OpenAI Gym meets GNU-Radio

Sascha Rösler
Ali Alouane
Tien Dat Phan
Technical University of Berlin
Department of Electrical Engineering and Computer Science
Germany, 13437 Berlin
Email: s.roesler@campus.tu-berlin.de
Email: ali.alouane@campus.tu-berlin.de
Email:t.phan@campus.tu-berlin.de

***Abstract*—Nowadays Reinforcement Learning (RL) shows promising advancement in fields like image recognition tasks by not relying on closed form solutions and finding solutions by applying e.g. inference techniques. Therefore, an agent tries out different actions on a given environment to find a nearly optimal solution. These approaches are used to solve more and more control problems. They are already used in the communication and networking domain. To evaluate transmission schemes/ network stacks an environment to train an agent is necessary.**
Additionally, researcher are using GNU-Radio as Software Defined Radio (SDR) tool to do rapid and low cost prototyping for new and existing wireless technologies. In this paper we introduce *gnugym*, a framework to use GNU-Radio as OpenAI Gym environment. Thereby, we enable RL for GNU-Radio. gnugym includes a general part, caring about communication and management of GNU-Radio. Via two interfaces a specific environment is loaded describing the specific Machine Learning (ML) model. This split allows rapid integration of new ML models within the framework.
As proof of concept we use an implementation of the IEEE 802.11p stack and run it in a loopback example. The Modulation and Coding Scheme (MCS) is set for given Received Signal Strength Indication (RSSI) values. During training the distance between sender and receiver varies. With some Supervised Learning (SL) algorithms there is successful training.

I. Introduction

The research in new wireless network technologies is still ongoing. As hardware implementations of new technologies are expensive, Software Defined Radio (SDR) becomes an important tool for prototyping and research topics. It allows low cost testing and evaluation of new technologies and improvements on existing technologies. There are software implementations for IEEE 802.11, Bluetooth, Zigbe, GSM, etc. available for GNU-Radio [7], a popular SDR implementation. Additionally, we see an increase of usage of Machine Learning (ML) techniques for control problems. With Reinforcement Learning (RL), deep learning becomes more and more popular and in the interest of researchers. Several problems in the communication domain are considered from a ML or especially from a RL point of view [8]. RL tools like OpenAI Gym provide interfaces for rapid prototyping and comparison of RL algorithms. Additionally, OpenAI Gym allows an exchange of agents and by this comparison of agents [9]. In this paper we introduce our framework *gnugym*. The framework combines both, GNU-Radio as SDR candidate and the rapid prototyping approach of OpenAI Gym. It manages the OpenAI Gym interface and the communication and execution of GNU-Radio. The framework allows the integration of new SDR scenarios as OpenAI Gym environment. It provides an interface to the agent and an interface for communication with the GNU-Radio program. gnugym acts as man in the middle between a userdefined GNU-Radio program, the agents and a userdefined specific scenario, describing the RL model.
Allowing an event based approach we evaluate the jitter of the reception of GNU-Radio data from GNU-Radio. We measure the delay of the different communication methods, too.
As proof of concept, we use the IEEE 802.11p stack implementation in GNU-Radio [1]. We add measurement tools to the loopback example and define a ML model. The ML model controls the Modulation and Coding Scheme (MCS) based on the measured Received Signal Strength Indication (RSSI) value. For simulation the distance between sender and receiver variates. The scenario uses a Gaussian channel.

II. Background

A. Reinforcement Learning

Nowadays ML sees an increasing interest in science and has shown promising performance especially in fields like image recognition. One subfield of ML is RL.
Figure 1 shows how such an agent interacts with its environment (here seen as Network Stack). By performing an action the RL-agent receives feedback in form of a reward. By doing this over and over the agent learns how to behave in certain environments.
All the advancements in RL motivated the integration of simulation software into RL-frameworks to enable further advancements in communication technologies.
In another work [3] RL has been used to "build detectors for systems where the underlying physical models of the channel are unknown or inaccurate", which fits into our project to try out which possibilities are possible with RL.

