Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
RESTART

A MULTI-ARMED BANDIT APPROACH IN UNDERWATER NETWORKS

Andrea Panebianco
andrea.panebianco@unipa.it
University of Palermo, CNIT Catania Research Unit, Italy

RESEARCH GROUP


Fabio Busacca


Laura Galluccio


Sergio Palazzo


Andrea Panebianco


Raoul Raftopoulos

OUTLINE

- Artificial intelligence
- Machine Learning
- Reinforcement Learning
- Multi-Armed Bandit
- Adaptive Modulation in Underwater acouStic nEtworks (AMUSE)
- Exercises

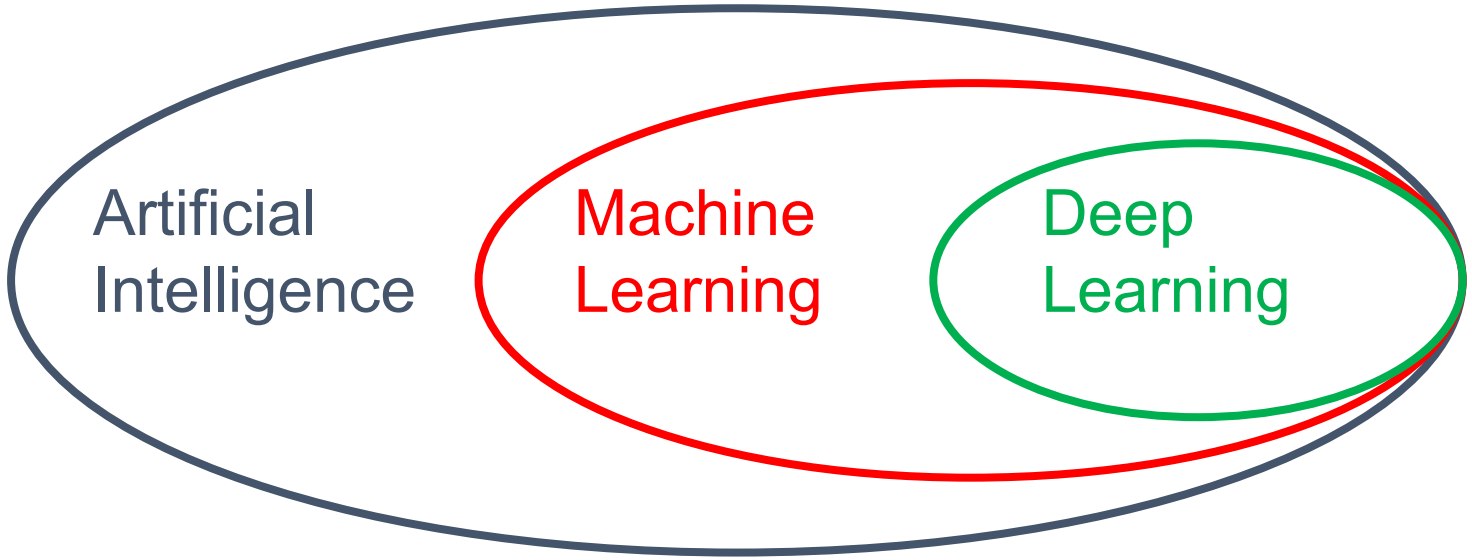
ARTIFICIAL INTELLIGENCE

Artificial Intelligence: "*The study of how to produce machines that have some of the qualities that the human mind has, such as the ability to understand language, recognize pictures, solve problems, and learn*''

- **Pros**: lower cost, reduced complexity and developing time
- **Cons**: potentially limited performance, not always applicable

Ref: https://dictionary.cambridge.org

ARTIFICIAL INTELLIGENCE



ARTIFICIAL INTELLIGENCE VS MACHINE LEARNING

Artificial Intelligence

- Simulate human behaviour
- Build systems which perform tasks the human way
- Learning and Reasoning
- Wide scope of application

Machine Learning

- Learn how to behave from data
- Teach machines to accurately perform tasks with data
- Learning
- Smaller scope of application

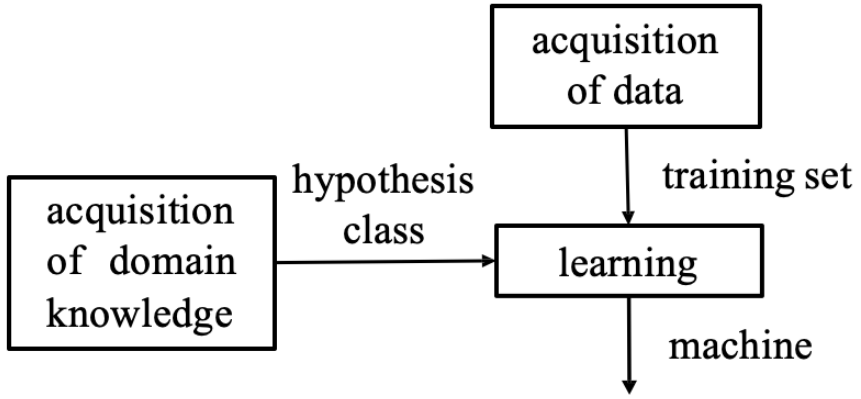
MACHINE LEARNING

Machine Learning: ”*A type of artificial intelligence in which computers use huge amounts of data to learn how to do tasks rather than being programmed to do them*”

