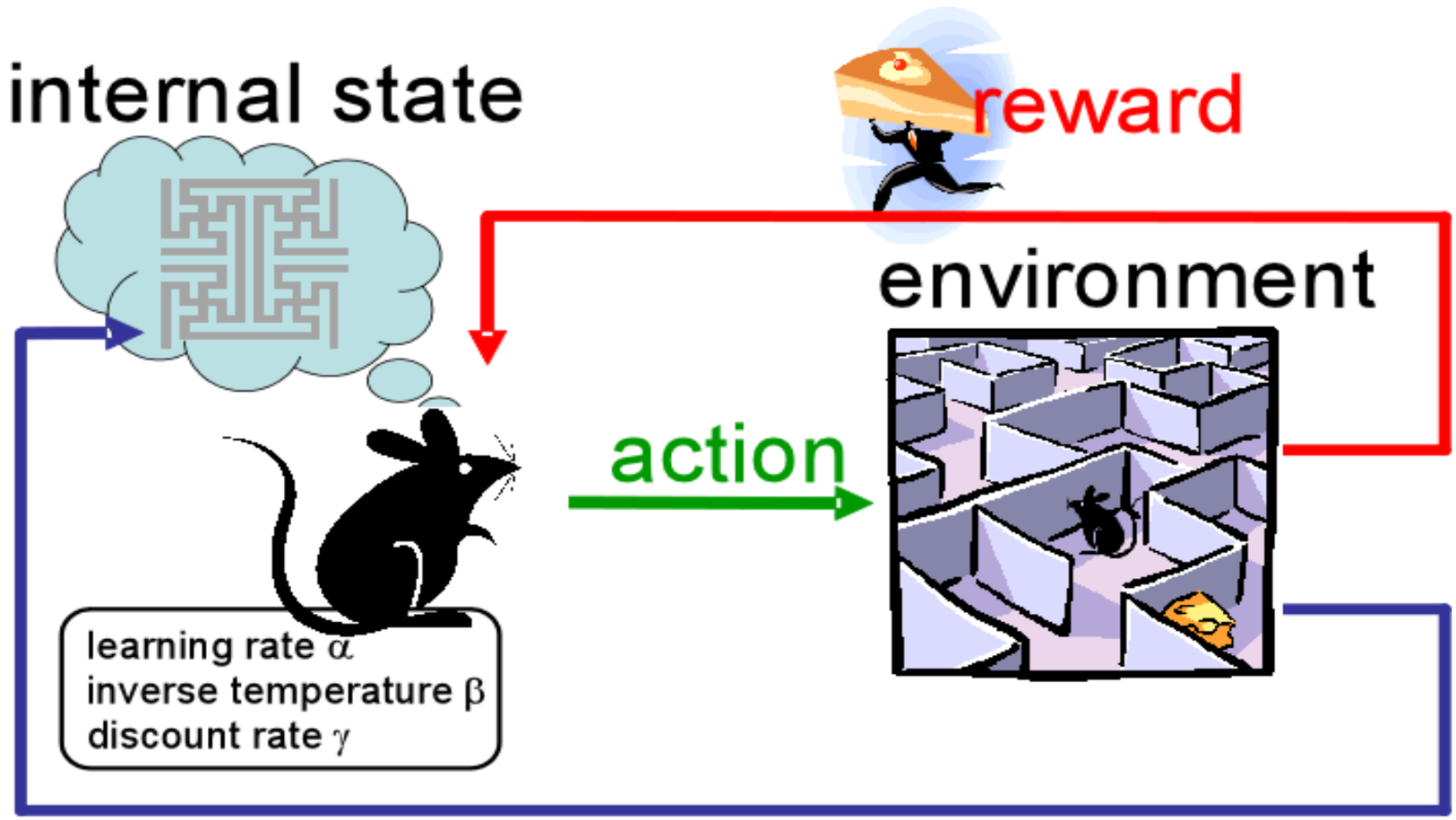
# ns3-gym – The Playground for Reinforcement Learning in Networking Research

