TECHNISCHE UNIVERSITÄT DRESDEN

ComNets
Deutsche Telekom Chair of Communication Networks

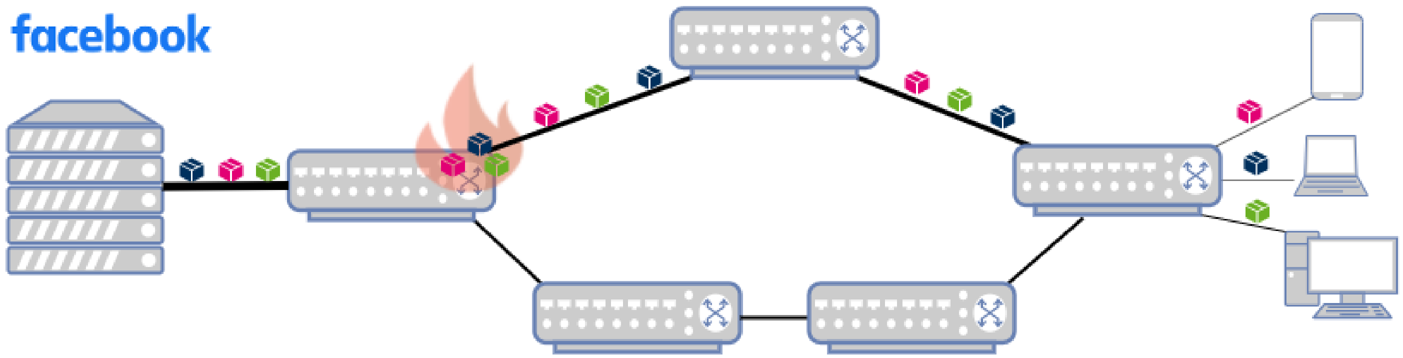
Peter Sossalla

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

Diploma Thesis

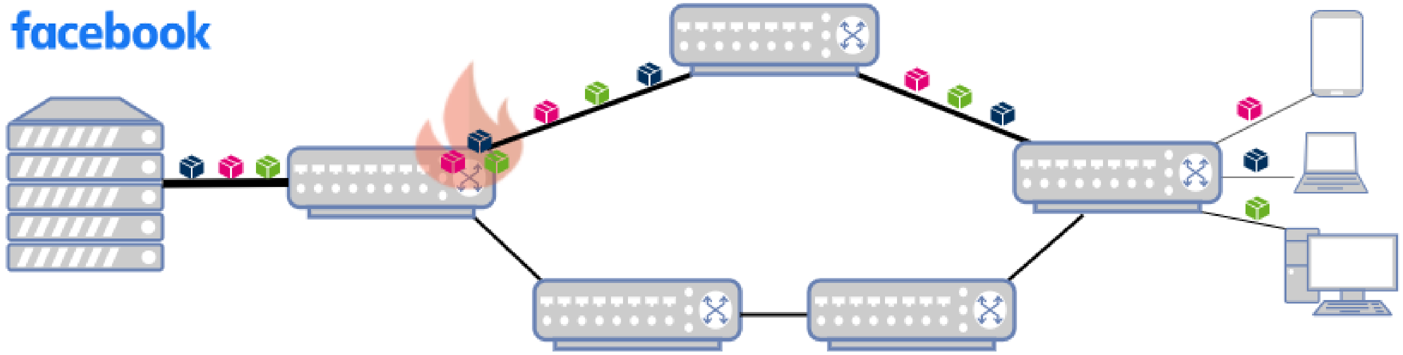
7.11.2019

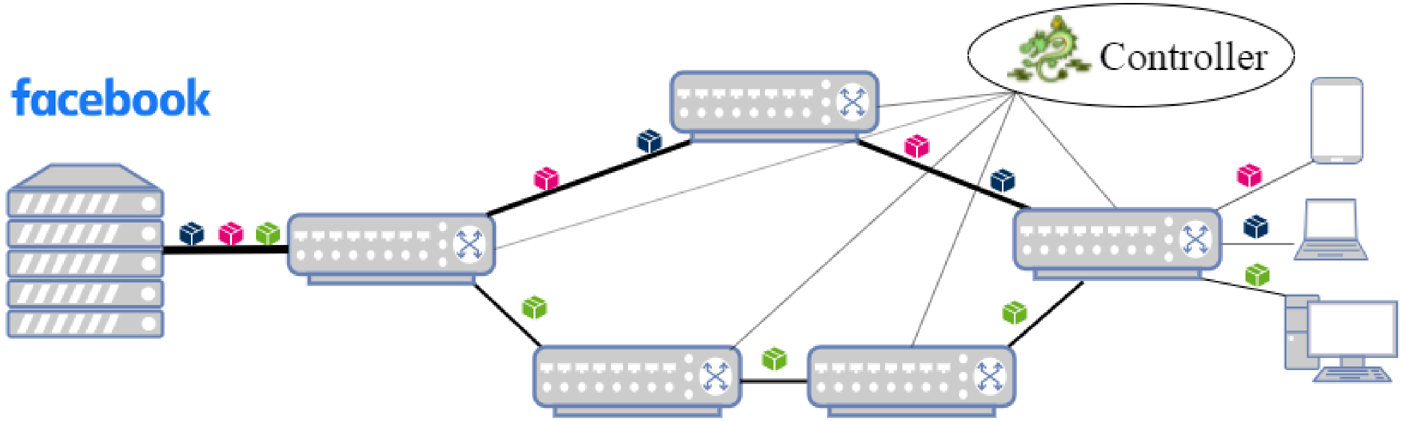
Motivation



- Problem: rising and varying traffic demand
- Objective: deliver traffic with the best possible performance
- SPF does not consider demand, other paths or the influence of its routing decisions
- Overprovisioning → higher costs
- Enterprises such as Facebook use Software-Defined Networking (SDN)

Motivation





Reinforcement Learning

Agent interacts with the environment

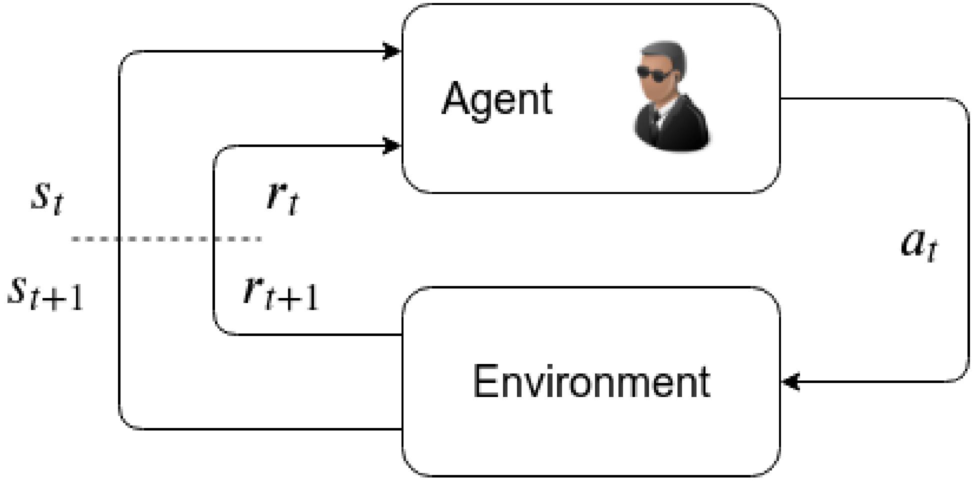
State s, Action a, Reward r

Q-value $Q(s,a)$ – Quality of action a in state s

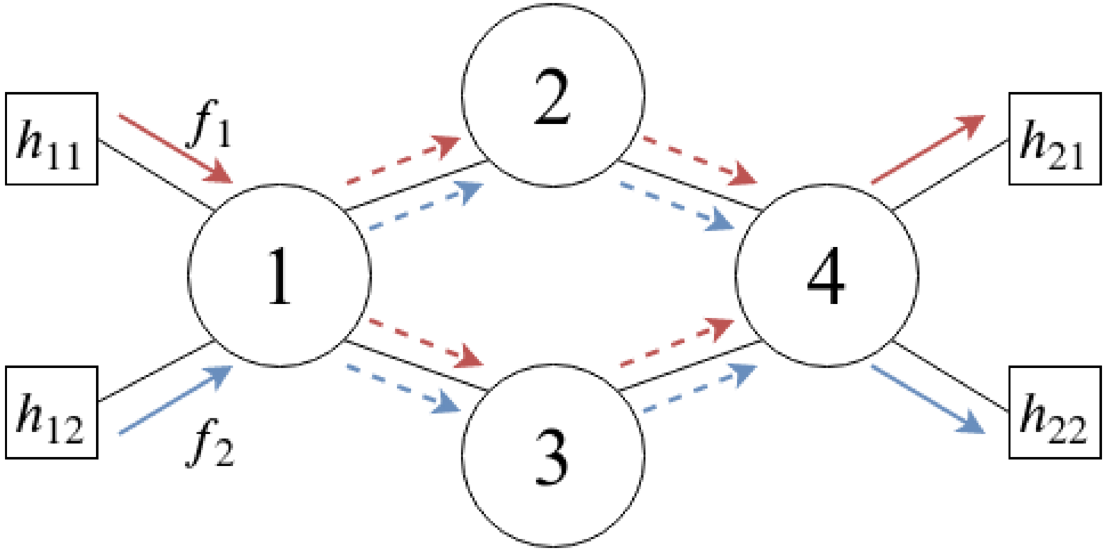
Q-learning for determining $Q(s,a)$

Q-table:

$Q(s,a)$	a_1	a_2
s_1	$Q(s_1,a_1)$	$Q(s_1,a_2)$
s_2	$Q(s_2,a_1)$	$Q(s_2,a_2)$



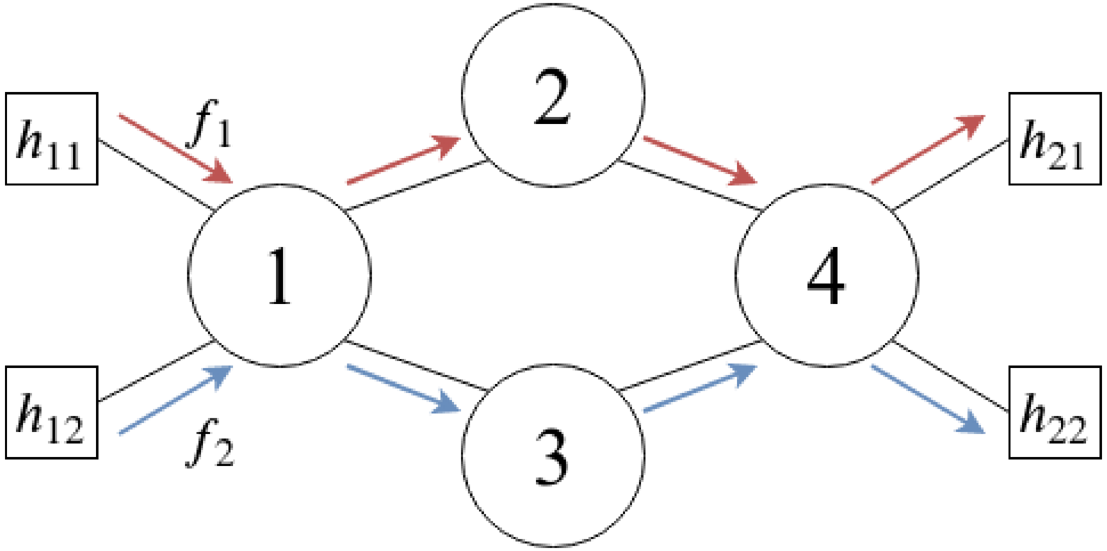
Implementation – States



Flow: [Path]

f_1: [1,2,4] f_2: [1,2,4]	f_1: [1,3,4] f_2: [1,2,4]
f_1: [1,2,4] f_2: [1,3,4]	f_1: [1,3,4] f_2: [1,3,4]

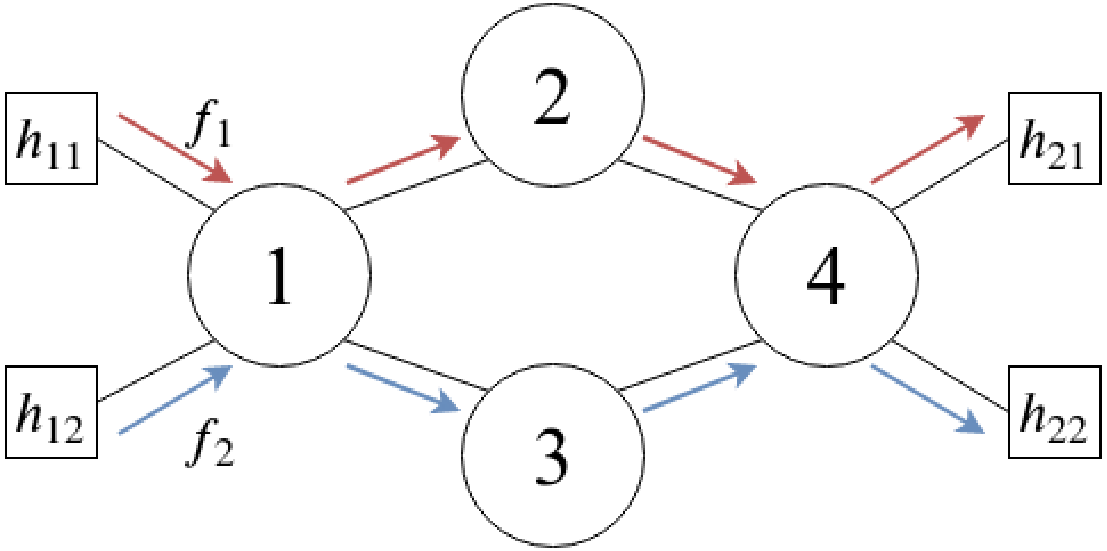
Implementation – States



Flow: [Path]

f_1: [1,2,4] f_2: [1,2,4]	f_1: [1,3,4] f_2: [1,2,4]
f_1: [1,2,4] f_2: [1,3,4]	f_1: [1,3,4] f_2: [1,3,4]