Ref: https://dictionary.cambridge.org

HOW MACHINE LEARNING OPERATES

- Data acquisition
- Selection of a general-purpose model and of a learning algorithm
- Learning to decide in future actions with a similar kind of data

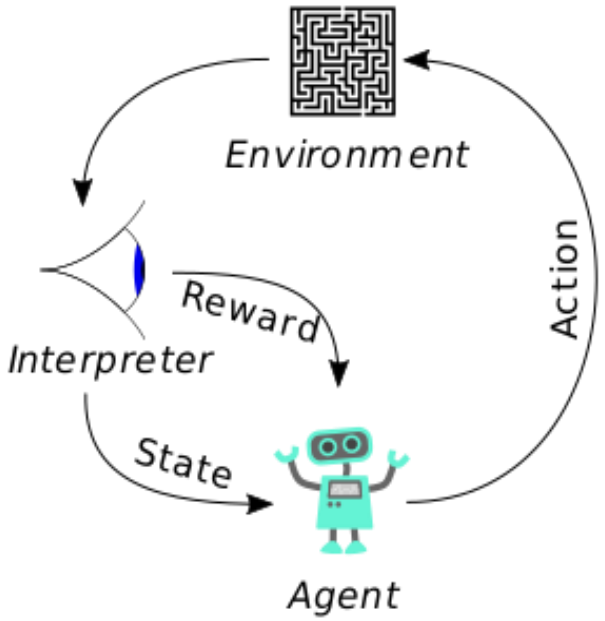


MAIN TYPES OF LEARNING

- **Supervised Learning:** the model is trained on a dataset of labeled data and learns to associate input data with corresponding output labels (e.g., classify images of animals into different species)
- **Unsupervised Learning:** the model is trained on a dataset of unlabeled data and learns to identify patterns and structure in the data, without the need for explicit labels (e.g., cluster data points into groups based on their similarities)
- **Reinforcement Learning:** the model learns by interacting with its environment and receives rewards or punishments for its actions, and it learns to take actions that maximize the expected reward over time (e.g., train a robot to navigate a maze)

REINFORCEMENT LEARNING

A model is obtained as a **Markov decision process**



HOW REINFORCEMENT LEARNING OPERATES

- **S**: set of environment and agent states
- **A**: set of agent actions

Action → $P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$

Reward → $R_a(s, s')$

Markov Property implication: What happens at instant $t + 1$ depends only on the previous state

DEEP REINFORCEMENT LEARNING

- **Deep Learning** is a type of machine learning that uses artificial neural networks to learn from data. Neural networks are made up of interconnected nodes that can learn to recognize patterns in data
- **Deep Reinforcement Learning** (DRL) is a subfield of machine learning that combines **Reinforcement Learning** (RL) and Deep Learning. DRL algorithms are able to process very large inputs (e.g., every pixel rendered on the screen in a video game) and decide which actions to take to optimize a goal (e.g., maximize the game score)

MULTI-ARMED BANDIT

- Multi-Armed Bandit (**MAB**) algorithms are online-learning algorithms where a decision-making agent explores different options (**arms**), to learn which is the best in a given context for future exploitation
- In a MAB problem, the action expected payoff or expected reward (**action-value**) is represented by $Q(a, n)$ and defines the average reward for each action at epoch n
- Each action has its own reward distribution, which is unknown to the agent. The agent must therefore learn this distribution over time

MULTI-ARMED BANDIT: REWARD KNOWLEDGE

- **Known**: the reward distributions are known to the agent, which can therefore choose the best action at every time step
- **Unknown**: the reward distributions are unknown to the agent, which must therefore learn them over time

MULTI-ARMED BANDIT: REGRET

- **Regret** is the difference between the reward that the agent could have obtained by choosing the best action at every time step and the reward that the agent actually obtained

MULTI-ARMED BANDIT: ALGORITHMS

There are many different algorithms for Multi-Armed Bandits, each with its own advantages and disadvantages. Some common algorithms include:

- **Epsilon-greedy**: it chooses a random action with probability ε and the best action with probability $1 - \varepsilon$
- **Thompson sampling**: it chooses an action based on its probability distribution
- **Upper Confidence Bound**: it chooses the action with the highest Upper Confidence Bound (**UCB1**) value, which is an estimate of the sum of the mean rewards of the actions plus the square root of the number of times they have been played

