the least hops do not mean the least end-to-end delay, that could lead to inefficient routing decision. Ignoring the link conditions can result in congestion or as it would be the case for TCP connections, in lower throughput. To tackle these problems, as mentioned before, the QoS extension have been introduced [18] but they it is often not implemented due to the interference of other network parts and routing loops. A more advanced approach is to expand the shortest path routing with adding constraints.
The so called Constrained Shortest Path (CSP) routing minimizes or maximizes a an end-to-end metric while being constrained by another metric. That can be extended to the Multi-Constrained Shortest Path (MCSP) routing which has to comply with bounds of multiple metrics. Joksch delivered in [41] the first formulation of the CSP problem based on integer linear programming. Therefore it is possible to solve it by dynamic programming [42].
A simple approach to solve the CSP and MCSP problem is the Fallback algorithm, introduced in [43]. It computes the path with the least cost and then verifies if the constraints have been met. The algorithm continuous to calculate paths by exchange the metrics and constraints until a suitable path is found. Another approach is to enumerate through all possible combinations of paths and select the best solution that satisfies all the constraints. This would lead to high computational complexity. In [44], Aneja *et al.* use implicit programming [45, p. 297-300] which gives the possibility to evaluate systematically all solutions without the necessity of evaluating all of them explicitly by fixing them.
An algorithm called Constrained Bellman-Ford (CBF) is introduced in [46]. The algorithm has the capabilities to discover possible paths to a set of destination nodes while complying with the destination's constraint. In contrast to it's name, the algorithm is based on the breadth-first search (BFS) search algorithm [31, p. 594-597]. The algorithm finds independent minimum cost paths between one source and a set of destination nodes subject to each destination's delay constraint. As CBF returns a set of possible paths for each destination node, it can be used for the subsequent optimization. Another advantage is that paths to multiple destinations are returned. In the case of unicast connections, in the case of for example a server and multiple clients, it could relieve the burden of calculating the paths for each client separately. Especially for multicast connections, the usage of CBF would be beneficial.
Another approach to solve the MCSP problem is an algorithm called A*Prune [47], which was developed by Liu and Ramakrishnan based on the A* (A-star) search [48]. A* relies on a guess-function which defines the nodes chosen which are expected to lead to the goal as fast as possibles. In A*Prune, it is assumed that a guess function can be adopted for all cost and constraints, in other words the network metrics. It constructs a heap [31, p. 151-152], so a data structure which also defines the priority of the data. From this, a priority queue can be derived that is then used for path discovery. The constraint values are projected and if they exceed their end-to-end bound, they are removed from the queue. After all, when the destination node is reached, the MCSP is found. One major advantage of A*Prune is that it finds MCSPs to other destination nodes on the way and the search can also be extended to find even more complying paths to other nodes.
Other approaches are based on Genetic Algorithms, an approach inspired by nature to solve complex optimization problems. In [49], a genetic algorithm is used for the basic routing problem in networks. Ahn [50] and Hamed [51] are using this kind of algorithms for solv-

ing the routing problem with constraints (i.e. CSP). For another kind of solutions, the Ant Colonization Optimization method [52] is used. It is derived by the idea that a collection of ants can find efficiently the shortest path to a source of food and then informing other ants by the deposition of pheromones. This behaviour can be applied, as shown firstly by Colomi in [53], for determining the shortest path in a network. An approach presented by Di Caro [54] does apply the optimization based on the behaviour of ants to the routing in packet switched networks. There are plenty of solutions to find the shortest path based on Genetic Algorithms, Ant Colony Optimization or other optimization methods, but this would be out of the scope of this work.
A good overview for unicast routing algorithms in Software-Defined Networkings delivers [55]. However, QoS routing algorithms are commonly greedy, meaning that they attempt to find a path that meets with the particular constraints, but ignoring the impact of their decisions on the whole network.

2.5.2 Pre- and postcomputed

The state of the network depends on the flows and their required bandwidth. If a new flow joins, the network does not have the knowledge how much bandwidth the flow wants to obtain. It is also possible that traffic rates change over time depending on the user behaviour. Therefore it is beneficial to optimize the network with the metrics of its current state. Several objectives are common to be chosen for the optimization. That includes Congestion Minimization, End-to-End delay Minimization , Packet Loss Minimization, network utility maximization or even Energy consumption Minimization [56].
In this work, the focus lays mainly on the congestion and End-to-End delay minimization. To prevent Congestion, different techniques can be applied. One is that different streams share the network resources by minimizing the utilization of the links by distributing the traffic over the network and avoiding congestion. To achieve a higher accuracy, the traffic flows could be split up on different paths.
In [57], Wang *et al.* calculate link weights for arbitrary flow splitting by using linear programming. Another is to block the access to congested network resources (i.e. links or forwarding devices). If a new demand is processed, congested resources are not taken into account for the path selection. Congestion would mean a higher end-to–end delay (i.e. latency) and packet losses. In other words, congestion minimization has a direct beneficial impact on the other two objectives and can considered as an useful general objective. But it also has to be noted that the distribution of the traffic can lead to higher latency, so depending on the situation, it can end up in a trade-off. As a result, proposed approaches generally concentrate on one independent objective.
Wang *et al.* propose in [58] a linear programming formulation to prevent congestion by minimizing the maximum of link utilization. In [59], Trimintzios *et al.* formulate a the problem in a way that they optimize for a lower delay and prevent overloading parts of the network at the same time. Additionally, constraints on the hop count and packet losses have been added to ensure meeting the QoS requirements.
Next to the solutions based on a multi-constrained problem formulation, others focus on modelling the network with queuing theory [60][61]. Using queuing theory, the complexity grows with the size of the network, therefore multi-hop and multi-point to multi-point networks are still open problems [9]. In addition, traffic is often assumed as a distribution, often the Poisson distribution [10]. However, this does not always correspond with the reality in complex networks such as the Internet where user demands and link usages are hard to predict.
A detailed summary of the application of different approaches to solve TE problems can be found in [21] and their application for SDN in [62].

2.5.3 Industrial Applications

With a rising traffic requirements through video streams and cloud application led to an increasing number of data centers of content providers distributed around the globe. These datacenters are connected via separated WAN networks. These networks are usually expensive and the content provider, usually considerably large enterprise, tend to achieve a high utilization for cost savings.
Jain and his colleagues at Google describe in [5] how they move away from costly overprovisioning, in their case 2-3 times the necessary bandwidth, to a near perfect utilization of their backbone network B4. Their network can handle standard routing protocols and has a TE application running on a SDN controller. It splits application flows up (i.e. Multipath forwarding) and allocate these while balancing priority and demands of the different application. An algorithm based on max-min fairness [63][64, p. 8], a model for the fair allocation of network resources in which it is tried to maximize the minimum rates. But the algorithm was extended to additionally satisfy the throughput demand of certain applications with prioritization and it can dynamically relocate bandwidth if link failures occur. Their success by achieving near 100% utilization in some links and the associated cost savings led to more research attraction on OpenFlow and SDN [56]. Microsoft has similar as a content provider similar to Google's B4, a backbone network connecting his datacenters.
Therefore, Huang *et al.* developed a system, called SWAN [6], to improve the utilization depending on the current traffic demand. It satisfies its set goals of max-min fairness and strict priority classes for applications by coordinating the sending rates of its services and a beneficial allocation of the data flows. It derives the flows into different classes and reserves the shortest path always for the ones high priority. Flows with lower priorities can be disaggregated (i.e. split up) or reallocated to achieve a high utilization. To prevent congestion due to incoming demands or ongoing traffic allocation, the algorithm leaves a scratch space (i.e. free space dedicated to temporal use) in each link. With the usage of their developed algorithm, they achieved a 60% higher utilization.
Another content provider which delivers different types of media with the help of distributed datacenters is Facebook. Edge Fabric [65] is an SDN-based system presented by Schlinker *et al.* which manages the traffic between their points of presence, in other words their datacenters. The main goal of Edge Fabric is to avoid congestion by monitoring the network state and allocate the traffic on alternative paths. The strong interest of reputable companies in this topic shows its relevance and the demand for a suitable solution.

2.5.4 Machine Learning

The algorithms and approaches described previously are on the one hand based on optimization with constraints or on complicated models that scale poorly. Machine Learning techniques have shown in previous publications and use cases that they can handle relatively large optimization problems.
One common example especially for Reinforcement Learning is the mastering of different games such as Go or Chess [66]. The special aspect is that they teach themselves the games, so they are capable to play effectively without the development of an algorithm. In combination with deep neural networks (i.e. Deep Reinforcement Learning (DRL)), it can even handle arcade games [67]. Other areas include autonomous driving [68], robotics [69] and the financial sector [70]. Also classic control problems such as temperature regulation of facilities [71] can be managed successfully. A comprehensive overview can be found in [72]. The advantages of learning to behave effectively in an environment without requiring a model suits well for the dynamic and complex behaviour of todays networks. In the following para-

graph, approaches that use machine learning for routing or traffic engineering are introduced.
An early occurrence of an application of machine learning for routing was the approach of Boyan and Littman [38]. In their algorithm, called *Q-routing*, they use the Q-learning algorithm, as introduced in section 2.4.3. A reinforcement learning module (i.e. a RL agent) is implemented into each node of the network. Information and statistics about the delivery times were gathered, stored locally in each node and used as a reward. By implementing the Q-learning algorithm, each node estimates the delivery time in the number of hops of a packet until it reaches its destination when sending it to a specific neighbor node. For evaluation, a discrete network simulator which can forward one packet each timestep. The Q-tables are predicting the delivery time, showing the similarities to the Bellman-Ford algorithm, that also keeps a cost function in a table. In Q-routing, the Routing policy is characterized by the Q-values stored in the Q-tables of each node in the network. For exploration, exploration algorithms were not used intentionally because of their negative influence on the network state by provoking congestion by forwarding packets randomly. Instead a method called "full echo" was introduced. In this method, a node requests to its neighbors when it has to make a forwarding decision. The neighbors then return a number that characterizes the neighbors estimate of the time the packet would need reaching the destination. Using this kind of information exchange shows clearly the similarities to the Bellman-Ford algorithm. Their results indicate that an algorithm based on Q-learning performs equally well under low load and even better than shortest path algorithms under high load.
Choi and Yeung in [73] or Peshkin and Savova in [74] provide further developments of Q-routing to improve its performance under high and low load. The trade-off is the short time of inefficiency when the agents learn the optimal behaviour, so how to forward packets in the most beneficial way. One drawback is the usage of local information and the dependence on integrity of the other nodes.
Global knowledge about the network metrics can gained by using an SDN controller. It follows a review over approaches to use the available monitoring and control capabilities for more effective routing and Traffic Engineering.

Based on Supervised Learning

One strength of Machine Learning is its ability to predict possible future states using the current information base. The approach by Azzouni *el al.* called Neuroute [19] uses this advantage in their framework for dynamic routing in SDN networks. The system is divided into three parts, an estimator for the traffic matrix, a traffic matrix predictor and the so called traffic routing unit. The estimator is based on OpenMeasure [75], which aims to retrieve the current network state in an efficient way by learning to optimize the placement of the measurement resources in the network. Based on the measurement, a neural network is trained to save the computed paths. Doing so, routing configuration can be saved in a match to the current network state.
A similar approach is presented in [76]. It introduces a supervised learning framework which is trained with previously computed paths showing like Neuroute that neural networks are capable to match routing decisions to network states. But the quality of a trained network depends strongly on the variety of the input data and the presented solutions are not capable to learn the behaviour of an environment on their own.

Based on Reinforcement Learning

Reinforcement Learning, on the other hand learns through interaction with the environment and consequently does not require any dedicated training data. In [77], Hu and Chen added to each network node a supervised reinforcement learning agent [78], which has the objective to balance the arriving traffic to prevent busy links. Supervised RL relates to the idea to speed up the learning by adding an additional entity which influences the agent additionally to the environment. As reward function, the distance to the destination in Hops and the relative load in the next link is chosen. A RL approach is used by Chavula *et al.* in [79] to optimize UbuntuNet, an education network in Africa. Each switch is seen as a state s with each connected neighbour switch as next state s'. The reward is calculated from a function of delay, capacity and number of flows between combinations of two switches. Additionally, the system was tested using multiple paths per flow with the Q-values as splitting ratios. The evaluation was not compared to other routing approaches and using the number of flows as a parameter for the reward calculation is not practical.
Xu *et al.* propose in [80] a data driven approach based on Deep Reinforcement Learning. As states, the performance parameters throughput and delay are chosen. The actions are split rations of the flows per link. They optimize for network utility maximization in a combination with α-fairness [64, p. 13 - 17] that is another model of the fairness in a network in which α is used as a trade-off between efficiency and fairness. Used for solving the problem, a mathematical framework called Gurobi Optimizer[5] were used. There are several things to consider. The approach does not solve the optimization problem on its own and uses mathematical programming for solving the network utility maximization problem. It takes the task of mapping calculated solutions for the problem to specific network states. Additionally, splitting paths can lead to better solutions, but a practical implementation like e.g. Multipath TCP [81] can be complicated in real life scenarios. It delivers comprehensive results in simulated environments, such as the used packet simulator ns-3 [82], but the implementation would be more complicated in a real setup (i.e. hardware).
The focus of [83] lies on optimizing the routing for SDN networks in a multi-layer hierarchical manner. It attempts to provide a solution for real networks such as the Internet, which are divided into individual Autonomous Systems, as described in section 2.2.1. Therefore, the control plane is divided into three different layers, one for the so called Super Controller, followed by Domain Controllers which retrieve the network metrics by Slave Controller that are placed in the third layer. Inside of a AS (i.e. Interdomain), the Domain Controller is responsible for the routing. If the communication takes place between or via different ASs, the Super controller is consulted, which will decides about the global forwarding direction. When a route for a new flow is demanded, the routing is calculated on a hop per hop basis. Each switch is defined as a state. The forwarding decision, in other words to which neighboring switch the packet should be sent, is defined as the action. For the action selection, the softmax policy, as described in section 2.4.4, is used and SARSA as learning algorithm. The reward function consists of different QoS aware functions: the packet loss, available bandwidth, Queuing delay in the current switch and the summed up transmission delay (i.e. the link latencies) till the destination. The influences of the individual components can be weighted by varying factors. In the evaluation, the approach is compared to Q-routing [38], which is very similar. Like Q-routing, the algorithm does not use all the knowledge available to the controllers through the SDN architecture. Additionally, the forwarding packet per packet, is not suitable to realistic networks. There is no guarantee that packets arrive in order, what can result in a collapsing congestion window for TCP [84]. To compute the forwarding decision for each packet could overload the switches and congestion if the switch-controller response delay is included.

[5] Gurobi Optimizer, https://www.gurobi.com

In [85], Al-Jawad *et al.* propose the RL-based framework LearnQoS. It attempts to learn a policy that optimizes according to the QoS requirements. As the state, the current traffic matrix is defined. The traffic in the links is measured by the SDN controller by requesting the switches for statistical information. Additionally, information about the throughput of the flows, the delay and packet losses are gathered. These metrics are then used to check if the required bandwidth is met for the video streams, what also defines the reward. The actions are defined as followed: the rates of best effort flows (i.e. the ones with the lowest priority) can be increased or reduced, best effort flows can be rerouted or not executing any change. Tabular Q-learning was selected as RL algorithm. For the evaluation, HTTP traffic were classified as best effort and video traffic as prioritized. LearnQoS is evaluated in an emulated network with the Floodlight controller against shortest path routing and shows the potential of gaining stable throughput for prioritized flows by using RL based Traffic Engineering. In their action space, the rerouting is not specifically defined and as a result, it can interfere again with other video streams. A simpler and common solution for prioritizing traffic would be to classify traffic and prioritize it, for example by using Differentiated services [86].
An interesting approach is AuTo [87] by Li *et al.* for traffic optimizing inner-datacenter networks. Even if it refers to a different use case than a WAN or inter-datacenter network like Google's B4, their work deals with a realistic problem that short lived flows often are already gone before traffic optimizing decision could have been made. First the state space was defined as a combination of all active and finished flows, the action was defined as their queuing priority and the reward was calculated as a ratio of the average throughput of all completed flows in two consecutive time steps and thus indicates how good a decision was with regard to the overall performance. Using this system, the underlying problem was identified that the processing delay of existing frameworks for Deep Reinforcement Learning is too high to satisfy data center related traffic. As solution, two different approaches have been introduced in which the flows were categorized and queued in an effective way. An assumption has been made, called *big-switch*, which means that the network is assumed to be non-blocking, non-congested and suitable load balanced. As a result, in this work, routing and traffic engineering is ignored and the focus lies on the scheduling of flows. The interesting parts are that optimization of networks can be complicated especially in real ones due to changing traffic patterns and short lived flows which can affect running optimization schemes even if they are already gone.
In [88], Sun *et al.* proposed TIDE, a DRL-based routing system. It has the objective to determine the link weights of the network-graph under consideration of different QoS criterias. The Floyd–Warshall algorithm [31, p. 693-699] is applied on this graph to determine the shortest path. Yu *et al.* provide in [89] delivers a similar approach like TIDE with regard to the determination of link weights and then applying a shortest path algorithm. In addition, the survey of [90] describes the utilization of ML in networking. Especially for applying ML methods to SDN, [91] and [92] provide a comprehensive survey.

2.6 Conclusion

The presented approaches focus on an efficient routing of traffic flows in networks. The efficiency can be defined by different optimization goals and by the constraints which are given. For example, the minimum latency or a maximum throughput, which should be achieved without overloading the links in the network. Furthermore, it can be distinguished by the point in time when the decisions have to be made. Some methods deal with routes of a newly emerging connection, while others deal with the subsequent rerouting of flows to achieve a specific optimization objective. The routing of flows based on the Shortest Path

First paradigm, even with constraints such as CSP or MCSP do not take the influence of their routing decision on the rest of the network into account. Especially flows which require a large bandwidth can negatively influence the network. Traffic Engineering approaches are based on solving complicated multi-constrained problem formulations or models which do not scale well, often only focus on one optimization objective, and require additional engineering efforts.
This is where machine learning methods come into play, which, in addition to their foresighted character, are also capable of solving large optimization problems. However, the presented ML approaches do not offer a complete solution for routing. Instead, they each solve parts while at the same time disregarding certain other aspects of the whole problem. For instance, Q-routing [38] does not use the global knowledge of an SDN controller and additionally is not feasible for today's networks speeds. Others, such as [80], assume the arbitrary splitting of traffic, which is not feasible in reality. The approaches [88] and [89] generate a weighted graph on which shortest path routing algorithms are performed. This means that they still rely on classic routing algorithms and only modify their behaviour by changing the graphs. This is due to the application of DRL methods such as Deep Deterministic Policy Gradient [93], which output continuous actions. For example AuTo [87] adds the big switch assumption and therefore simplifies the network behaviour.
In short, the current approaches based on Machine Learning do not provide a complete solution for routing or traffic engineering by unrealistic assumptions or restriction to partial problems.

3 Implementation

3.1 Overview

The objective of this work is to evaluate a data driven approach by the usage of Reinforcement Learning (RL) to determine a effective routing policy. Reinforcement Learning is based on a Markov decision process, which is based on a Markov Decision Process (MDP), which consists of states $\mathcal{S}$, actions $\mathcal{A}(s)$ and feedback of the environment, the reward signal r. In the use case in this work, the environment is a Software-Defined Networking. It consists of a controller entity and several switches under its authority. The switches forward packets, which arrive sequentially (i.e. flows), on rules set by the controller. To define it as an optimization approach for the Multi-commodity non splittable flow problem, as described in section 2.3), the controller should find the optimal combinations of flows that maximize a specific metric, typically QoS criteria such as End-to-End delay (i.e. latency) or throughput. In current networks, a low latency gets increasingly important for applications like in health care, autonomous driving or industrial automation [94]. Therefore the main focus of this work lies on the minimization of the end-to-end delay of the flows.