Piotr Gawłowicz
gawlowicz@tu-berlin.de


AMLD, 28.01.2019, Lausanne

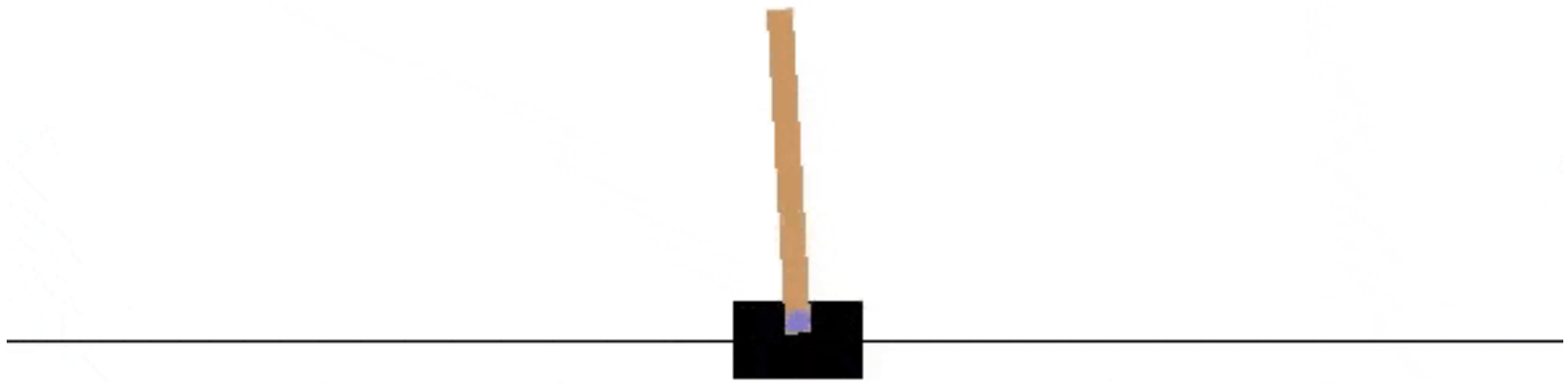
# I. Reinforcement Learning Primer

# Reinforcement Learning



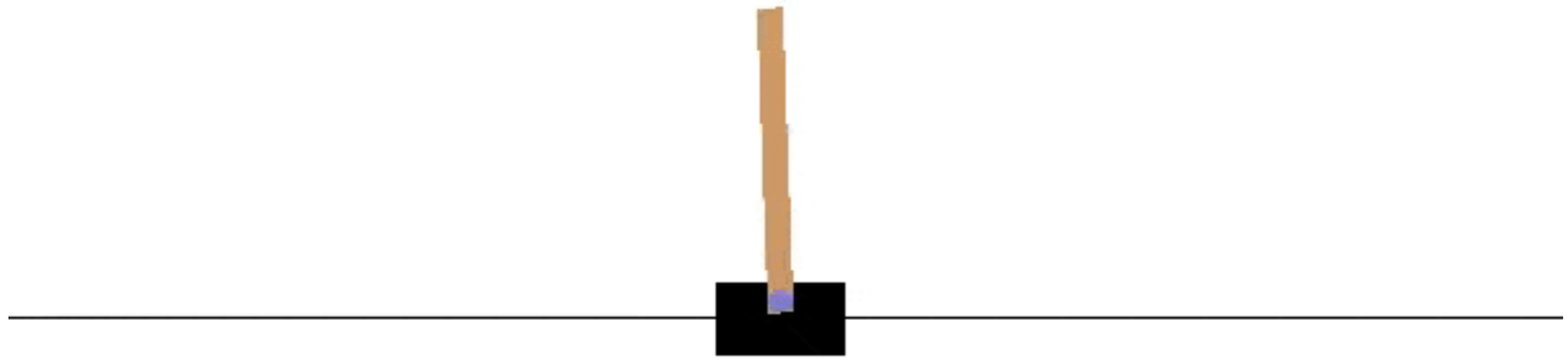

- Reinforcement learning is branch of Machine Learning where an agent learn how to behave in an environment by observing the state of the env, performing actions and seeing the results

# RL-based control



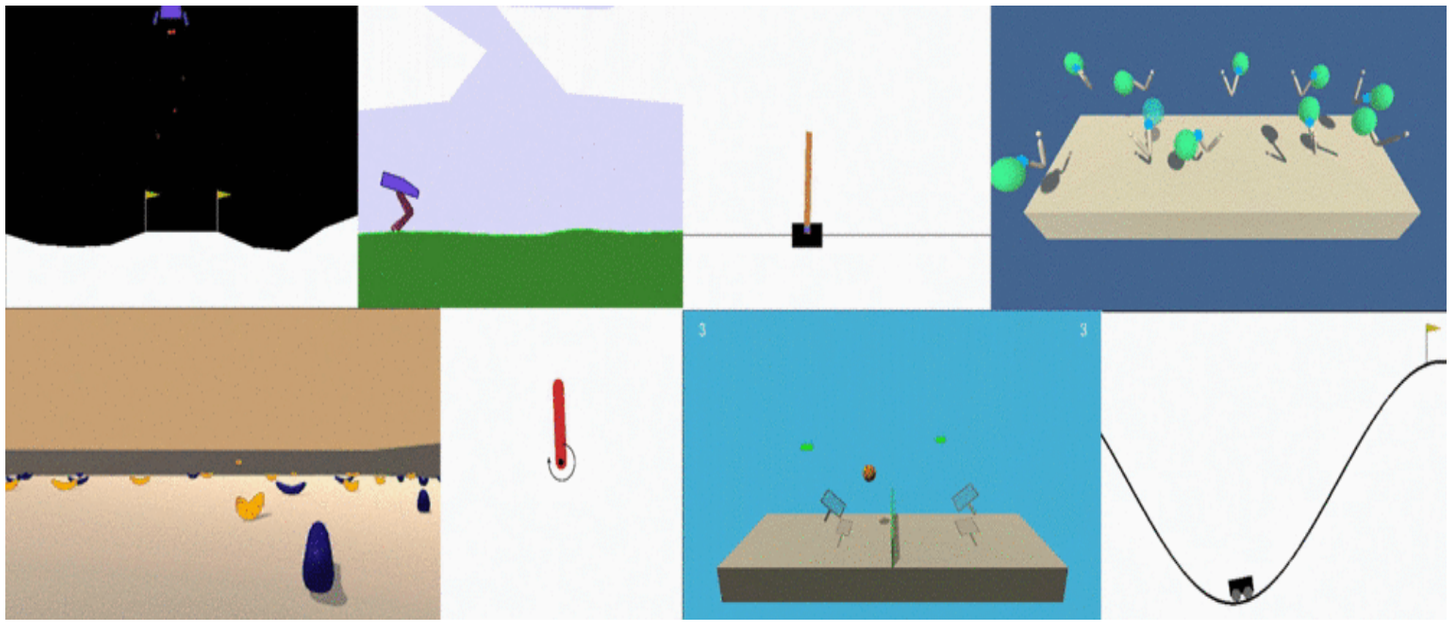
**Environment**: Cart-Pole - a pole attached to a cart
**Goal**: prevent the pole from falling down
**Observation**: current position of the pole
**Actions**: move the cart left or right