UPPER CONFIDENCE BOUND

- The most widely used method for MAB problem solution is **UCB1**, in which the more uncertain it is about an action, the more important it becomes to explore it
- Each time an action is selected, the number of times that action has been selected increases, and the uncertainty estimate accordingly decreases. After an initial phase of exploration, the MAB algorithm will always select the action with the biggest Q-Value, i.e., the estimated best one

Adaptive Modulation in Underwater acouStic nEtworks (AMUSE)

- The **AMUSE** agent is implemented in Python and runs as a separate module, external to the DESERT simulator. The Python script encompasses the UCB1 algorithm, and the interfaces required to communicate with the DESERT framework
- It considers the total number of packets transmitted and received in the previous decision epoch, and according to both the current epoch and past history, identifies the supposedly best modulation
- This choice is finally broadcast through feedback packets to all network nodes for usage in the next decision epoch

F. Busacca, L. Galluccio, S. Palazzo, A. Panebianco, R. Raftopoulos, "Adaptive Modulation in Underwater acouStic nEtworks (AMUSE): a Multi-Armed Bandit approach", to appear in IEEE ICC MWN Symposium, Denver, CO, USA

AMUSE: DECISION PROCEDURE

- At each decision epoch, the agent identifies the modulation that will be applied by all nodes. After the action is executed, the agent will receive a variable reward depending on the throughput and the Packet Delivery Ratio (**PDR**) experienced by the nodes
- Let P_n^r be the number of packets received during the time interval between two decision epochs $n-1$ and n

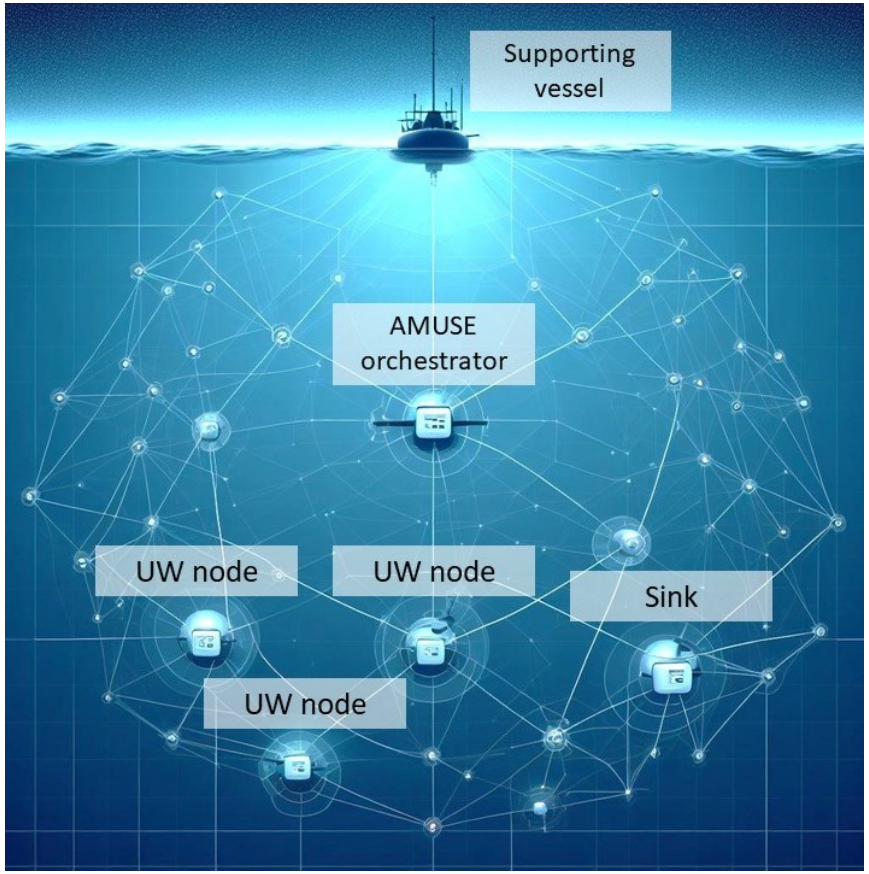
$$P_n^r = \sum_{\forall t \in n-1} p_t^r \quad \text{where} \quad p_t^r = \begin{cases} 1 \; a\ packet\ has\ been\ received\ by\ the\ Sink\ at\ time\ t \\ 0 \; otherwise \end{cases}$$

- Recalling that the objective of the system is to increase the overall network throughput, the reward accounts for the number of packets correctly received during each decision epoch, that is:

$$r_n^{(M)} = {}^{P_n^r}/_{\Delta} = {}^{\sum_{\forall t \in n-1} p_t^r}/_{\Delta}$$

