Using AI/ML frameworks with ns-3

Hao Yin

Collaborators:
Pengyu Liu, Keshu Liu, Xun Deng (Huazhong University of Science and Technology, HUST)
Lyutianyang Zhang, Liu Cao, Collin Brady, Sachin Nayak (University of Washington, UW)
Advisors:
Xiaojun Hei, Yayu Gao (HUST), Sumit Roy, Thomas R. Henderson (UW)

Outline

- AI/ML in communication
- Introduction of the ns3-ai module
- Basic usage of the ns3-ai module
  - Basic functions and usage
  - Demos and instructions with code
- Example: Wi-Fi Rate Control

Introduction

AI/ML in Communication

❑ Motivation

Challenges for the traditional communication system:

> PHY/MAC Complexity: Multi-User Operation, Temporal Dynamics → hard to model analytically

> Clear model, but space of algorithms very large → complex search for suitable algorithm

> Model/Algorithm deficiencies due to imperfect information → learn/adapt

Aligned with:

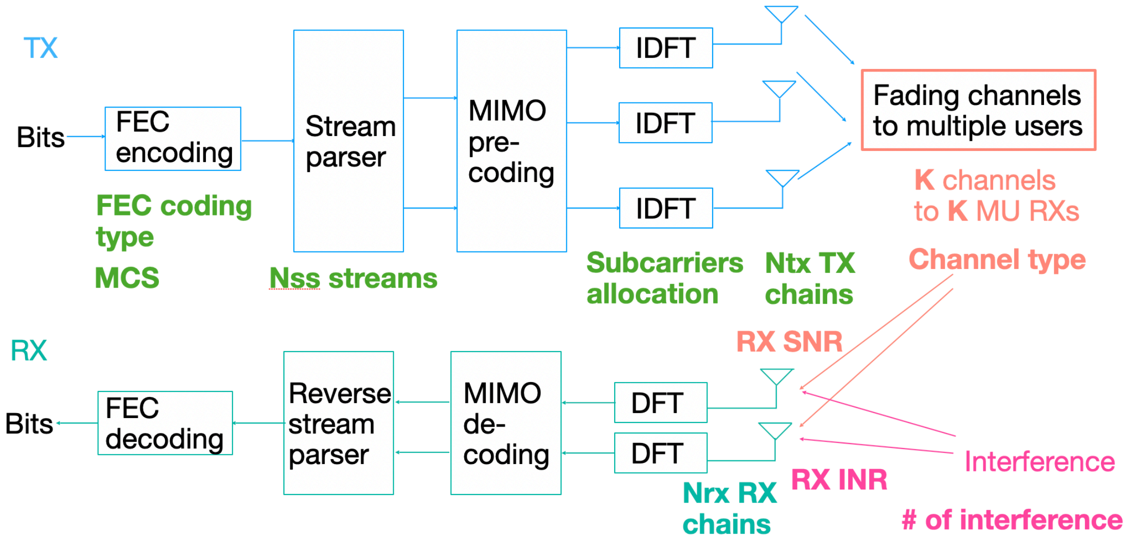
> Availability of large data sets (supervised/un-supervised)

> Fast Computing (GPU, FPGA)

> Mapping Algorithmic onto ML computation architecture

AI/ML in Communication

❑ PHY Layer



PHY layer procedures in MIMO system

Exploding Complexity

> Increasing MIMO dimensions

> Complex channels

> Greater co-channel interference (denser networks)

> Higher order of MU transmissions

Applications of Deep Learning in PHY layer improvements

> DeepTurbo: Deep Turbo Decoder [1]

> Deepcode: Feedback Codes via Deep Learning [2]