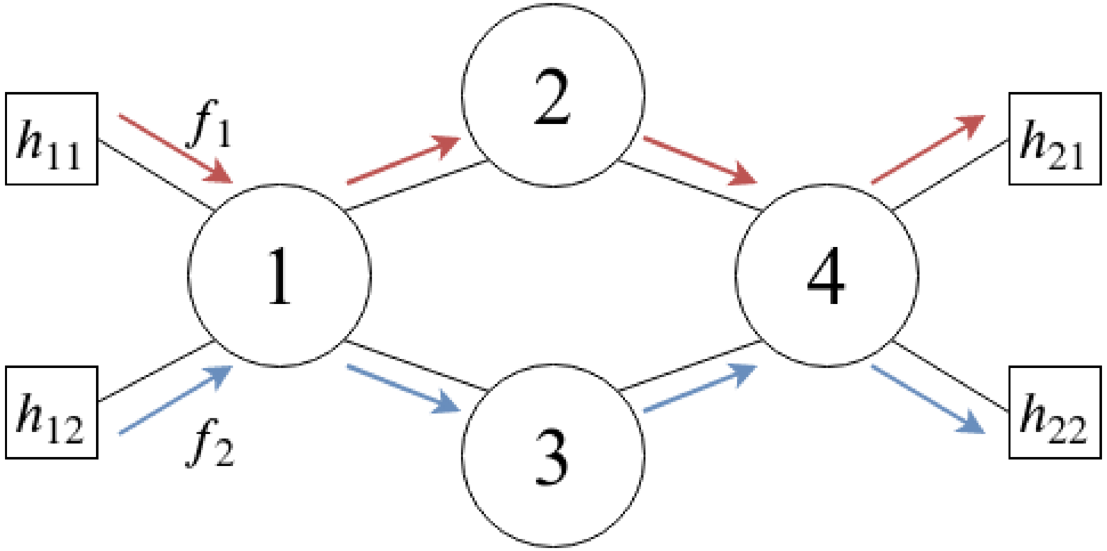
Implementation – Actions



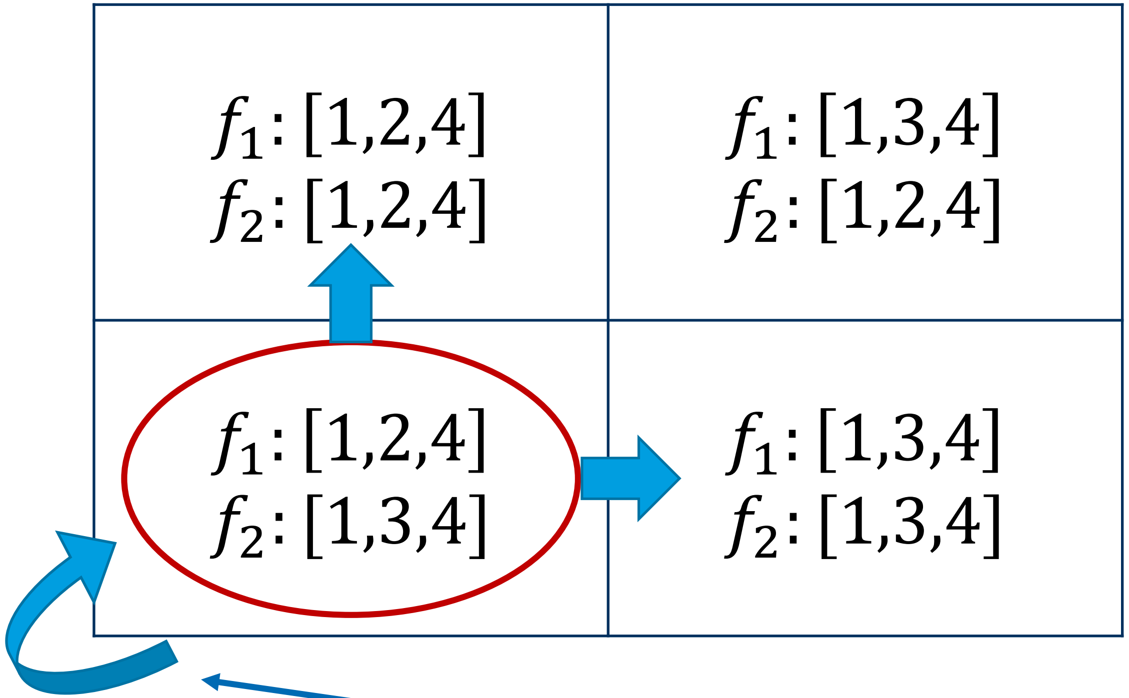
Flow: [Path]

f_1: [1,2,4] f_2: [1,2,4]	f_1: [1,3,4] f_2: [1,2,4]
f_1: [1,2,4] f_2: [1,3,4]	f_1: [1,3,4] f_2: [1,3,4]

Implementation – Actions

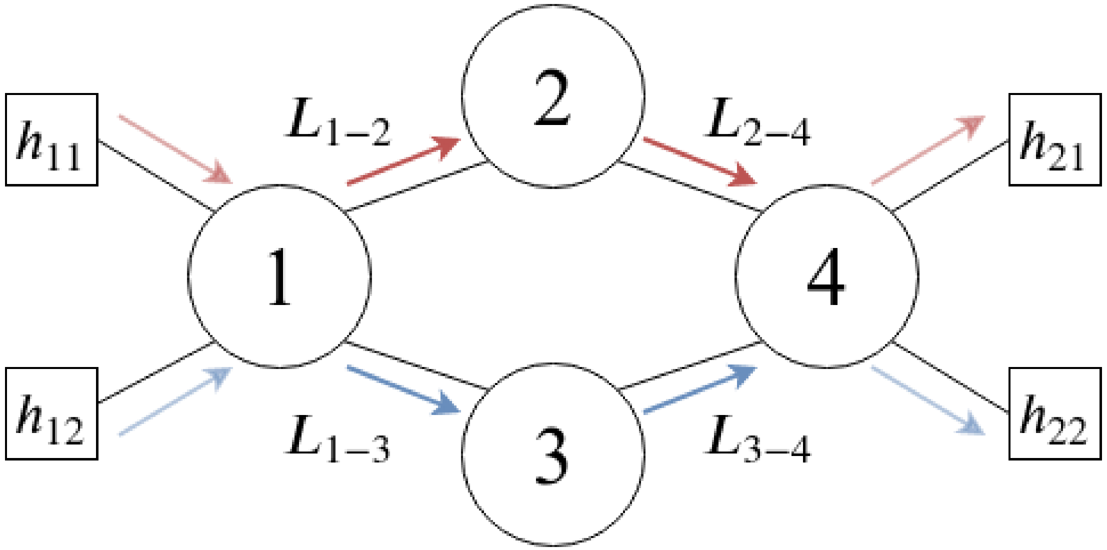


Flow: [Path]



Additionally $\boxed{\text{“No Transition”}}$ action

Implementation – Reward



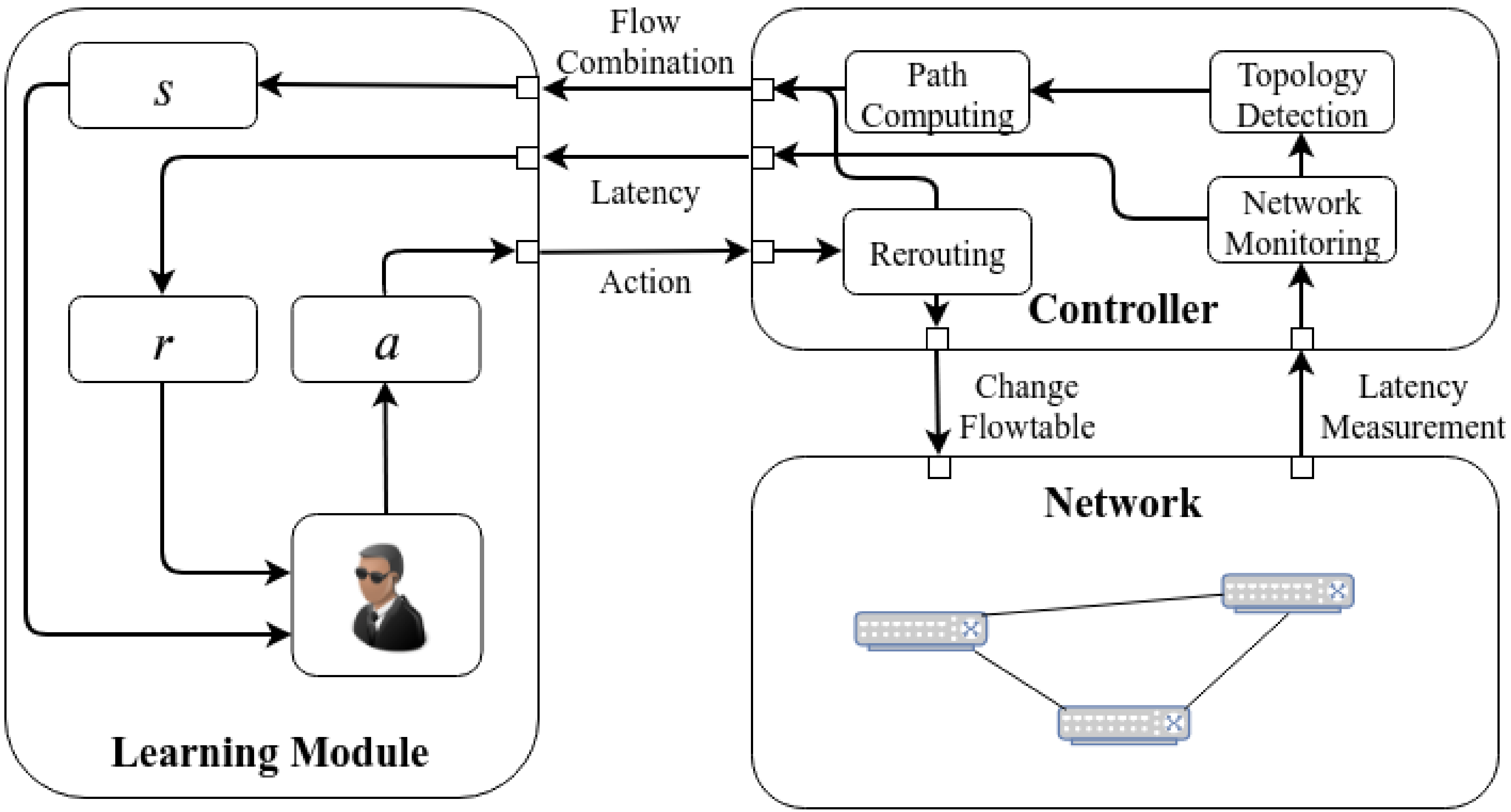
Latencies determined by measurements:

$$L(f_1) = L_{1-2} + L_{2-4}$$
$$L(f_2) = L_{1-3} + L_{3-4}$$

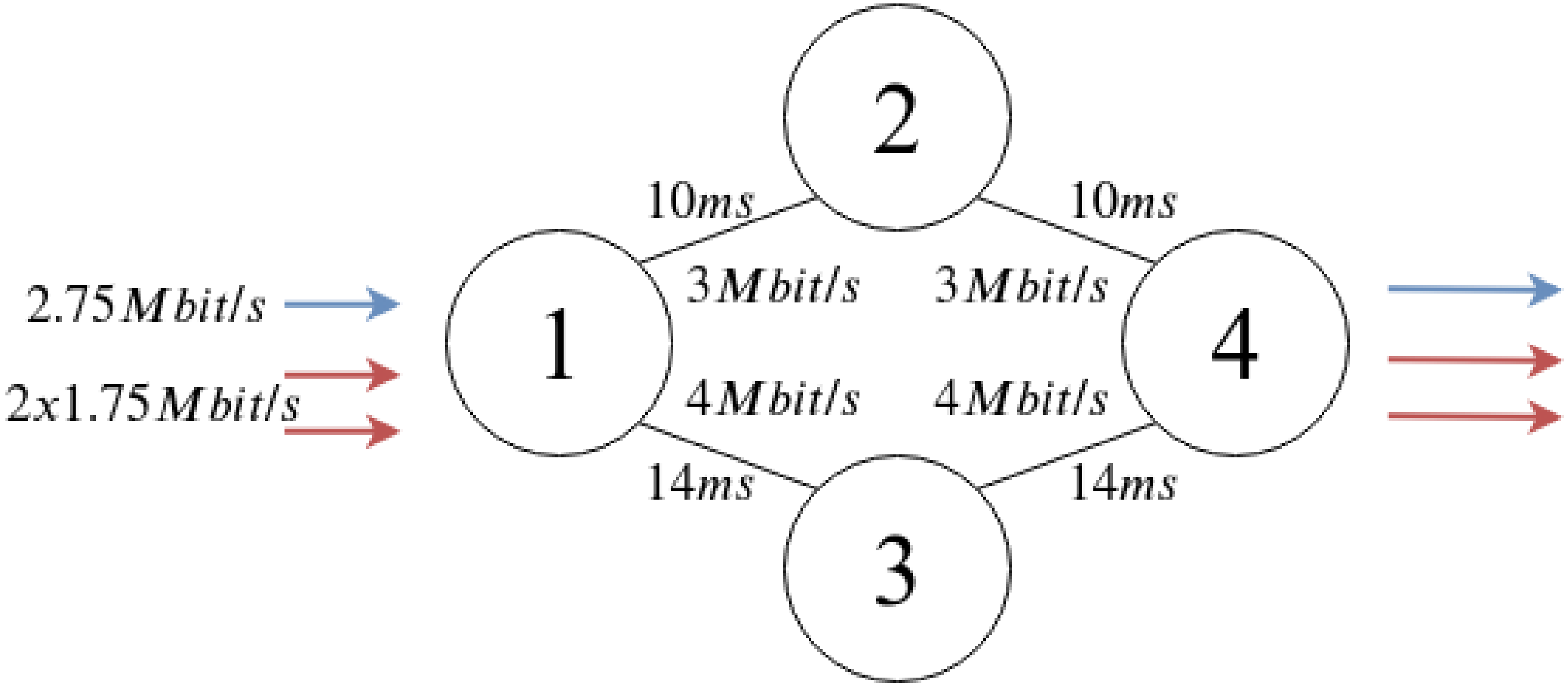
Reward: Root mean square of latencies

$$r = -\sqrt{\frac{L(f_1)^2 + L(f_2)^2}{2}}$$

Implementation - System



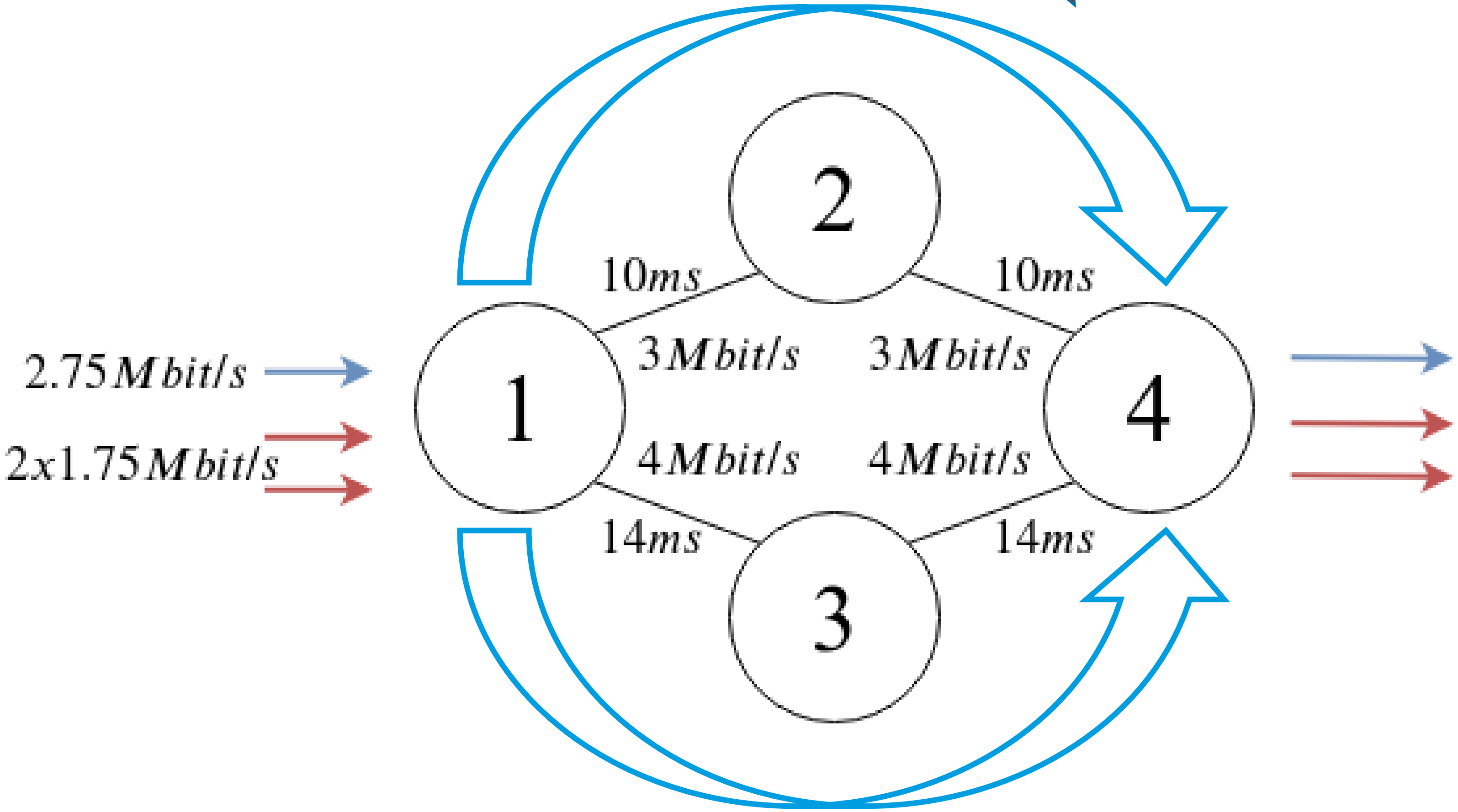
Evaluation - Topology



Evaluation - Topology

Path 1: 20ms

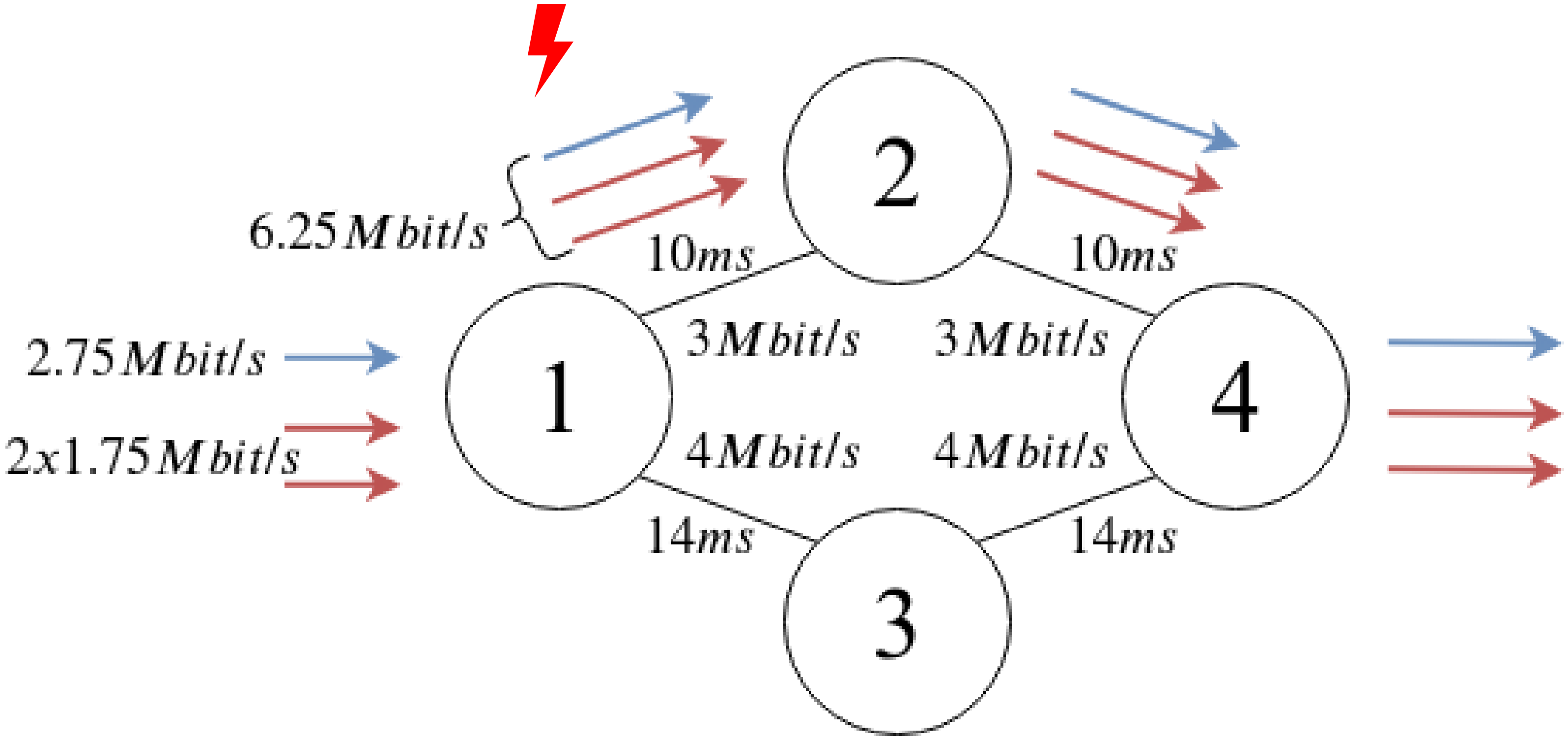
Path with the lowest latency



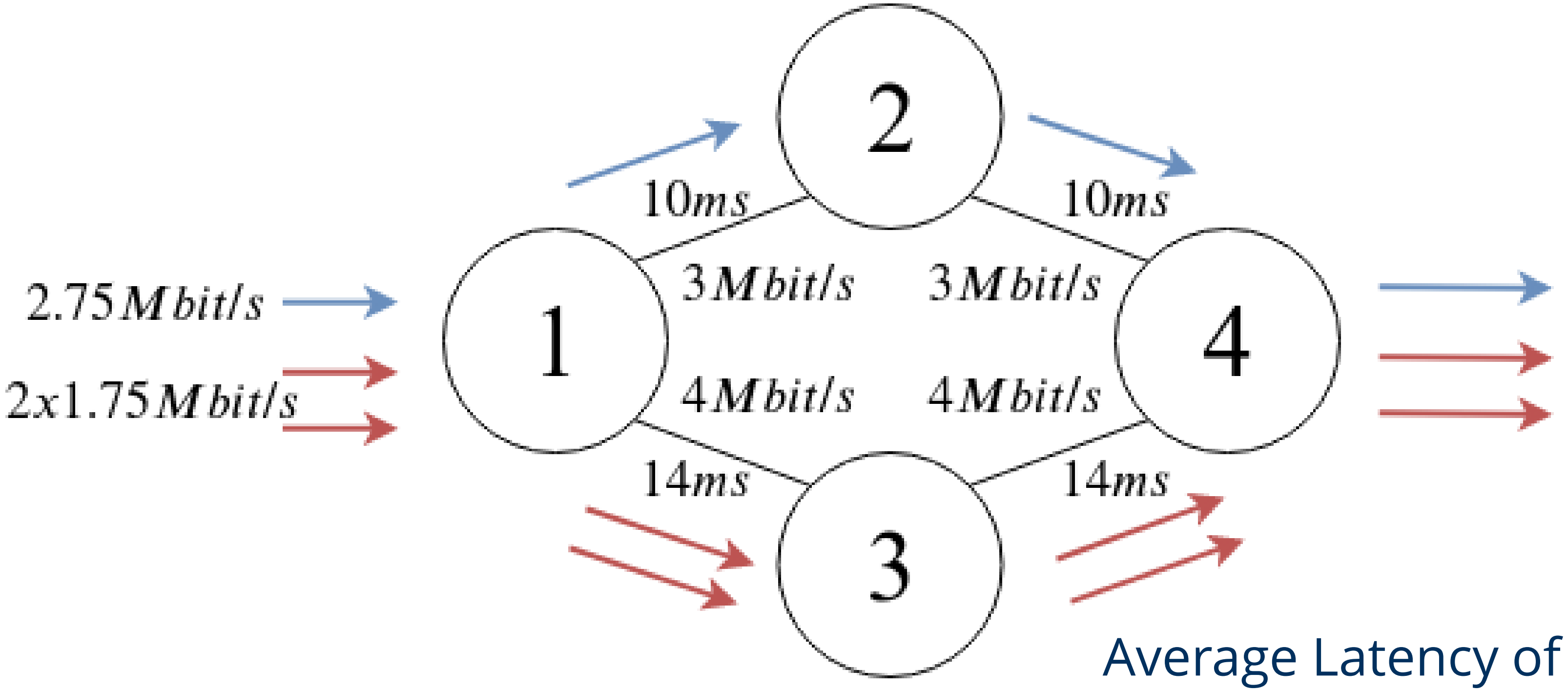
Path 2: 28ms

Q-table size:
$|Q| = 32$

Evaluation – Topology (SPF)



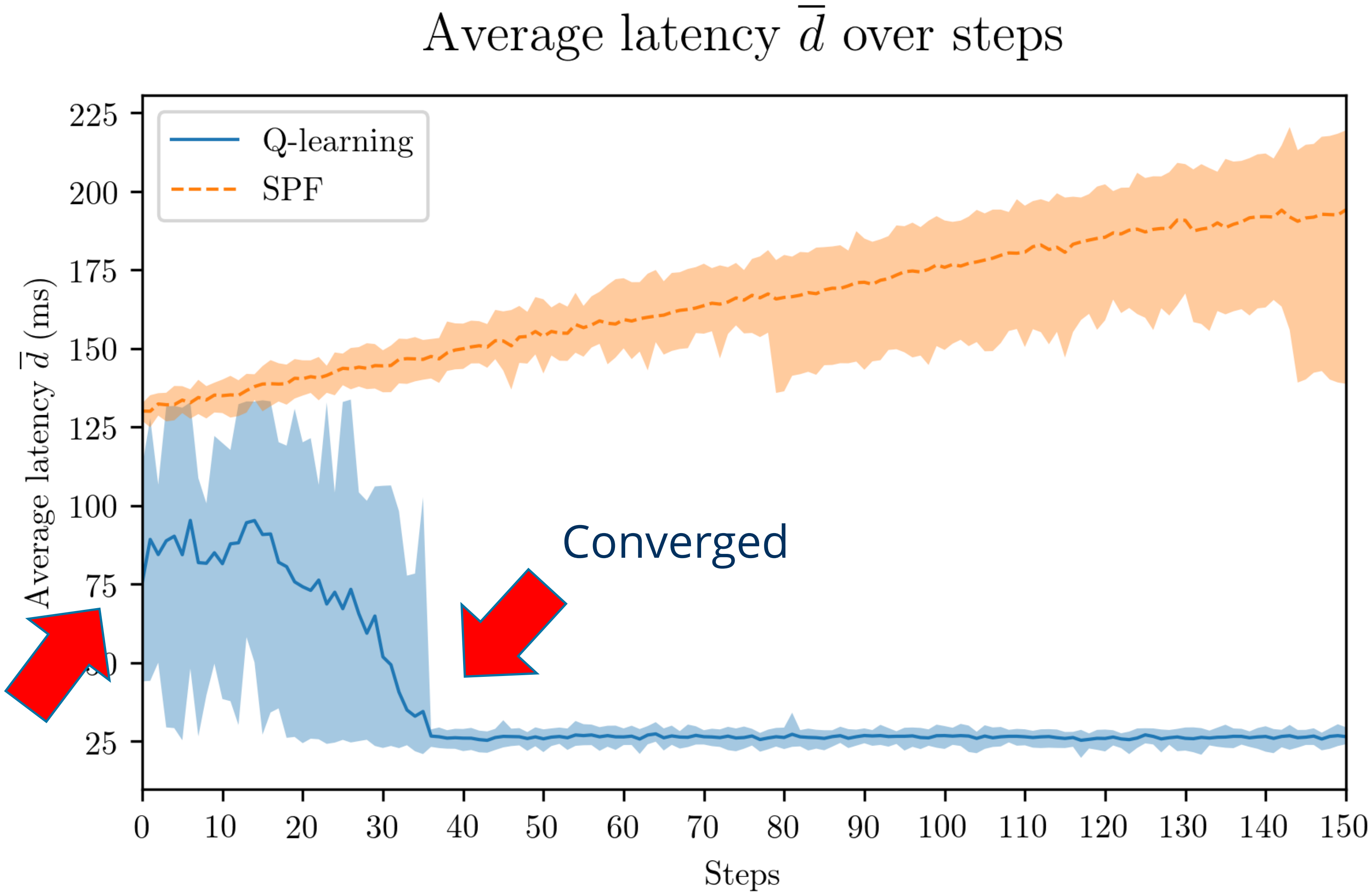
Evaluation – Topology (Optimal)