SYSTEM ARCHITECTURE

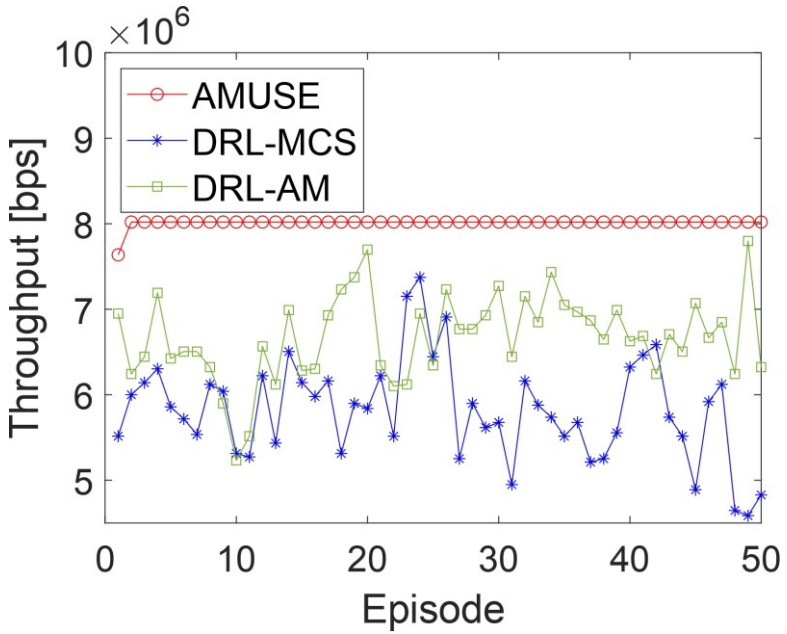
- An UWA network, where multiple nodes communicate with a destination node (**Sink**)
- A central node (**AMUSE orchestrator**) running the **AMUSE agent** and leveraging on the statistics of the UW links status to train and execute it
- Each node periodically receives indication by the orchestrator about the modulation scheme to use (**BPSK, 8PSK, 16PSK**)



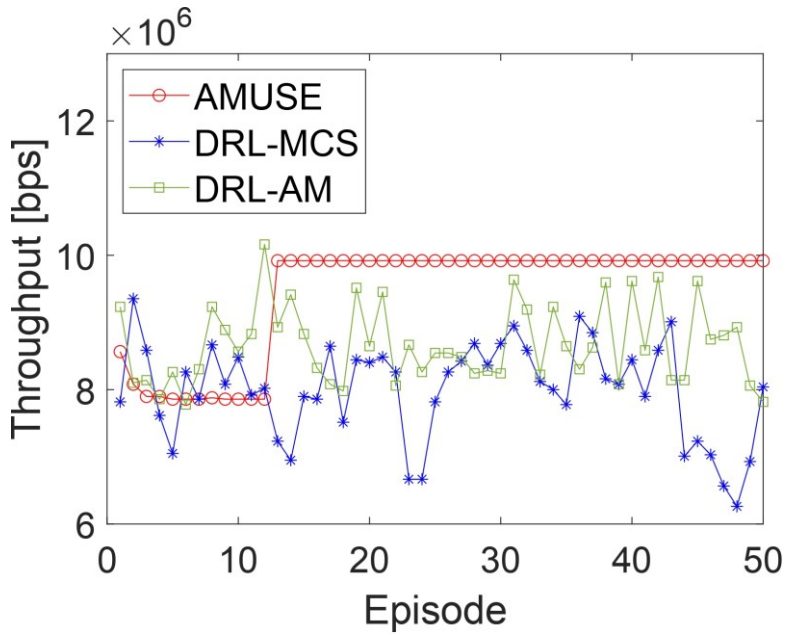
AMUSE: COMPARISON

- Comparison of the AMUSE approach with two alternative state-of-the-art methodologies, to test its effectiveness:
  - Deep Reinforcement Learning-based Adaptive Modulation (**DRL-AM**) relies on a DRL framework to adapt modulation schemes as an alternative to non-linear Auto-Regressive Moving Average (ARMA) models
  - DRL-based intelligent Modulation and Coding Scheme selection (**DRL-MCS**) the appropriate modulation and coding schemes in the aim of minimizing the impact of interference coming from secondary users