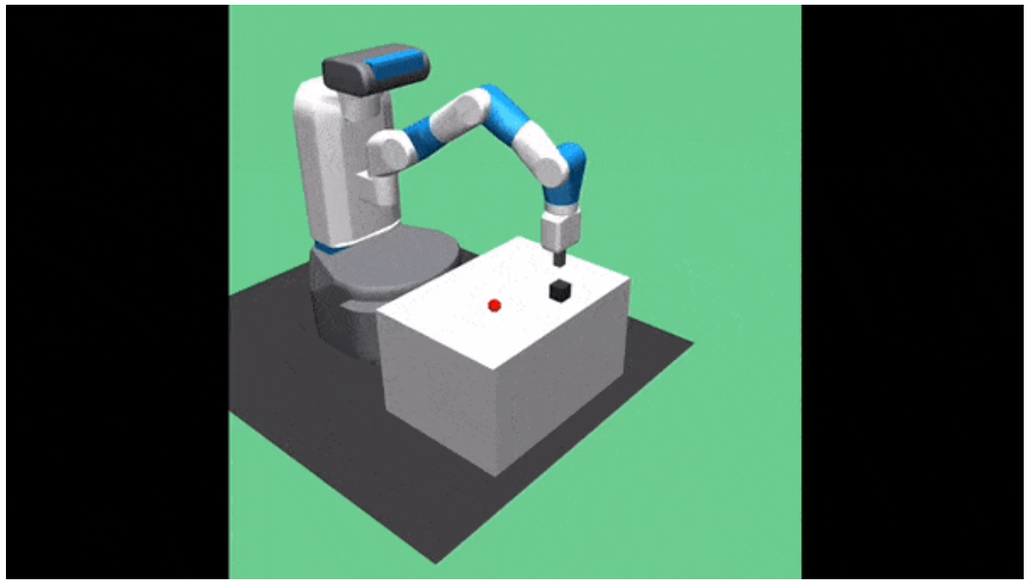
# RL-based control

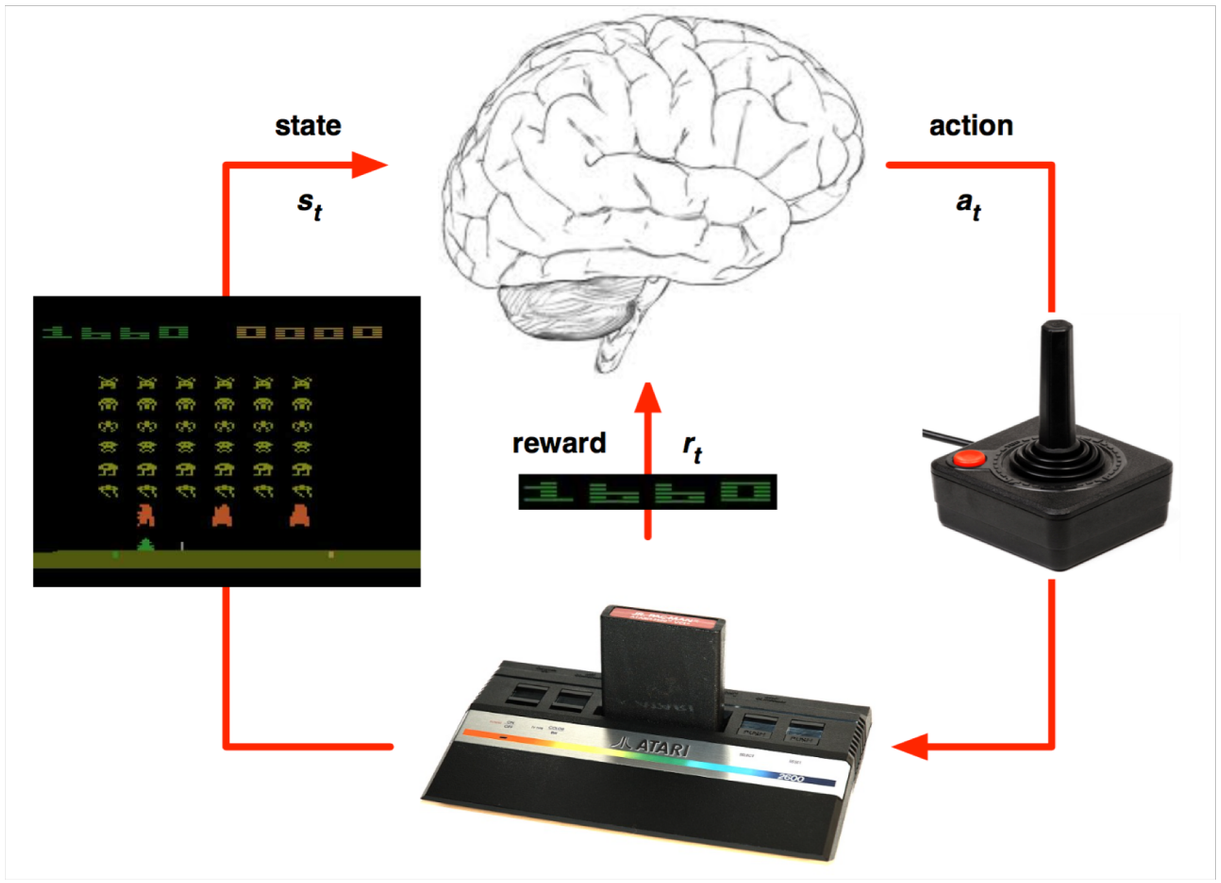


**Environment**: Cart-Pole - a pole attached to a cart
**Goal**: prevent it from falling over
**Observation**: current position of the pole
**Actions**: move the cart left or right

# Environment examples








# OpenAI Gym

- **Gym** is the **open-source** Python library with a vast set of standardized environments including algorithmic examples, Atari games and 3D robots

- **Gym** allows for developing and comparing reinforcement learning algorithms in the same virtual conditions

- Basic concepts:
  - environment (the world)
  - agent (an algorithm that one has to create)