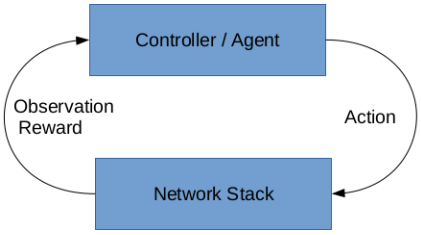

Fig. 1. Reinforcement Learning Agent

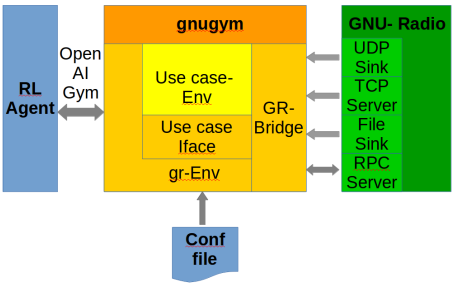

Fig. 2. System Architecture

B. OpenAI Gym

OpenAI Gym [4] is a toolkit that allows users to test and train RL algorithms with predefined environments. Users can create environments on their own, too.

C. GNU-Radio

GNU-Radio [5] is a toolkit that allows us to implement software radios with signal processing blocks. For example, there are blocks for an IEEE 802.11p network stack [6]. To generalize even further, GNU-Radio allows the definition of a loopback channel in software, too. Using real hardware is still allowed. This offers the possibility of creating data for ML algorithms even more versatile.

D. IEEE 802.11p

IEEE 802.11p is an amendment to the IEEE 802.11 standard to define the communication between vehicular communication systems. Furthermore, it defines enhancements to support Intelligence Transportation Systems applications. In this project we use an implementation [6] of an IEEE 802.11 a/g/p transceiver to test the implementation of the communication between GNU-Radio and OpenAI Gym. To train our RL agent we put our attention on the modulation scheme which is used to transmit data. Therefore BPSK, QPSK, 16QAM and 64 QAM in each case with the possibility of two error correcting codes making a total of eight possible actions are available [2].

III. Design Principles

The main goal of our work is the simplification of RL prototyping for GNU-Radio applications. Therefore, the framework fulfils the following design principles:

- Separation: Separation of a general part and an adaptive part (*specific environment*). The specific environment should implement the ML model. For new scenarios within GNU-Radio just a new specific environment is necessary.
- Rapid Prototyping: The framework should allow rapid prototyping of ML models using GNU-Radio programs.
- Abstract: There is no specialization to one specific GNU-Radio example. The provided framework should work for different GNU-Radio scenarios.
- Low Implementation Overhead: Caring about communication to GNU-Radio is hidden within the general part of framework. The generic part cares about compilation and execution of GNU-Radio, too. The integration of a new model should not care about these general parts.
- Interchangeable: Model and agents are separated by the generic part and OpenAI Gym. This approach allows interchanging of agents and model. Thereby, different agents can be compared on different models.
- Simple: The framework provides a very detailed interface from the specific environment to the agent. The detailed interface allows separate implementation of each part of the model. This approach keeps the writing of the specific environment simple.

IV. System Design

A. Overview

The system design consists on three main parts. There are the RL-agents, our gnugym environment and the GNU-Radio scenarios (Figure 2). The GNU-Radio part and the RL-agents depend on the specific use case and are no part of our general work. The gnugym part consists of different modules. First there is the generic part `GR-Env`. It implements the OpenAI Gym interface. The generic part initiates the communication to GNU-Radio via `GR-Bridge`. The generic part does the lifecycle management of a `step` (Figure 3) and of a `reset`, too. Whenever model specific information and definitions are required it does a call to specific environment. This specific part is interchangeable.

B. Features

The core feature of gnugym is the connection between the OpenAI Gym interface and the GNU-Radio scenario. gnugym takes care about the communication and provides interfaces for specific environments and implements the OpenAI Gym interface to the agent. It allows

- writing own specific environments. These environments implement the specific ML model describing action, observation and reward.
- the control of parameters in the GNU-Radio program via XMLRPC

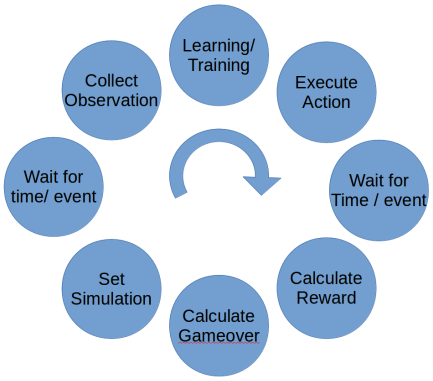

Fig. 3. Livecycle of a step

- listening on streams of variables via named pipes, Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) connections.
- automatic compilation and execution of the GNU-Radio flowchart and automatic start of the execution.
- dynamic includes of the specific environments.
- usage of a selectable simulation mode. A userdefined simulation can be enabled if needed.
- usage of a timed based approach and an event based approach for the control. Via a configuration file the mode is selectable. In an event based approach the specific scenario can wait for the next arriving data. In the timed approach gnugym waits for a configurable step time and after changing the simulation for a configurable simulation time.