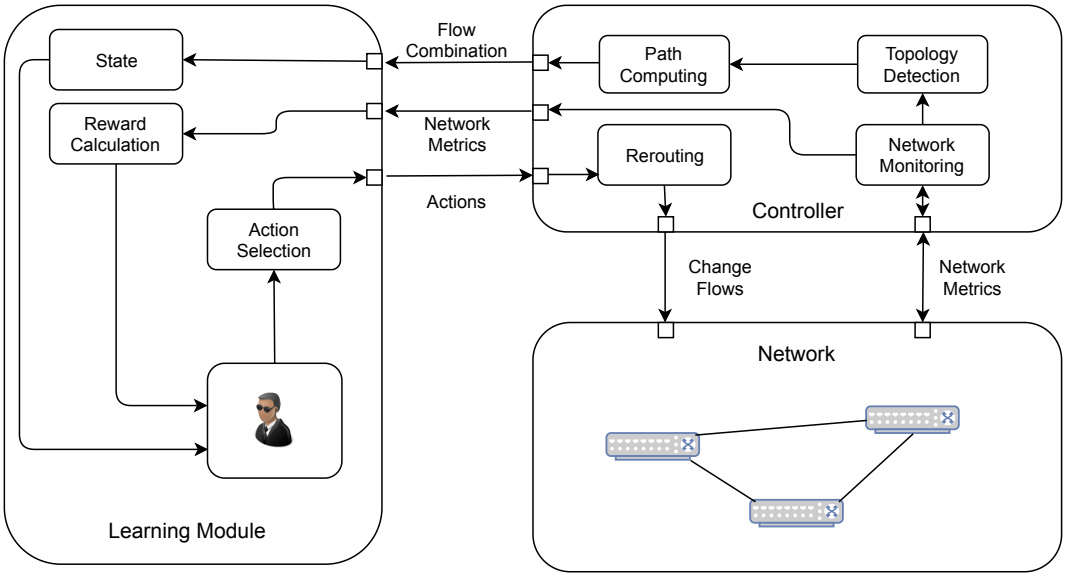


Figure 3.1: Overview of the implemented system with each of its components.

First, the implementation of the agent is described. Containing the definition of the MDP with its state space, action space and reward signal. To execute the agent's actions or gain a meaningful reward signal, an entity needs to carry out the actions or retrieve necessary network metrics. This task is handled by the SDN-controller, which measures the network metrics and translates the agent's actions into feasible modifications of the network. The composition of the controller module is described in the second part. To evaluate the behaviour and performance of the RL-agent, an emulated network is used, which is described in the third part. Figure 3.1 gives an overview of the interaction of the individual components within the overall system.

3.2 Reinforcement Learning

To create an effective agents, different design choices are necessary. As part of the Markov Decision Process, a suitable definition of the states, actions and the reward needs to be found. Additionally an effective way of maintaining sufficient exploration and using the gained knowledge beneficially is required, as described in section 2.4.4.

3.2.1 Markov Decision Process

Designing a Markov Decision Process can be a challenging task and has a major influence on the effectiveness. In particular, the definition of states $\mathcal{S}$, actions $\mathcal{A}$ and rewards $\mathcal{R}$ needs to be defined properly. The learning rate α and discount factor γ also influence the behaviour of the agent.

State space

The state space should represent the condition of the current network on which the agent decides how to act. As described in 2.3, the network consists of flows $\mathcal{F}$ between different hosts. Depending on the topology, the flows can take different paths with specific switches Sw in their way. It is defined as multi-commodity flow problem with non splittable flows, each flow $f_{h_{src},h_{dst}}$ can always only take one path.

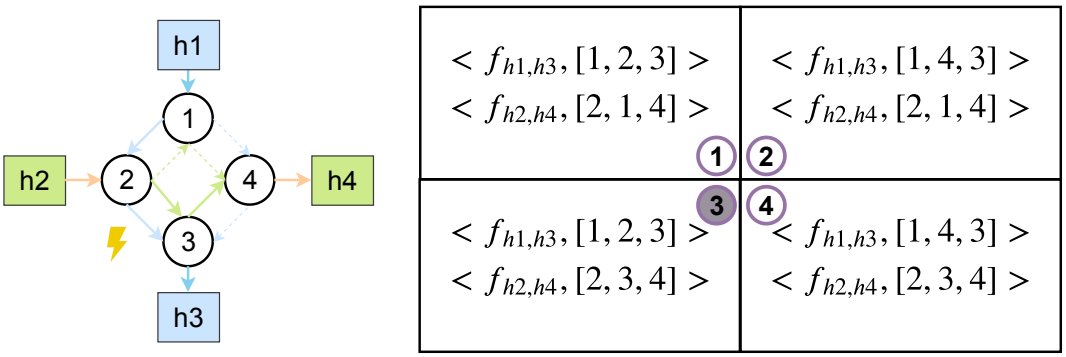


Figure 3.2: Representation of possible flow constellations in a network with four switches and two flows.

One state s is therefore defined as a combination of all tuples (i.e. an ordered pair) of flows with their chosen path $< f_{h_{src},f_{dst}}, [Sw_{src}, ..., Sw_{dst}] >$. The state space $\mathcal{S}$ is defined as all com-

binatorial options of flow to path tuples. Figure 3.2 shows an example that contains four switches, four hosts and two flows. Each flow represents a one-directional transmission between two hosts. The blue one from host $h1$ to $h3$ and the magenta colored from $h2$ to $h4$. In this example, the currently selected paths for each flow are marked as solid thicker lines. On the right, all possible combinations are shown with the currently chosen one marked.

As it is a combinatorial problem, the number of states, so the cardinality $|\mathcal{S}|$ of the state space $\mathcal{S}$ scale with $\mathcal{F}$ as a set of all flows and $\mathcal{P}(f)$ as a set of all possible paths a flow f could take:

$$|\mathcal{S}| = \prod_{f \in \mathcal{F}} |\mathcal{P}(f)| \tag{3.2.1}$$

In the case of the example shown in our If in an assumption, both flows combined need more bandwidth than the capacity of the link between switch Sw_2 and Sw_3, the constraint 2.3.2 would be broken and result in congestion.

Action space

The agent can interact with the environment through different actions. In finite MDPs, the value (i.e. the expected return) of a specific action a in a state s is defined as $q(s, a)$, with its estimate $Q(s, a)$. To influence the network, the possible actions that the agent can carry out through the SDN controller are changing the paths of the flows. Thereby the flow tables of the switches are modified. The presented actions also result in a change of the state transition probability as described in section 2.4.2. The state transitions end up being deterministic, meaning $p(s'|s, a) = 1$. Two kinds of actions are considered, the *direct change* and the *one flow change*. A direct change would give the agent the opportunity to go from one state straight into another. For the previous example (figure 3.2), the possible actions for each action mode are shown in figure 3.3. In the case of a direct change, the agent in state 3 could change to all other states 1, 2 and 4.

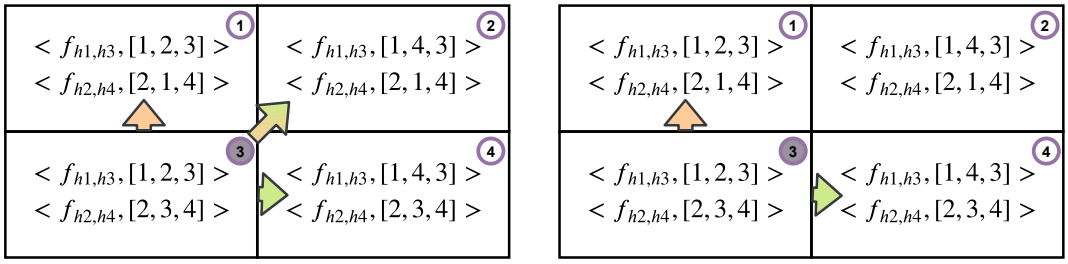


Figure 3.3: Possible actions with direct change on the left and one flow change on the right.

If only one flow can be changed, as shown on the right, the agent could perform the transition from state 3 to states 1 and 2. The task of the agent is to find the combination of flow to path mapping that optimizes the system to a particular target. Since it is a process that does not end with reaching a pre-defined goal or terminal state, a no-transition action *NoTrans* is introduced. This gives the agent the ability to remain in a state that was perceived as optimal by the agent. As can be seen from the simple example in figure 3.3, both variants scale differently in terms of the total amount of actions in each state $|\mathcal{A}(s)|$ and as a result the number of possible Q-values for the whole system $|\mathcal{Q}|$, for the tabular case the number of Q-table entries. For the direct case, the number of actions for each state is the size of the state space without the current one $|\mathcal{S}_{-s}|$ plus the No-Transition action, so the size of the

state space:

$$|\mathcal{A}(s)| = |\mathcal{S}_{-s}| + 1 = |\mathcal{S}| \tag{3.2.2}$$

$$|\mathcal{Q}| = |\mathcal{S}|^2 \tag{3.2.3}$$

As a result, the total number of actions and with it the size of the Q-table, scales polynomial depending on the state space. This increases the time it takes the agent to evaluate all actions. To reduce the increase in the number of Q-values, the actions are limited to the respective flows. The learning agent can change the path for each flow of the state space $p(f)_s \rightarrow p \in \mathcal{P}(f)_{-s}$. In figure 3.3, the one flow change is shown on the right side. The number of actions of each state $\mathcal{A}(s)$ depends on the number of possible paths for each flow:

$$|\mathcal{A}(s)| = \sum_{f \in \mathcal{F}} (|\mathcal{P}(f)| - 1) + 1 \tag{3.2.4}$$

$$|\mathcal{Q}| = |\mathcal{S}| * |\mathcal{A}(s)| \tag{3.2.5}$$

In addition to reducing the total number of Q-values, it also reduces the number of changes in flows that would be necessary for a direct state change. This additionally reduces the required number of flow table modifications.

Reward

To determine the approximated value $Q(s, a)$ for the action a in a state s the reward r, in other words the feedback signal of the environment as described in 3.2.1, is necessary. It defines the optimization objective and could depend on one or various QoS parameters. For the case of this work, the goal is to minimize the end-to-end delay (i.e. the latency) between the hosts of each flow. The latency between the hosts is determined by the sum of all measured latencies by the method described in 3.3.1 of all links in the currently selected path. One option for reward calculation could be to simply perform an average of all values $\overline{d} = \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} d(f)$. For gaining more fairness [95], so very high values for one or several flows are prevented. To reduce the latency and gaining additionally fairness, the quadratic mean is chosen:

$$D = \sqrt{\frac{\sum\limits_{f \in \mathcal{F}} d(f_i)^2}{|\mathcal{F}|}} = \sqrt{\frac{d(f_1)^2 + d(f_2)^2 + ... + d(f_{|\mathcal{F}|})^2}{|\mathcal{F}|}}. \tag{3.2.6}$$

Latency is a cost function, so it is desirable to minimize it what stays in contrast to Reinforcement Learning, which typically has the objective to maximize a specific metric. Therefore, the reward is defined as the negated quadratic mean from equation 3.2.6:

$$r = -D = -\sqrt{\frac{\sum_{f \in \mathcal{F}} d(f)^2}{|\mathcal{F}|}}. \tag{3.2.7}$$

Therefore, the agent tries to minimize the latency by maximizing the reward. Another option would have been to calculate the reward by creating the inverse of the delay function: $r = 1/d$, but it would result in a nonlinear dependency. To keep the dependency linear, a possibility would be to select a upper limit, such as $d_{up} = 1000ms$, to calculate the reward on the difference to the upper limit $r = d_{up} - d$. As a consequence, it would be necessary to choose an appropriate value based on the topology, the location of the hosts and the link latencies. This would mean more engineering effort and explicit knowledge about the topology. One of the advantages of the developed system is that no previously collected topology or latency information is necessary. As a result, the option of calculating the reward as the negative delay 3.2.7 was chosen.

Q-Table

As described in section 2.4.3, the Q-values (i.e. the quality of an action in a specific state) are saved in a tabular way. Due to using negative rewards, calculating the Q-value by equation 2.4.17 results in negative Q-values. The $\max_a Q(s_{t+1}, a)$ operation is performed to estimate the value in the next state by taking the highest Q-value. Generally, Q-values in the tables are initialized with zeros. Therefore, the Q-learning algorithm would always take the initial Q-values values of zero for the value estimation instead of the already calculated Q-values. To prevent that not intended behaviour, the initial Q-values are set to negative infinity $-inf$. Therefore, the non-selected values are not preferred by the $\arg\max_a$. Additionally, using a negative reward and as a result negative Q-values, additionally affects the exploration.

Exploration - ε-greedy

As described in section 2.4.4, an action is selected greedily by an arg max with a probability of $1 - \varepsilon$. By changing the initialization to $-inf$, no values of zero are falsely preferred. It could be argued that using initial Q-values of zero would lead to an enforced exploration, but it was decided to not change the purpose of the strategy.

Exploration - Softmax

Exploration based on the softmax functions maps the Q-values $Q(s, a)$ to action selection probabilities. As described in 2.4.4, the temperature τ defines the degree of exploration. As described before in section 3.2.1, a negative reward is selected what results in negative Q-values $Q(s, a)$. Additionally, the Q-table is initialized with negative infinity $-inf$, what makes a modification of the softmax exploration necessary, because using $-inf$ would result in $\sum_{b \in \mathcal{A}(s)} Q(s, b)/\tau = 0$ and therefore in a division by zero. The original behaviour of function should be conserved and hence the softmax for negative values is defined as followed:

$$p(a|s) = \frac{\exp\left[-1/(\tau Q(s, a))\right]}{\sum_{b \in \mathcal{A}(s)} \exp\left[-1/(\tau Q(s, b))\right]}. \tag{3.2.8}$$

A higher temperature still leads to a stronger exploration and reasonable choice of the size of the temperature depends on the expected Q-values. In figure 3.4, the probabilities of a varying u in a set of Q-values $[u, -200, -200]$ with different temperatures τ is displayed.

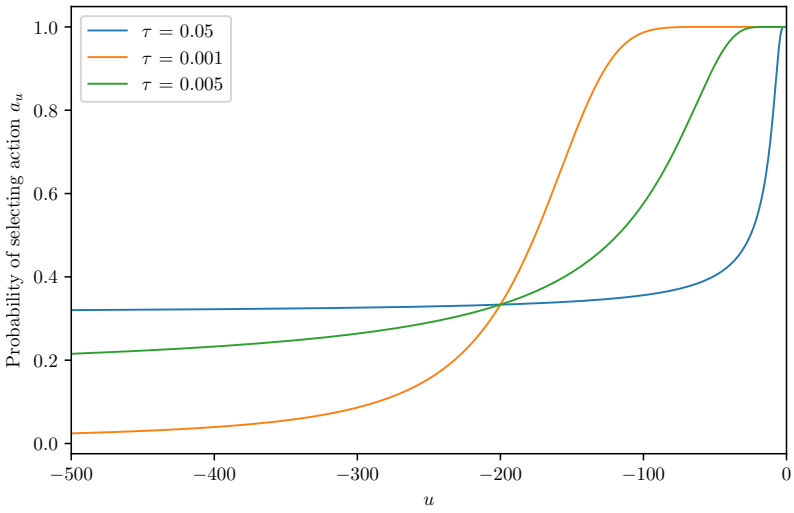


Figure 3.4: Probability of choosing action u in a set of Q-values [u, –200, –200].

To design the RL system in the right way, an appropriate temperature parameter needs to be selected depending on the expected Q-values. Figure 3.4 also demonstrates that even with negative Q-values, the property of a growing exploratory character with increasing τ remains legit.

Exploration - UCB

Using the Upper Confidence Bound exploration method as introduced in 2.4.4, the bonus b^+ provides the enforced exploration. b^+ depends on the count how often actions are selected in a state in comparison to the state visits of the agent. With larger values for $Q(s, a)$, the weighting (i.e. influence) of the bonus b^+ must also be increased. This can be achieved by adjusting parameter c. The bonus has a similar effect with negative Q-values as with positive ones.

Negative Rewards

Another option would have been to perform a minimization $\min_a Q(s_{t+1}, a)$ to gain the approximate of the next state. That would also result in changes in the Q-table initialization and the exploration methods. Initializing the table with zeros would also give the agent the intention to estimate the future rewards on zero values. In the case of the ε-greedy method, the arg max would have to be changed into an arg min. Additionally, the Softmax functions would have to be modified. For UCB, the parameter c would have to be set negative, which contradicts the definition of $c > 0$. The need for strong changes in exploration strategies has also contributed to the decision for a negative reward ultimately.

3.2.2 Merging Q-Tables

The possible states $\mathcal{S}$ and actions $\mathcal{A}$ and thus the Q-table depend on the total flows registered. Then, the Q-table then expresses which actions (i. e. which flow table changes) lead to the optimal state with the highest reward. If a new flow joins the network, the Q-table is

no longer valid because the possible states as a combination of flows changed. As a logical consequence, the Q-table would be reset, or in other words: *reinitialized*. Consequently, the agent would have to start learning from the beginning. This is not very efficient and would result in a long adaption period. Therefore the framework was extended by the feature of merging the previous and new Q-tables. The action values $Q(s, a)$ of the most similar previous combination are adopted to the new one. For the values of the action of rerouting the newly joined flow, the initialization value is set. Figure 3.5 illustrates the merging operation.

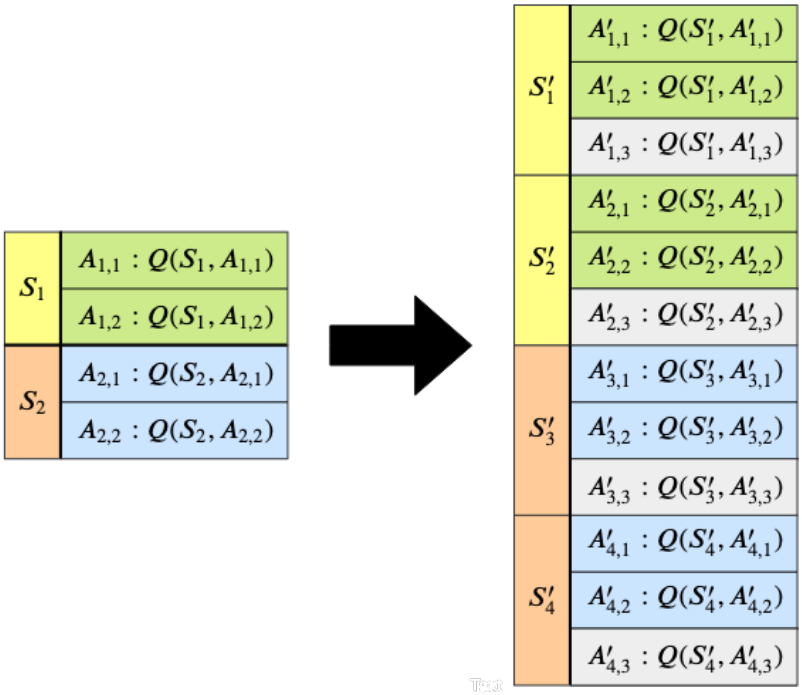


Figure 3.5: Illustration of the merging operation.

This feature was implemented under the expectation of waiving the converging time by merging in comparison of the complete relearning of the Q-table.

3.2.3 Initialization

If a new flow joins the network, the possible paths are calculated using a search algorithm with their respective costs (e.g. the latency). There are two options to route the flow directly. One is to select one path possible path randomly. This would ignore the available knowledge about the topology. The second way is to use the shortest of the calculated paths to route the flow. This initialization is the approach to route the flow in such a way that from the beginning it is close to the optimal combination and thus the convergence time is shortened. This allows to use the existing knowledge about the network metrics.

3.3 Controller

In the implemented system, the controller has two main responsibilities. It should monitor the network and provide the data to the learning module for calculating the reward. Additionally, it executes the actions which are selected by the RL-agent. In this section, the design

steps to implement the controller module are described.

3.3.1 Measuring the Latency

For obtaining the current latency in the network, an active measurement mechanism was implemented. The approach is based on the idea of Phemius and Bouet [96] in which they use probing packets to measure the delay between two switches.

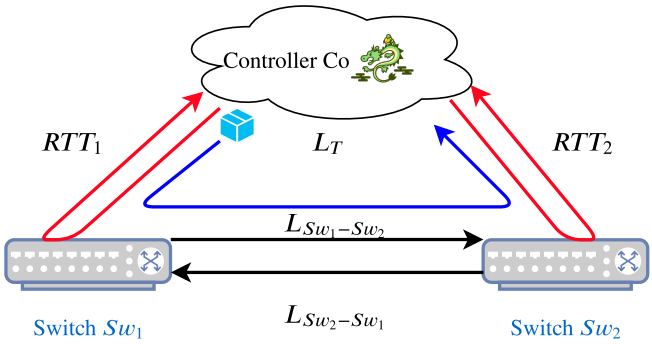


Figure 3.6: Illustration of the latency measurement mechanism.