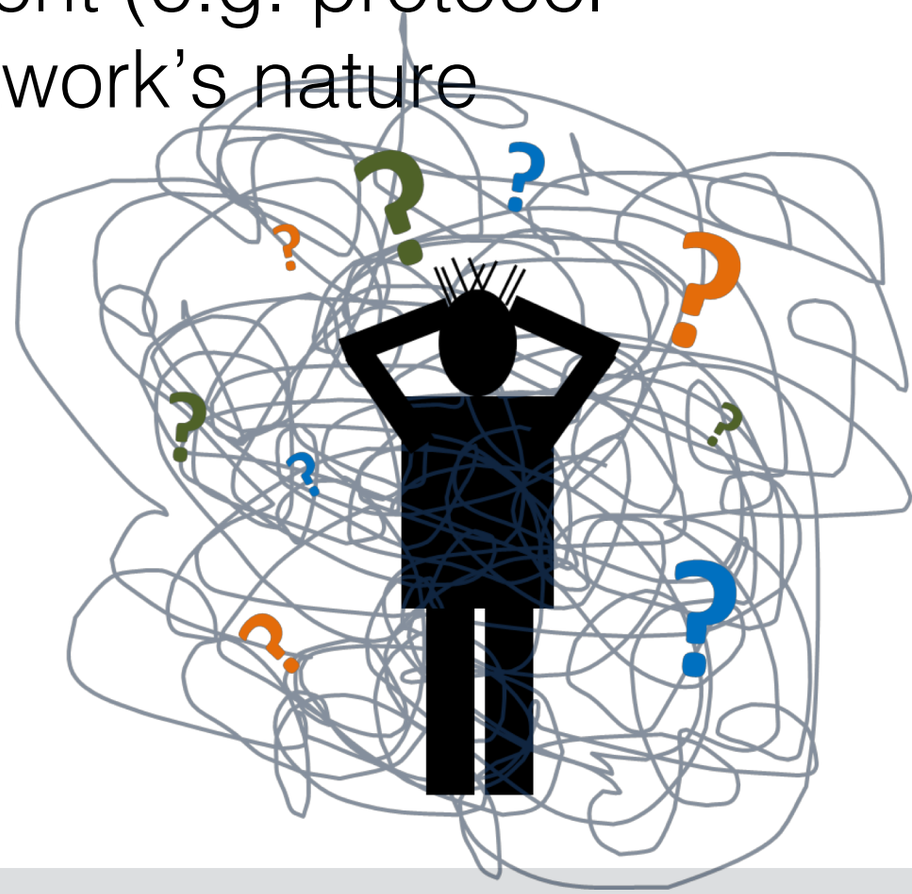
- Unified environment interface:
  - reset()
  - step()
  - render()
  - + data containers

```
import gym

env = gym.make('CartPole-v0')
obs = env.reset()
agent = MyGreatAgent()
done = False

while not done:
    action = agent.get_next_action(obs)
    obs, reward, done, info =
env.step(action)
```

# II. RL in Networking

# Motivation

- Modern communication networks have evolved into extremely complex and dynamic systems
- Traditional (rule-based) design of solutions is based on (over)simplified models, hence they bring only limited performance gains
  - Mostly focused on a single component (e.g. protocol layer) neglecting the end-to-end network's nature
- Networks generate a large amount of monitoring data, that can help to improve them. How to process it?



# ML+Networking= 

Technische Universität Berlin

- From the collected data ML can derive and provide estimated models with tunable accuracy
- Especially, RL fits because of its ability to learn control tasks
  - We can set a policy!
- The proposed RL-based control solutions overtake traditionally designed ones in terms of performance and efficiency (e.g. QTCP, DASH video clients, resource scheduling)
- but…

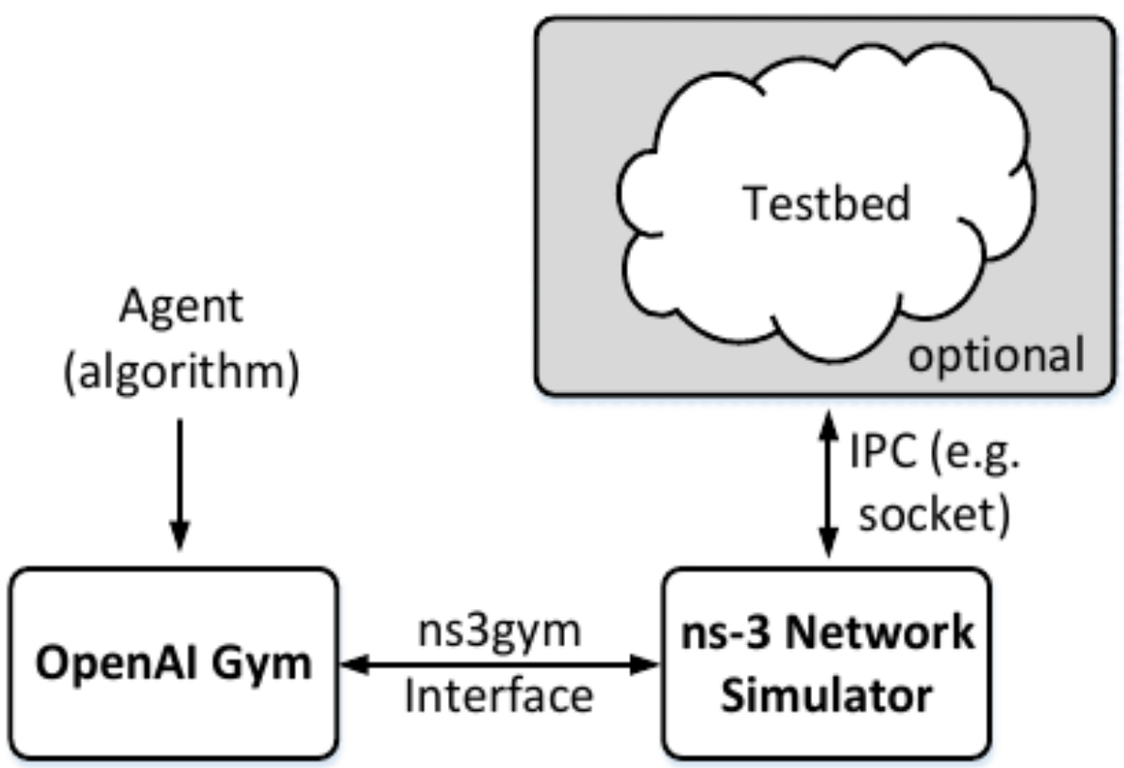
# Unfavorable conditions

Technische Universität Berlin

- RL in networking research is slowed down by:
  - The existence of a knowledge gap between ML and networking communities
  - Lack of training environments
    - Problems of testbeds and real network deployments
  - The need for reliable benchmarking to compare and track progress
    - currently case-by-case solutions