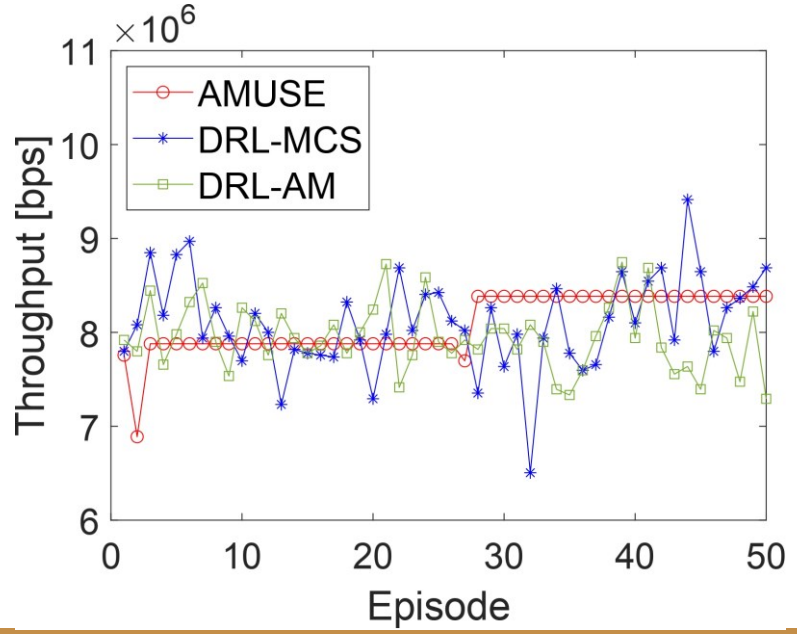
RESULTS



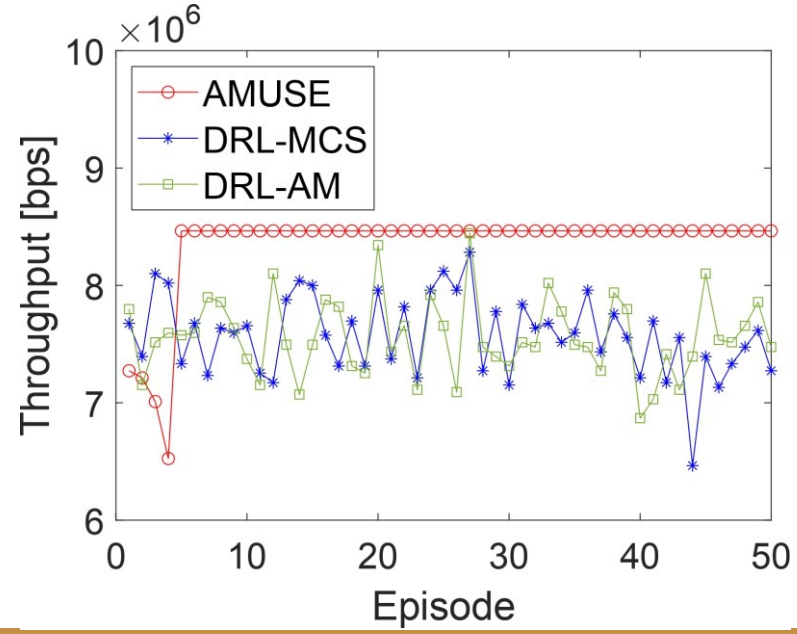
4-nodes scenario



6-nodes scenario

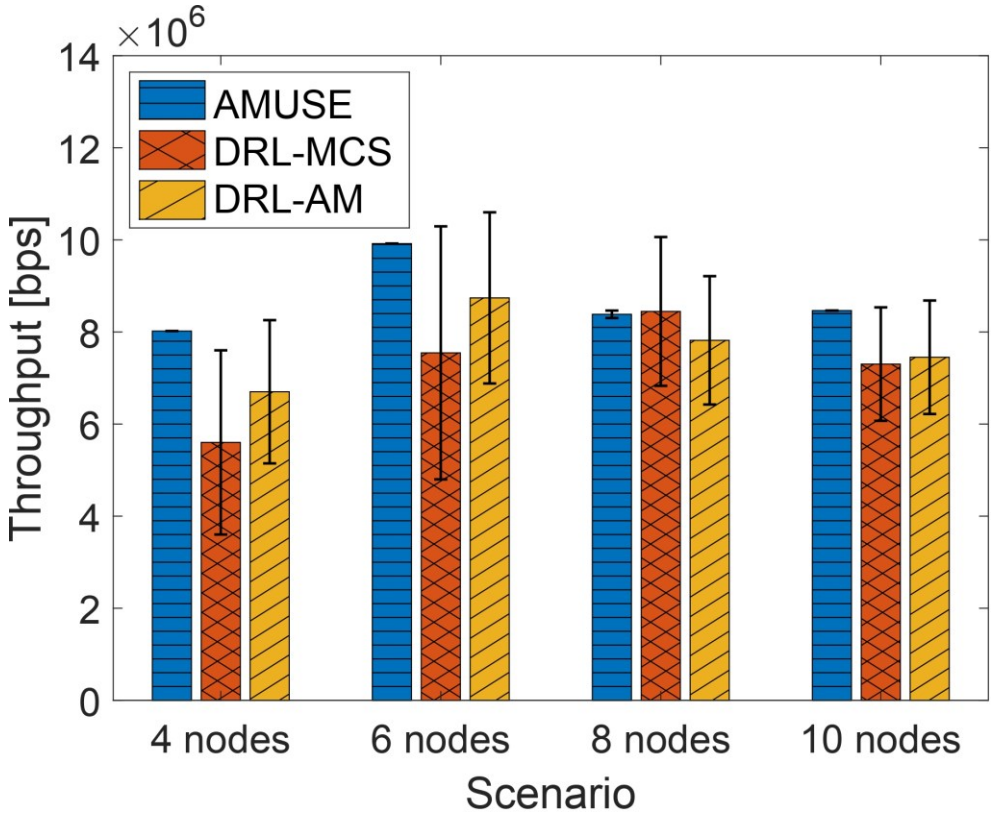