Figure 3.6 demonstrates the measurement with a simple topology of two switches. The controller *Co* sends a probing package to a switch Sw_1 with a `PACKET_OUT` message and the command to flood it out of all its ports. The measurement package is an Ethernet frame with a value of 0*x*07*c*3 for the Ethernet-Type. This value is chosen arbitrary and is not included in the registered numbers Ethernet-Types of the IEEE 802.3 standard. Using a specific number as Ethernet-type makes the packet clearly identifiable as a probing packet for the controller. Additionally it contains a timestamp and the switch datapath identification number as payload. The unique datapath ID is given to each switch by the controller when they are connected to each other for the first time. Altogether, the size of the total packet is the minimum packet size defined by the Ethernet standard, which is 64 byte. Figure 3.7 shows the composition of the packet .

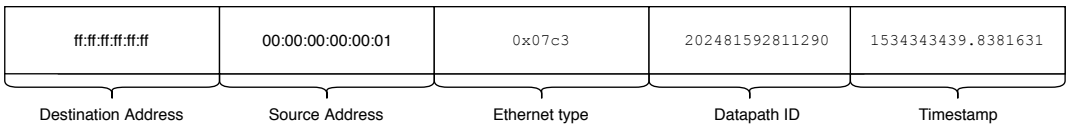


Figure 3.7: Composition of the latency measurement packet.

After Sw_1 sent out the packet, switch Sw_2 receives the measurement packet. Sw_2 does not have any table entry for that specific Ethernet-Type and sends it as a `PACKET_IN` message to the controller *Co*. When the controller receives the message containing the probing packet, it can derive the origin by the datapath ID. By the included timestamp and the arrival time, *Co* can additionally determine how long the packet required to return. The total travel time of the packet is defined as latency L_T. As determining the one-way delay (i.e. latency) $L_{Sw_1-Sw_2}$ is the objective, it can be derived by subtracting the time the packet needs to be sent from the controller *Co* to Sw_1 and from Sw_2 to *Co*, so L_{Co-Sw_1} and L_{Sw_2-Co}, respectively. Between controller and switch, an uncongested link is assumed. This means that the link is symmetric, so the delays are the same in both directions. So the end-to-end delay (i.e. the latency) can be determined as half of the round-trip time *RTT*. The round-trip time is the length of time

the packet needs to travel from a source to a destination plus the time an answer from the destination needs to be received.

$$L_{Sw-Co} = L_{Co-Sw} = \frac{RTT}{2} \tag{3.3.1}$$

This allows the calculation of the latency $L_{Sw_1-Sw_2}$:

$$L_{Sw_1-Sw_2} = L_T - L_{Co-Sw_1} - L_{Sw_2-Co} = L_T - \frac{RTT_1}{2} - \frac{RTT_2}{2} \tag{3.3.2}$$

The one-way delay $L_{Sw_2-Sw_1}$ could be determined by the controller through sending the probing packet to switch Sw_2 which then sends it out to Sw_1, where it results in a `PACKET_IN` message. An advantage of sending the timestamp within the packet is that the controller does not have to save the time when the packet is sent. If losses would occur in the link between the switches and a packet would be lost, a timeout would be necessary. This would lead to more orchestration and development effort. Additionally, if a link gets congested and the measurement interval is smaller than the resulting link latency, wrong values would be assumed if the newer timestamp would be chosen. The contained timestamp solves this problem and for each probing packet, the necessary time to return to the controller can be derived directly.

In a topology which has more than two switches, sending out the probing packets out of every switches can derive a matrix that represents all latency values between the switches. An example matrix for three interconnected switches would be:

$$\mathsf{L} = \begin{vmatrix} 0 & L_{Sw_1-Sw_2} & L_{Sw_1-Sw_3} \\ L_{Sw_2-Sw_1} & 0 & L_{Sw_2-Sw_3} \\ L_{Sw_3-Sw_1} & L_{Sw_3-Sw_2} & 0 \end{vmatrix} \tag{3.3.3}$$

This matrix can be used as cost matrix for a weighted graph. Additionally, the actual topology can be derived and combined in an unweighted graph. On the resulting graphs, search algorithms can be performed and the path with the lowest cost, in the case of delay the shortest path, can be found out.

3.3.2 Routing

In addition to tracking network metrics such as latency, the controller also has the task of executing the agent's actions. The actions are the initialization and modification of the paths of the flows in the network.

OpenFlow Commands

In order to create a connection between the different hosts, the forwarding rules must be configured to switches located on the path. The rules, which are saved in the flow tables, are modified with the `OFPT_FLOW_MOD` message provided by the OpenFlow protocol[1]. Different types of modification requests allow to add, change or delete flow table entries (via the `OFPFC_ADD`, `OFPFC_MODIFY` or `OFPFC_DELETE` key).

[1] OpenFlow Specification, opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

Locating the hosts

Initially, the controller that controls the Layer 2 network does not have the information to which switches hosts are connected. Only if one of the hosts wants to establish a connection to another host, an Address Resolution Protocol (ARP) [97] request is sent. Since the host-connected switch has no entry for the specific constellation of destination IP, source IP, and ARP protocol type, the switch sends a `PACKET_IN` message to the controller. The controller also does not know which switch the host with the searched destination IP is connected to, the ARP request flooded until an ARP response is received. If this occurs, the locations of the hosts are found out, clearly identified by their Media Access Control (MAC) and IP -address, and saved with the ports of the switch they are connected to.

Path search

As described in section 3.2.1, each flow has different possible paths and their combinations define the state space $\mathcal{S}$. To find out all possible paths for each flow, algorithm 3 based on Depth-first search (DFS) [31, p. 603-610] is deployed. For each discovered vertex, a tree is created and the path to this vertex is saved on a stack. When the destination node is reached, a possible path is found. Using the previously created weighted graph, the costs of each path can be determined. This allows to find the path with the lowest cost (i.e. the shortest path). The ports that serve the links to other switches have been determined by

Algorithm 3 Path Search

```
1:  function SearchingPaths(adjacencyMatrix, src, dst)
2:      if src == dst then return src
3:          paths = []
4:          stack = [(src, [src])]
5:          while stack do
6:              (node, path) = stack.pop()
7:              neighbors = adjacencyMatrix[node]            ▷ All neighbors of vertex
8:              forwardNeighbors = SET(neighbors)-SET(path)  ▷
    Neighbors without origin path
9:              for next in forwardNeighbors do
10:                 if next == dst then
11:                     paths.append(path + [next])
12:                 else
13:                     stack.append((next, path + [next]))
14:                 end if
15:             end for
16:         end while
17:     end if
18:     return paths
19: end function
```

the latency measurement. Then flow table entries are added for all switches on the route via the `OFPT_FLOW_MOD` message with type `OFPFC_ADD`.

Reroute

The path search found different possible routes between two hosts. One path gets selected, randomly or the one with the lowest cost, and the path is established by modifying the flow table of the switches. As explained in section 3.2.1, the actions the RL agent can perform are changing the paths of one (one flow change) or more flows (direct change). To change a path to another one it is necessary to change the forwarding tables of the switches. Figure 3.8 shows this process in a topology with a branch. The connection between switch number 1 and 7 is changed from the previous connection via switches 3 and 4 (1-2-3-4-7) to a connection via switches 5 and 6 (1-2-5-6-7).

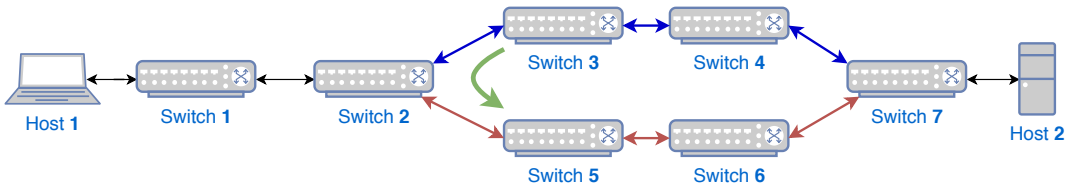


Figure 3.8: Process of changing a route in a specific topology.

The following commands need to be sent to the switches:

1. Adding flow table entries in switches 5 and 6.
2. In switch 2 the output-port of the existing flow is changed to the one of the link to switch 5.
3. The flows of the specific connection in switch 3 and 4 are deleted.

It is important that the sequence is followed to ensure a continuous flow of packets without interruptions. In the example shown in Figure 3.8, it is not necessary to change the flow table in switch 1. Therefore, the rerouting procedure, shown in algorithm 4, was developed.

First, three lists are initialized, one for each of the three types of `OFPT_FLOW_MOD`, so one for Add, Delete and Modification operations. Then it is iterated over the new path that should be deployed. The first switch in the path, which is directly connected to the source host, is not considered in the iteration. It is checked if the currently selected switch from the new list is also in the old list. If this is the case, the system checks whether the previous switch matches in the old and the new path. If not, the previous switch in the new list is set to the modification list. In case the current switch is not in the old path, it will be placed on the Add list and its predecessor on the FlowMod list. This is continued until the end of the new list is reached. After that the procedure will be followed as defined previously. First the flow table entries are added to the switches of the FlowAdd list. This has no effect on the switches in the current path. Then the FlowMod list is processed from the back. Thus it is ensured that there are no interruptions in the packet flow or that no paths lead to an unconfigured switch. Finally the switches are determined in which flow table entries have to be deleted. For this the difference between the set of the old switches and the new ones is determined. Deleting the old flow entries is necessary to avoid complications when new flows are added at the next rerouting.

3.3.3 OSPF

As a comparison to the proposed approach based on RL, routing is implemented using Open Shortest Path First. OSPF computes the path between two nodes in which the sum of the

Algorithm 4 Rerouting

```
1:  procedure RouteDeployment(oldPath, newPath, flowID)
2:      flowAddList ← [ ]                    ▷ Switches in which flow table entries are added
3:      flowModList ← [ ]                    ▷ Switches in which flow table entries are modified
4:      flowDelList ← [ ]                    ▷ Switches in which flow tables entrie are deleted
5:      for index, switch in Enumerate(newPath) do
6:          if switch in oldPath then
7:              oldIndex ← GetIndex(oldPath, switch)
8:              if oldPath[oldIndex-1] == newPath[index-1] then    ▷ If same previous switch
9:                  continue
10:             else
11:                 if newPath[index-1] not in flowAddList then
12:                     flowModList ← flowModList + newPath[index – 1]
13:                 end if
14:             end if
15:         else
16:             flowAddList ← flowAddList + switch
17:             if newPath[index-1] not in flowAddList then
18:                 flowModList ← flowModList + newPath[index – 1]
19:             end if
20:         end if
21:     end for
22:     for switch in flowAddList do                    ▷ Adding flow table entries
23:         followingSwitch ← newPath[getIndex(newPath, switch) + 1]
24:         addFlowSwitch(switch, flowID, followingSwitch)
25:     end for
26:     for switch in reversed(flowModList) do          ▷ Modify flow table entries
27:         followingSwitch ← newPath[getIndex(newPath, switch) + 1]
28:         modFlowSwitch(switch, flowID, followingSwitch)
29:     end for
30:     flowDelList ← SetDifference(oldPath, newPath)
31:     for switch in flowDelList do                    ▷ Delete flow table entries
32:         delFlowSwitch(switch, flowID)
33:     end for
34: end procedure
```

weights of the constituent edges is minimized. The link weights or costs in this case are the one way delays between the nodes. For this the one way delays are measured in the first timestep and a graph, which represents the existing latencies in the links, is generated. The graph is used to determine the shortest path when a new datastream between two hosts is added to the network.

3.4 Practical Implementation

For the controller implementation, Ryu was chosen as a framework because it offers extensive documentation and supports OpenFlow. Ryu is written in the programming language python and so the rest of the programs were also developed using python. As described in section 3.1, the implementation is divided in two modules, the learning module and the controller. The controller is responsible for collecting the measured values and executing the

actions given by the learning module. For the latency measurement, as described in section 3.3.1, the point in time at which the packet returns to the switch is measured. Since latency values are in the millisecond range, it is important that these measurements are accurate. Therefore, the measurements should not be disturbed by other parts of the implementation, in this case the learning module. Ryu has a threading module *ryu.lib.hub*, which is basically a wrapper of *eventlet*, a lightweight threading library. Unfortunately in python it is not possible to do multithreading (via system threads) because of the so-called global interpreter lock. The global interpreter lock should make sure that shared data structures are accessed by only one thread at a time for avoiding race conditions. A solution to divide the processes also on system level is the application of the python multiprocessing[2] library. It allows spawning processes and can therefore effectively bypass the global interpreter lock. This also allows the use of multiple processors. To ensure that the measurements are not blocked by the learning module, they are divided into different processes. When the controller is launched, it also starts the process for the learning module. Since the learning module requires the measured current metrics of the network and the actions selected by the agent must be transmitted to the controller for execution, communication between the individual processes is necessary. The multiproccessing library offers a pipe function for the exchange of information between two processes. The function provides two connection objects which represent the two ends of the pipe. Each side has a receive and send function with which the data can be exchanged in the form of objects. Figure 3.9 shows the structure schematically.

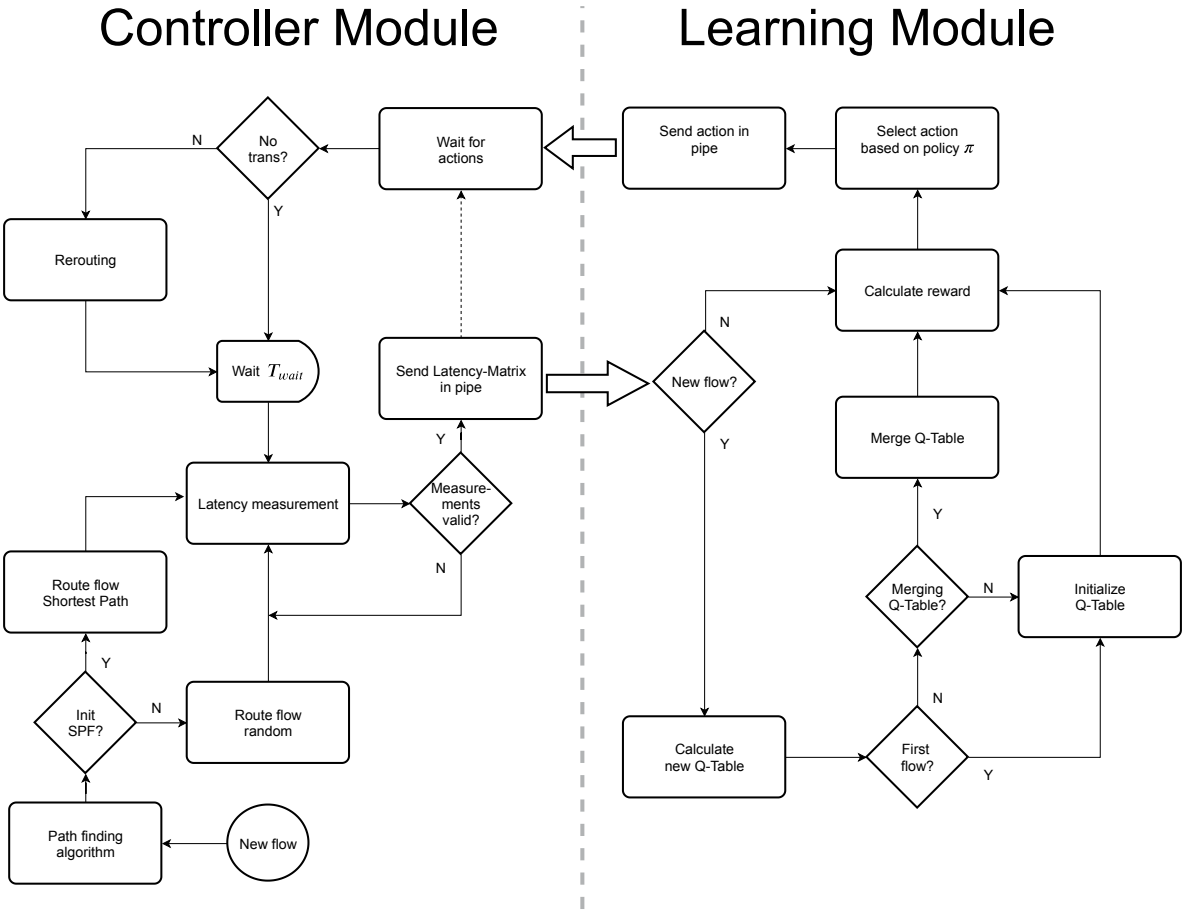


Figure 3.9: Flowchart of the controller and learning module.

[2] python, docs.python.org/3.5/library/multiprocessing

The metrics are transmitted to the learning module in the form of a matrix, which can then determine the current latencies for the paths. The one-way delays (i.e. latencies) are determined by the active measurement system described in section 3.3.1. The probing packets are transmitted in fixed intervals and the matrix is updated when measurement packets are returned by the `PACKET_IN_MESSAGE`. In addition, a time T_{wait} is waited before the measurements are recognized as valid. This is necessary to ensure that the system has settled to the desired state. This will be discussed in more detail in the following sections. After the waiting interval T_{wait}, it is checked whether all latencies were successfully measured. It can happen that measurement packets are dropped in a congested (i.e. full) queue. Due to the packet loss, the measurement may not be performed. Therefore after elapse of T_{wait} the controller additionally waits for all measurements for the links. When the latency matrix has been filled with valid measurements, the matrix is transmitted to the learning module via the pipe. The agent then determines the next action to find the optimal state and, if necessary, to stay in it. For the actions, the learning module transmits the paths to be changed to the controller. Depending on the type of action, i.e. one flow change or the direct change between the states, it can be one or more changes. The actions are processed by the controller and executed by the rerouting as described in 3.3.2.

4 Measurements & Evaluation

4.1 Topologies & System

For evaluation, a network is emulated with Mininet [98], a tool which allows the creation of prototypes of large networks. It allows flexible and scalable emulation of switches, hosts, controllers and the links in between. Additionally, Mininet supports the creation of OpenFlow switches with the same scheme as a hardware switch. Hosts are emulated by network namespaces, which allow the execution of processes. In summary, Mininet gives the possibility to easily create realistic prototypes of networks containing OpenFlow switches. In addition, it is possible to create customized networks. For example, bandwidths of links can be limited and other network parameters such as packet losses or delays can be added. For this purpose, Mininet uses the Linux tool TC (Traffic Control) [99]. TC creates a queue in which the data packets are processed with the so-called queuing discipline (qdisc). The queuing rules used by the TC module to implement the flow control functions can be divided into two categories, the classless and classful queuing disciplines. The classless queuing discipline is relatively simple. Packets are all treated the same and no classification or prioritization takes place. On the other hand, the classful queuing discipline divides the packets into different classes with filters. The rescheduling, delaying or discarding then takes place within a class, meaning that packets are not treated equally. In the evaluation network an equal treatment of the packets is desired. The probing packets should experience the same delay as the data packets in the links. Thus the probing packets can recognize congestion. Therefore a classless queuing discipline called Token Bucket Filter (TBF) is used. The queuing algorithm works by outputting so-called tokens according to the desired data rate. The packets are collected in the queue and the tokens are given to the packets. The packets need, depending on their size, a certain number of tokens to pass the network. When a queue is full, the packets are dropped (i.e. discarded). This can lead to packet loss in the flows, but also measurement packets can be dropped. TBF is a pure shaper and does not reschedule packages. Meaning that TBF has the capabilities to limit the bandwidth, but not adding specific delay. Therefore, additionally NetEm, the Linux Network Emulator Module is used to add the desired delay for a link.

4.1.1 Scenarios

For the evaluation, a simple and understandable topology, shown in figure 4.1, is introduced. The network consists of four switches. A connection between the switches Sw_1 and Sw_4 can

take place via two paths over Sw_2 and Sw_3 respectively. Three hosts h_{11}, h_{12} and h_{13} are connected to switch Sw_1 and hosts h_{41}, h_{42} and h_{43} are connected to switch Sw_4.