C. Generic Environment

The generic environment cares about the communication and the handling of requests. It connects the RL-agent with the specific environment and GNU-Radio. The generic environment creates the agent side of the communications channels with GNU-Radio using the GR-Bridge. It compiles and starts the GNU-Radio flowchart and starts the protocol stack at a `reset` of the environment. It handles the correct order of the execution of subroutines when there are `step` and `reset` calls. The generic environment imports the configuration file and provides the data to the specific environment.

D. Specific Environments

As each scenario of control application uses another model, the specific environment has to hide the complexity of the scenario from the agent. Thus, the agent just sees the simple OpenAI Gym interface. The specific environment implements the ML model. To not get lost in too complex coding, gnugym provides an interface of short steps. For example, the `step` method is split into six smaller methods. Executing an action, calculating the reward, calculating a game over, getting the observation and some information, doing a reset or changing the simulation parameters are tasks for several methods. Calculation of observation space and action space are done by extra methods, too. Additionally, the interface provides a GR-Bridge object and a parameter object. In most cases it is not necessary to use additional interfaces. To go with this interface each specific environment should inherit from `grgym.envs.gnu_case`.

Listing 1. interface for the specific environment

```
def __init__(self, gnuradio, args):
def get_observation_space(self):
def get_action_space(self):
def get_obs(self):
def get_reward(self):
def get_done(self):
def render(self):
def reset(self):
def get_info(self):
def simulate(self):
```

In timed mode none of these methods should act blockingly. The timing is to the general environment. In event based mode `get_reward` and `get_observation` should do some blocking to wait for the specific events.

E. Communication with GR-Bridge

The communication between the environments and GNU-Radio is managed by the GR-bridge. GR-bridge acts as XML-RPC client and can set and read variables in the GNU-Radio flowchart. Additionally, it can run a listener for continuing data streams and can buffer the last value. The listener can listen on named pipes, UDP or TCP sockets. For UDP, the listener acts as UDP server. Listening on a TCP stream, the listener is a TCP client. To listen on data, there has to be a subscription request to GR-bridge before. GR-Bridge runs separate threads for each subscription, listening on the given source. The received data is interpreted as numpy array. The data representation is announced at subscription.
For reading and writing variables, waiting for new data and for subscription of variables, GR-Bridge provides a simple interface to the environment using four methods (see Listing 2). Each method takes `gr_var` as parameter. This is the name of the GNU-Radio variable or an internal name of the data streams.
The implementation of GR-Bridge has an adaptive structure. Additional connection types can be added easily. For each new connection type a definition of how to start, how to do a blocking read and how to close is required.

Listing 2. GR-Bridge interface

```
gnuradio.subscribe_parameter
    ('gr_var', 'path/to/pipe;server:port',
     numpy dtype of var, length of array,
     BridgeConnectionType)
(value, timestamp) =
     gnuradio.get_parameter('gr_var')
gnuradio.set_parameter('gr_var', value)
gnuradio.wait_for_value('gr_var')
```

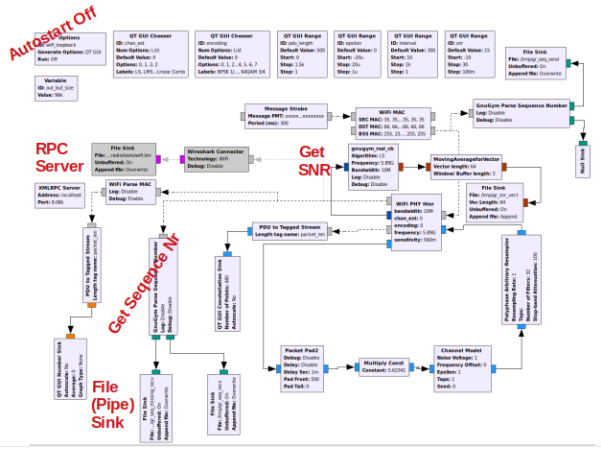

Fig. 4. GNU-Radio flowchart

1) Communication in GNU-Radio flowchart: For a successful communication some blocks are required in the GNU-Radio flowchart. For changing variables via XMLRPC a XMLRPC Server block has to be placed in the flowchart. For each variable a variable or a GUI block is required. To bring data streams out of GNU-Radio to the environment a File-Sink block can be used to put data into a named pipe. A TCP-Server sink can send the data via a TCP stream. For UDP connection, there is the UDP sink block.