# ns3-gym framework

Technische Universität Berlin




- A Gym agent observes environment created in ns-3 network simulator
- Playground for ML and networking researchers to work together -> faster progress in RLN

# Why simulations?

- Simulations are more practical in comparison to the real world experiments:
  - **easily accessible** – no HW required, just download, develop and test your ideas
  - **fully controllable** - god's view on the network
  - **faster** – run multiple simulations in parallel
  - **safer** – exploration without consequences

- **Curriculum learning** – start with simple model, then add more and more complexity

# Why ns-3?

Technische Universität Berlin

- ns-3 is a discrete-event network simulator for networking systems
- Shipped with a vast set of models for Internet protocol stacks, wireless propagation, wireless technologies (including WiFi, LTE, WiMAX, ZigBee)
- ns-3 community strives to make its models reflect reality as close as possible
- It became de-facto a standard in networking research and is accepted by the community

# Basic example

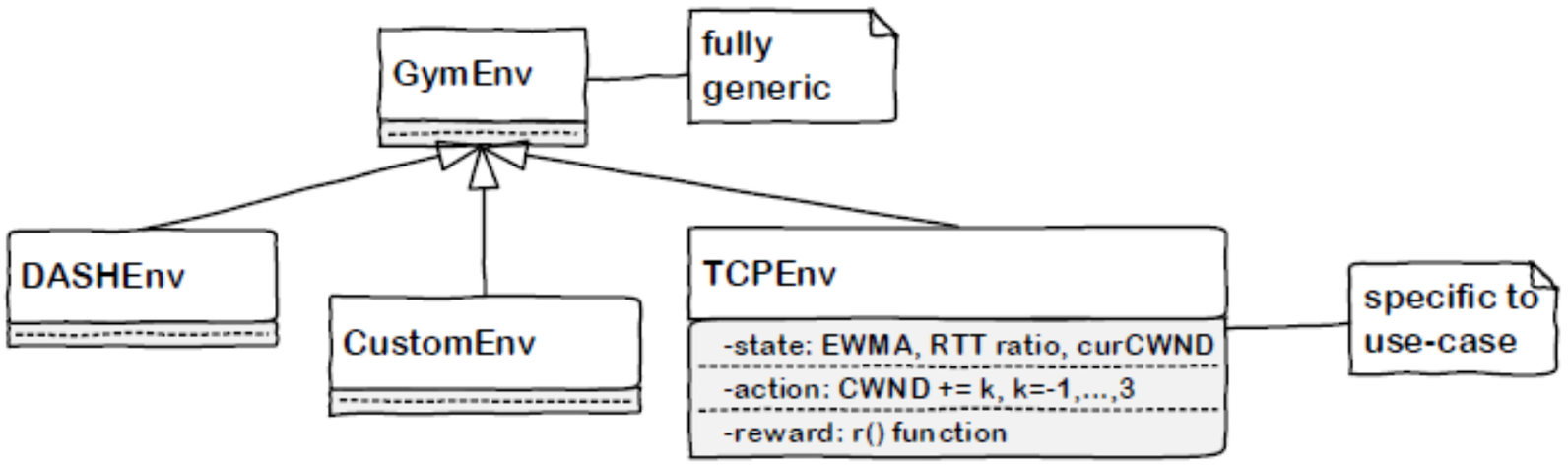
```
import gym
import ns3gym

env = gym.make('ns3-v0')
obs = env.reset()
agent = MyGreatAgent()
done = False

while not done:
    action = agent.get_next_action(obs)
    obs, reward, done, info = env.step(action
```

# Generic Environments

- The environment is defined entirely in ns-3 simulation script
  - any simulation script can be used as a Gym environment
- This requires only to instantiate *OpenGymInterface* and implement the ns3-gym C++ interface with following functions:
    - Ptr<OpenGymSpace> GetObservationSpace();
    - Ptr<OpenGymSpace> GetActionSpace();
    - Ptr<OpenGymDataContainer> GetObservation();
    - float GetReward();
    - bool GetGameOver();
    - std::string GetExtraInfo();
    - bool ExecuteActions(Ptr<OpenGymDataContainer> action);
- ns3-gym – automatically maps corresponding C++ and Python functions and hide the entire complexity behind easy to use API

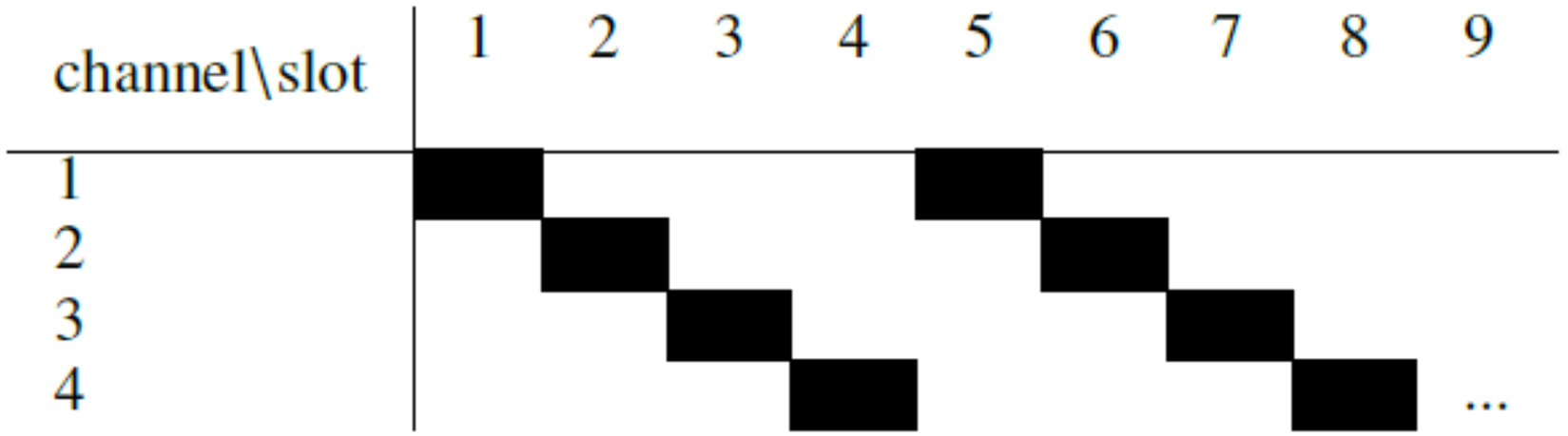
# Custom Environments

- We provide the first set of example problems along with baseline solutions for:
  - Linear 802.11-based mesh topology
  - Interference pattern learning
  - TCP Congestion Control
  - more are coming…