Average Latency of the flows:

$$\bar{d} = \frac{20\ ms + 2 \cdot 28\ ms}{3} = 25.3\ ms$$

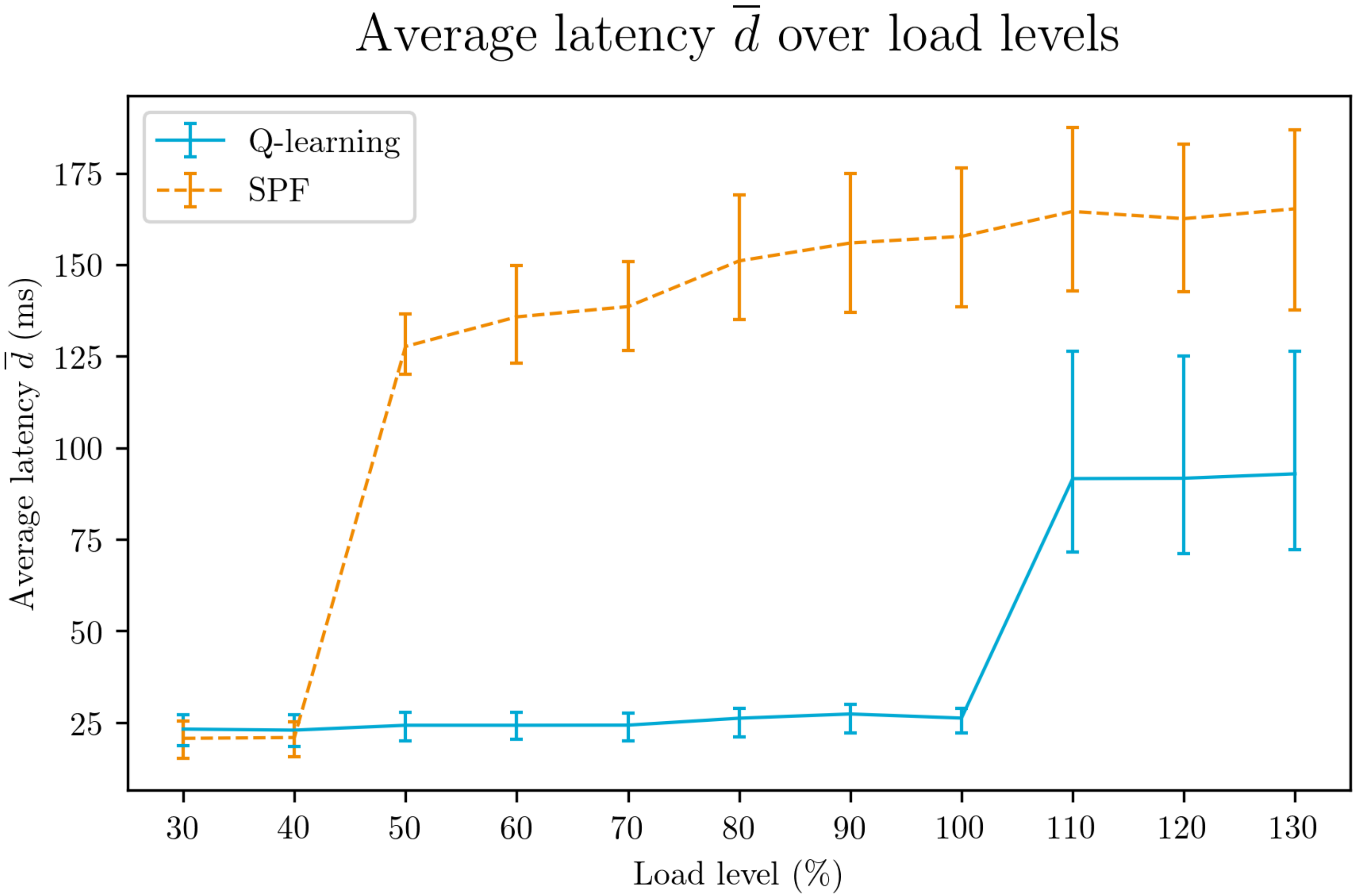
Evaluation – Learning



Paths are selected randomly

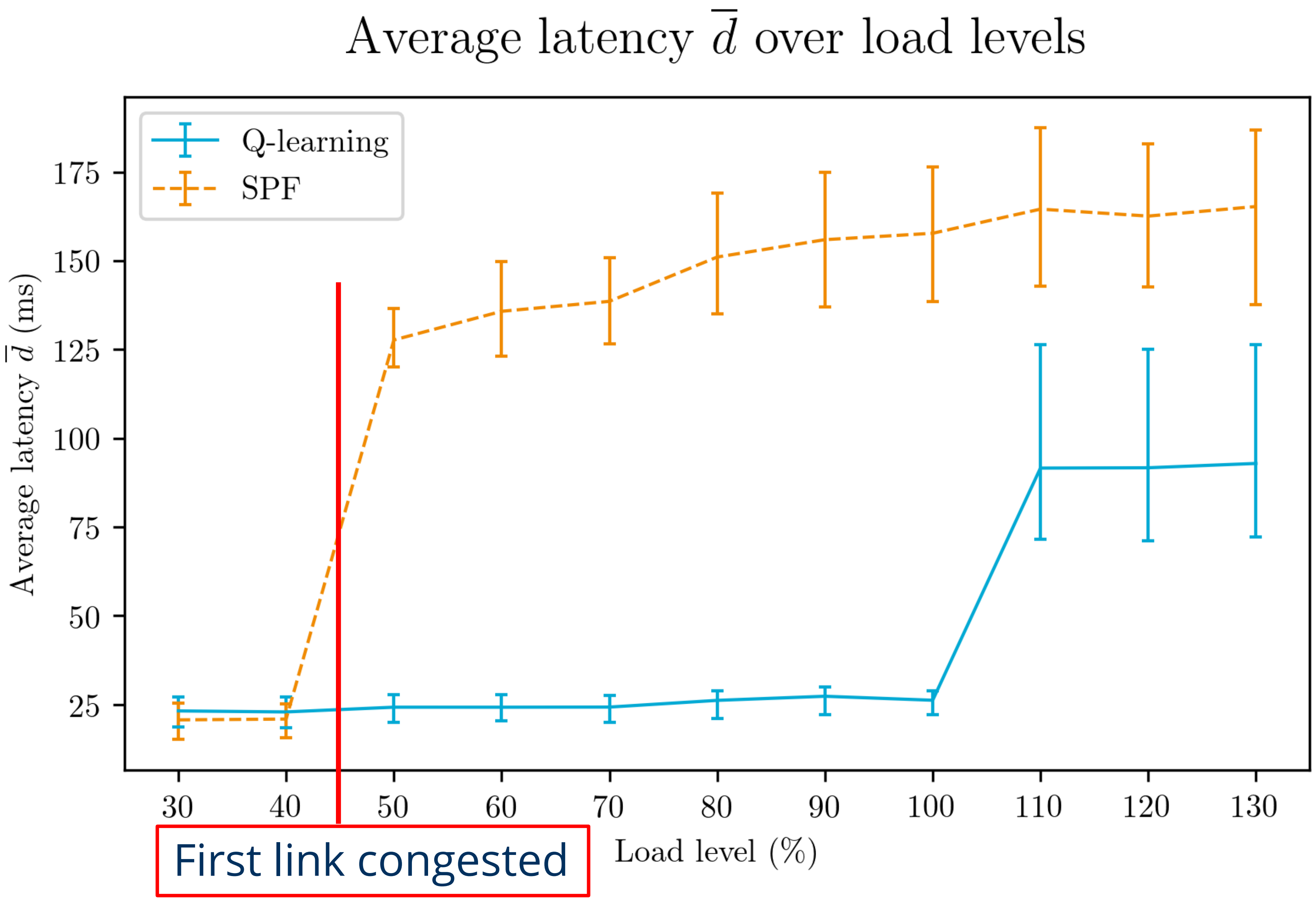
Evaluation – Load Level

How does the system perform for different loads?



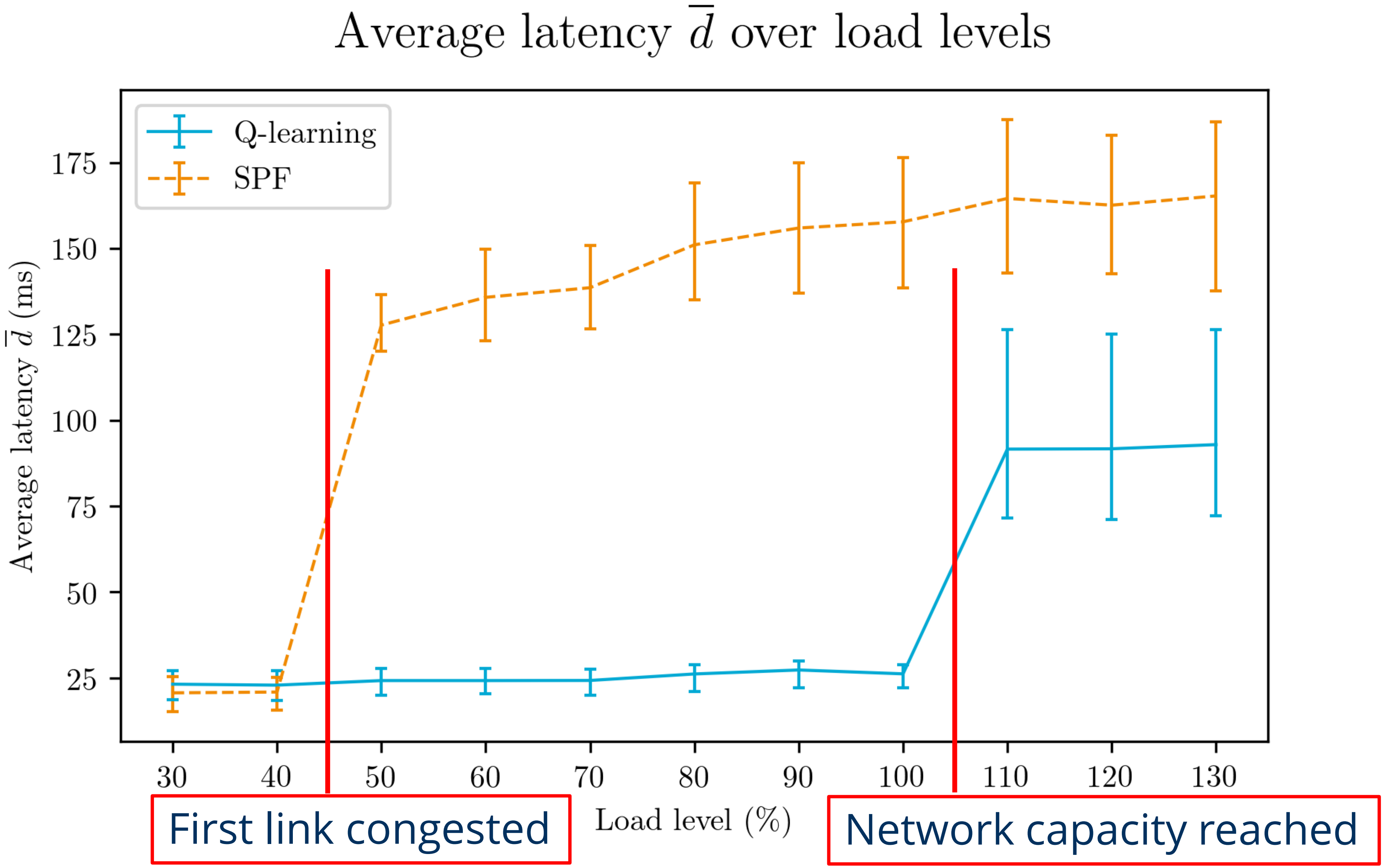
Evaluation – Load Level

How does the system perform for different loads?



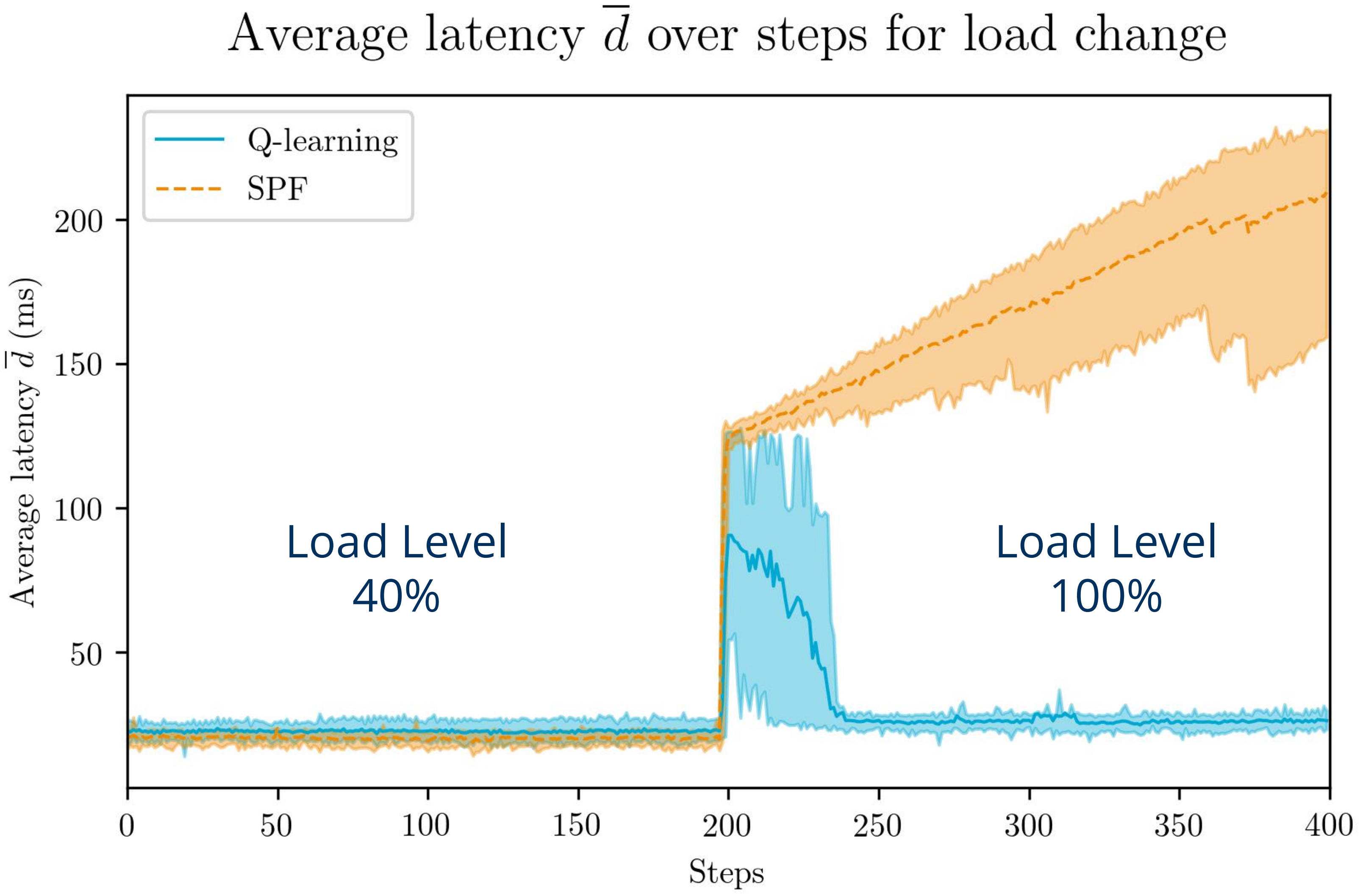
Evaluation – Load Level

How does the system perform for different loads?