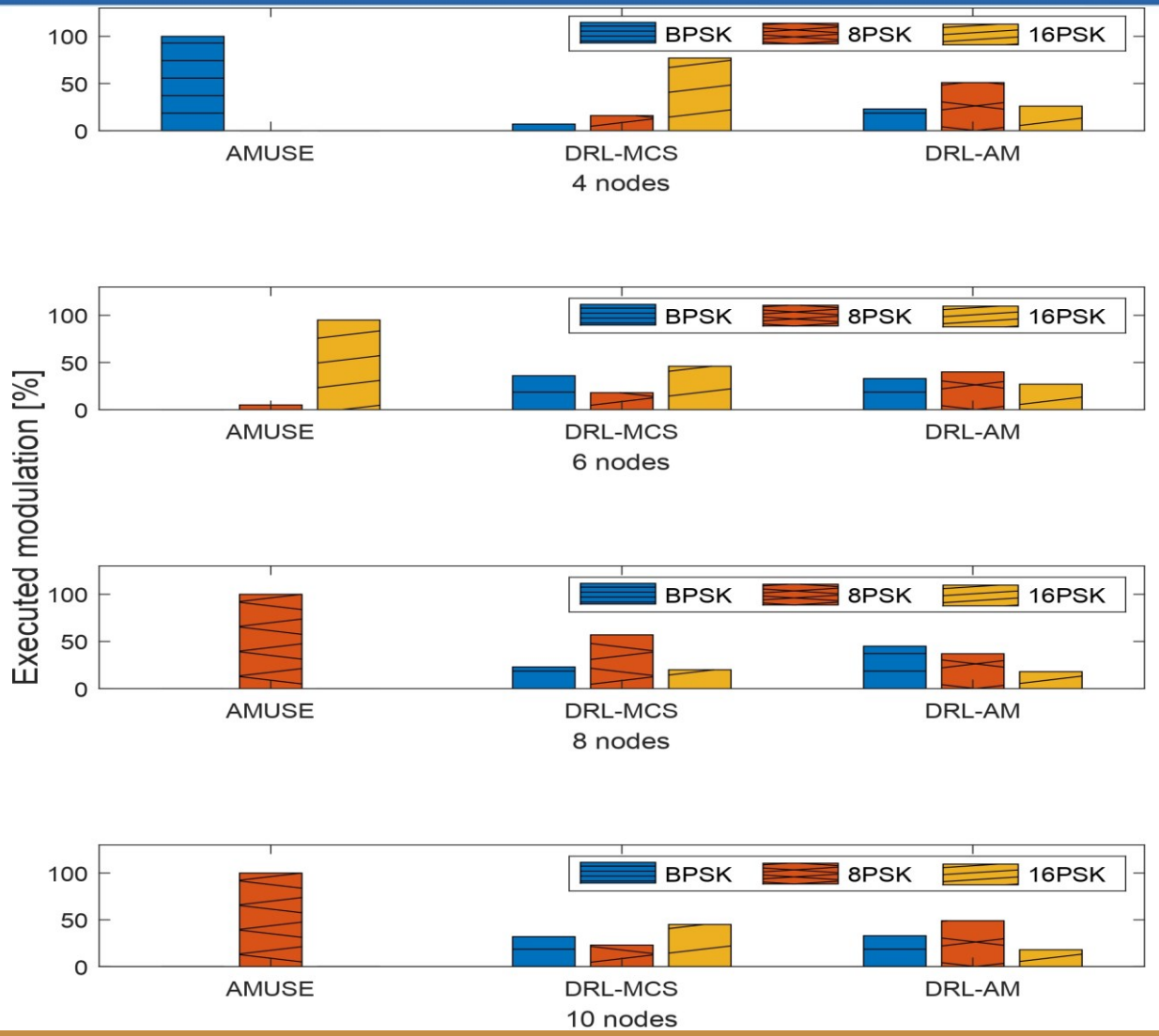
8-nodes scenario



10-nodes scenario

RESULTS


×10^6
14
12
10
8
6
4
2
0
Throughput [bps]
AMUSE
DRL-MCS
DRL-AM
4 nodes
6 nodes
8 nodes
10 nodes
Scenario