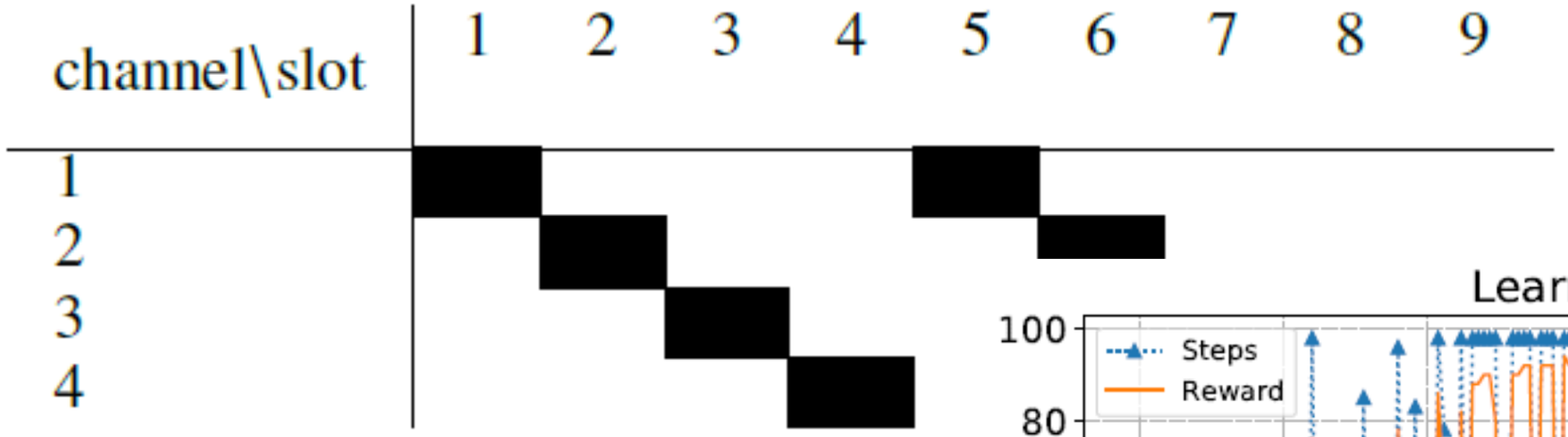
# Example I: Interference pattern

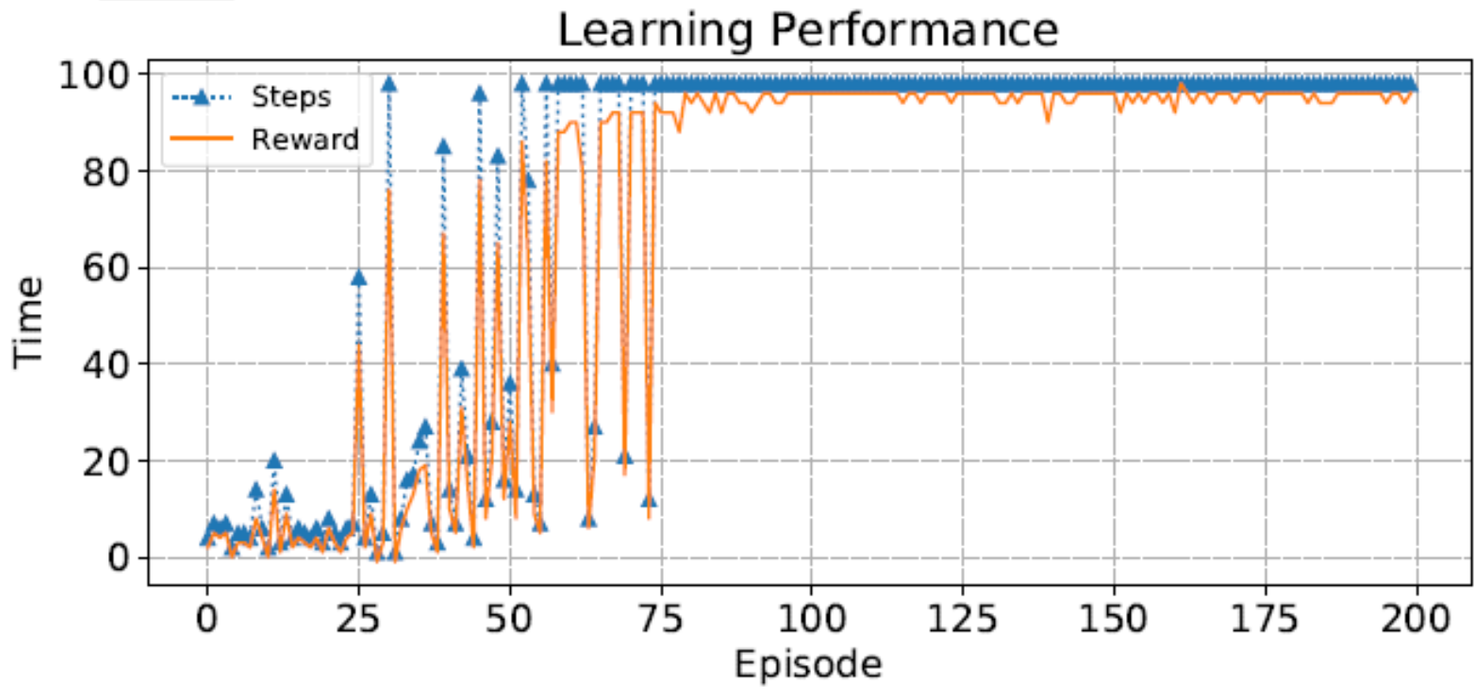
- **Scenario**: 802.11 network collocated with interferer
- **Observation**: occupation on each channel in the current time slot
- **Action**: select the channel to be used for the next time slot
- **Reward**: +1 in case of no collision with interferer; otherwise -1
- **Gameover**: three collisions during the last ten time-slots