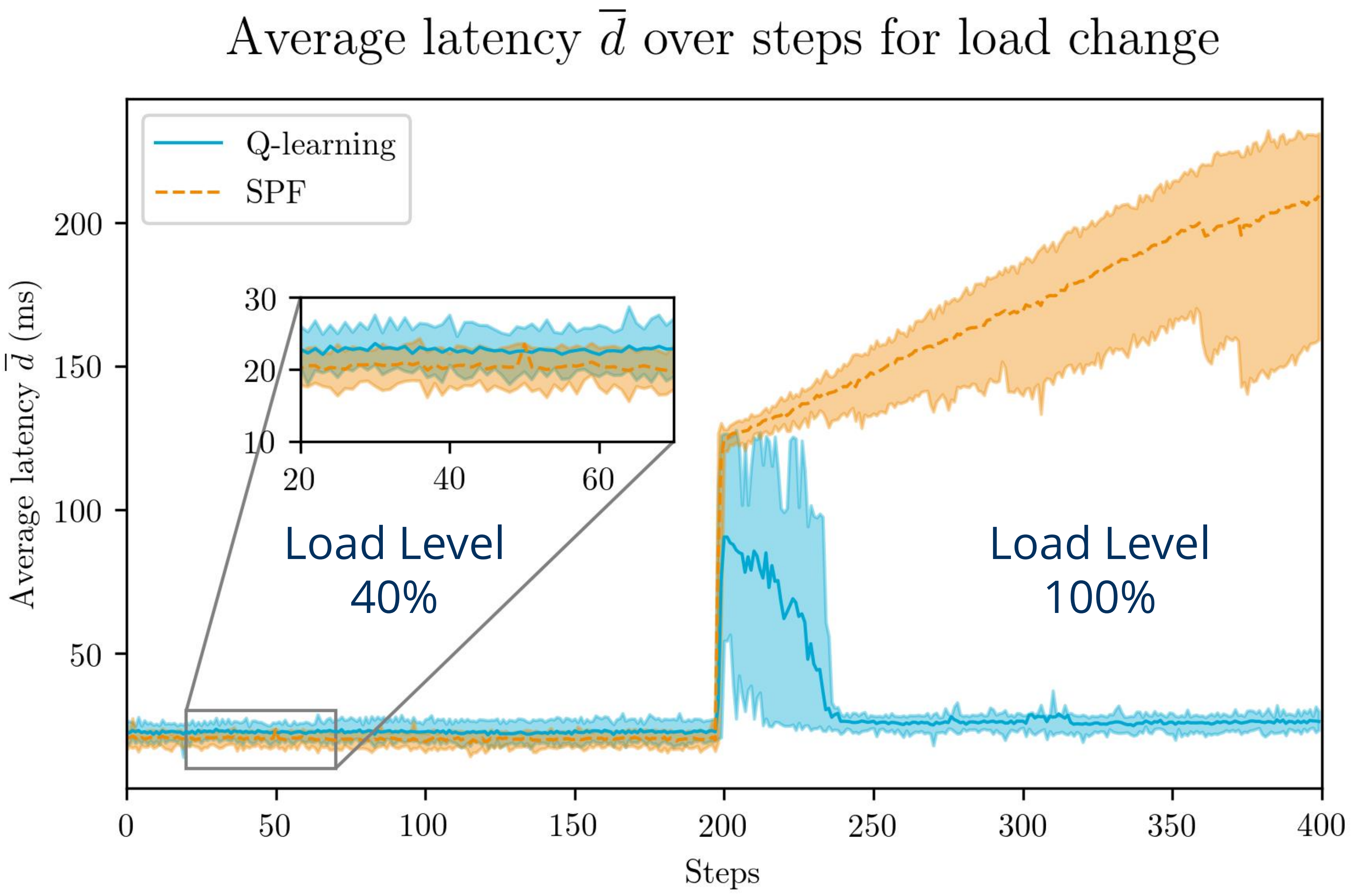
Evaluation – Load Change

How does the system perform in a dynamic load change?



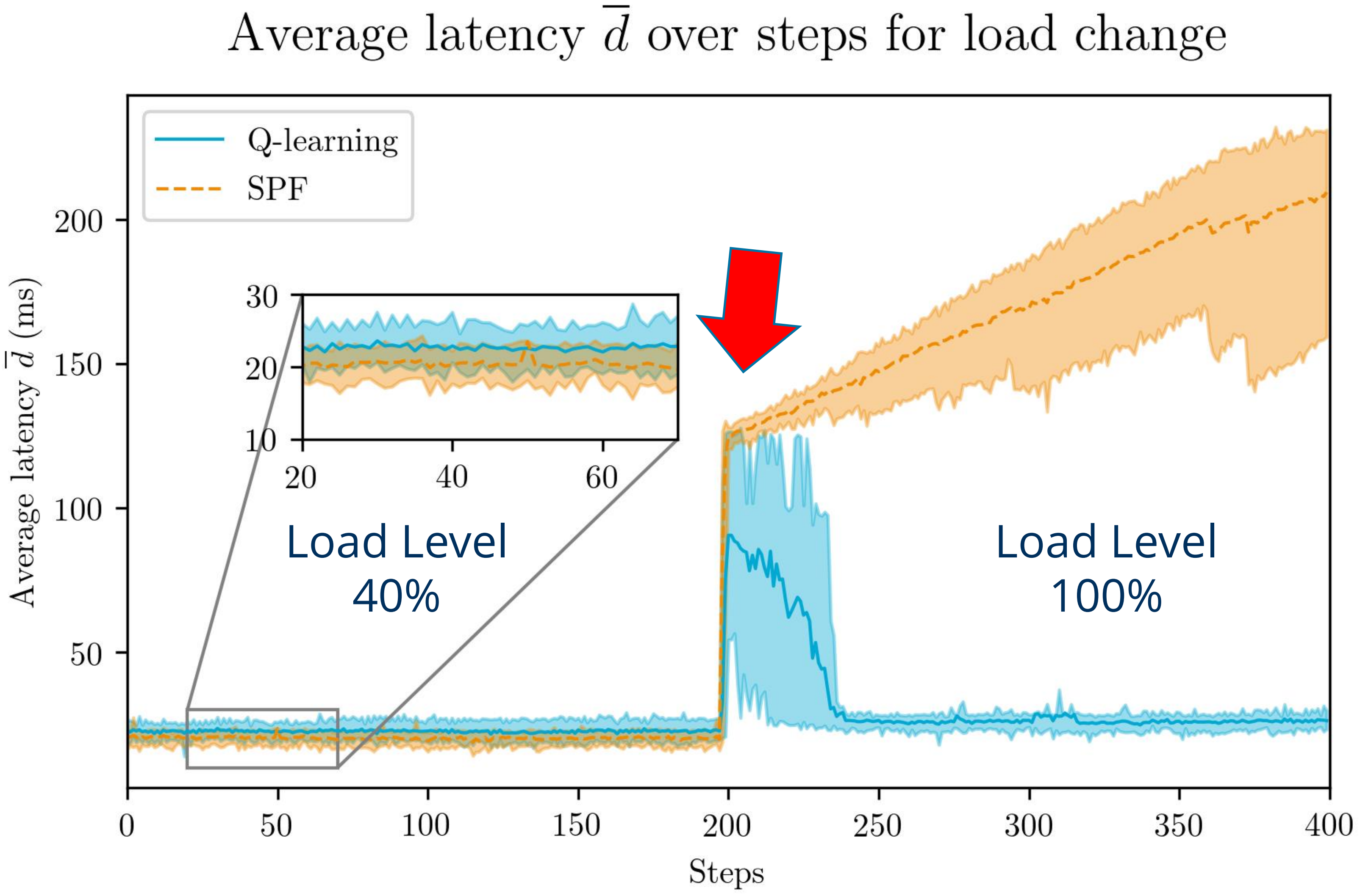
Appendix – Load Change

How does the system perform in a dynamic load change?

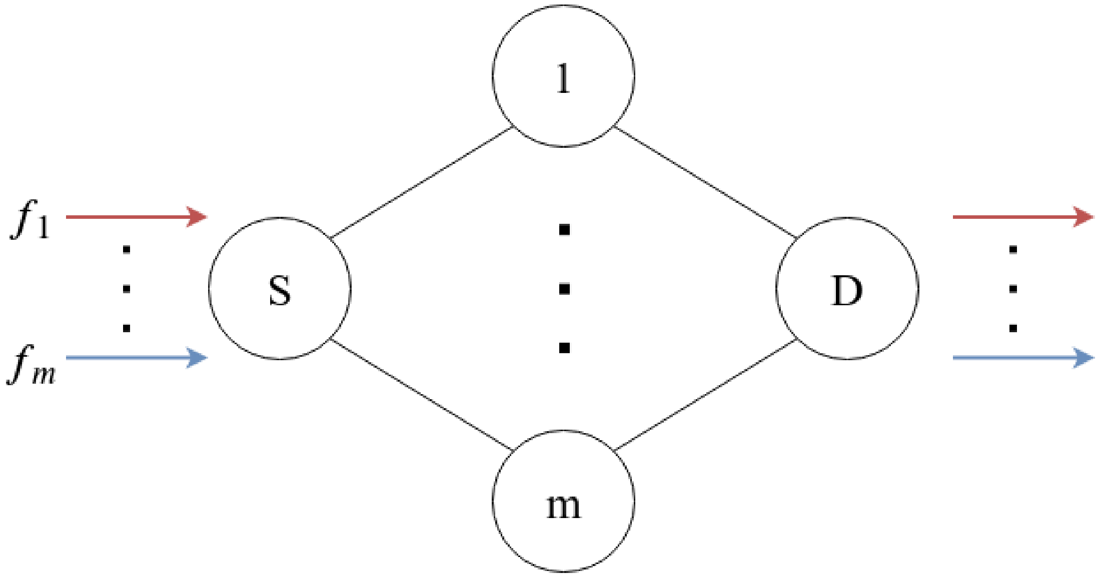


Appendix – Load Change

How does the system perform in a dynamic load change?



Evaluation – Scalability



Scalability level m - Number of flows and intermediate switches

Equal path latencies

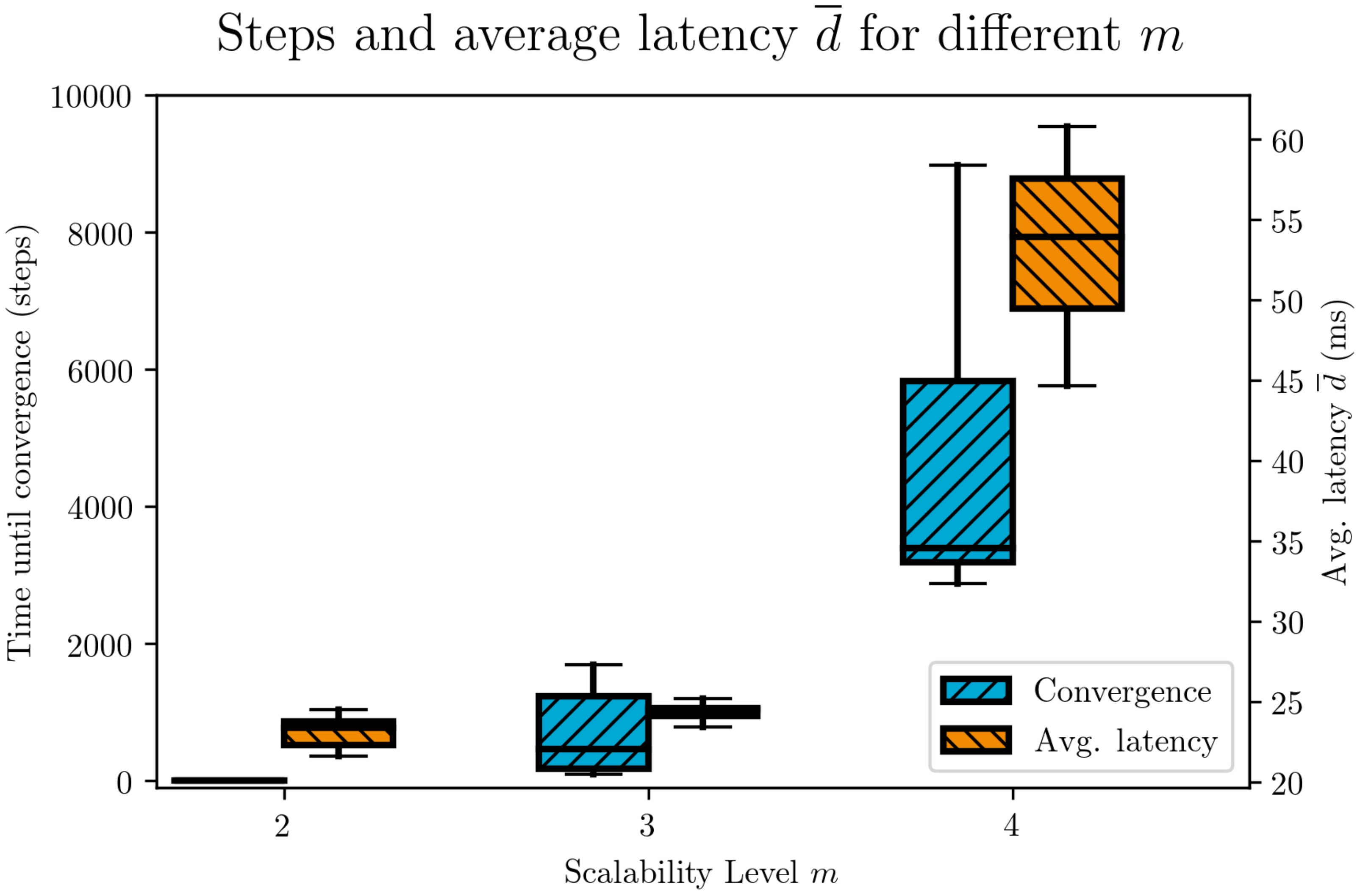
Single uncongested routing state

Q-table entries:

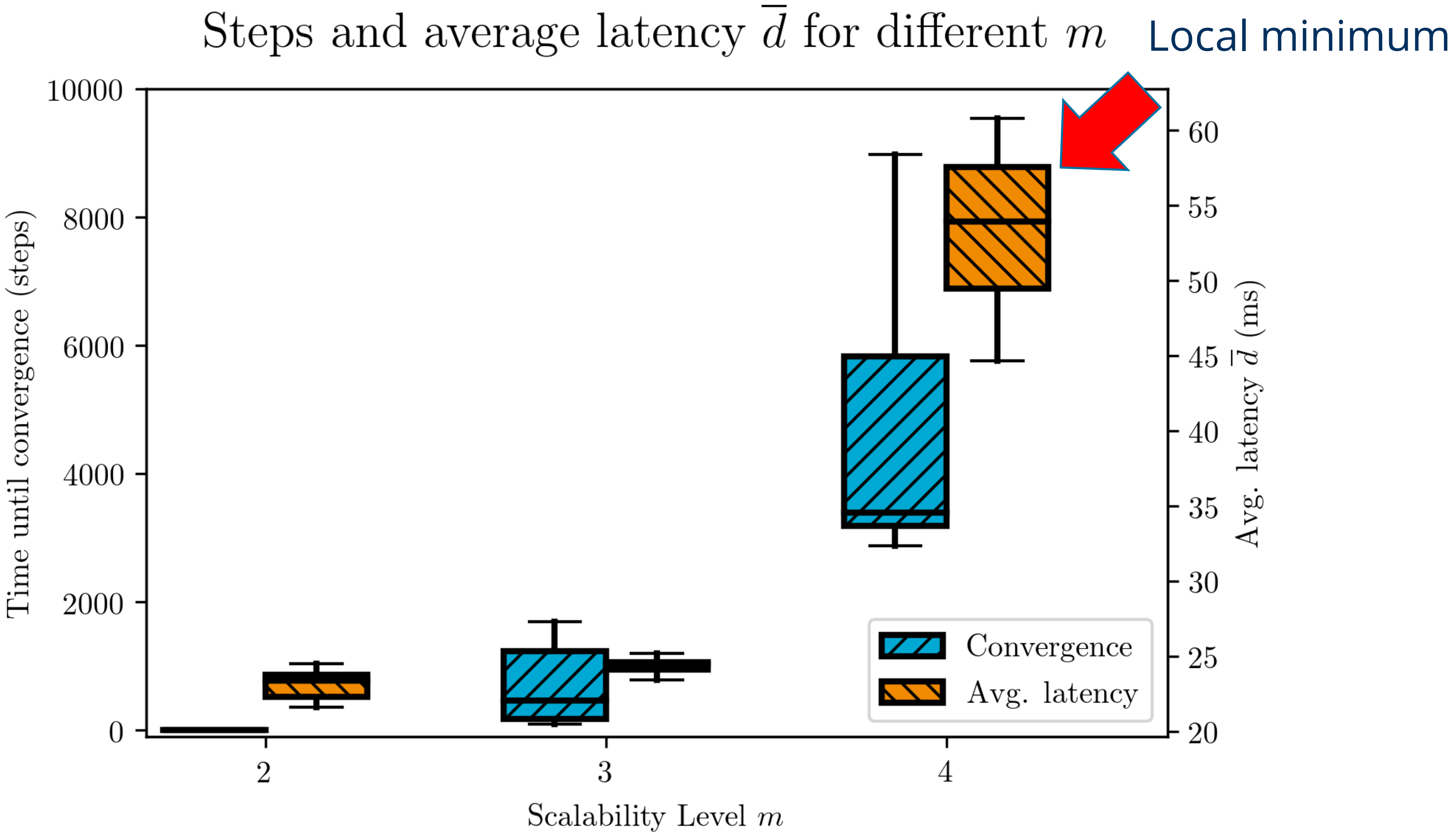
$$|Q| = m^m \cdot (m \cdot (m-1) + 1)$$

→ Number of entries scales exponentially

Evaluation – Scalability



Evaluation – Scalability



Contribution

- Developed latency optimization with Reinforcement Learning
- Framework
  - Capable of latency measurement and dynamic routing
  - Adapts on changes of network loads
  - Can be used for hardware switches
  - Easily expandable or modifiable for further research
- Evaluated in an emulated environment with Mininet

Further Research Questions

- Scalability:
  - Limitation
  - Generalization
  - Bandwidth in state space
- Reward modification → different performance objectives
- Real network topologies
- Hardware switches

Thank you for your attention

Appendix - Q-table

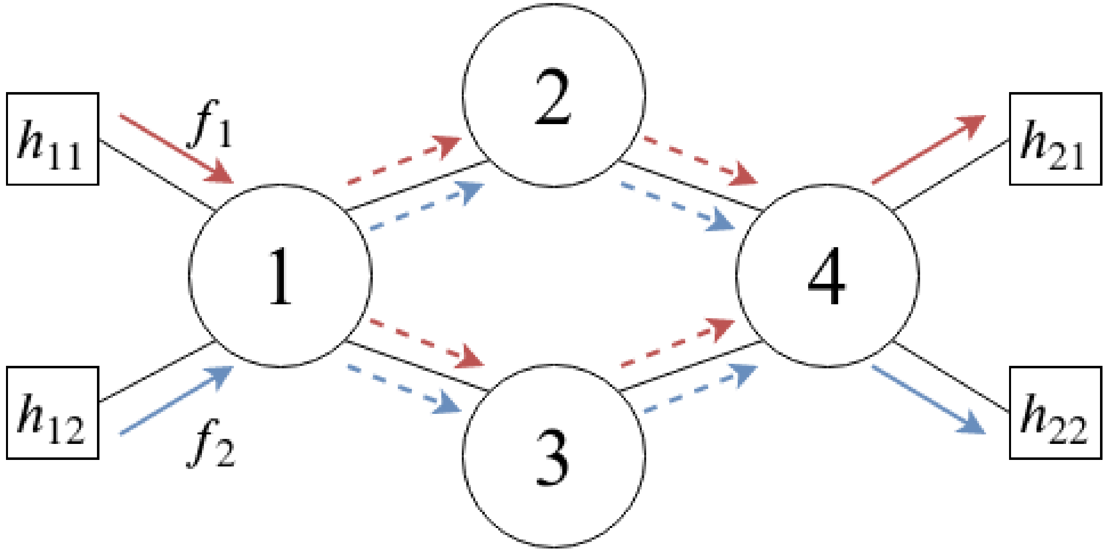
Number of actions:

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

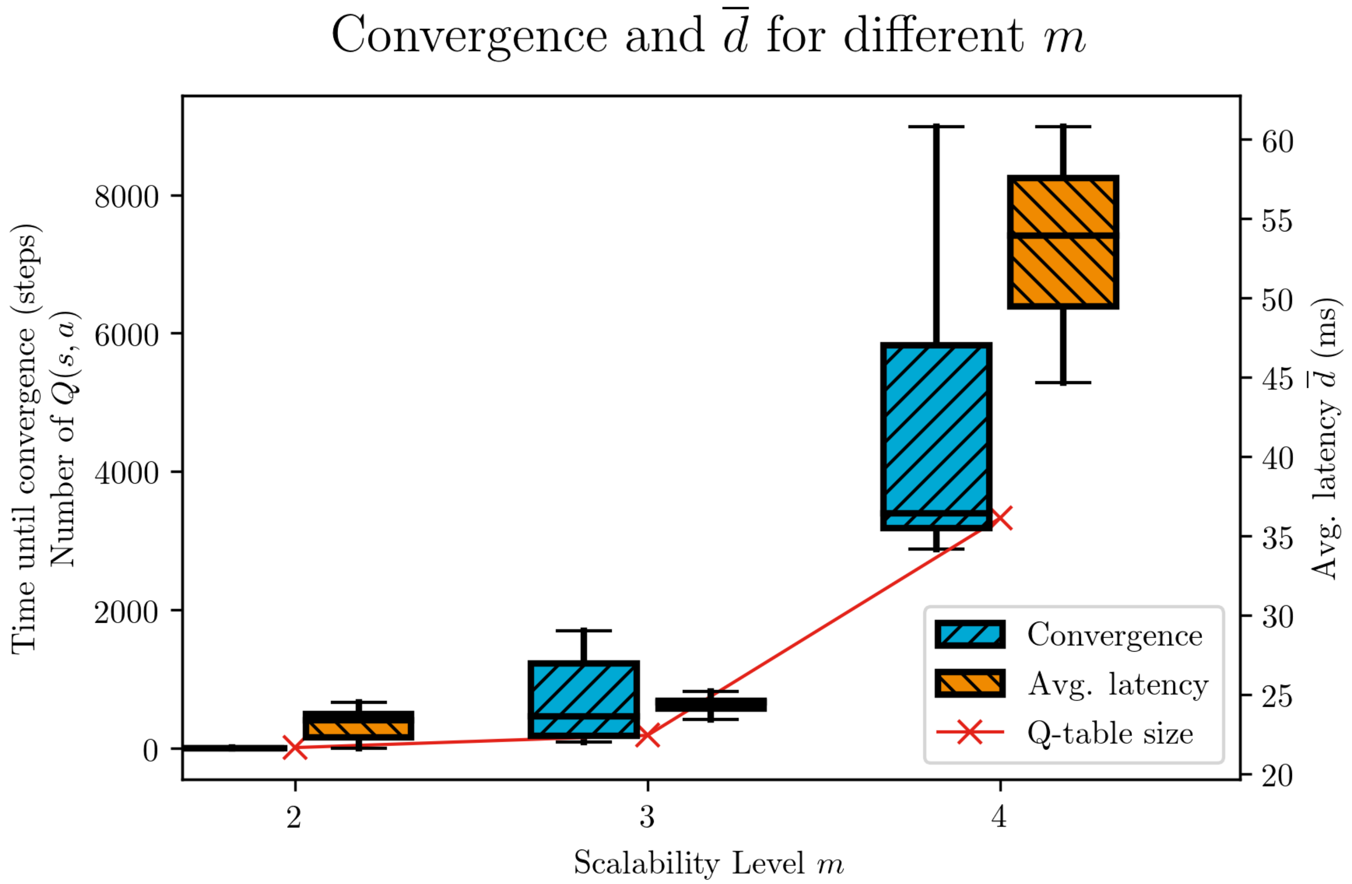
$$|A(s)| = \sum_{f \in F} (|P(f)| - 1) + 1$$