Executed modulation [%]
100
50
0
BPSK
8PSK
16PSK
AMUSE
DRL-MCS
DRL-AM
4 nodes
6 nodes
8 nodes
10 nodes

EXERCISES

AMUSE SETUP

- Install the CLI and Python library for interacting with the Weights and Biases API

```
sudo pip install wandb
```

- Next, log in and paste your API key when prompted

```
wandb login 465f070138c47bc63fff5775cbf64d55078319b1
```

- Download the AMUSE and DESERT scripts

andreapanebianco/UNWIS-2024: AMUSE scripts for UNWIS 2024 (github.com)

AMUSE SETUP

- `AMUSE.py` is the AMUSE agent
- `AMUSE_DESERT_simulation.tcl` is the DESERT simulation
- `Bash_simulation.sh` is the executable with which to repeat n training epochs for AMUSE
- `rewards.csv` contains the rewards for AMUSE
- `actions.csv` contains the suggested arms
- `synchronization.csv` allows the synchronization between DESERT & AMUSE at each time step of the DESERT simulation
- `done.csv` allows the synchronization between DESERT & AMUSE to finish each epoch

DESERT CONFIGURATION: SYNC WITH AMUSE

```
puts "Received Packets: $sum_cbr_rcv_pkts"
puts "Packet Error Rate: [expr ((1 - double($sum_cbr_rcv_pkts)/ $sum_cbr_sent_pkts) * 100)]"
set reader [open "~/file_path/synchronization.csv"]
set data_synchro [read $reader]
close $reader
set data_synchro [split $data_synchro ","]
set step [lindex $data_synchro 0]
set rcv [lindex $data_synchro 1]
puts "step: $step"
puts "rcv: $rcv"
```

Remember to change the file path

DESERT CONFIGURATION: CALCULATE THE IMPROVEMENT

```
set rcv_temp [expr $sum_cbr_rcv_pkts - $rcv]
set rcv $sum_cbr_rcv_pkts
puts "rcv_temp: $rcv_temp"
```

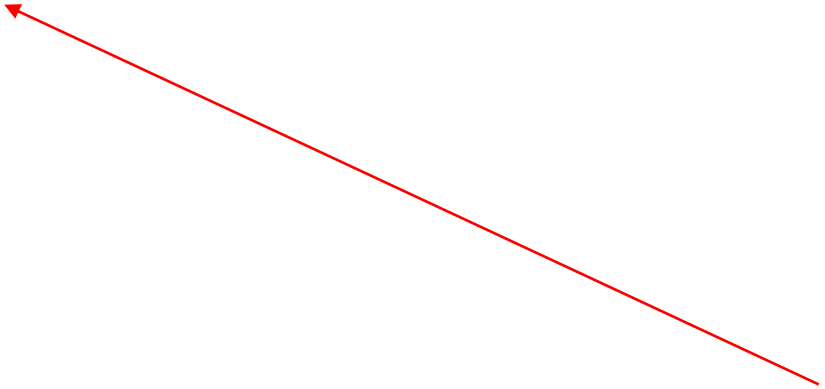
DESERT CONFIGURATION: WRITING TO AMUSE

```
set writer [open "~/file_path/rewards.csv" w+]
puts $writer "$step, $rcv_temp"
close $writer
puts "Wrote to rewards.csv: $step, $rcv_temp"
after 500
```

Remember to change the file path

DESERT CONFIGURATION: ACCESSING TO AMUSE INSTRUCTION

```
while 1 {
puts "Entering while loop"
set reader [open "~/file_path/actions.csv"]
set data [read $reader]
close $reader
```