V. Implementation

The complete environment is written in Python. The file structure follows the structure of an OpenAI Gym environment.

VI. Use case: Control Modulation and Coding of IEEE802.11p Stack

The environment is evaluated on an IEEE 802.11p stack scenario [1]. Sender and receiver are implemented in the same GNU-Radio flowchart and connected by a simulated channel. There is no test on a real channel. In this scenario the MCS is controlled. In the simulation the distance between sender and receiver variates. The GNU-Radio flowchart is given in Figure 4. It is the WiFi-Loopback example from [1] extended by blocks to get all needed data and the transmission to the environment. The changes are marked in red.

1) Identify Model: To do the ML, at first a model is required. For the IEEE 802.11p scenario reward, action and observation are defined as follows:

- Action: The action is the MCS of the packets. IEEE 802.11p supports 8 different possibilities (see Table I). In the implementation of the IEEE 802.11p stack the possibilities are represented by an integer value. `Discrete(8)` is the corresponding action space.
- Reward: The reward is set as effective data rate. It is the number of received packets during the step multiplied by the data rate of the current MCS. In the simulation, all packets have same length.

TABLE I
Data rates based on MCS [2]

Modulation	Coding Rate	Data Rate
BPSK	1/2	3 Mbps
BPSK	3/4	4.5 Mbps
QPSK	1/2	6 Mbps
QPSK	3/4	9 Mbps
16-QAM	1/2	12 Mbps
16-QAM	3/4	18 Mbps
64-QAM	1/2	24 Mbps
64-QAM	3/4	27 Mbps

- Observation: The observation is a vector of either the RSSI per subcarrier or the Signal to Noise Ratio (SNR) per subcarrier of the last received packet. As IEEE 802.11p uses 64 subcarriers the observation space is `Box(shape=(64, 1), dtype=np.float32)`. Four subcarriers are pilot tones. At the border of the spectrum there are null tones [2]. The observation is measured during the synchronisation phase of packet reception. As each frame starts with the same MCS the observation does not depend on the last action.

2) Design new GNU-Radio Blocks: To calculate reward and observation GNU-Radio has to send some variables. First, knowledge of the last send and received sequence number and the number of missing packets at the receiver is required. Second, the measurement of the RSSI or the SNR values is required, too. Therefore, we introduce new GNU-Radio blocks. To get the RSSI and the SNR values a measurement at a clearly defined point of time is useful for comparison. Therefore, the new blocks extract the data at the first or the first and the second sample of the synchronization word of the packet. The RSSI block returns the absolute value of the Fast Fourier Transformation (FFT) output. The SNR block calculates the difference between the FFT value of the first two samples as noise and the sum of the first to values as signal. The resulting SNR is the division of signal and noise, then. For the sequence numbers a block has to parse the passing packets and has to extract the numbers from the header.

The described tasks are already part of the block in the IEEE 802.11p stack. Thereby, extracting SNR and RSSI is a functionality out of `WiFi Frame Equalizer`, parsing MAC headers is already implemented in `WiFi Parse MAC`.