$$|Q| = \prod_{f \in F} |P(f)| \left(\sum_{f \in F} (|P(f)| - 1) + 1 \right)$$

$$= (2 \cdot 2) \cdot (2 \cdot (2 - 1) + 1) = 12$$

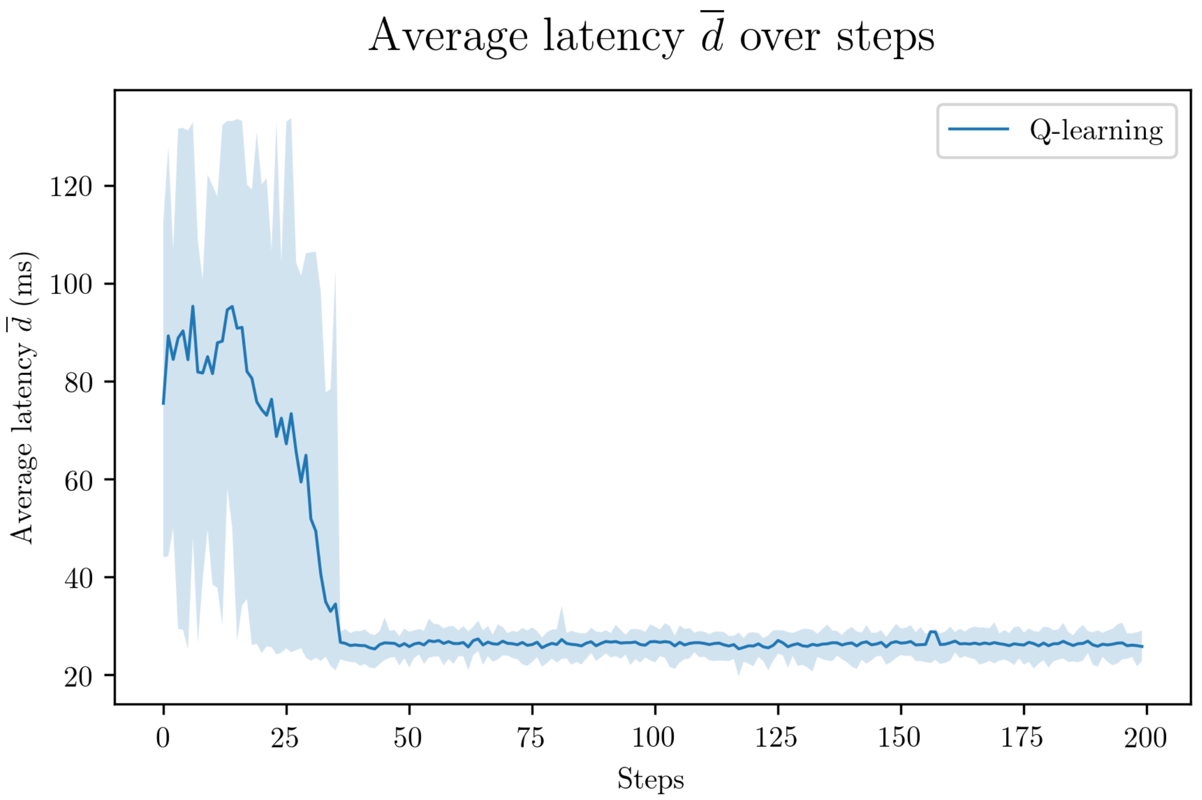


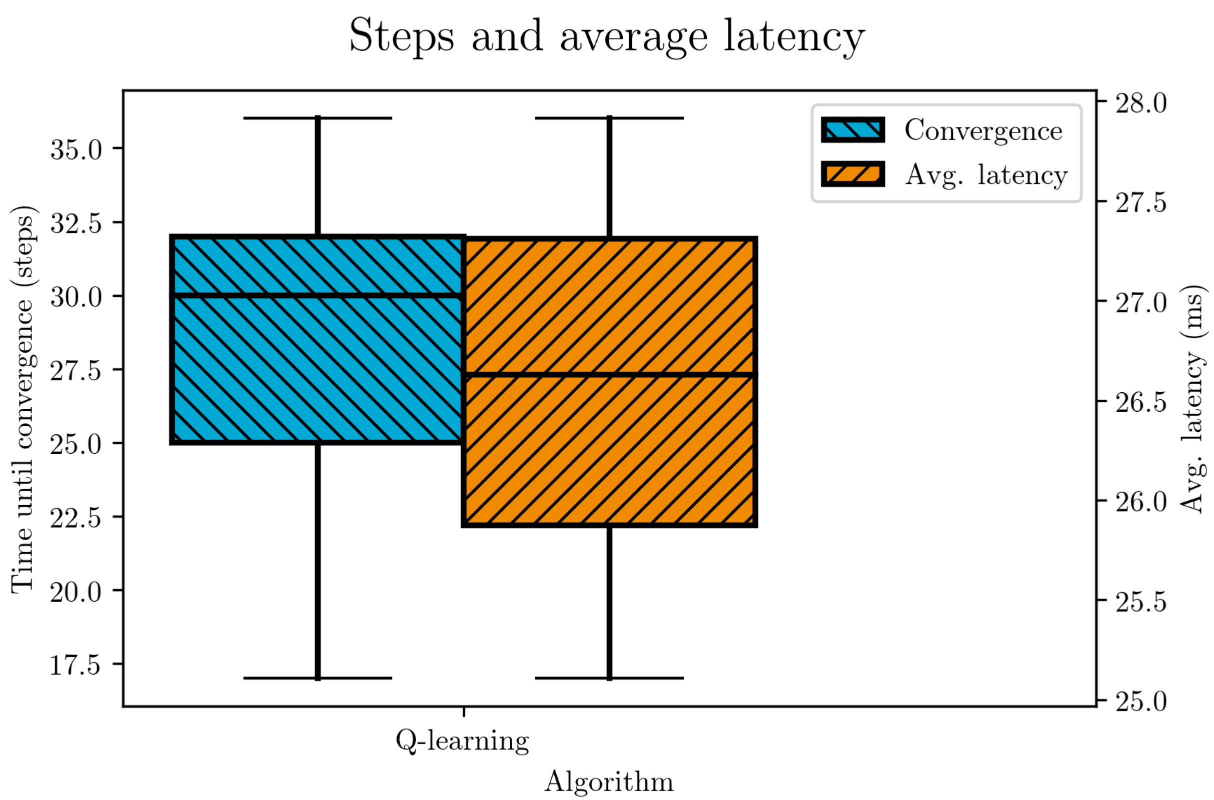
Appendix - Scalability



Level m	Q-table entries	Median Convergence steps
2	12	6.00
3	189	468.50
4	3328	3396.00

Appendix - Learning

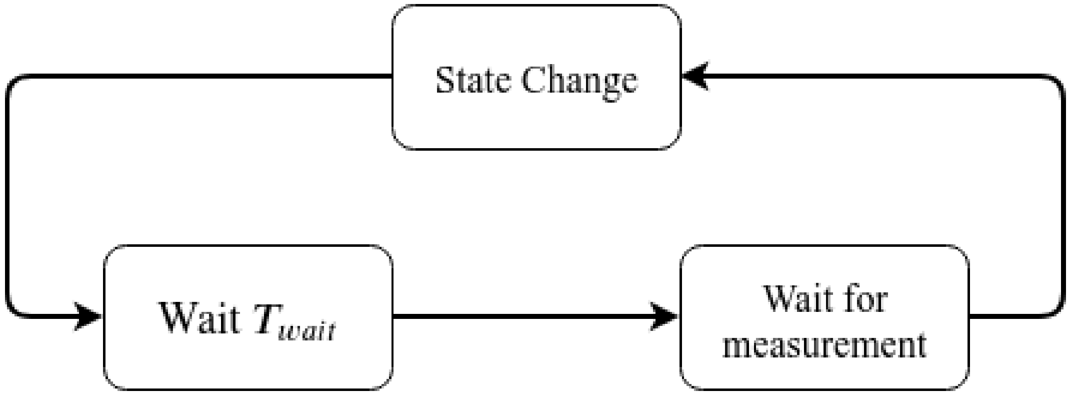




Appendix – Time and Steps

After state change a time is waited to ensure that stationary state reached (queues were emptied or filled)

Then it is waited until all latencies were measured successfully, because the measurement packets could be dropped due to congestion



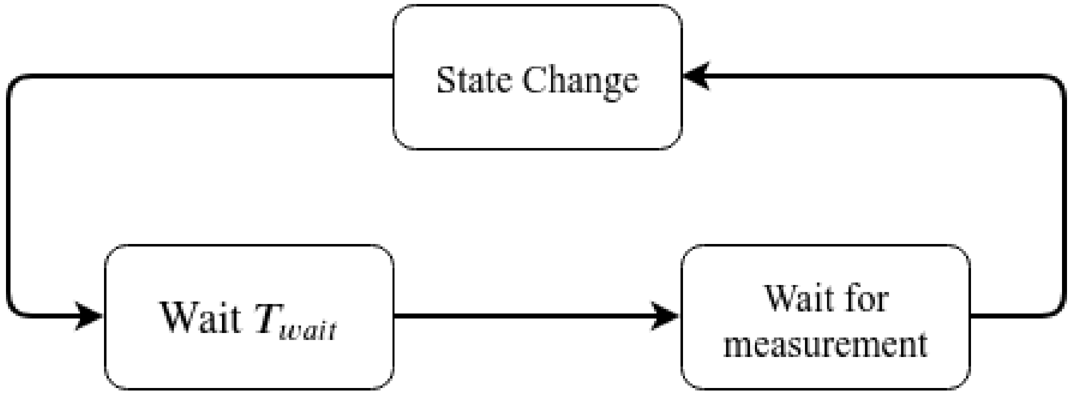
Appendix – Time and Steps

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

$$r_{empty} = \frac{b_{diff}}{k_{UDP}} = \frac{0.25\ Mbit/s}{1512\ byte\ * 8\ bit/byte} = 21.67\ Hz$$

$$T_{empty} = \frac{K}{r_{empty}} = \frac{30}{21.67} s = 1.38\ s$$

$$T_{delay} = \frac{K\ *\ k_{UDP}}{C(l)} = 115.54\ ms$$

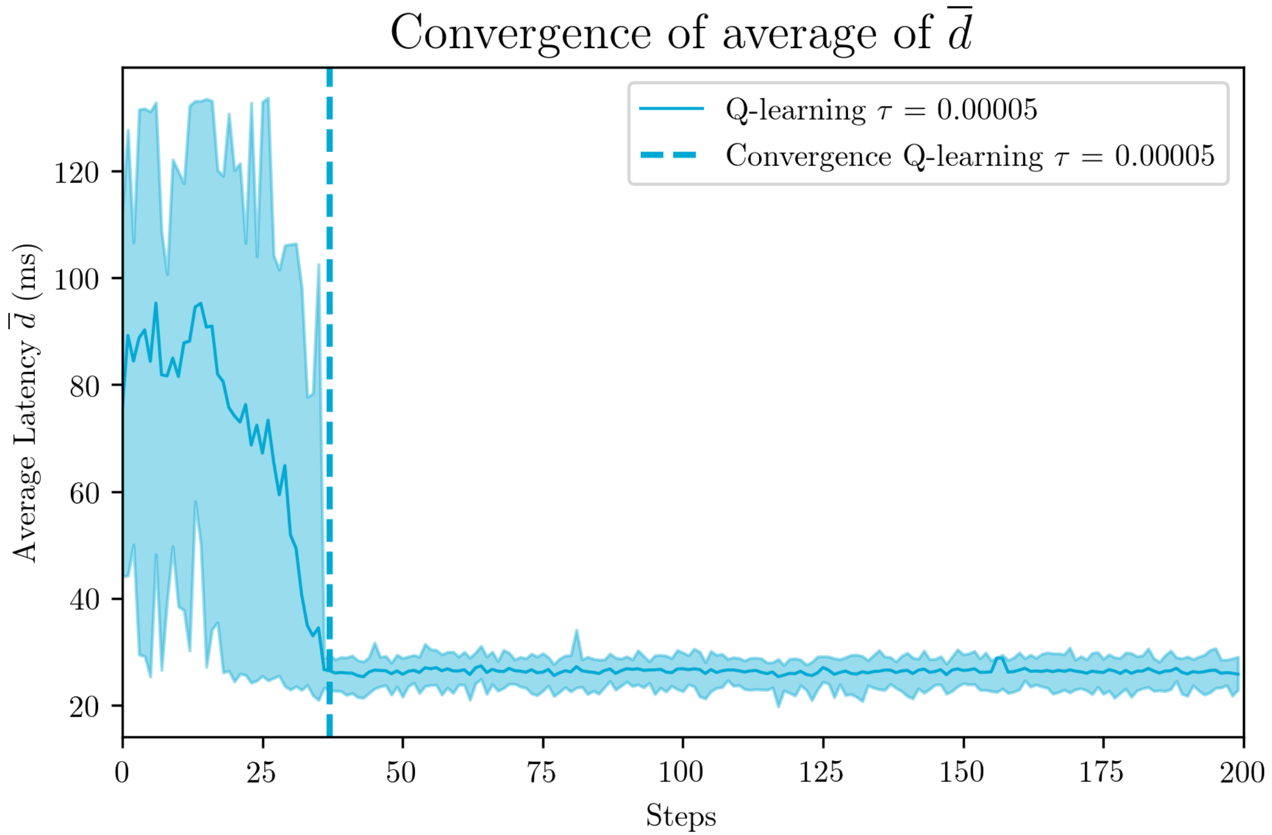


Appendix – Convergence Criterion

Moving average with N=40

$$|\bar{d}(t) - \overline{d_c}| < \epsilon$$

Smallest t_c with a value $\overline{d_c}$ in which all following values $\bar{d}(t)$ are within the range



Appendix – Exploration vs. Exploitation

Exploitation: selecting the most promising action

Exploration: Probing another candidate action

Softmax:

$$a = \max_{a \in A(s)} \frac{\exp(Q(s,a)/\tau)}{\sum_{b \in A(s)} \exp(Q(s,b)/\tau)}$$

Modified Softmax:

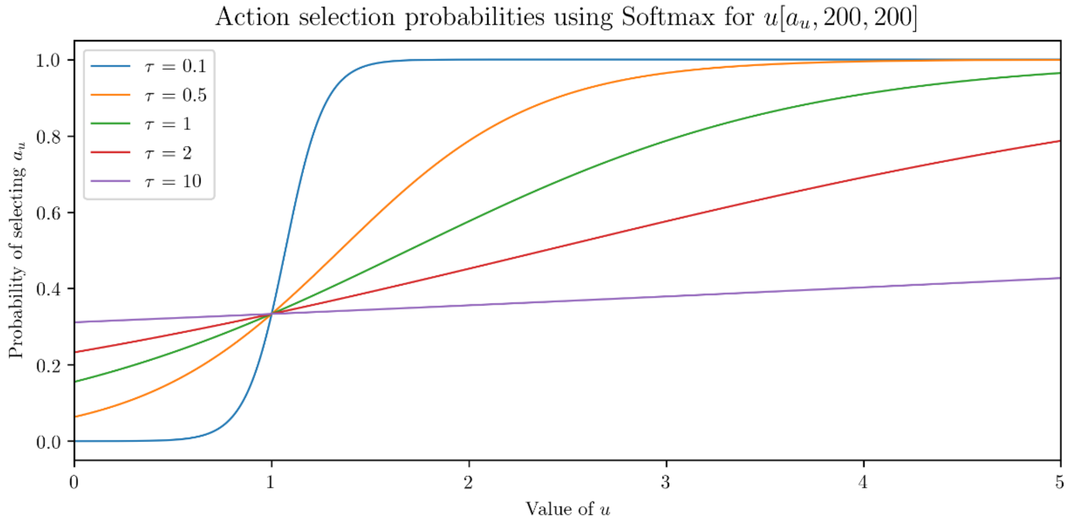
$$a = \max_{a \in A(s)} \frac{\exp(-1/(\tau Q(s,a)))}{\sum_{b \in A(s)} \exp(-1/(\tau Q(s,b)))}$$

Maps Q-values to selection probabilities

Appendix – Exploration vs. Exploitation

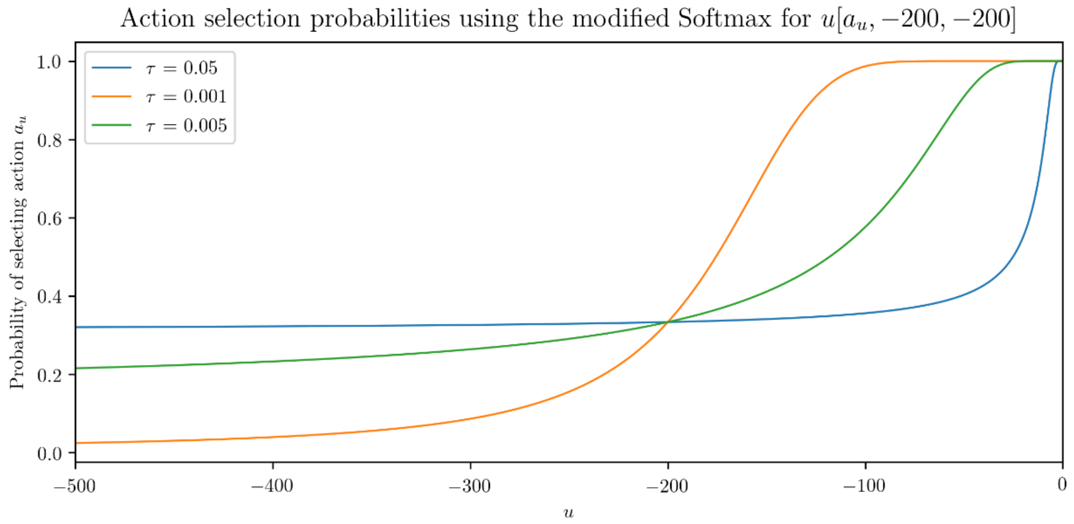
Softmax:

$$a = \max_{a \in A(s)} \frac{\exp(Q(s,a)/\tau)}{\sum_{b \in A(s)} \exp(Q(s,b)/\tau)}$$