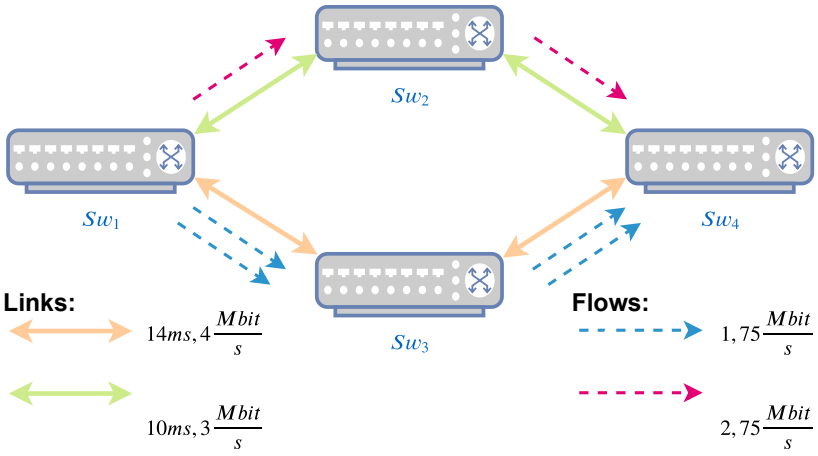


Figure 4.1: Topology containing four switches and three flows.

A latency of $10ms$ and a maximum bandwidth of $3Mbit/s$ are added to the links between Sw_1 and Sw_2 or Sw_2 and Sw_4. For links between Sw_1 and Sw_3 as well as Sw_3 and Sw_4, a delay of $14ms$ is added and the bandwidth limit is $4Mbit/s$. For the experiments, connections are established between the hosts attached to Sw_1 and Sw_4. For this purpose iperf [100] is used, a tool to perform performance measurements in networks. It offers the ability to generate data streams between different hosts. In the scope of this test setup UDP data streams are created. The added traffic streams (i.e. flows) and their set rates are as follows: $f_{h_{11},h_{41}}$ with a rate of $2.75Mbit/s$ and $f_{h_{12},h_{42}}$ respectively $f_{h_{13},h_{43}}$ with $1.75Mbit/s$. The total capacity between Sw_1 and Sw_4 is unidirectional 7 $Mbit/s$ and the only possible routing without causing congestion would be to route flow $f_{h_{11},h_{41}}$ over the path Sw_1-Sw_2-Sw_4 and the remaining flows over Sw_1-Sw_3-Sw_4. The bandwidths of the flows were selected lower than the capacity of the links, because it should be possible for the queues to empty when changing states. Otherwise the optimal state would not be recognized as optimal. Assuming the flows are in the optimal state, the expected average latency $\bar{d}$ is $22.67ms$

Joining flows

To evaluate the behaviour of the system on a joining flow, another scenario is tested. For this, two of the three flows are randomly selected from the set $F = \{f_{h11,h41}, f_{h12,h42}, f_{h13,h43}\}$ which are present from the start. In the following time, the system learns to route the two flows optimally. After the learning period, the third flow (i.e. the previously not selected flow) is added to the network. When a flow is added, it can either be routed by randomly selecting a path or by routing via the shortest path previously determined by the path search (see section 3.2.3). Additionally, the state space changes and therefore the Q-table. It can be decided whether the Q-table should be merged or reinitialized. These two features were introduced in sections 3.2.2 and 3.2.3. All combinations of initial routing and Q-table initialization are tested.

Changed link bandwidth

In order to test the influence of the link bandwidths, the previously introduced setup is modified. The bandwidths are reassigned as follows: $C(l_{Sw_1,Sw2}) = C(l_{Sw_2,Sw_4}) = 4Mbit/s$ and $C(l_{Sw_1,Sw_3}) = C(l_{Sw_3,Sw_4}) = 3Mbit/s$. Thus the total capacity of the network between Sw_1 and Sw_4 remains the same. Also the added latencies for the links are not changed and remain the same $d_{Sw_1,Sw_2} = d_{Sw_2,Sw_4} = 10ms$ and $d_{Sw_1,Sw_3} = d_{Sw_3,Sw_4} = 14ms$. An illustration of this scenario can be found in figure 5.1 in the appendix.

Scalability

To test the scalability of the approach, another test setup is introduced. Therefore, the connection between a source switch Sw_S and a destination switch Sw_D can take place over m different paths via intermediate switches $Sw_1 ... Sw_m$. To each of Sw_S and Sw_D, a number of m hosts $h_{S1} ... h_{Sm}$ and $h_{D1} ... h_{Dm}$ are connected. These hosts create m unidirectional flows between each other. Link capacities for each path range from $2Mbit/s$ to $m \cdot 2Mbit/s$ and flow bandwidths from $1.75Mbit/s$ to $(2m - 0.25)Mbit/s$. This setup ensures that only one uncongested state exists. m defines the number of possible paths and number of flows in one direction at the same time. The total number of action values $|\mathcal{Q}|$ in relation of the scalability level m with one flow change can be determined as follows:

$$\begin{aligned} |\mathcal{Q}| &= |\mathcal{S}| \cdot \left(|\mathcal{F}| \cdot (|\mathcal{P}(f)| - 1) + 1 \right) \\ &= m^m \cdot \left(m \cdot (m-1) + 1 \right) = m^m \cdot (m^2 - m + 1) \end{aligned} \tag{4.1.1}$$

An illustration 5.6 of the topology can be found in the appendix.

4.1.2 Queue Sizing and Learning Steps

The system should be able to recognize how to optimally set the paths to generate the lowest possible latency for all flows and prevent congestion. Congestion is caused by exceeding the capacity of a link and is reflected by the accumulation of packets in the queue. The size of the data in the queue can be calculated by multiplying the number of packets by their size. In the case of iperf, the default payload of the generated packets is 1470 bytes. Together with the IP- (20 bytes), UDP- (8 bytes) and Ethernet header (14 bytes), this results in a total packet size of $k_{UDP} = 1512byte$. As explained in section 3.4, after each state change the RL system is assumed to be in a steady state after time T_{wait}. It should be possible for all queues to be filled or emptied during this time. In the case that the flow $f_{h_{11},h_{41}}$ with a bandwidth of $b^{f_{h_{11},h_{41}}} = 2.75Mbit/s$ is present in a link l_{Sw_1,Sw_2} with $C(l_{Sw_1,Sw_2}) = 3Mbit/s$, the difference is $b_{diff} = C(l_{Sw_1,Sw_2}) - b^{f_{h_{11},h_{41}}} = 0.25Mbit/s$. For a waiting time $T_{wait} = 2s$ and a packet size of $k_{UDP} = 1512byte$, this would mean that packets can flow out of the queue with the rate:

$$r_{empty} = \frac{b_{diff}}{k_{UDP}} = \frac{0.25 \frac{Mbit}{s}}{1512byte \cdot 8 \frac{bit}{byte}} = 21.67Hz. \tag{4.1.2}$$

The queue length should be sufficiently high to make congestion measurable and at the same time small enough to not wait too long after the state transitions to empty the queues. Therefore the queue length is set to $K = 30$ packets and the queue can be unloaded in:

$$T_{empty} = \frac{K}{r_{empty}} = \frac{30}{21.67s^{-1}} = 1.38s. \tag{4.1.3}$$

If the queue is full, packets are dropped independently to their origin. This can also cause the measurement packets to be dropped. The probability for a packet to be dropped when it arrives in a congested (i.e. full) queue with limited size can be calculated by

$$p(dropped) = \begin{cases} 1 - \frac{C(l)}{b^f(l)} & \text{if } b^f(l) > C(l) \\ 0, & \text{otherwise} \end{cases} \tag{4.1.4}$$

Before the SDN controller forwards the measured latencies to the RL module, the controller waits for them to be valid. Thus the number of the decisions of the RL agent, called *learning steps*, might not equal the total measuring time T_{meas} divided by T_{wait}.

The delay caused by the congested queue can be determined by

$$T_{delay} = \frac{k_{UDP} \cdot K}{C(l)} \tag{4.1.5}$$

For the case of a queue length $K = 30$ and a $C(l) = 3Mbit/s$, the caused delay is

$$T_{delay}(K = 30, C(l) = 3\frac{Mbit}{s}) = \frac{(1512byte \cdot 8\frac{bit}{byte}) \cdot 30}{3\frac{Mbit}{s}} = 115.54ms \tag{4.1.6}$$

This results in an expected delay of approximated $(2 \cdot 10 + 116)ms$ from the connection of $Sw_1 - Sw_2 - Sw_4$.

4.1.3 Load Level and Average Latency

To perform measurements at different loads in the network, the term load level LL is introduced. It represents a scaling factor for the bandwidth during maximum load on the network.

$$b^f(LL) = LL \cdot b^f, \qquad LL \geq 0 \tag{4.1.7}$$

If the load level is changed, the bandwidths of the flows in the network are modified accordingly. For the previously introduced topology containing four switches and six hosts, for the flow between h_{11} and h_{41} with a load level of 40%, the new rate $b^{f_{h11,h41}}(0.4) = 0.4 \cdot 2.75Mbit/s = 1.1Mbit/s$ would result.

As described in section 3.2.1, the reward is calculated using the quadratic mean. To make the measurements easier to understand, the average one-way-delay $\overline{d}$ over all flows is used for the plots instead.

$$\overline{d} = \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} d(f) \tag{4.1.8}$$

4.1.4 Convergence Criterion

To evaluate the time until convergence and average flow latency as performance metrics quantitatively, a convergence criterion needs to be introduced. To the best of our knowledge, there is no common definition for convergence in Reinforcement Learning. Therefore a convergence criterion based on a threshold ε is introduced. The moving averages of the measurements are regarded. At the end of a measurement the system, and therefore also the last measured value, is considered converged. Starting from the back, all values that

are closer to a convergence limit than a threshold ε are also called converged. ε is defined relative to the last measurement value. Convergence time t_c is the smallest time with the corresponding value of the average latency of all flows $\overline{d}(t_c)$, where there is a convergence limit $\overline{d}_c$, so that equation 4.1.9 is true for all $t > t_c$.

$$|\overline{d}(t) - \overline{d}_c| < \varepsilon \tag{4.1.9}$$

Figure 5.2 in the appendix shows the calculated convergence time for the average of $\overline{d}$ for an example measurement with one flow change, Q-learning and softmax exploration method.

4.1.5 Latency Measurement

As described in section 3.3.1, an active measurement of the latency is carried out by using probing packets. The method is based on the idea of Phemius and Bouet, who describe in [96] an varying offset of the measured latency in comparison to measurements when using the ping utility, which delivers accurate measurements. They also used Mininet for their evaluation and found out that the moving average of the offset stays constant over time and sinks with a higher computing power of the host machines. Therefore, Phemius and Bouet relate the offset mainly to processing delay of the controller.

4.1.6 Evaluation Setup

As previously described in section 3.4, a time T_{wait} is waited after each state change until reaching a stationary state. Due to the realism achieved by Mininet, measurements with multiple iterations take correspondingly longer, ranging from hours to days. Running multiple instances of Mininet and ryu can lead to problems, for example by using the same ports or interface names. One possibility would be using additional hardware, in other words, additional working stations. However, these should have the same hardware and software configurations to ensure the credibility of the measured latency values. Since there was not enough hardware available and the effort of installing and maintaining the same Linux distributions would be high, virtualization was chosen. VirtualBox was not an option as the measured latency values fluctuated strongly. Therefore Linux KVM (Kernel-based Virtual Machine) [101] was selected, which offers better performance than VirtualBox [102].

KVM is built into Linux and allows the kernel to act as a hypervisor, so the host system can start and manage multiple virtual machines.

Listing 4.1: Default configuration

```
config.vm.provider :libvirt do |libvirt|
    libvirt.driver = "kvm"
    libvirt.cpus = 2
    libvirt.cpu_mode = "host-passthrough"
    libvirt.memory = 4096
  end
```

As a management tool for the platform virtualization, libvirt[1] was used. Vagrant[2] was used as a management platform for the virtual machines. Through Vagrant it is possible to configure virtual machines through a file, the so called Vagrantfile. In listing 4.1 the configuration

[1] Libvirt, `libvirt.org`
[2] Vagrant, `vagrantup.com/docs`

for the provided resources is shown. Each of the virtual machines is provided with four gigabytes of RAM and two CPUs. By the setting "host-passthrough", libvirt instructs KVM to pass the host CPU without modifications[3]. This provides a better performance, and can be important to some applications that check low level CPU details. As operating system, Debian with kernel version 4.19.67-2 was used. The hostsystem on which the Virtual machines are deployed is defined in table 4.1.6.

Component	Specification
Operation System	Ubuntu 18.04 LTS
Kernel	4.15.0-43
CPU	Intel Xeon W-2155 @ 20x 4.5GHz
RAM	128 GB

Table 4.1: Host system configuration

The system has 10 physical cores that result in 20 threads, meaning the number of parallel executable instructions. Therefore a total number of 9 virtual machines were started on the host with two threads remaining for the host's operation system.

4.2 Measurements

In order to evaluate the system with its implemented functions and parameters, it is tested in different scenarios regarding convergence time and the latency after convergence. First, the different action types, i.e. one flow change and direct change as introduced in 3.2.1, are compared with each other. Then, the On-Policy algorithm SARSA is compared to the Off-Policy algorithm Q-learning. This is followed by the influence of the different exploration strategies ε-greedy, softmax and UCB varying their different parameters. These are compared based on their convergence time and latency after convergence. Then, the behavior in case of a change of the network load is tested. The RL approach is compared with SPF. Next, the approach of merging Q-tables, i.e. as described in section 3.2.2, is tested. Subsequently, the initialization of a path with SPF is tested. Finally, measurements are performed over an increasing number of flows and switches. This evaluates the scalability of the approach. Before the evaluation of the exploration strategies, the softmax exploration method with $\tau = 5 \cdot 10^{-5}$ was used to test the different action styles and learning algorithms. The learning rate α and discount factor γ were set to 0.8. Unless not stated otherwise, each measurement was conducted 30 times.

4.2.1 Action types

Two possibilities for the execution of the actions were presented in section 3.2.1. The first, called one flow change, means that up to one flow can be rerouted with each action. The second approach is the direct change between states, so the rerouting of multiple flows. Figure 4.2 shows the average latency $\overline{d}$ of all three flows $f_{h_{11},h_{41}}$, $f_{h_{12},h_{42}}$ and $f_{h_{13},h_{43}}$ over steps of the learning process. The line represents the average of all iterations. The shading in the background represents the 5% and 95% percentile respectively.

[3] Libvirt configurations, `libvirt.org/formatdomain.html`

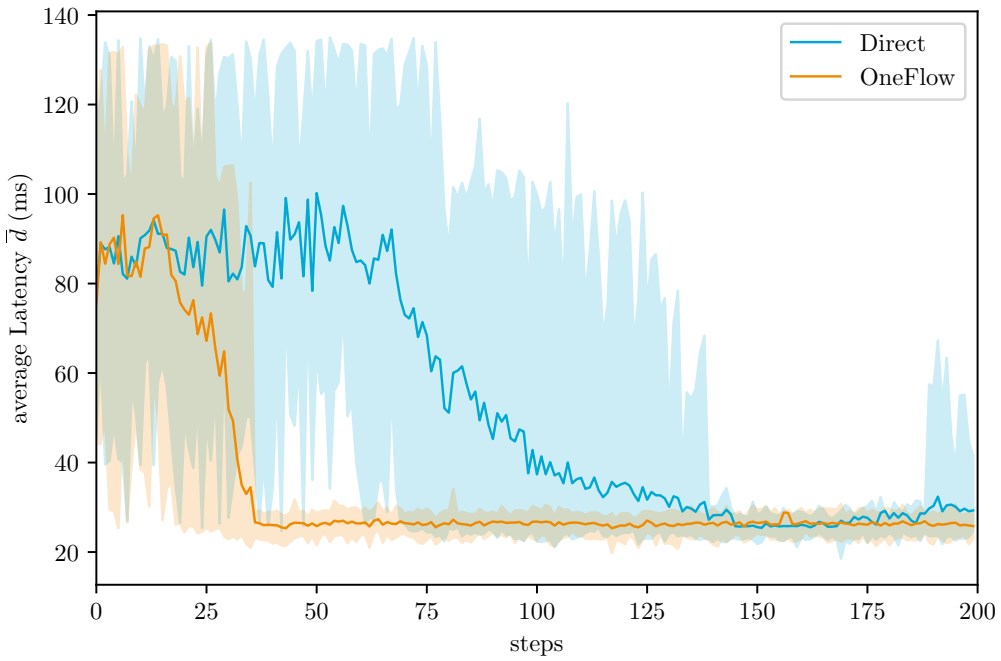


Figure 4.2: Average latency $\overline{d}$ over steps for different action types.

As can be seen in the graph, $\overline{d}$ decreases faster when using one flow change. This is due to the fact that the number of possible actions per state, and thus also the number of action-values $|\Omega|$, is higher in case of the direct change. As a result, the agent has to explore more actions before the NoTrans action in the optimal state can be found. Figure 4.3 shows the time the system needs to converge and the average latency after converging for each of the methods in form of a boxplot. The box extends from the lower to the upper percentile values with a line demonstrating the median. The whiskers show the range of the data.

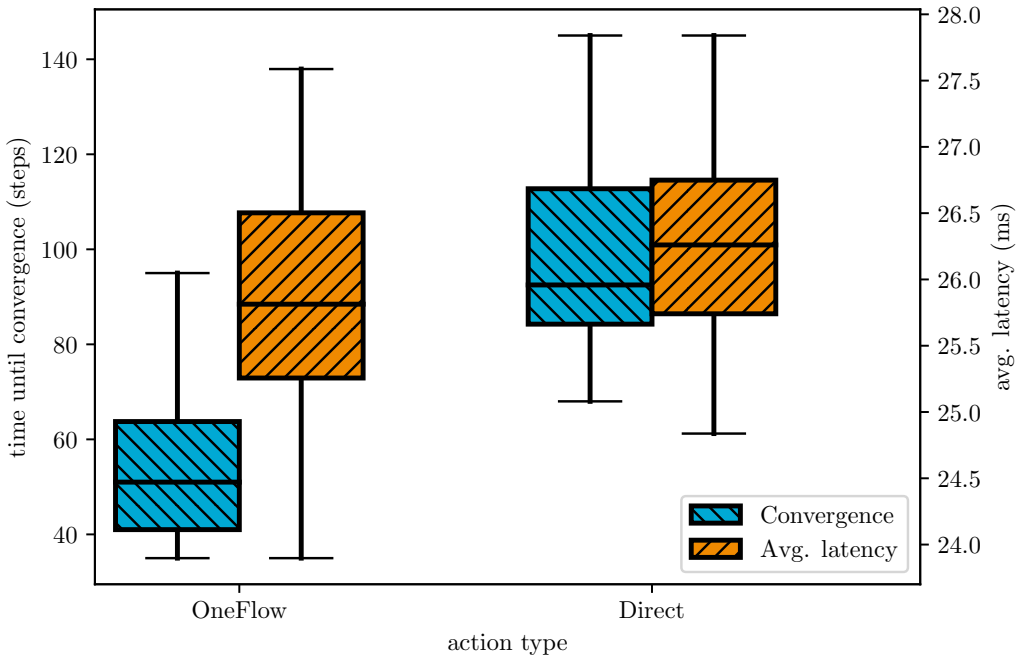


Figure 4.3: Performance over time steps for the different action types.

The method used to detect convergence is described in section 4.1.1. The threshold relative to the last measured value $\overline{d}(T_{meas})$ was set to $\varepsilon = 0.05 \cdot \overline{d}(T_{meas})$. The convergence criterion is applied for each iteration and the values following $\overline{d}(t_c)$ are taken as average latency after the convergence. Due to the faster convergence the one flow change approach is chosen for the further measurements.

4.2.2 On- or Off-Policy

The methods presented in section 2.4.3 for estimating the Q-value Q(s,a) are SARSA and Q-learning. As a reminder, SARSA follows the behaviour policy π, in this case ε-greedy, to estimate the value of the next state. In other words, the Q-value selected by π is used for the calculation of $Q(s, a)$ and the amelioration of π. In contrast, Q-learning uses the highest value $\arg\max_a Q(s_{t+1}, a)$ of the following state for the estimation. The two approaches are compared directly to each other in figure 4.4.