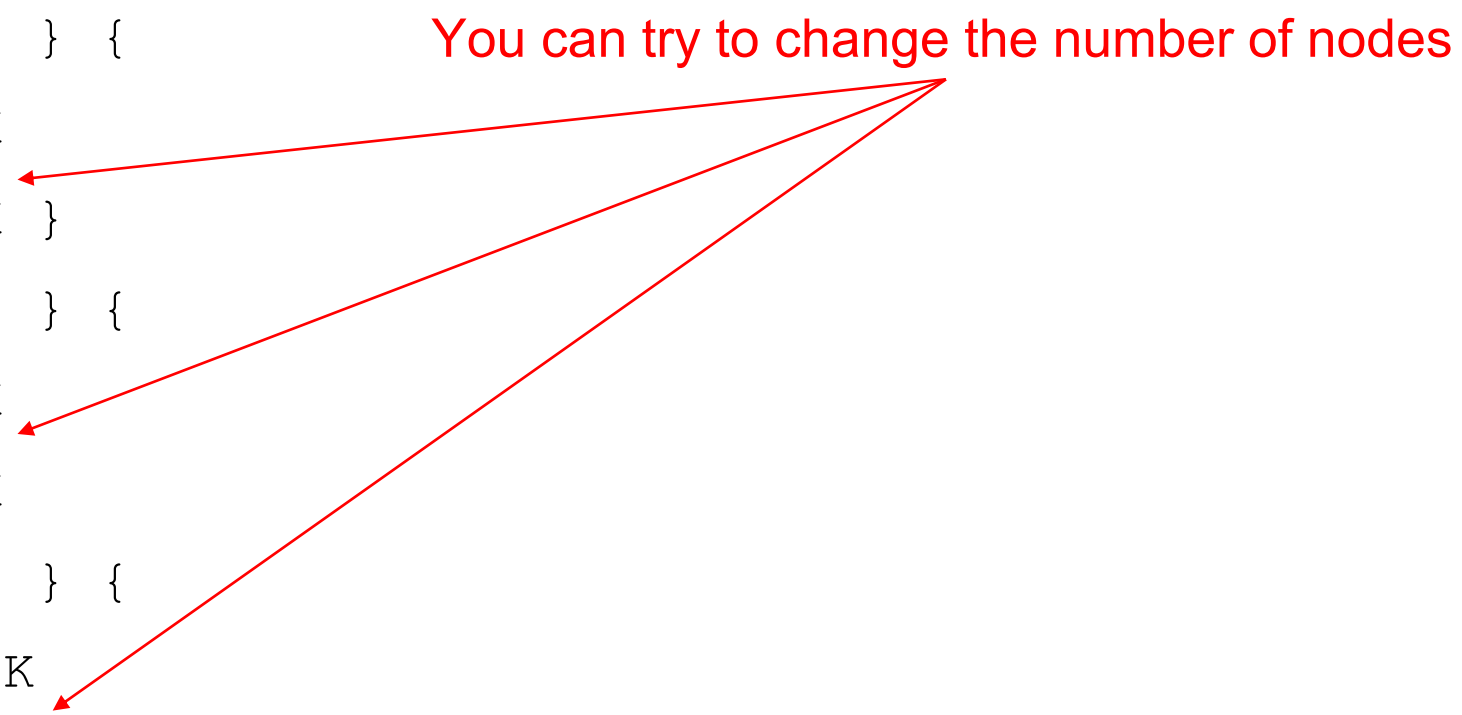
Remember to change the file path

DESERT CONFIGURATION: READING THE MODULATION

```
set item [split $data ","]
set int_step [lindex $item 0]
set modulation [lindex $item 1]
puts "int_step is: $int_step, modulation is: $modulation, int_step
has to match step: $step"
if { $int_step == $step } {
puts "breaking from while loop"
break }
after 100 }
puts "setting modulation to $modulation"
```

DESERT CONFIGURATION: SETTING THE MODULATION

```
if { $modulation == "0" } {
$phy(0) modulation BPSK
$phy(1) modulation BPSK }
if { $modulation == "1" } {
$phy(0) modulation 8PSK
$phy(1) modulation 8PSK
if { $modulation == "2" } {
$phy(0) modulation 16PSK
$phy(1) modulation 16PSK }
```



DESERT CONFIGURATION: UPDATING THE SYNCHRONIZATION

```
incr step
puts "Incrementing step: step = $step"
set writer [open "~/file_path/synchronization.csv" w+]
puts $writer "$step, $rcv"
close $writer
puts "wrote step: $step, rcv: $rcv to synchronization.csv"
after 500 }
```

Remember to change the file path

DESERT CONFIGURATION: STARTING THE SIMULATION

```
for {set i 1} {$i<$opt(stoptime)/$opt(interrupttime)} { incr i } {
$ns at [expr $opt(interrupttime)*$i + 250.0]  “proc;”
}
$ns run
```

DESERT CONFIGURATION: ASSURING DESERT & AMUSE SYNCHRONIZATION

Remember to change the file path

```
while 1 {
puts "Entering while loop"
set reader [open "~/file_path/done.csv"]
set value [read $reader]
close $reader
puts "Value is: $value"
if { $value == 1 } {
puts "Breaking from while loop"
break }
after 100 }
```

DESERT CONFIGURATION: BASH FILE

```
#!/bin/bash
for i in {0..15}
      do
            sleep 2
            echo 1, 0 > ~/file_path/synchronization.csv
            ns AMUSE_DESERT_simulation.tcl
      done
```

You can try to change the number of epochs

Remember to change the file path

TRY THE SIMULATION WITH AMUSE

In the folder, open two terminals; in the first one:

- `chmod +x Bash_simulation.sh` to set the bash file as executable
- `./Bash_simulation` to start the simulation for *n* epochs and train AMUSE

in the second one:

- `python3 AMUSE.pip` to start AMUSE

Remember to start AMUSE always before the bash file