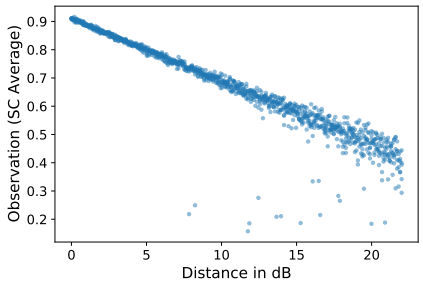

Fig. 5. Normalized Observation in Relation to the Distance

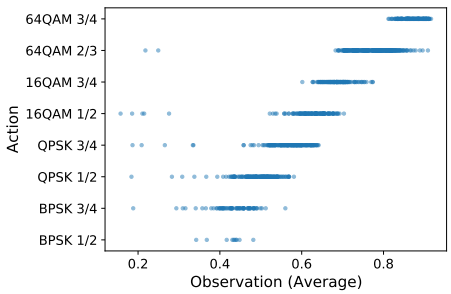

Fig. 6. Best Action in Dependency of the Distance

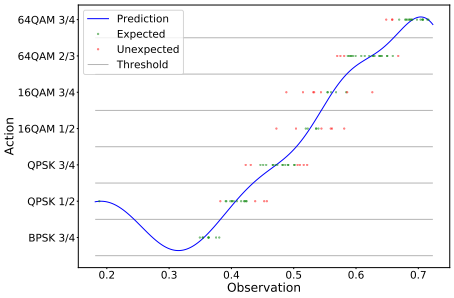

Fig. 8. Result of Gaussian Process Ression (GPR)

However, the new blocks are required, as the values are not accessible from outside the IEEE 802.11p blocks.

3) Model evaluation: Before evaluating the agent a view on the model is useful. Figure 5 shows the dependency of the average over the RSSI values of the pilot tones on the variation of the distance. In the measurement there is some noise. Nevertheless, there is a clear linear dependency. This makes the RSSI values to a good candidate for the observation. The SNR shows the same linear curve. Indeed, the value increases with the distance.

Next there was a measurement of the best action (action with highest reward) for given distances. In Figure 6 the data is shown in relation to the average RSSI value. There is no clear correlation of the data, as there are observations where different actions fit best. It looks like the observation do not show the complete channel characteristic. Resulting from this measurement training will be complicated and the accuracy of the agent can be around 70% maximum. This is a bound, we get with Supervised Learning (SL) algorithms.

4) Train the Agent: To evaluate the model we take three different RL algorithms and try to train it on the IEEE 802.11p scenario. There are these three algorithms:

- DQN: DQN trains one neuronal network based on the given reward. It includes the neuronal network of Figure 7. Regarding to a random variable and a threshold (exploration rate) ε, it takes a random action or the best fitting action based on the neuronal network [10].
- Proximal Policy Optimization (PPO): PPO method takes advantage of the data by sampling actions according to the latest version of its stochastic policy [11].
- Soft Actor Critic (SAC): SAC is an off-policy method. The important feature of SAC is entropy regularization. The policy is trained to maximize a trade-off between expected return and entropy. The term of Entropy is a measure of randomness in the policy function. Thus, a high entropy in the policy encourages exploration [12].

With none of the three algorithms a successful training was possible. The number of training steps was increased to 600000. One Problem can be the strongly divergent reward, depending on the channel conditions. To get hands on the problem we perform a measurement on the scenario. Therefore, variating the distance between 0 and $10m$ we collect reward and observation for all possible MCS. Using this data the best action is calculated. On this result two SL algorithms are trained. We try a GPR and a SL-NN algorithm (see Figure 7). In both cases the result is a stair function (see Figure 8 and Figure 9). As the best actions overlap there is no error free result possible. An accuracy around 70% in both cases looks like an upper limit.

VII. Timing Evaluation

In an event based approach it is possible to trigger the agent on the reception of packets. A comparison of the delay of XMLRPC, named pipes and ZeroMQ ports in Figure 10 shows, that the limiting factor is XMLRPC (around $1ms$ delay). Indeed, XMLRPC is required to set the variables. The

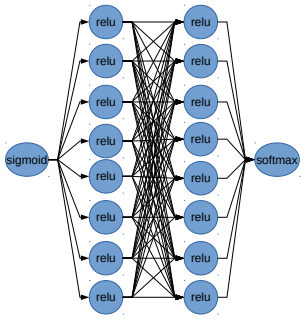

Fig. 7. Neuronal Network for Supervised Learning (DQN)

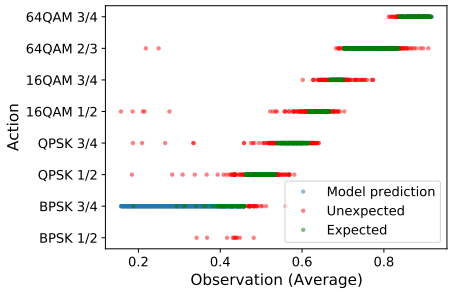

Fig. 9. Result of SL-Neuronal Network (NN)