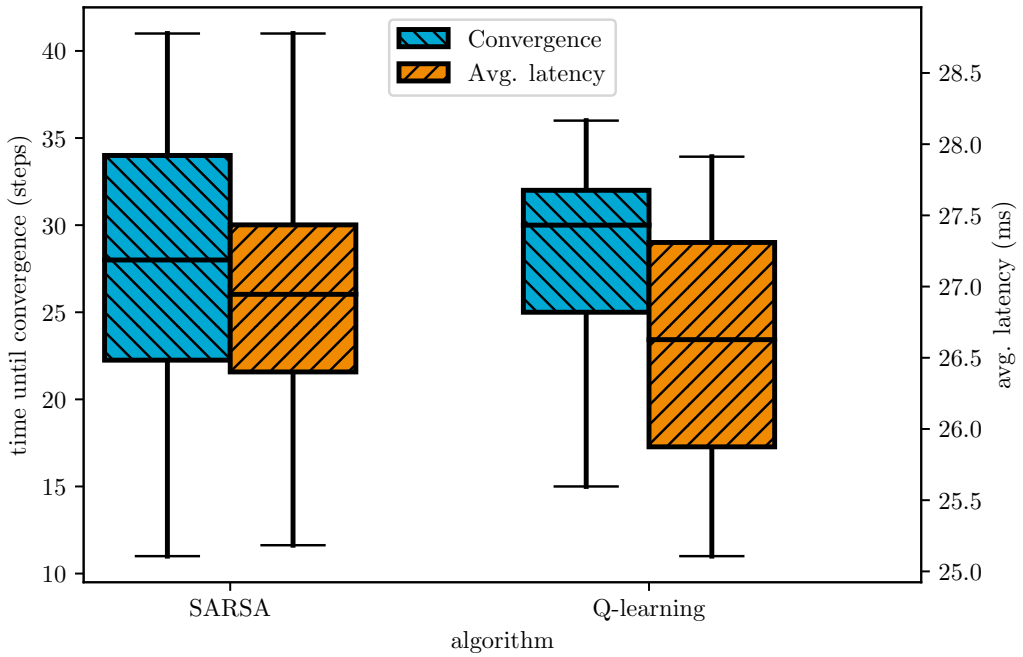


Figure 4.4: Comparison of the performance of SARSA and Q-learning.

The plot shows that the two algorithms do not differ significantly in their performance in terms of necessary steps for convergence. In terms of the average latency $\overline{d}$ after convergence, Q-learning performs better and is therefore chosen for the rest of this work.

4.2.3 Exploration strategies

The different exploration strategies, as described in section 2.4.4, are compared now. First, the individual parameters of the ε-greedy, softmax and UCB methods are varied. Then the individual strategies applying the best performing exploration parameter are compared.

ε-greedy

Using the ε-greedy method as a policy means that the agent takes a random action with a probability ε and otherwise the action with the highest action-value $Q(s, a)$. In figure 4.5, the convergence times and the resulting average latency $\overline{d}$ after convergence are compared for different values of ε.

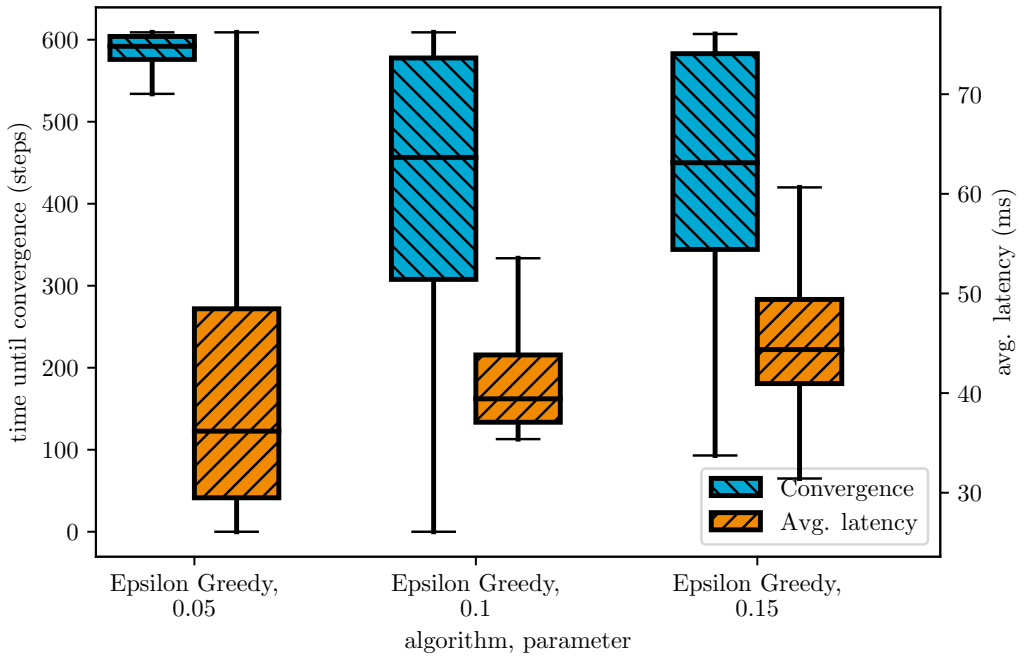


Figure 4.5: Performance of the ε-greedy algorithm in terms of convergence time and following latency regarding for different values of ε.

Due to its random nature the ε-greedy method is not able to converge. Sooner or later non-beneficial actions will be chosen, depending on exploration factor ε. As all measurements are considered converged by the end, the last bad action was longer ago for small ε, so the convergence appears to be earlier. Figure 5.4 in the appendix shows $\bar{d}$ over timesteps with different values for ε.

Softmax

Softmax provides the opportunity to map the Q-values $Q(s, a)$ to probabilities. As described in section 3.2.1, a negative reward is used which leads to negative $Q(s, a)$. To handle the negative values as well as negative infinity – inf, the softmax function was modified, as explained in section 3.2.1. In addition to the adjustment of the softmax function, the values for the temperature τ have to be selected appropriately. The creation of a graph that shows the selection probabilities in relation to the Q-values, such as section 3.4, provides orientation with the knowledge of the expected Q-values $Q(s, a)$ in the table. As mentioned before, because of the modified softmax function, it is possible to use initialization values of $-inf$. Unfortunately, no value could be found for τ which provides sufficient exploration for Q-values near $-inf$ and can effectively distinguish between values closer to zero. Due to the idea that convergence can be accelerated by selecting optimistic initialization values for the Q-table, as Sutton described in [1, p. 34 -35], a finite initialization value was therefore chosen.

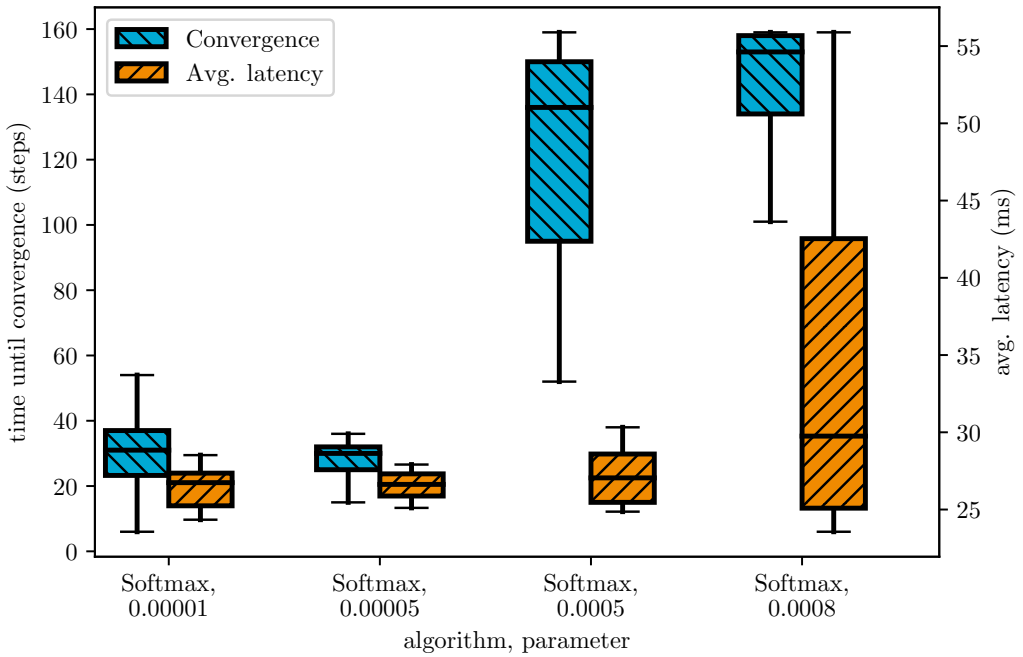


Figure 4.6: Performance of the softmax exploration with a varying τ.

In the case of a measured maximum latency of 28*ms*, an initialization value of –140 is assumed. The value was determined by assuming that the learning agent finds the optimal state and selects the NoTrans action endlessly. When applying the Q-learning algorithm, the Q-values converge over time to a value, which is around –140 for a reward $r = -28$. Figure 5.8 in the appendix shows the curve of the resulting Q-values over time for different continuously received rewards. In graph 4.6 the convergence time for different values of the temperature τ is shown. For $\tau = 0.0005 = 5 \cdot 10^{-5}$, the values for the average latency of all flows and the convergence time are determined as the most promising. Therefore this value is chosen for the temperature for the following measurements. Figure 5.5 shows the average latency over the time steps with varying τ.

UCB

Upper Confidence Bound is based on the idea to prioritize actions that have not been selected previously. Therefore the bonus b^+ was introduced, which can be weighted by the degree of exploration c. Figure 4.7 shows the performance of UCB varying the parameter c.

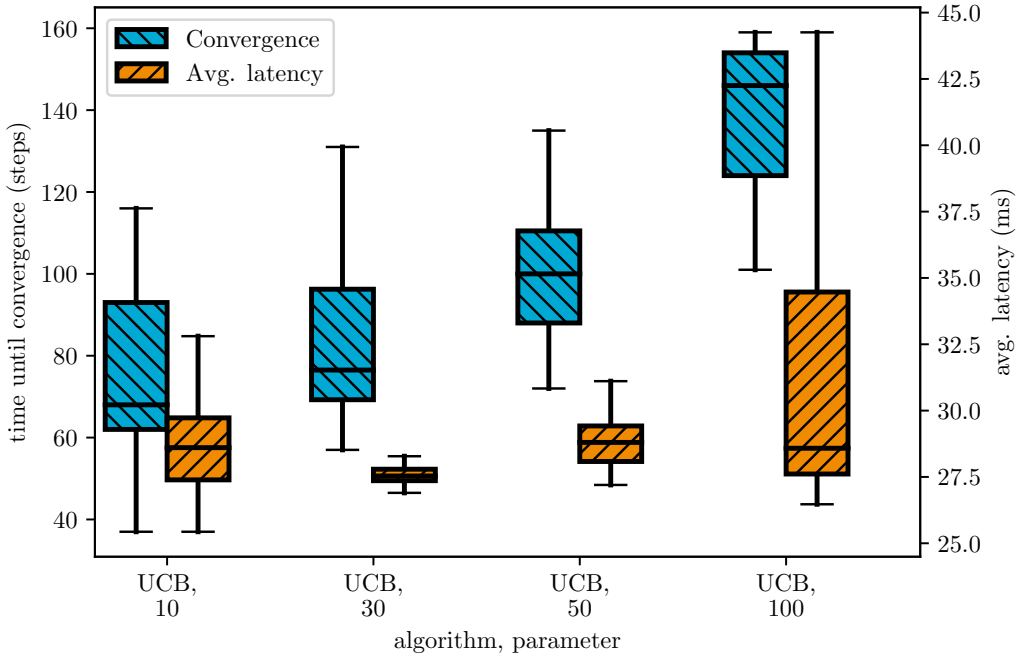


Figure 4.7: Performance of the UCB exploration strategy with different values for c.

The graph shows clearly the influence of c. A high exploration is necessary to find the global optimum, but higher values result in longer convergence times and higher regrets. Therefore a trade-off has to be made. Graph 4.7 shows that a medium value of $c = 30$ achieves the best overall results.

4.2.4 Recapitulation

In the previous sections, the different action types, learning algorithms and exploration methods were compared. As best performing action type and learning algorithm respectively one flow change and Q-learning were used for later measurements. For the exploration method several options remain. The epsilon greedy method has not been found to be a suitable method for efficient exploration. For softmax the temperature of $\tau = 5 \cdot 10^{-5}$ and for UCB $c = 30$, is selected.

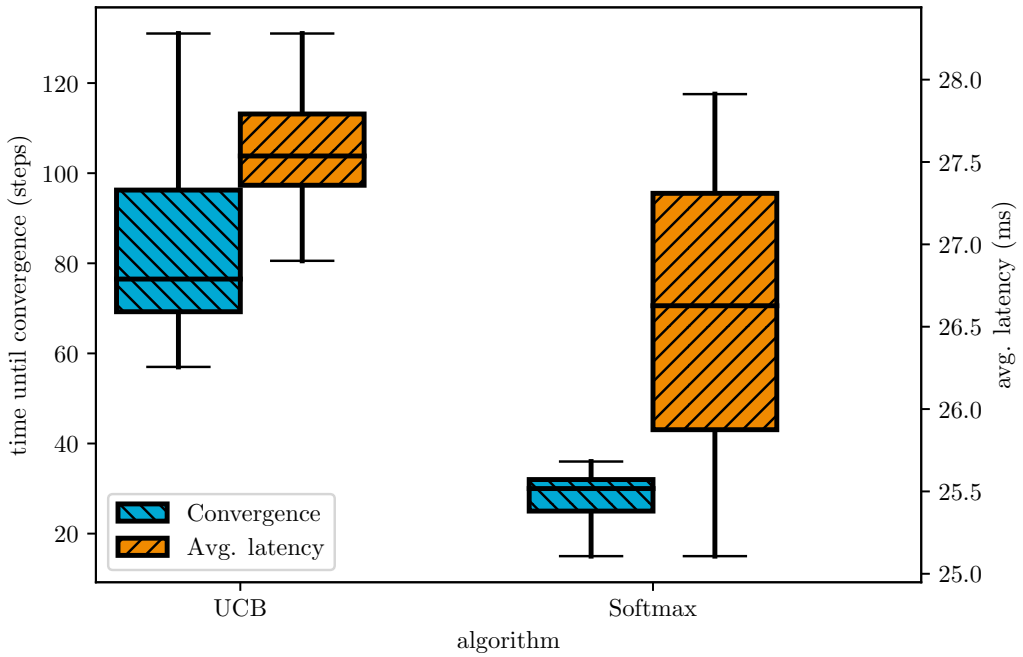


Figure 4.8: Comparison of the performance of softmax with the temperature of $\tau = 5 \cdot 10^{-5}$ and UCB with $c = 30$.

The direct comparison in figure 4.8 demonstrates that softmax delivers the best performance, in regard to convergence time and regret. Therefore, the following configurations as shown in table 4.2.4 are chosen for the next measurements:

Configuration	Selection
Action Type	One Flow
Learning Algorithm	Q-learning
Exploration Method	Softmax
Tau	10^{-5}

Table 4.2: Configuration of the RL framework.

For the tests, the values were passed to the system via a configuration file, which can be found in the appendix in listing 5.1.

4.2.5 Load Change

In order to investigate how the RL approach behaves in a dynamic environment, the load of the network is changed. In figure 4.9, the system based on RL, with the two different exploration methods softmax and UCB, are directly compared to OSPF routing, as described in section 2.2.3. For softmax, the temperature was set to $\tau = 5 \cdot 10^{-5}$ and for UCB an exploration degree of $c = 30$ was defined. For OSPF, latency is selected as the cost function and the path $Sw_1 - Sw_2 - Sw_4$ was determined by the shortest path algorithm and the three flows $f_{h_{11},h_{41}}, f_{h_{12},h_{42}}, f_{h_{13},h_{43}}$ were routed over this path. First a load level of $LL = 40\%$ is set, i.e. all flows can be routed over the shortest path without causing congestion. After 200 steps, the load level is increased to $LL = 100\%$, the average latency $\bar{d}$ for the OSPF routing method spikes.

This behavior is due to the congestion in link l_{Sw_1,Sw_2} and the resulting full queue. A load level of 100% results in an accumulated bandwidth of 6.25*Mbit/s* for all three flows which flow into link l_{Sw_1,Sw_2} that only has a bandwidth of 3*Mbit/s*. This leads to a bandwidth difference of $b_{diff} = -3.25Mbit/s$ and therefore in a congested state. In addition to the spike, which is caused by the quickly filled queue of the link between Sw_1 and Sw_2, the average latency continuous to rise. The traffic rate arriving at switch Sw_2 and forwarded to Sw_4 through l_{Sw_2,Sw_4} corresponds to the previously limited rate of 3*Mbit/s*. This means that the queue of the Sw_2-Sw_4 connection is at its upper limit with a bandwidth limitation of 3*Mbit/s*. In addition, the measurement packets are sent out once per second with a packet size of 64 byte (i.e. 512 bit). This results in a growing queue for the link l_{Sw_2,Sw_4} up to the configured length of $K = 30$ and as a consequence an increasing delay. For some measurements, the latency suddenly dropped due to unpredictable behaviour of linux traffic control. It was observed that without foreseeable reason, queues were emptied during the measurements. Therefore, the latency shows stronger variations over time in protracted congested cases.

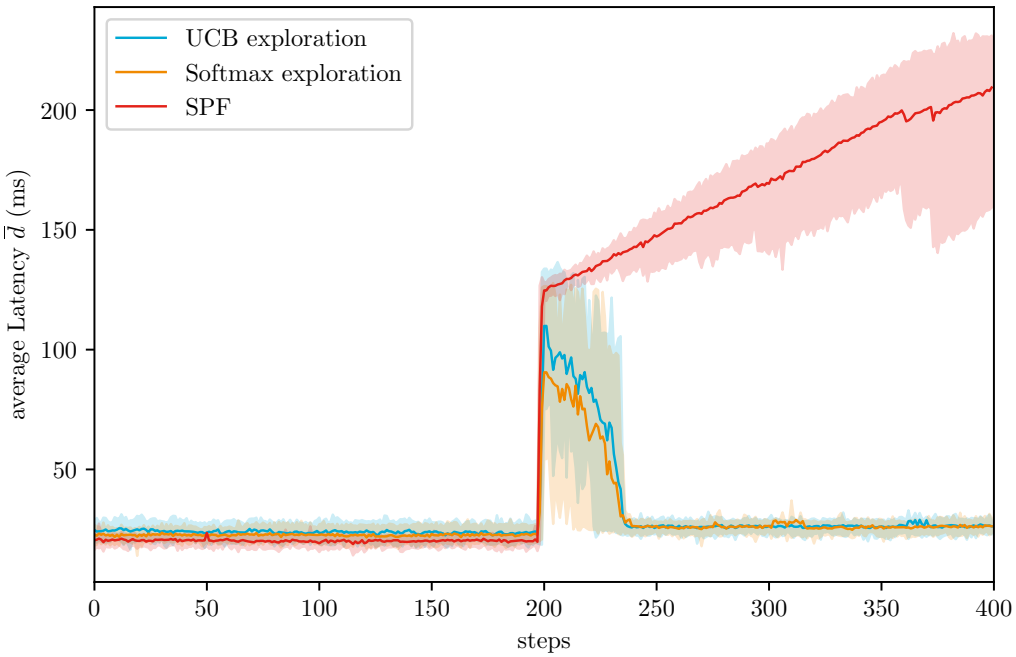


Figure 4.9: Average latency $\overline{d}$ over steps with a load change after 200 steps.

On contrary to SPF, after a short adaption period, both RL solutions converge to an uncongested state resulting in a lower $\overline{d}$. The agent recognizes the changing load at time step 200 through receiving a lower reward and resulting lower Q-values which in turn encourages exploration. When he finds the state with the lowest latency, he remains in it except for actions that serve further exploration (i.e. the regret). From graph 4.10, it can be derived that softmax has a lower convergence time than UCB.