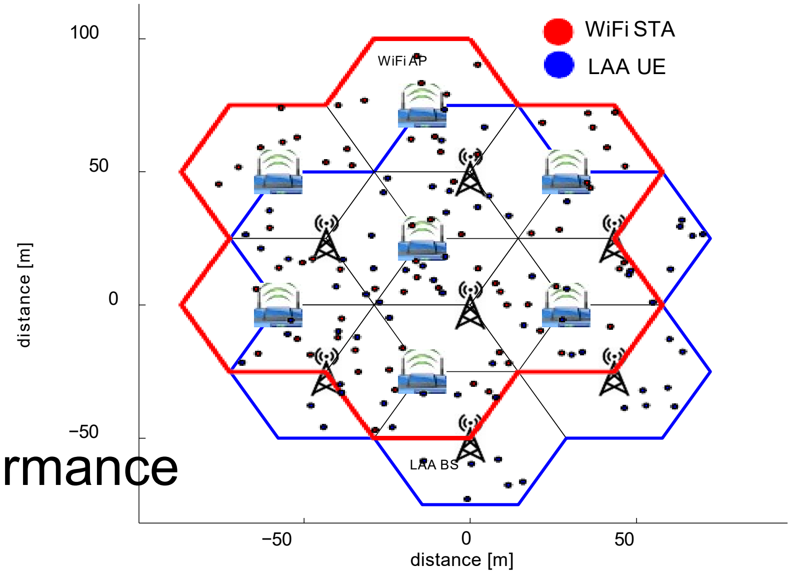
[1] Y. Jiang, S. Kannan, H. Kim, S. Oh, H. Asnani and P. Viswanath, "DEEPTURBO: Deep Turbo Decoder. "
[2] H. Kim, Y. Jiang, S. Kannan, S. Oh and P. Viswanath, "Deepcode: Feedback Codes via Deep Learning. "

AI/ML in Communication

❑ MAC Layer

Dense overlapping Networks

- > Interference between different BSSs
- > Scheduling and resource allocation scheme to improve the performance
- > Link adaptation and channel access scheme
- > Coexistence of WiFi, LTE, and 5G



Dense Overlapping networks

Applications of Reinforcement Learning in MAC layer

- > FDRL in WiFi: Reinforcement Learning for WiFi channel access [3]
- > DRL in Resource Slicing for eMBB and URLLC Coexistence in 5G [4]

AI/ML is promising for the next-generation wireless networks!

[3] L. Zhang, H. Yin, Z. Zhou, S. Roy and Y. Sun, "Enhancing WiFi Multiple Access Performance with Federated Deep Reinforcement Learning."
[4] M. Alsenwi et al, "Intelligent Resource Slicing for eMBB and URLLC Coexistence in 5G and Beyond: A Deep Reinforcement Learning Based Approach."

Ns-3 Simulation with AL/ML algorithms

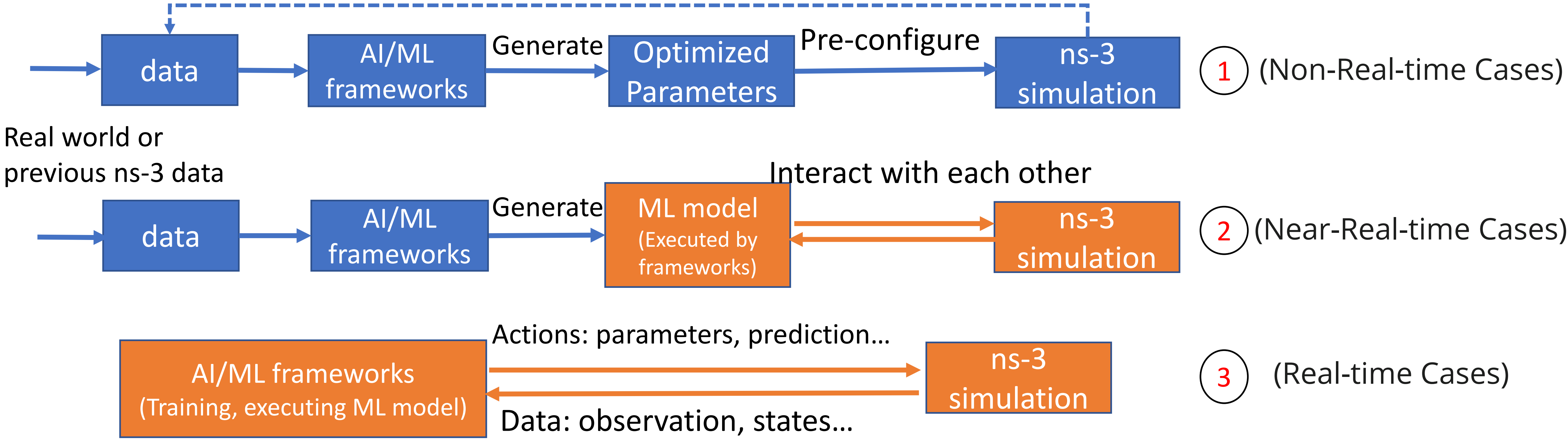
> Easier to scale network dimension (e.g. dense overlapping networks) for performance evaluation, compared to building test bed.