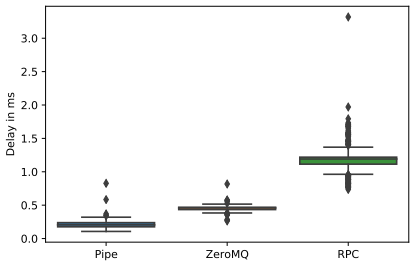

Fig. 10. One Way Delay

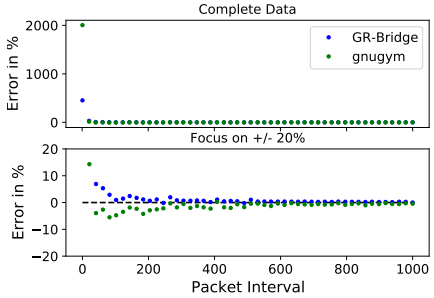

Fig. 12. Error in % of packet interval reception

delay for ZeroMQ and named pipes is in the same range ($0.4ms$ delay). The delay for UDP and TCP will be in the range, too.

In a second test, the jitter of the packet transmission is evaluated. Therefore, the interval between two consecutive packets varies in GNU-Radio. In the `GR-Bridge` the interval between the received data of the packets is measured. The testing machine is a five years old Acer Laptop with an i3 processor (Figure 11). The interval is stable until a given interval of around $50ms$, as shown in Figure 12.

The timing duration of a complete step cycle is presented with green dots. Based on a different point of starting time, there is a shorter interval than expected. Again, for packet intervals above $50ms$ there is just small jitter.

VIII. Conclusion

This paper is the report of a student project. We introduce a framework combining GNU-Radio with OpenAI Gym. The separation of the framework into a specific environment and a generic part allows the usage of the framework for many scenarios in GNU-Radio. Both, a timed approach and an event based approach are possible and have only low constraints because of delay and jitter. gnugym cares about the compilation and the execution of the GNU-Radio programs, too. Additionally, the framework includes a communication module handling writing and reading of GNU-Radio data. Thus, gnugym enables researchers to bring their own GNU-Radio example as ML model into the environment.

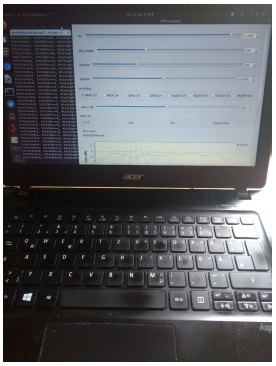

Fig. 11. Experiment Setup

A test with the IEEE 802.11p stack was not successful. Different RL algorithms do not converge. Indeed, there were results using a SL approach. These algorithms are trained on a collected dataset from the environment. However, a maximum training accuracy around 70% can be achieved. This value is limited to the chosen model and the overlapping best actions depending on the observation.

Acknowledgment

We want to give many thanks to our project supervisor Dr. Anatolij Zubow and the TKN group at TU Berlin.

References

[1] Bloessl, Bastian, et al. "Performance assessment of IEEE 802.11 p with an open source SDR-based prototype." IEEE transactions on mobile computing 17.5 (2017): 1162-1175.

[2] Abdelgader, Abdeldime MS, and Wu Lenan. "The physical layer of the IEEE 802.11 p WAVE communication standard: the specifications and challenges." Proceedings of the world congress on engineering and computer science. Vol. 2. 2014.

[3] N. Farsad and A. Goldsmith, *Detection Algorithms for Communication Systems Using Deep Learning* arXiv:170.08044v2, 31 Jul 2017

[4] https://openai.com/about/, 17.07.2020

[5] https://www.gnuradio.org/about/, 17.07.2020

[6] https://github.com/bastibl/gr-ieee802-11, 17.07.2020

[7] Valerio, Danilo. "Open source software-defined radio: A survey on gnuradio and its applications." Forschungszentrum Telekommunikation Wien, Vienna, Technical Report FTW-TR-2008-002 (2008).

[8] Gawłowicz, Piotr, and Anatolij Zubow. "ns3-gym: Extending openai gym for networking research." arXiv preprint arXiv:1810.03943 (2018).

[9] Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).

[10] Gawłowicz, Piotr, and Anatolij Zubow. "Ns-3 meets openai gym: The playground for machine learning in networking research." Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems. 2019.

[11] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. Klimov, O. (2017). "Proximal Policy Optimization Algorithms.". CoRR, abs/1707.06347. (2017).

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." CoRR, abs/1801.01290, (2018).