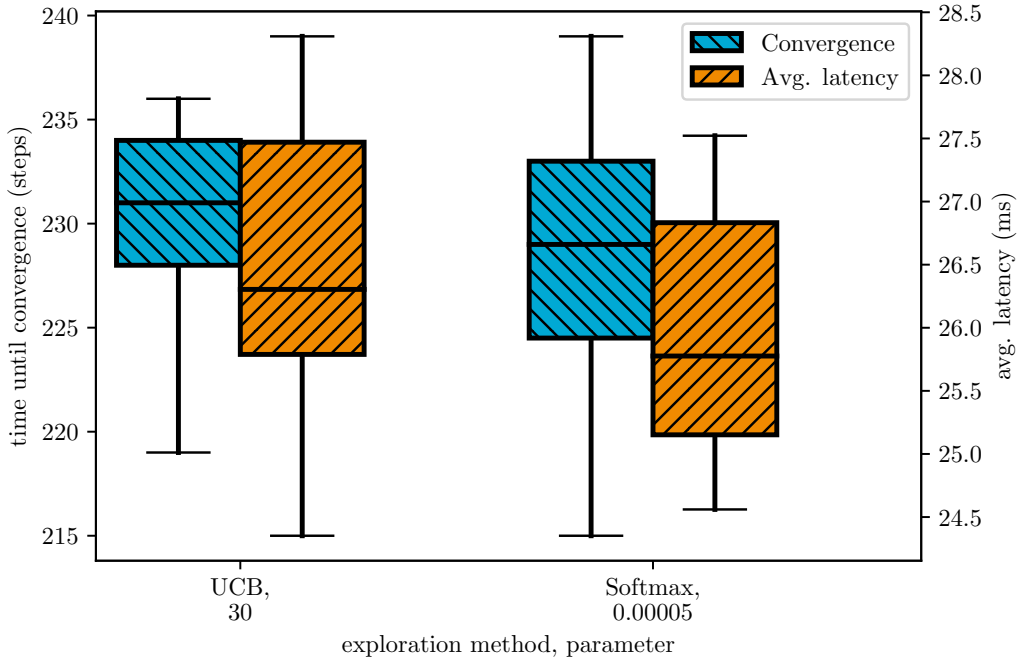


Figure 4.10: Comparison of the convergence time and the regret after the load change for UCB ($c = 30$) and softmax $\tau = 5 \cdot 10^{-5}$.

This clearly shows the advantage of a system which dynamically adapts based on the current state of the network. Reinforcement Learning makes this possible by using the feedback the agent receives from the environment through the reward.

4.2.6 Load Levels

Next, the performance of the implemented solution compared to routing based on OSPF is examined as a function of the load level in figure 4.11. In this case, the measurement consists of 20 iterations. The plot shows the average flow latencies with its corresponding 5% and 95% percentiles as whiskers of 30 iterations. OSPF uses the shortest path determined by the constructed weighted graph. As before, the flows are routed over the path with the lowest overall latency, Sw_1-Sw_2-Sw_4. The RL solution shows slightly higher average values than SPF on a low network load due to the continued exploration. With a link bandwidth between Sw_1-Sw_2 of $3Mbit/s$ and with a cumulative bandwidth of all flows of $b_{total}(l_{Sw_1,Sw_2}) = \sum_f b^f(l_{Sw_1,Sw_2}) = 6.25Mbit/s$ under 100% load, congestion occurs on a load level of $LL_{congested} = (3Mbit/s) \,/\, \left(b_{total}(l_{Sw_1,Sw_2})\right) = 0.46 = 46\%$. As a result, SPF routing shows a recognizably higher average latency of all flows $\overline{d}$ starting at the measurement point of $LL = 50\%$. RL-based routing, on the other hand, adapts to the change in the load and can thus successfully prevent congestion.

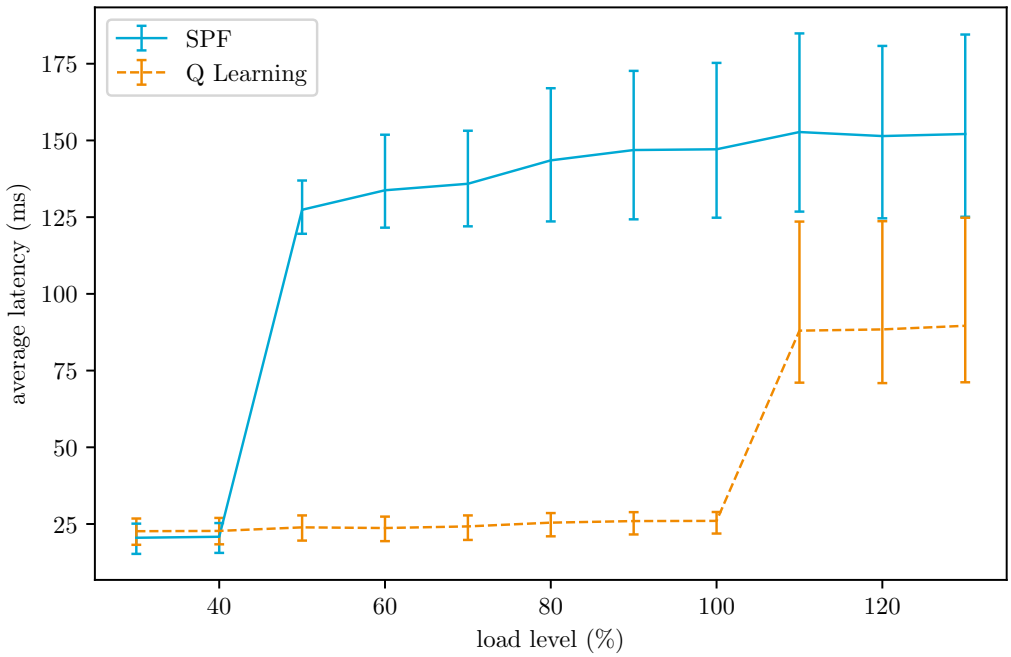


Figure 4.11: Average Latency $\overline{d}$ after convergence of the softmax exploration with $\tau = 5 \cdot 10^{-5}$ and SPF depending on load level *LL*.

What can be recognized additionally in graph 4.11 is the fact that the average latency for the RL approach still gives better results than SPF routing with load levels above 100%, in which no uncongested state is possible anymore. This is because the latencies of all flows were included in the reward calculation. Therefore the agent learns to route as many flows as possible over a path in which they do not cause congestion. As described in 4.1.1, the traffic rates of all flows were set in a way that the queues could be emptied within T_{wait} and a stationary state could be reached. At a load level of 110%, the flows $f_{h_{12},h_{42}}$ and $f_{h_{13},h_{43}}$ would have a bandwidth of $1.925 Mbit/s$. These two flows could be routed over the connection $Sw1$ – $Sw3$ – $Sw4$ without causing congestion. However, due to a smaller b_{diff}, the unloading time T_{empty} is longer than T_{wait} and therefore the agent is not capable to find the route. Therefore, the agent does not find the optimal state and converges to a state with two congested flows. The congested flows are included in the calculation, resulting in an increase in average latency at a load of $LL = 110\%$. This shows that the approach based on RL routes the flows in such a way that the largest possible number of flows do not experience congestion under the assumption of static states.

4.2.7 Merging & Initialization

As described in section 3.2.2, the state- and action-spaces change when a new flow is added to the network. This would result in resetting the Q-table which would require the agent to be re-trained to find the optimal state in the network. An approach to prevent this, is using the Q-values $Q(s, a)$ with the greatest similarity to the previously calculated Q-table. To evaluate this functionality, first the joining flow scenario is evaluated. Secondly, the topology is customized by swapping the bandwidths of the links. These two scenarios are introduced in section 4.1.1 Two flows are first selected randomly from the set $F = \{f_{h_{11},h_{41}}, f_{h_{12},h_{42}}, f_{h_{13},h_{43}}\}$. After a short time which is sufficient to learn the optimal state, the third flow joins the network. There are two ways, to reinitialize the Q-table or to use the table which contains the

Q-values for the setup of two flows (i.e. merge them). Also the two possibilities of initial routing, as presented in section 3.2.3, are evaluated. Graph 4.12 shows the performance in terms of convergence time and average latency after the addition of the third flow of the four combinations: Merging Q-table and resetting the Q-table, each with or without SPF initialization with the introduced topology shown in figure 4.1. Since the time steps are counted from the experiment's beginning, the moment of the addition of the third flow should be subtracted to get absolute convergence times. In the plots, the convergence times can be relatively compared. The graph implies that no considerable performance gains are recognizable for the flow table merging or the SPF initialization.

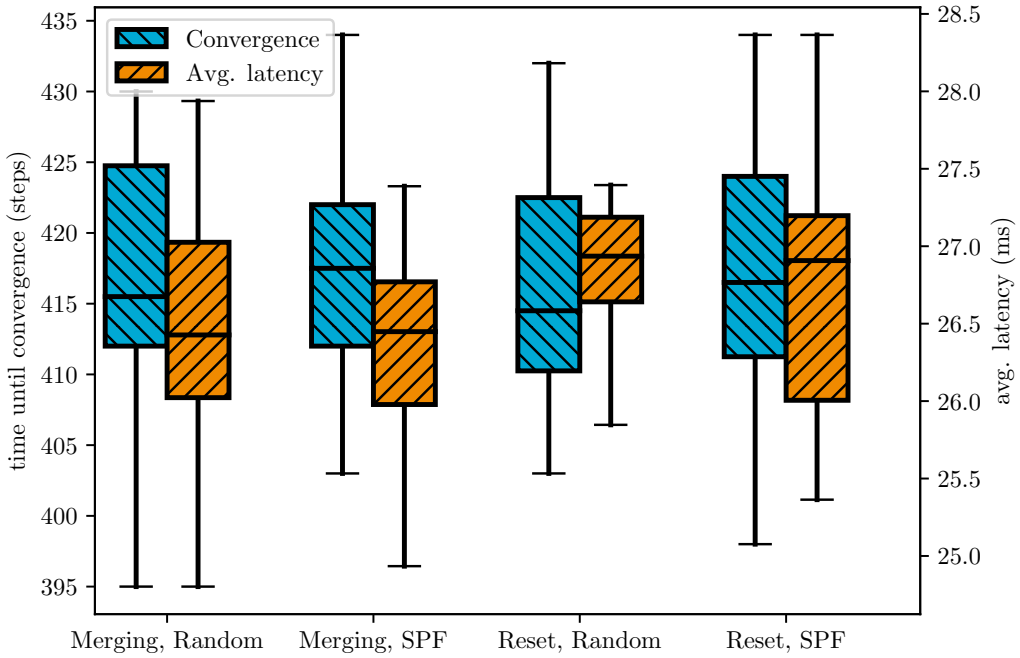


Figure 4.12: Convergence time steps and latency $\overline{d}$ after convergence of the different combinations of reactions on a joining flow for the original topology.

To test whether the performance depends on the topology, the bandwidths of the links were flipped like described in section 4.1.1. Graph 4.13 shows the performance of the features in the modified topology.

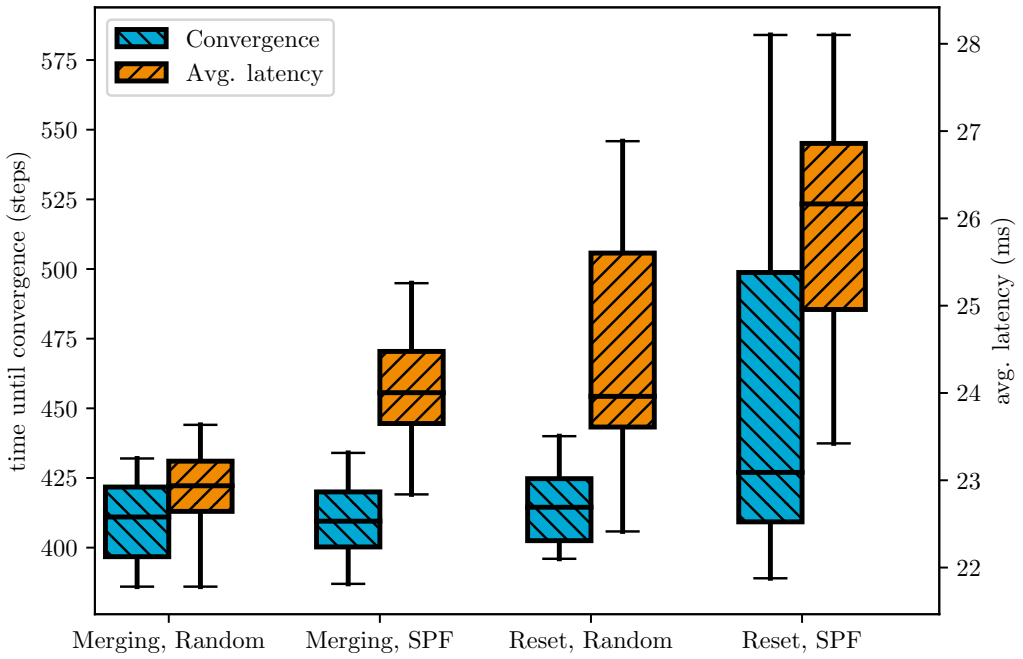


Figure 4.13: Convergence time steps and latency $\overline{d}$ after convergence of the different combinations of reactions on a joining flow for the modified topology.

The results show that the performance of the merging feature depends on the topology and delivers better performance in the modified topology. Initialization via SPF on the other hand does not provide any benefits.

4.2.8 Scalability

The previous measurements were applied to a relative small topology and therefore in correspondingly small state and action spaces. To show the influence of the number of the action values $|\Omega|$ on the convergence time, the extended topology as described in section 4.1.1 is used. The scalability level m defines the number of flows $|\mathcal{F}|$ and additionally the possible paths for each flow $|\mathcal{P}(f)| = m$. Figure 4.14 shows the convergence times in relation to the scalability level m. For the exploration, softmax with a temperature of $\tau = 5 \cdot 10^{-5}$ was used. The theoretical average latency after convergence should be considered as close to $20ms$, although as described in section 4.1.1 the measurement uncertainty caused by the latency measurement should be taken into account. The boxplot 4.14 shows that the convergence time does not scale linearly with the scalability level m. Additionally, the boxplots show that for a scalability level of $m = 4$ the expected latency value was clearly exceeded, which is due to the fact that the exploration was not sufficiently strong.

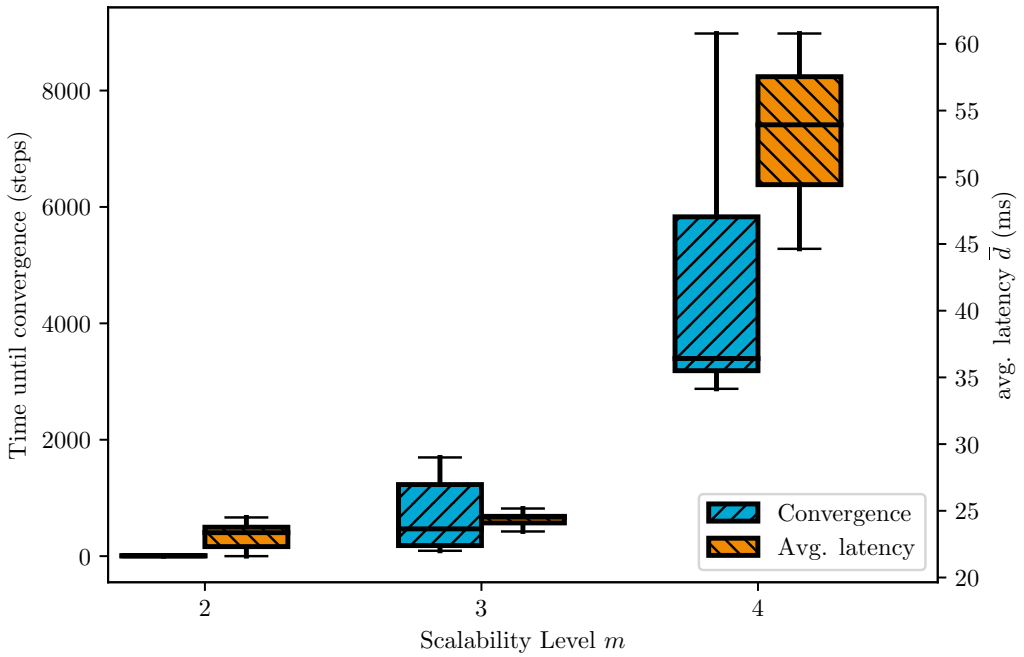


Figure 4.14: Convergence time steps and following average latency in relation to the scalability level.

Table 4.2.8 shows the number of Q-values $|\mathcal{Q}|$, which can be calculated by the equation 4.1.1, in comparison to the median and the average of the convergence times.

Level	$\|\mathcal{Q}\|$	Average	Median
2	12	5.95	6.00
3	189	687.75	468.50
4	3328	4885.07	3396.00

Table 4.3: Average and median values for time steps until convergence.

In figure 5.7 in the appendix the average of the average latency of all flows $\overline{d}$ over time steps is also shown. This shows that after an alleged minimum has been found, the latencies do not change far from it. To sum it up, this evaluation scenario demonstrates the need for sufficient exploration, optimally combined with annealing, to find the global minimum.

5 Conclusion & Outlook

5.1 Conclusion

In this thesis a new approach for routing and traffic engineering in Software-Defined Networks was presented and evaluated. The existing solutions struggle to keep up with the growing network size and number of devices either due to unrealistic distributions of traffic in networks or due to high computational complexity. In addition, the implementation of these solutions is often complex in design and hence involves additional engineering effort. Therefore a purely data-driven different approach was chosen and the network should be optimized without a model only by using experiences gained from the interaction with the network. For this purpose a system based on Reinforcement Learning was implemented. It is capable to record the current metrics of the network, such as the latency. Using the actual network data, the RL agent learns how to optimize the network with the objective of latency minimization and congestion prevention. As the state space, a combination of chosen paths of the flows in the network was chosen. To define the possible actions which the agent can perform, the rerouting of the flows to different paths were selected. Therefore two different approaches have been defined, the possible rerouting up to multiple flows or the introduction of a limitation of allowing only one flow to be rerouted. For the determination of the possible paths of an end to end connection a path search algorithm based on DFS was implemented. To enable the rerouting of flows in the network, an algorithm was developed and added to the solution. The algorithm has the advantage of not interrupting packet flows, what would result in losses in the connection, by executing the changes of the flow table entries in a specific order. As a reward function (i.e. the optimization objective), the quadratic mean of the latencies of the flows have been chosen. This results in a fair optimization of all flows in the network without favouring an individual one. Using RL, the system learns independently the optimal constellation of the possible paths of flows in the network leading to a state in which the average latency of all flows in the network is minimized. It has been shown that the proposed solution is capable to handle different network topologies and number of traffic flows. Additionally, the RL system also successfully detects load changes and performs actions to adjust the routes of the flows accordingly. Features such as Optimistic Initialization, SPF initialization and the merging of the Q-tables have been implemented and tested. The optimistic initialization can speed up the learning process if the values are chosen wisely. It have been shown that merging the Q-tables and the initialization can lead to a faster convergence in specific cases that depend on the topologies and flows in the network. To briefly summarize, a framework was created in which an RL agent

learns and maintains the optimal placement of flows independently and without requiring engineering effort. In addition to recording the latencies in the network, it also includes the efficient routing of flows and can be used simply for further research. Various methods, features and parameters of the RL implementation were successfully evaluated in this framework.

5.2 Outlook

In this work, RL is applied successfully to optimize the latency of connections between hosts. The experiments on the scalability showed that the non-linear scaling of the state and action-space in relation to topology size and flow count could lead to long convergence times in real-sized networks. One possibility to tackle the growing state and action-space could be the usage of switch-to-switch, so entry- and exit-switch to the network, connections instead of taking directly the flows. In other words, flows with the same entry and exit-switch would be bonded and treated as one connection. However, this would imply less effective RL due to limited actions, that the agent could perform. An easy solution would be to set limitations, such as a maximum number of paths a flow could take. But as a result, limiting the possible actions of the agent also limits its capabilities to find a optimal state. Another option could be the generalization of the actions, for example changing the flow with the largest delay to the second precalculated shortest path. Therefore the action space would not grow in relation with the state space but also limit the effectiveness of the agent's actions.