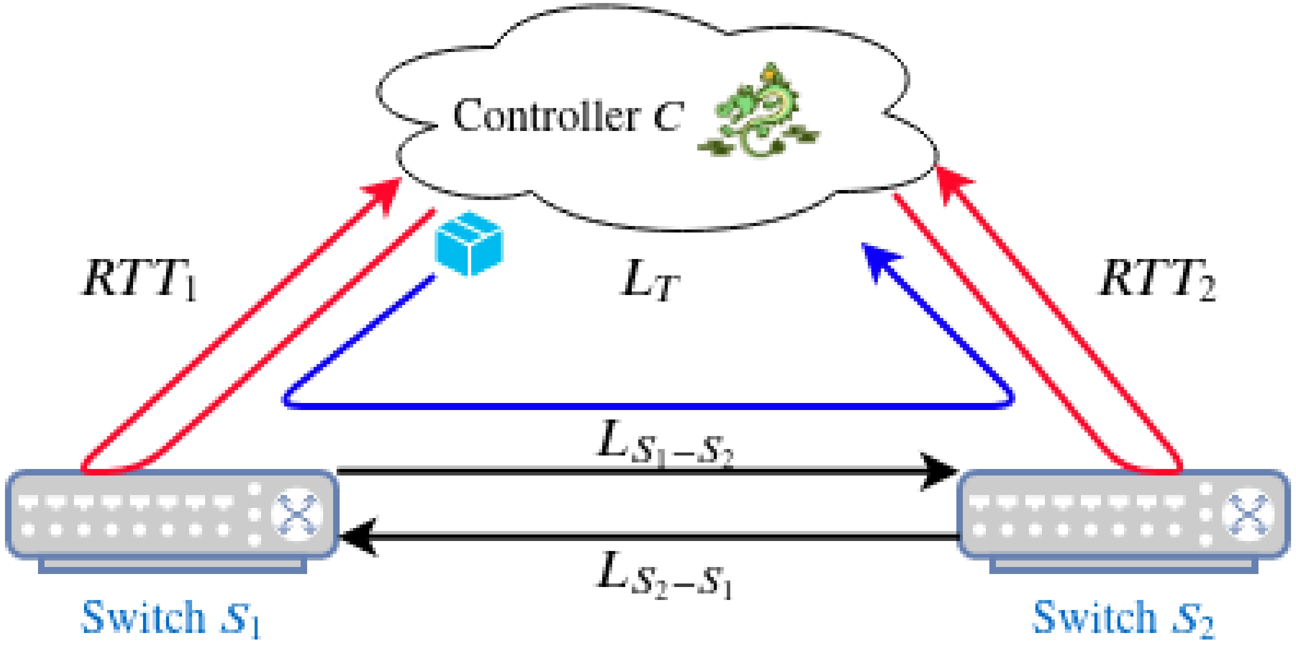


Modified Softmax:

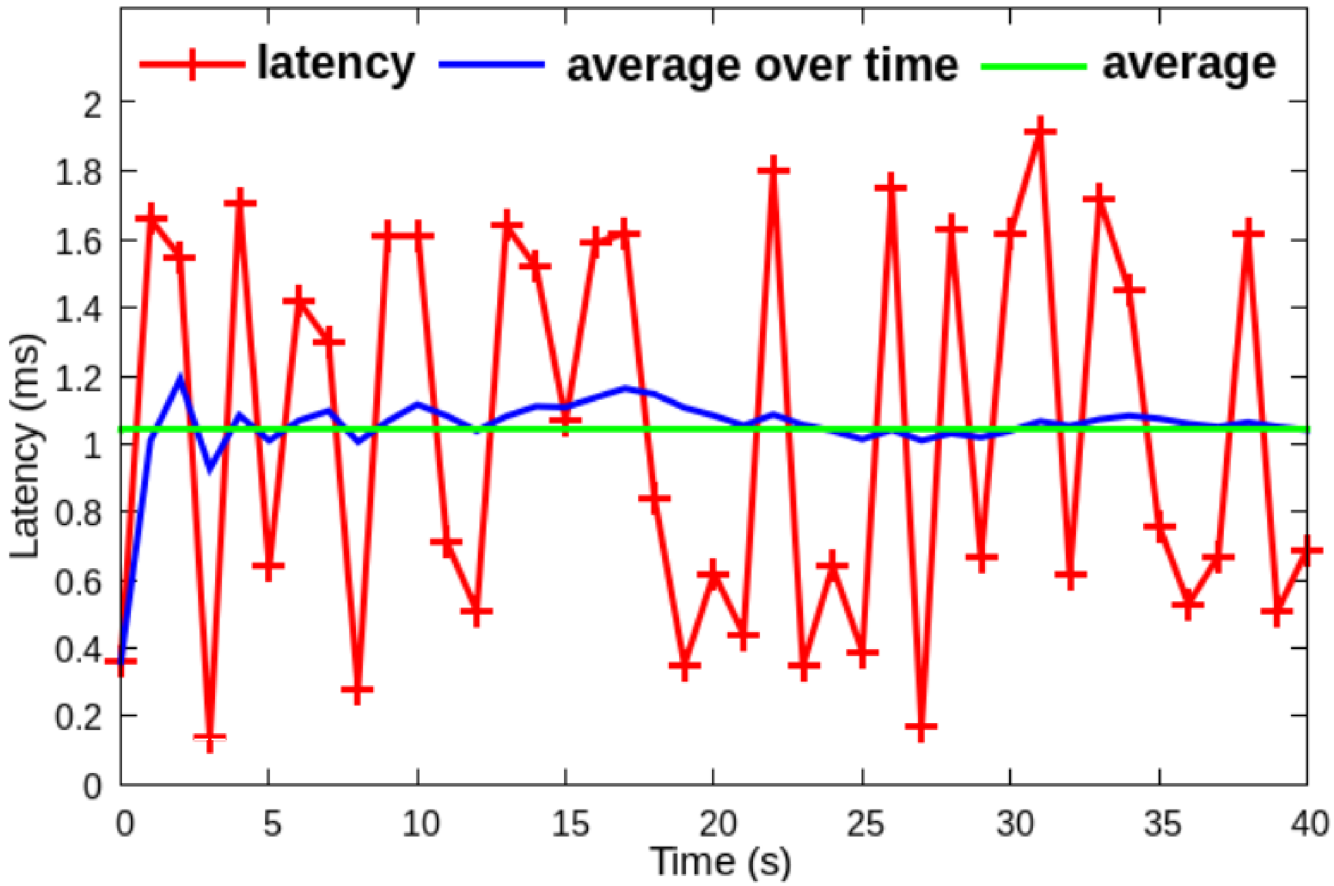
$$a = \max_{a \in A(s)} \frac{\exp(-1/(\tau Q(s,a)))}{\sum_{b \in A(s)} \exp(-1/(\tau Q(s,b)))}$$



Appendix – Latency Measurement



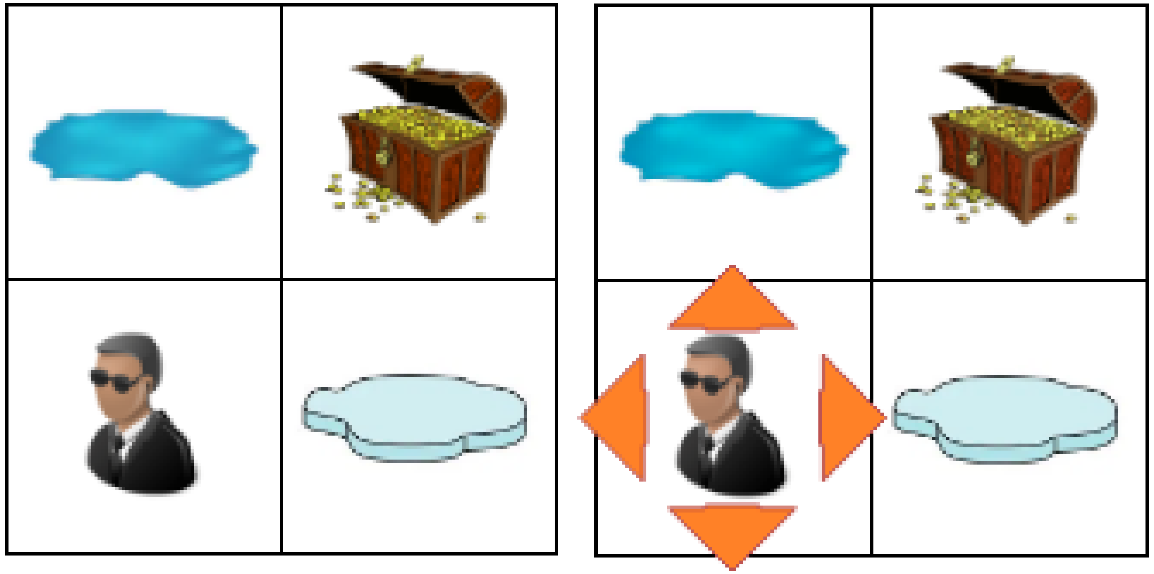
Appendix – Latency Measurement Uncertainty



K. Phemius and M. Bouet, "Monitoring latency with OpenFlow”, 2013

Appendix – Q-learning

$$Q(s,a) = Q(s,a) * \ \alpha\,(r + \gamma \operatorname*{argmax}_a Q(s',a) - Q(s,a))$$



Appendix - Routing

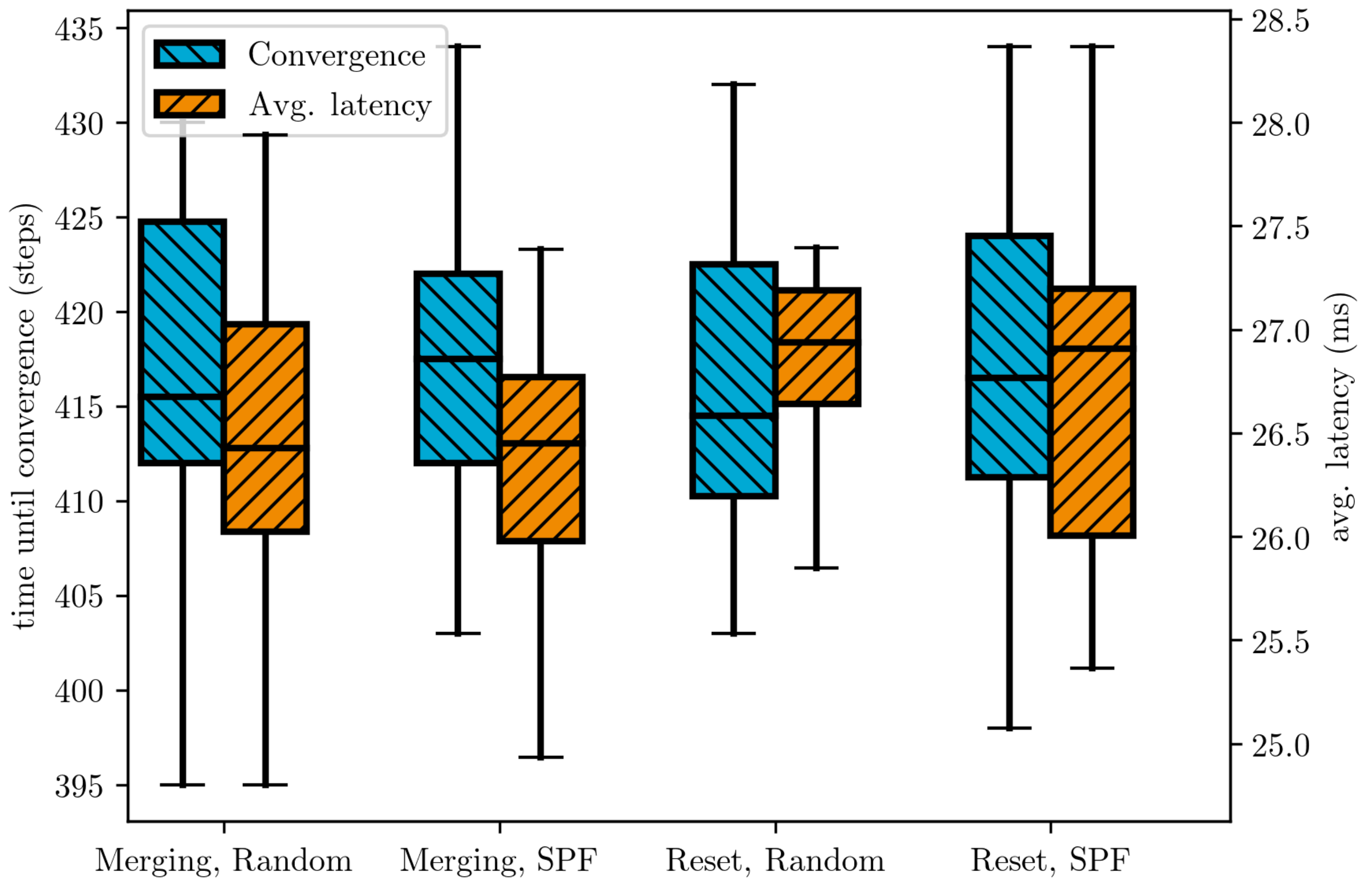
Algorithm 3 Path Search

```
function SEARCHINGPATHS(ADJACENCYMATRIX, SRC, DST)
    if src == dst then return src
        paths = []
        stack = [(src, [src])]
        while stack do
            (node, path) = stack.pop()
            neighbors = adjacencyMatrix[node]            ▷ All neighbors of vertex
            forwardNeighbors = SET(neighbors)-SET(path)  ▷ Neighbors without origin path
            for next in forwardNeighbors do
                if next == dst then
                    paths.append(path + [next])
                else
                    stack.append((next, path + [next]))
                end if
            end for
        end while
    end if
    return paths
end function
```

Algorithm 4 Rerouting

```
procedure ROUTEDEPLOYMENT(OLDPATH, NEWPATH, FLOWID)
    flowAddList ← [ ]        ▷ Switches in which flow table entries are added
    flowModList ← [ ]        ▷ Switches in which flow table entries are modified
    flowDelList ← [ ]        ▷ Switches in which flow tables entrie are deleted
    for index, switch in ENUMERATE(newPath) do
        if switch in oldPath then
            oldIndex ← GETINDEX(oldPath, switch)
            if oldPath[oldIndex-1] == newPath[index-1] then   ▷ If same previous switch
                continue
            else
                if newPath[index-1] not in flowAddList then
                    flowModList ← flowModList + newPath[index − 1]
                end if
            end if
        else
            flowAddList ← flowAddList + switch
            if newPath[index-1] not in flowAddList then
                flowModList ← flowModList + newPath[index − 1]
            end if
        end if
    end for
    for switch in flowAddList do                              ▷ Adding flow table entries
        followingSwitch ← newPath[GETINDEX(newPath, switch) + 1]
        ADDFLOWSWITCH(switch, flowID, followingSwitch)
    end for
    for switch in REVERSED(flowModList) do                    ▷ Modify flow table entries
        followingSwitch ← newPath[GETINDEX(newPath, switch) + 1]
        MODFLOWSWITCH(switch, flowID, followingSwitch)
    end for
    flowDelList ← SETDIFFERENCE(oldPath, newPath)
    for switch in flowDelList do                              ▷ Delete flow table entries
        DELFLOWSWITCH(switch, flowID)
    end for
end procedure
```

Appendix – Joining flows



Different flow initializations and if Q-table is merged