# Example I: Interference pattern

- **Scenario**: 802.11 network collocated with interferer
- **Observation**: occupation on each channel in the current time slot
- **Action**: select the channel to be used for the next time slot
- **Reward**: +1 in case of no collision with interferer; otherwise -1
- **Gameover**: three collisions during the last ten time-slots



# Conclusions

- ns3-gym - toolkit that simplifies the usage of RL in networking problems
- It is based on OpenAI Gym and the ns-3 simulator
- We plan to extend the set of available environments
- We hope for research community to grow around it
  - got example agent implementations from external users

# Thank you!

# Q&A

Check ns3-gym

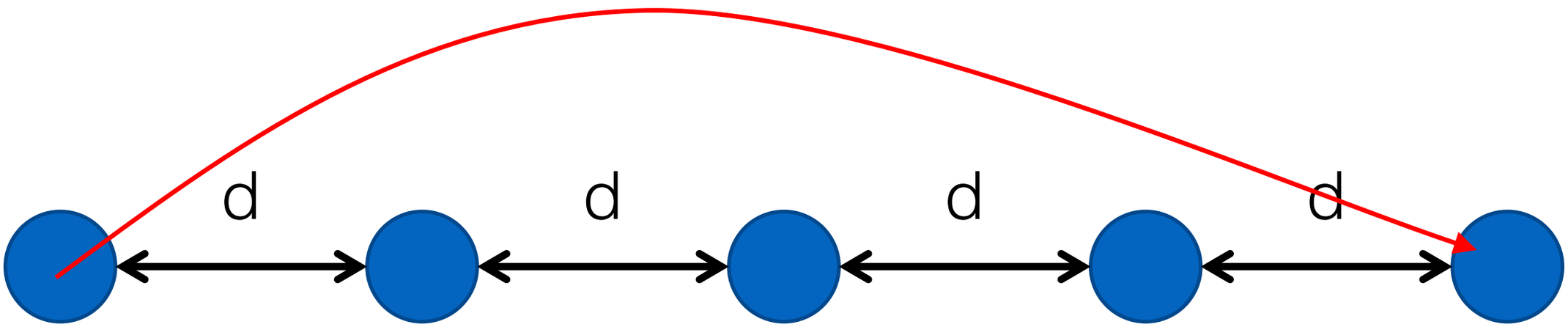
on GitHub

https://github.com/tkn-tub/ns3-gym

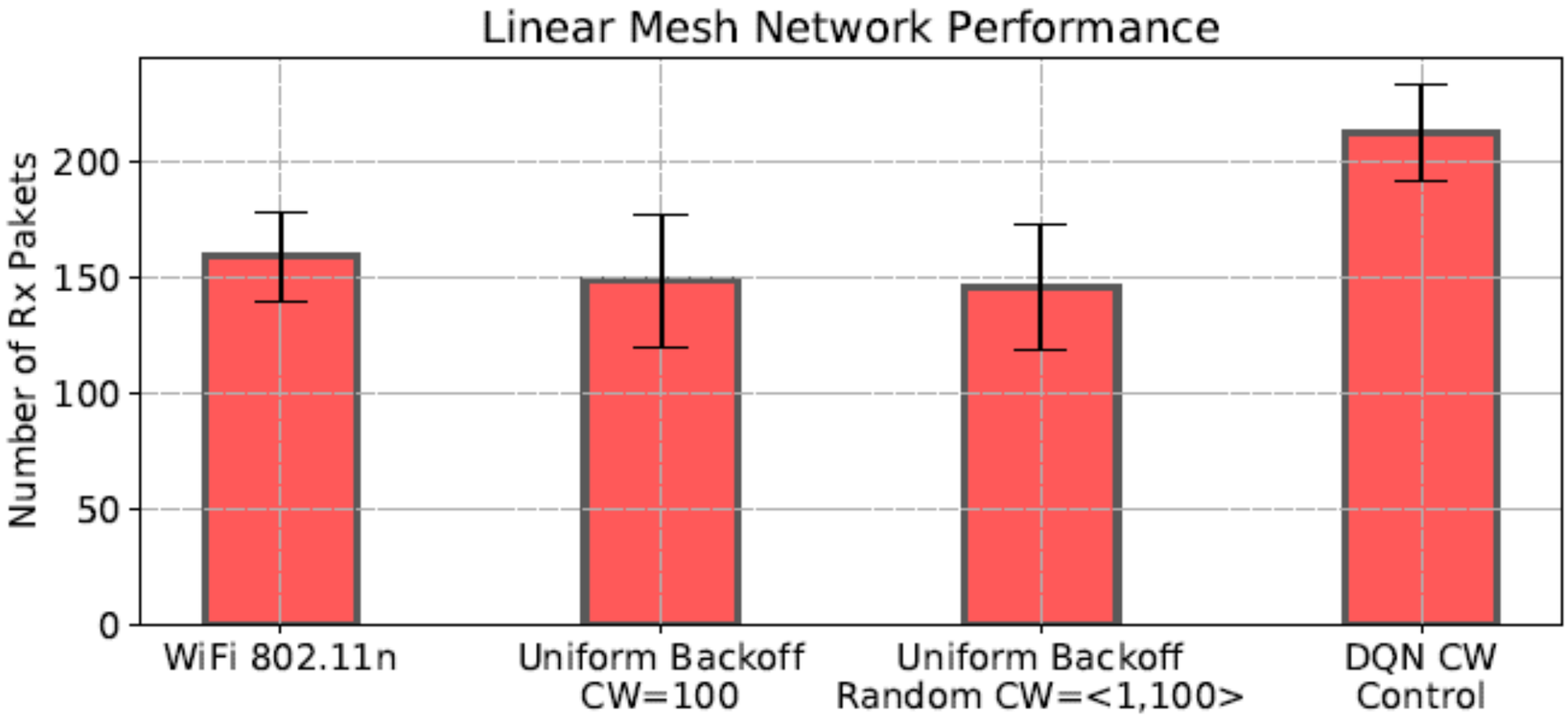
# Example II: Deep Q-learning for CW tuning