The current approach does not include any information about the bandwidth, the natural cause of congestion in links. One possibility would be to add the current bandwidth of flows or links to the state space. Since the bandwidth of the flows and their route are independent, the resulting state space would be even bigger with discrete bandwidth levels lvl_{bw} in $|\mathcal{S}| = \prod_{f \in F} |P(f)| \cdot |lvl_{bw}|$. As a result, the number of Q-table entries would also grow which in turn would result in longer exploration. Therefore it would be beneficial to separate the state and action space by only including the bandwidth information of the links or flows in the state space.

By using tabular Q-learning, a discretization of the continuous values would be necessary and could result in loss of information. Function approximators, such as neural networks [103], could provide a solution [1, p. 190 ff.]. A bigger topology or high number of connections would still lead to an exploding action space. An interesting approach to handle large state-action spaces with Deep Reinforcement Learning, the combination of Reinforcement Learning with Deep Neural Networks, gives Dulac-Arnold *et al.* in [104]. Therefore they embed the discrete actions into a continuous space instead of sampling from a categorical distribution. Then after applying a continuous policy, the closest discrete action depending on a metric is executed. This allows to handle larger networks as well. For a faster convergence, annealing could be applied. This would mean to start with a higher degree of exploration and reducing it over time. Essentially, this could lead to an efficient exploration of all network states and later to a lower regret.

An option to apply Reinforcement Learning to on-demand routing would be to take the current network state as a state and to learn how to route a newly incoming flow. In this case, the actions would be the possible paths, the new flow could be routed on.

Another interesting aspect would be to modify the reward function to let the agent follow a different optimization objective such as network utility maximization or even a mixture of objectives that could be several QoS criteria. Additionally, the fairness of the agents actions should be considered. The test case with increasing load levels from section 4.2.6 shows

that the RL agent tries to maximize the number of uncongested flows when the network is overloaded. On the one hand, this leads to flows with lower bandwidth requirements being preferred. On the other hand, all flows for which no uncongested path can be found are routed via the same link, so that the packet drop rate rises sharply. The performance could be evaluated with fairness metrics, such as max-min-fairness [63] or α-fairness [64, p. 13 - 17]. For this work, flows containing UDP packets have been chosen for the evaluation. Adding TCP flows would require a modification of the Reinforcement Learning setup. Due to the congestion window, it is possible to route all flows over the shortest path, but under a lower bandwidth of all flows. Finally, the application of the presented approach on real network could be investigated.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[2] Sundar Iyer, Supratik Bhattacharyya, N. Taft, and C. Diot, "An approach to alleviate link overload as observed on an ip backbone," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, vol. 1, pp. 406–416 vol.1, March 2003.

[3] C. Fraleigh, F. Tobagi, and C. Diot, "Provisioning ip backbone networks to support latency sensitive traffic," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, vol. 1, pp. 375–385 vol.1, March 2003.

[4] B. Naudts, M. Kind, F. Westphal, S. Verbrugge, D. Colle, and M. Pickavet, "Techno-economic analysis of software defined networking as architecture for the virtualization of a mobile network," in *2012 European Workshop on Software Defined Networking*, pp. 67–72, Oct 2012.

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally deployed software defined wan," in *Proceedings of the ACM SIGCOMM Conference*, (Hong Kong, China), 2013.

[6] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, (New York, NY, USA), pp. 15–26, ACM, 2013.

[7] J. Rexford, *Route Optimization in IP Networks*, pp. 679–700. Boston, MA: Springer US, 2006.

[8] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman, "Polynomial time approximation algorithms for multi-constrained qos routing," *IEEE/ACM Transactions on Networking*, vol. 16, pp. 656–669, June 2008.

[9] F. Ciucu and J. Schmitt, "Perspectives on network calculus - no free lunch, but still good value," *Computer Communication Review*, vol. 42, 10 2012.

[10] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, pp. 226–244, June 1995.

[11] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology." RFC 7426, Jan. 2015.

[12] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "e.a.: Extending networking into the virtualization layer," in *In: 8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII).New YorkCity,NY(October 2009*, 2009.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.

[14] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, pp. 18–26, Nov 2014.

[15] J. A. Hawkinson and T. J. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)." RFC 1930, Mar. 1996.

[16] J. Moy, "OSPF Version 2." RFC 2328, Apr. 1998.

[17] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, Dec 1959.

[18] D. Williams, D. R. Guerin, T. Przygienda, S. Kamat, G. Apostolopoulos, and A. Orda, "QoS Routing Mechanisms and OSPF Extensions." RFC 2676, Aug. 1999.

[19] A. Azzouni, R. Boutaba, and G. Pujolle, "Neuroute: Predictive dynamic routing for software-defined networks," *CoRR*, vol. abs/1709.06002, 2017.

[20] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *2013 Proceedings IEEE INFOCOM*, pp. 2211–2219, April 2013.

[21] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys Tutorials*, vol. 10, pp. 36–56, First 2008.

[22] W.-Y. Huang, T.-Y. Chou, J.-W. Hu, and T.-L. Liu, "Automatical end to end topology discovery and flow viewer on sdn," *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, pp. 910–915, 2014.

[23] "Ieee approved draft standard for local and metropolitan area networks - station and media access control connectivity discovery," *IEEE P802.1AB-REV/D1.2, November 2015 (Revision of IEEE Std 802.1AB-2009)*, pp. 1–143, Jan 2015.

[24] W. R. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms." RFC 2001, Jan. 1997.

[25] D. Medhi and K. Ramasamy, *Network Routing, Second Edition: Algorithms, Protocols, and Architectures*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2nd ed., 2017.

[26] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1 ed., 1997.

[27] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, pp. 76–80, Jan 2003.

[28] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "High-performance neural networks for visual object classification," 2011.

[29] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 44, pp. 206–226, Jan 2000.

[30] E. Alpaydin, *Introduction to Machine Learning*. The MIT Press, 2014.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[32] R. Bellman, "The theory of dynamic programming," *Bull. Amer. Math. Soc.*, vol. 60, pp. 503–515, 11 1954.

[33] C. J. C. H. Watkins, *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.

[34] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, May 1992.

[35] D. L. Poole and A. K. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*. New York, NY, USA: Cambridge University Press, 2nd ed., 2017.

[36] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, p. 237–285, May 1996.

[37] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, Dec 2016.

[38] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, (San Francisco, CA, USA), pp. 671–678, Morgan Kaufmann Publishers Inc., 1993.

[39] S. Kumar and R. Mukkulainen, "Confidence based dual reinforcement q-routing: An adaptive online network routing algorithm," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, (San Francisco, CA, USA), pp. 758–763, Morgan Kaufmann Publishers Inc., 1999.

[40] A. V. Goldberg and R. E. Tarjan, "Expected performance of dijkstra's shortest path algorithm," *NEC Research Institute Report*, 1996.

[41] H. Joksch, "The shortest route problem with constraints," *Journal of Mathematical Analysis and Applications*, vol. 14, no. 2, pp. 191 – 197, 1966.

[42] H. Greenberg, "A dynamic programming solution to integer linear programs," *Journal of Mathematical Analysis and Applications*, vol. 26, no. 2, pp. 454 – 459, 1969.

[43] W. C. Lee, M. G. Hluchyi, and P. A. Humblet, "Routing subject to quality of service constraints in integrated communication networks," *IEEE Network*, vol. 9, pp. 46–55, July 1995.

[44] Y. P. Aneja, V. Aggarwal, and K. P. K. Nair, "Shortest chain subject to side constraints," *Networks*, vol. 13, no. 2, pp. 295–302, 1983.

[45] S. Bradley, A. Hax, and T. Magnanti, *Applied Mathematical Programming*. Addison-Wesley Publishing Company, 1977.

[46] R. Widyono *et al.*, *The design and evaluation of routing algorithms for real-time channels*. Citeseer, 1994.

[47] Gang Liu and K. G. Ramakrishnan, "A*prune: an algorithm for finding k shortest paths subject to multiple constraints," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, vol. 2, pp. 743–749 vol.2, April 2001.

[48] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, July 1968.

[49] M. Munetomo, Y. Takai, and Y. Sato, "A migration scheme for the genetic adaptive routing algorithm," in *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*, vol. 3, pp. 2774–2779 vol.3, Oct 1998.

[50] Chang Wook Ahn and R. S. Ramakrishna, "A genetic algorithm for shortest path routing problem and the sizing of populations," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 566–579, Dec 2002.

[51] A. Y. Hamed, "A genetic algorithm for finding the k shortest paths in a network," *Egyptian Informatics Journal*, vol. 11, no. 2, pp. 75 – 79, 2010.

[52] Kwang Mong Sim and Weng Hong Sun, "Ant colony optimization for routing and load-balancing: survey and new directions," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 33, pp. 560–572, Sep. 2003.

[53] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," 1991.

[54] G. Di Caro and M. Dorigo, "Ant colonies for adaptive routing in packet-switched communications networks," in *Parallel Problem Solving from Nature — PPSN V* (A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, eds.), (Berlin, Heidelberg), pp. 673–682, Springer Berlin Heidelberg, 1998.

[55] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast qos routing algorithms for sdn: A comprehensive survey and performance evaluation," *IEEE Communications Surveys Tutorials*, vol. 20, pp. 388–415, Firstquarter 2018.

[56] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero, "A survey on the contributions of software-defined networking to traffic engineering," *IEEE Communications Surveys Tutorials*, vol. 19, pp. 918–953, Secondquarter 2017.

[57] Yufei Wang, Zheng Wang, and Leah Zhang, "Internet traffic engineering without full mesh overlaying," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, vol. 1, pp. 565–571 vol.1, April 2001.

[58] Yufei Wang and Zheng Wang, "Explicit routing algorithms for internet traffic engineering," in *Proceedings Eight International Conference on Computer Communications and Networks (Cat. No.99EX370)*, pp. 582–588, Oct 1999.

[59] P. Trimintzios, T. Baugé, G. Pavlou, P. Flegkas, and R. Egan, "Quality of service provisioning through traffic engineering with applicability to ip-based production networks," *Computer Communications*, vol. 26, pp. 845–860, 2003.

[60] Y. Li, A. Papachristodoulou, M. Chiang, and A. R. Calderbank, "Congestion control and its stability in networks with delay sensitive traffic," *Computer Networks*, vol. 55, no. 1, pp. 20 – 32, 2011.

[61] F. Paganini, Zhikui Wang, J. C. Doyle, and S. H. Low, "Congestion control for high performance, stability, and fairness in general networks," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 43–56, Feb 2005.

[62] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in sdn-openflow networks," *Computer Networks*, vol. 71, pp. 1 – 30, 2014.

[63] E. Danna, A. Hassidim, H. Kaplan, A. Kumar, Y. Mansour, D. Raz, and M. Segalov, "Upward max-min fairness," *J. ACM*, vol. 64, pp. 2:1–2:24, Mar. 2017.

[64] R. Srikant, *The Mathematics of Internet Congestion Control (Systems and Control: Founda-*

tions and Applications). SpringerVerlag, 2004.

[65] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng, "Engineering egress with edge fabric: Steering oceans of content to the world," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, (New York, NY, USA), pp. 418–431, ACM, 2017.

[66] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[67] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. G Bellemare, A. Graves, M. Riedmiller, A. K Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

[68] A. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, pp. 70–76, 01 2017.

[69] J. Kober and J. Peters, *Reinforcement Learning in Robotics: A Survey*, pp. 9–67. Cham: Springer International Publishing, 2014.

[70] Y. Li, C. Szepesvári, and D. Schuurmans, "Learning exercise policies for american options," in *AISTATS*, 2009.

[71] T. Wei, Yanzhi Wang, and Q. Zhu, "Deep reinforcement learning for building hvac control," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, June 2017.

[72] X. Xu, L. Zuo, and Z. Huang, "Reinforcement learning algorithms with function approximation: Recent advances and applications," *Information Sciences*, vol. 261, pp. 1 – 31, 2014.

[73] S. Choi and D. yan Yeung, "Predictive q-routing: A memory-based reinforcement learning approach to adaptive traffic control," in *In Advances in Neural Information Processing Systems 8 (NIPS8*, pp. 945–951, MIT Press, 1996.

[74] L. Peshkin and V. Savova, "Reinforcement learning for adaptive routing," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, vol. 2, pp. 1825–1830 vol.2, May 2002.

[75] C. Liu, A. Malboubi, and C. Chuah, "Openmeasure: Adaptive flow measurement amp; inference with online learning in sdn," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 47–52, April 2016.

[76] L. Yanjun, L. Xiaobo, and Y. Osamu, "Traffic engineering framework with machine learning based meta-layer in software-defined networks," in *2014 4th IEEE International Conference on Network Infrastructure and Digital Content*, pp. 121–125, Sep. 2014.

[77] Z. Hu and H. Chen, "Network load balancing strategy based on supervised reinforcement learning with shaping rewards," in *2013 Fourth International Conference on Intelligent Control and Information Processing (ICICIP)*, pp. 393–397, June 2013.

[78] J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, *Supervised ActorCritic Reinforcement Learning*. IEEE, 2004.

[79] J. Chavula, M. Densmore, and H. Suleman, "Using sdn and reinforcement learning for

traffic engineering in ubuntunet alliance," in *2016 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, pp. 349–355, Nov 2016.

[80] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, Apr 2018.

[81] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath tcp," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, (Berkeley, CA, USA), pp. 99–112, USENIX Association, 2011.

[82] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*, pp. 15–34. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[83] S. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "Qos-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," in *2016 IEEE International Conference on Services Computing (SCC)*, pp. 25–33, June 2016.

[84] C. Barakat, E. Altman, and W. Dabbous, "On tcp performance in a heterogeneous network: a survey," *IEEE Communications Magazine*, vol. 38, pp. 40–46, Jan 2000.

[85] A. Al-Jawad, P. Shah, O. Gemikonakli, and R. Trestian, "Learnqos: A learning approach for optimizing qos over multimedia-based sdns," in *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pp. 1–6, June 2018.

[86] F. Baker, D. L. Black, K. Nichols, and S. L. Blake, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers." RFC 2474, Dec. 1998.

[87] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, (New York, NY, USA), pp. 191–205, ACM, 2018.

[88] P. Sun, Y. Hu, J. Lan, L. Tian, and M. Chen, "Tide: Time-relevant deep reinforcement learning for routing optimization," *Future Generation Computer Systems*, vol. 99, pp. 401 – 409, 2019.

[89] C. Yu, J. Lan, Z. Guo, and Y. Hu, "Drom: Optimizing the routing in software-defined networks with deep reinforcement learning," *IEEE Access*, vol. 6, pp. 64533–64539, 2018.

[90] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, p. 16, Jun 2018.

[91] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges," *IEEE Communications Surveys Tutorials*, vol. 21, pp. 393–430, Firstquarter 2019.

[92] Y. Zhao, Y. Li, X. Zhang, G. Geng, W. Zhang, and Y. Sun, "A survey of networking applications applying the software defined networking concept based on machine learning," *IEEE Access*, vol. 7, pp. 95397–95417, 2019.

[93] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.

[94] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "5g-enabled tactile internet," *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 460–473, March 2016.

[95] T. Bonald and L. Massoulié, "Impact of fairness on internet performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 29, pp. 82–91, June 2001.

[96] K. Phemius and M. Bouet, "Monitoring latency with openflow," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pp. 122–125, Oct 2013.

[97] "An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware." RFC 826, Nov. 1982.

[98] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, (New York, NY, USA), pp. 19:1–19:6, ACM, 2010.

[99] W. Almesberger and E. Ica, "Linux network traffic control - implementation overview," 07 2001.

[100] C.-H. Hsu and U. Kremer, "Iperf: A framework for automatic construction of performance prediction models," in *Workshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France*, Citeseer, 1998.

[101] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "Kvm: the linux virtual machine monitor," in *In Proceedings of the 2007 Ottawa Linux Symposium (OLS'-07*, 2007.

[102] D. Beserra, F. Oliveira, J. Araujo, F. Fernandes, A. Araújo, P. Endo, P. Maciel, and E. D. Moreno, "Performance evaluation of hypervisors for hpc applications," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 846–851, Oct 2015.

[103] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[104] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," 2015.

Appendix

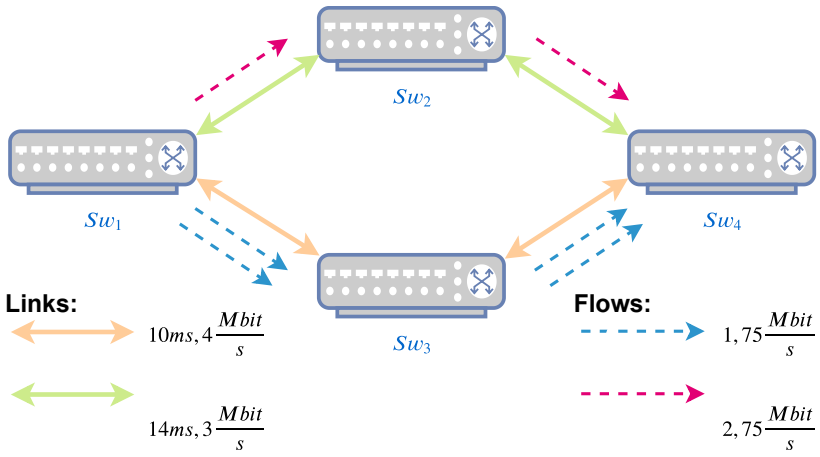


Figure 5.1: Topology containing four switches and three flows with a capacity of $4Mbit/s$ over the path with the lowest delay of $20ms$.