Technische Universität Berlin

- **Scenario**: linear wireless topology
- **Observation**: queue length of each node
- **Action**: set CW (channel access probability) for each node
- **Reward**: the number of packets received at the flow's ultimate destination during last step interval
- **Gameover**: end of simulation time



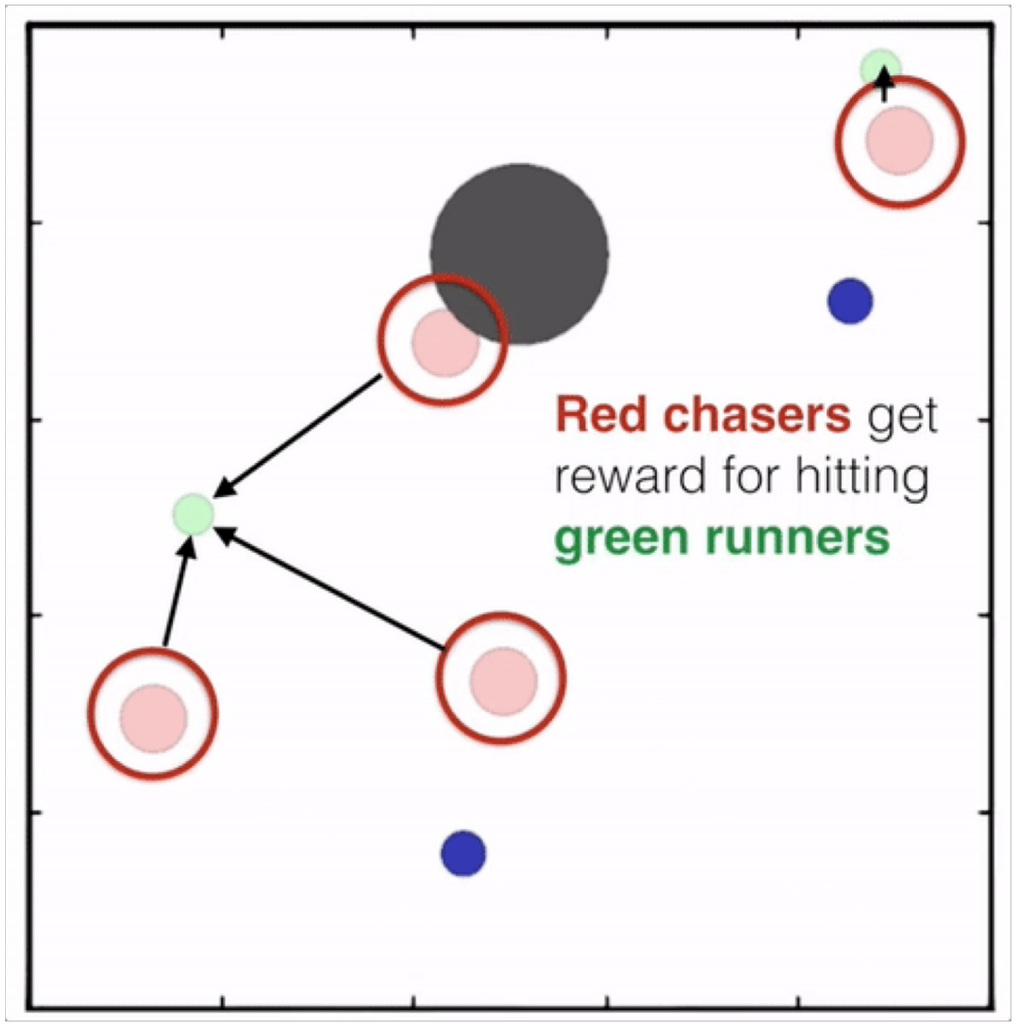
# Example II: Performance Comparison



# Multi-agent environments

- They belong to the most complex branch of RL research
- A number of agents must collaborate or compete for resources with others
- Traditional RL approaches fail to learn
  - each agent tries to predict the actions of other agents
  - randomness
  - selfish behavior
- A communication network by nature is a multi-agent environment.

# Learning to collaborate




- Collaboration with and without exchange of information.
- Spectrum Collaboration Challenge by DARPA in USA

# Reproducibility crisis

- *A Nature survey [1] indicated that more than 70 percent of researchers have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments.*
- This is mainly caused by:
  - source code not being released or only partially released
    - missing documentation – hard to understand and execute properly
    - lack of standardization among the papers
- How to track the progress in the area?

[1] Monya Baker, "Is there a reproducibility crisis?", Nature, 2016, https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970

# Example functions

```
Ptr<OpenGymSpace> GetObservationSpace()
{
      uint32_t nodeNum = NodeList::GetNNodes ();
      float low = 0.0;
      float high = 100.0;
      std::vector<uint32_t> shape = {nodeNum,};
      std::string type = TypeNameGet<uint32_t> ();
      Ptr<OpenGymBoxSpace> space =
                        CreateObject<OpenGymBoxSpace>(low,high,shape,type);
      return space;
}
```

# Example functions

```
Ptr<OpenGymDataContainer> GetObservation()
{
	uint32_t nodeNum = NodeList::GetNNodes ();
	std::vector<uint32_t> shape = {nodeNum,};
	Ptr<OpenGymBoxContainer<uint32_t>> box =
                                    CreateObject<OpenGymBoxContainer<uint32_t>>(shape);

	uint32_t nodeNum = NodeList::GetNNodes ();
	for (uint32_t i=0; i<nodeNum; i++) {
		Ptr<Node> node = NodeList::GetNode(i);
		Ptr<WifiMacQueue> queue = GetQueue (node);
		uint32_t value = queue->GetNPackets();
		box->AddValue(value);
	}
	return box;
}
```