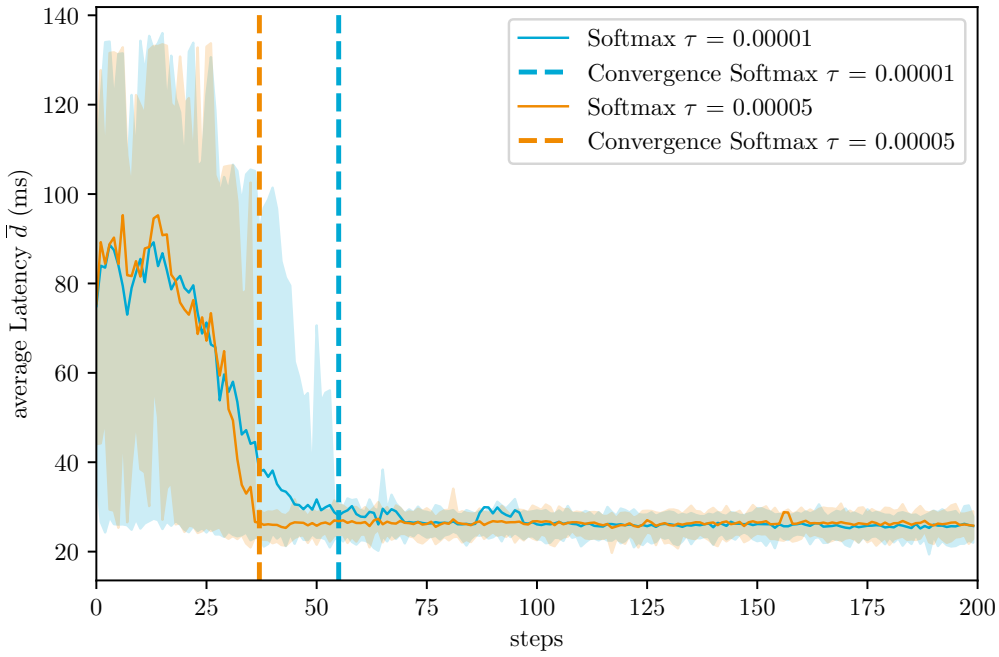


Figure 5.2: Convergence times of average of $\overline{d}$

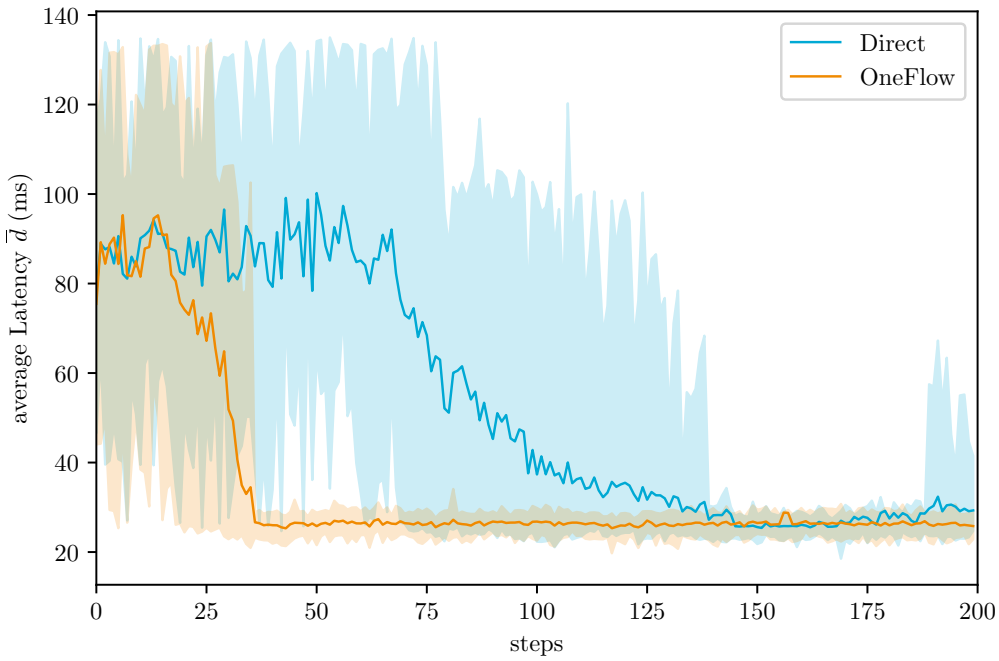


Figure 5.3: $\bar{d}$ over steps for Q-learning and SARSA

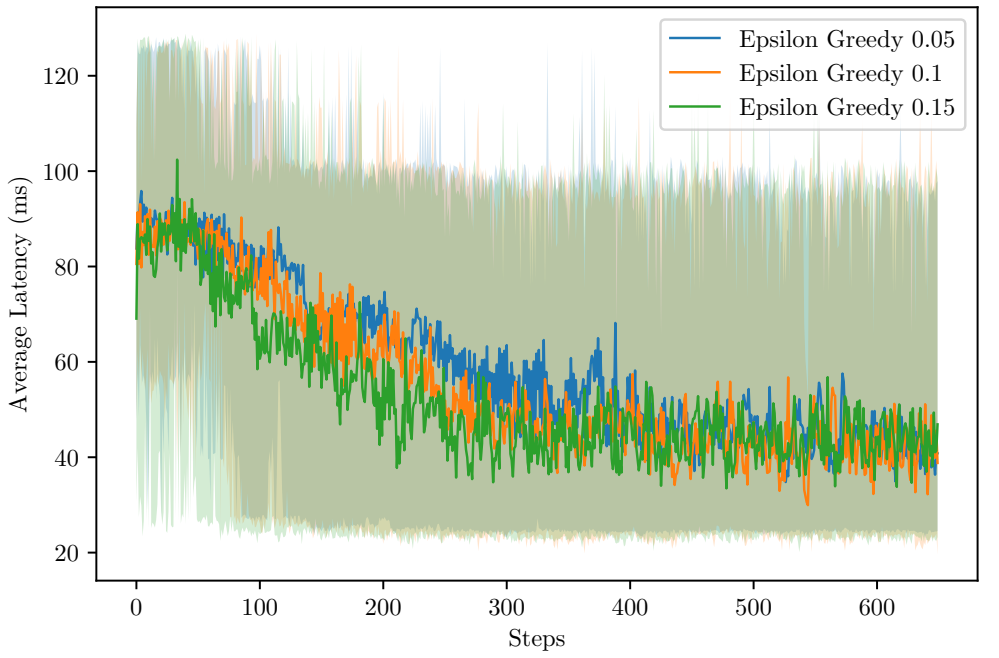


Figure 5.4: $\bar{d}$ over steps with ε-greedy method with a varying ε

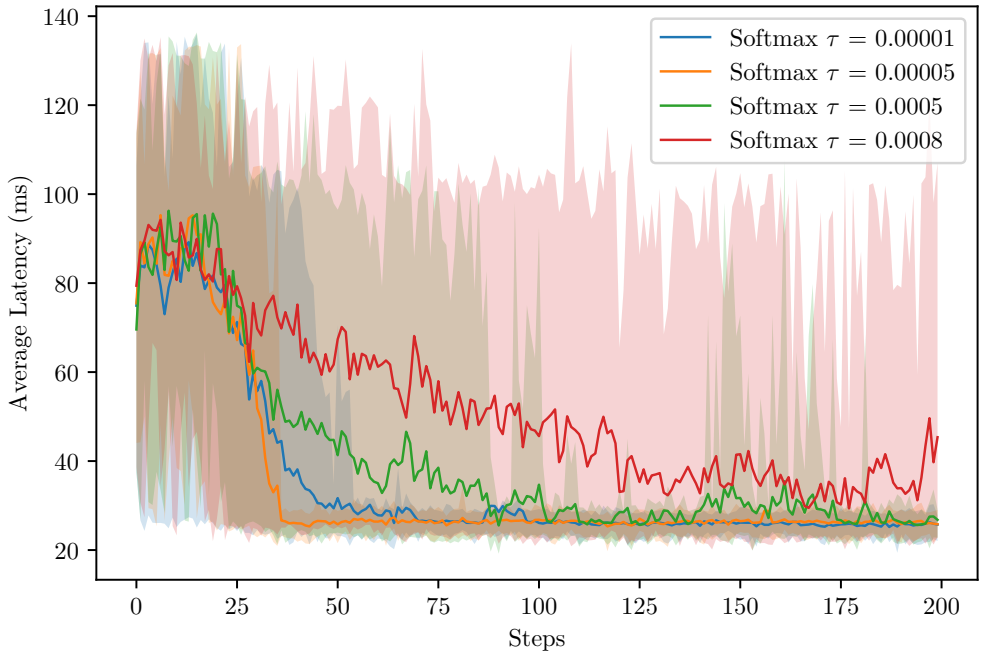


Figure 5.5: $\overline{d}$ over steps with the softmax strategy with different values for τ

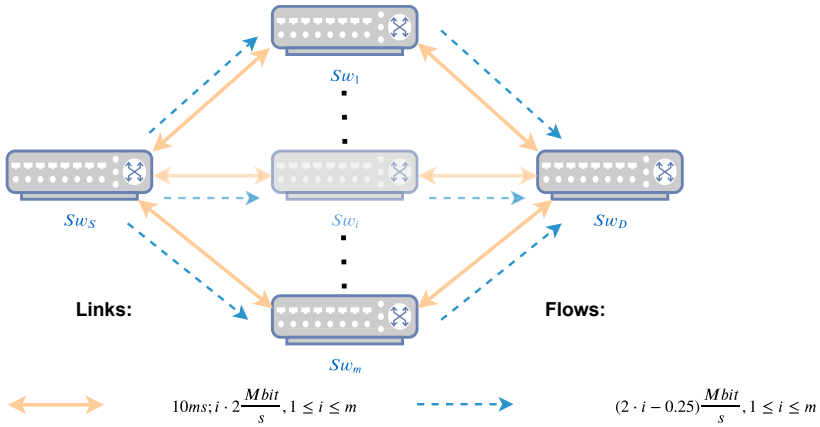


Figure 5.6: Topology with m intermediate switches for the scalability scenario

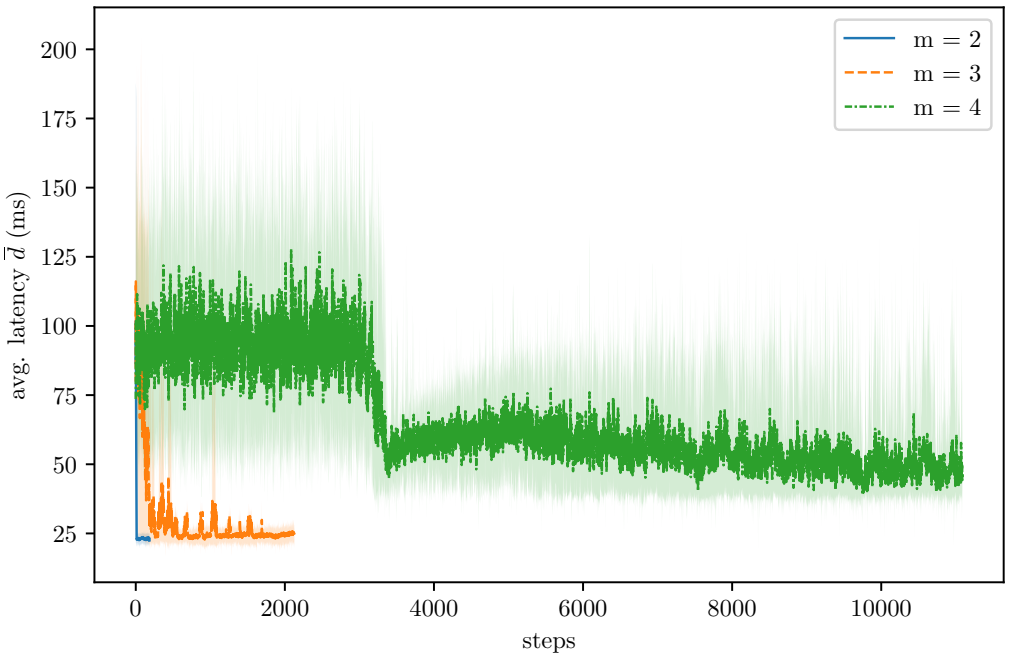


Figure 5.7: $\overline{d}$ over steps of the RL system with a varying scalability level

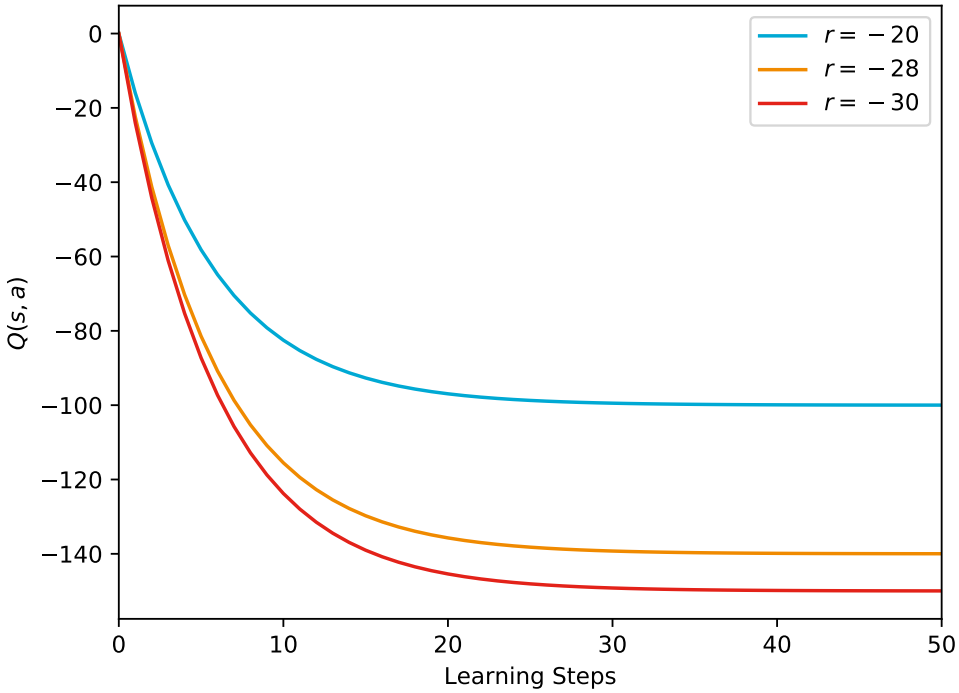


Figure 5.8: Progression of $Q(s, a)$ over time when receiving a specific reward r every step for different rewards

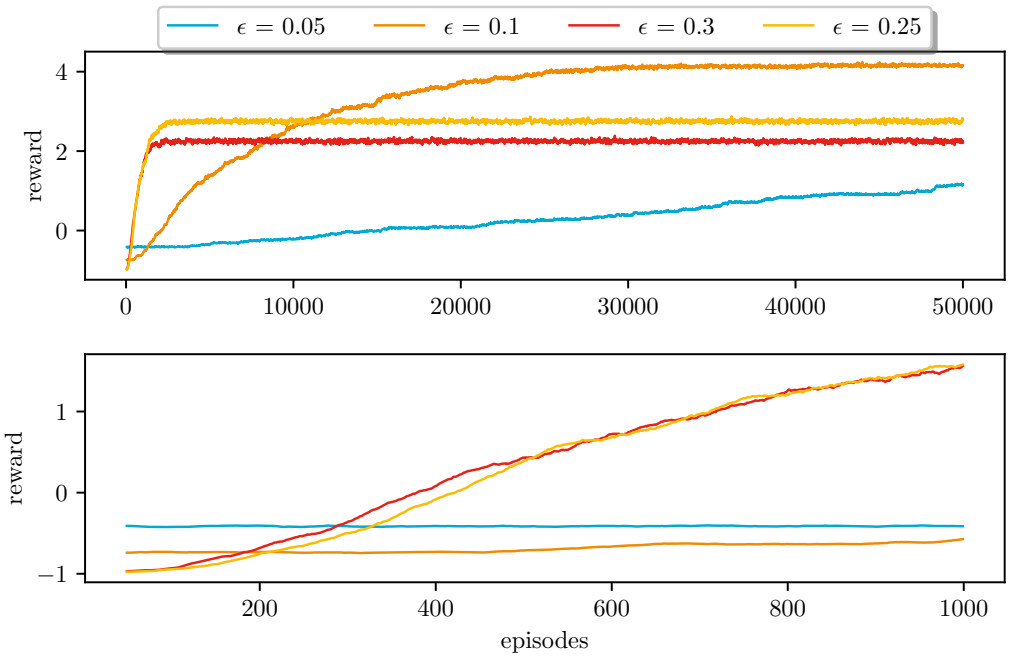


Figure 5.9: Rewards per episodes over time as an average of 200 iterations with ε - greedy exploration method for different ε

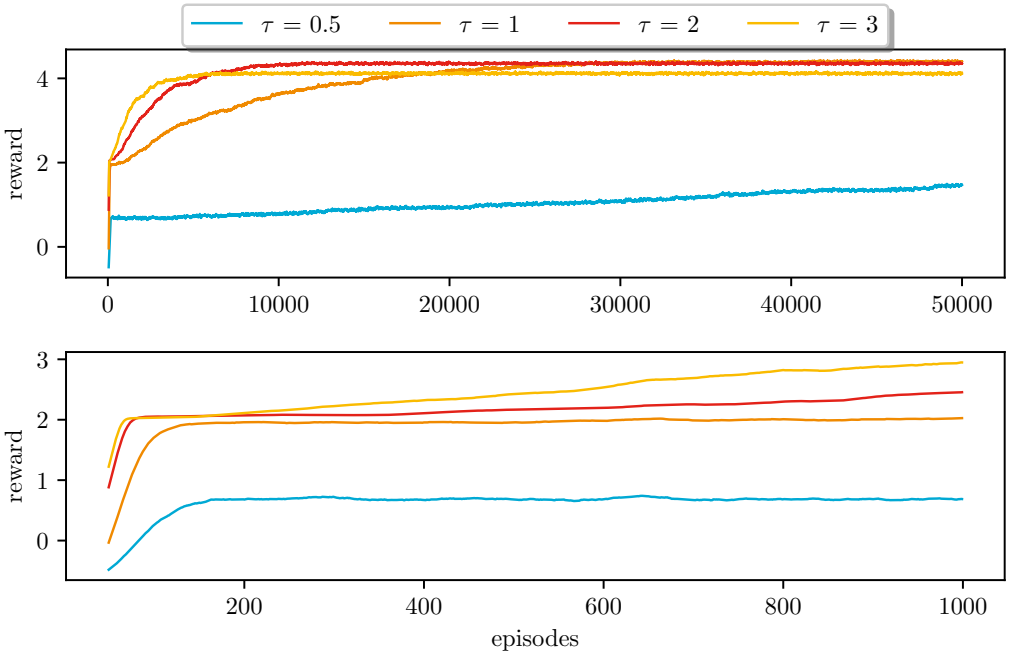


Figure 5.10: Rewards per episodes over time as an average of 200 iterations with softmax exploration method for different τ

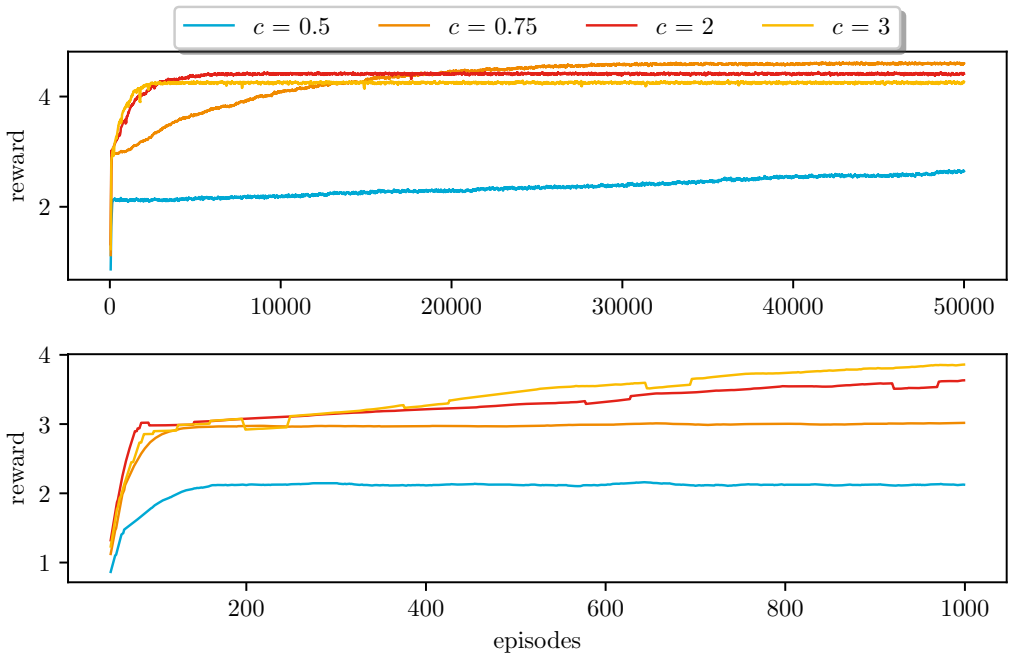


Figure 5.11: Rewards per episodes over time as an average of 200 iterations with UCB exploration method for different c

Listing 5.1: Default configuration

```
class LMode(Enum):
  SHORTEST_PATH = -1
  Q_LEARNING = 1
  SARSA = 2
class ExplorationMode(Enum):
  CONSTANT_EPS = 0
  FALLING_EPS = 1
  SOFTMAX = 2
  UCB = 3
class PathInitialization(Enum):
  SPF = 1
  RANDOM = 2
class ActionMode(Enum):
  ONE_FLOW = 1
  DIRECT_CHANGE = 2
class Config(object):
  learning_mode = LMode.Q_LEARNING
  # Learning rate
  alpha = 0.8
  # Discount Factor
  gamma = 0.8
  # Eps-greedy
  epsilon = 0.05
  # Softmax
  temperature = 0.00005
  # UCB
  explorationDegree = 5
  # Exploration Mode
  exploration_mode = ExplorationMode.SOFTMAX
  # Action mode
  action_mode = ActionMode.ONE_FLOW
  # how long to wait until starting to gather new rewards
  T_wait = 2
  # measurement time for each load level in each iteration
  run_time = 1440
  # load levels
  load_levels = [10]
  # number of iterations per measurement
  iterations = 1
  # init_value for Softmax
  softmax_init_value = - float('inf')
  # if LoadLevel Test Case
  resetQTestFlag = True
  # splitting up - each load level different log file
  splitUpLoadLevelsFlag = False
  # if merging QTables when new flow joins
  mergingQTableFlag = False
  # initialise with shortest path first or with a random selected path
  path_initialization = PathInitialization.RANDOM
```