> Allows cross layer (PHY/MAC) simulation → e.g. cross-layer optimization like WiFi rate control.

> Generate data and traces → use ns-3 to generate training data.

> Ns-3 is open source → easy for the researcher to modify and test the AI/ML algorithms.

AI/ML model training with ns-3

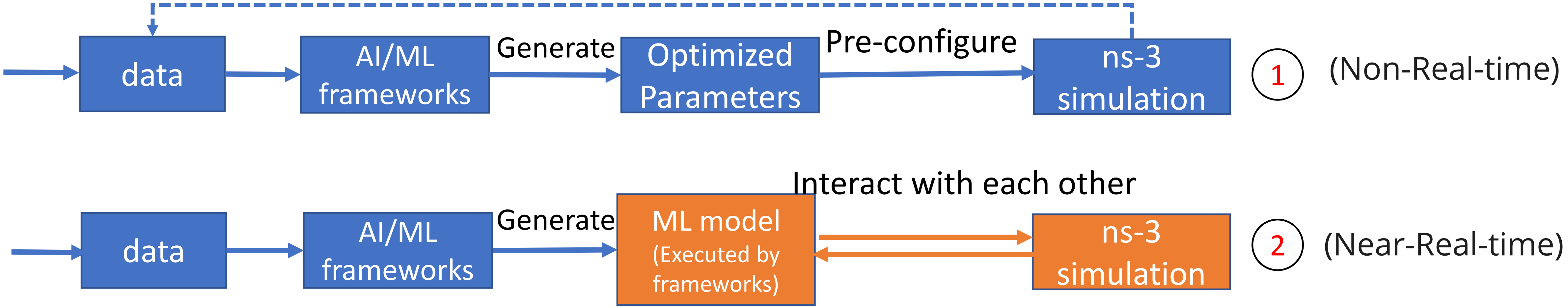


1 2 Offline Training

3 Online Training

AI/ML model training with ns-3

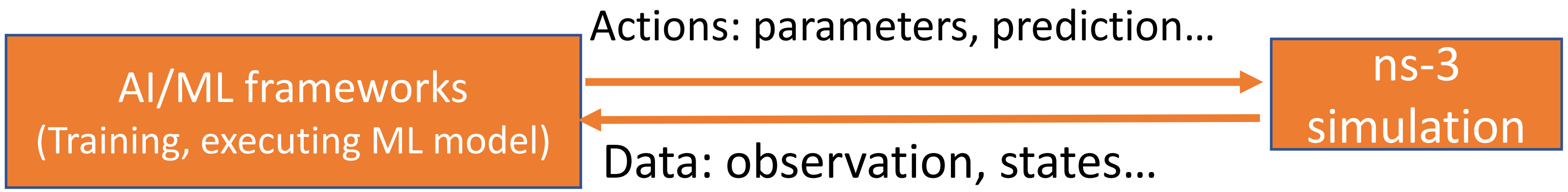
❑ Offline training



- Offline training: The model is first trained with offline data (from real world or ns-3).
- Two ways to use the model in the network/simulation.
  - 1: Configure the parameters before the simulation: power, ACK timeout threshold, sensing power.
  - 2: Make decisions dynamically during the simulation: execute the model to adjust the parameters in simulation.

AI/ML model training with ns-3

❑ Online Training



3 (Real-time)

> Scenarios such as reinforcement learning, where the model ‘learns’ as it is executing in the network.
> Interact with ns-3 in real time.
> Examples: CQI, RL-TCP, WiFi Rate Control

Using AI/ML frameworks with ns-3

❑ How to connect ns-3 with AI/ML frameworks

> Find C++ frameworks – compile with ns-3
- Hard to development ML model with C++
- Compile issues
- C++ API found insufficient

> Python Bounding
- Not support for all the models – rewrite the bounding code
- Not good with call back functions

> Using Python based frameworks – Inter-process communication (IPC)
- Easier way to develop ML models
- IPC: sockets, shared memory…
- Take time to develop the IPC models

Using AI/ML frameworks with ns-3

Methods / Aspects	Compiling with the C++ library versions of AI frameworks	ns3-ai	ns3-gym	Python Bounding
AI frameworks supported	Re-Compile C++ version for different frameworks	**ALL python-based**	OpenAI-Gym	ALL python-based
Develop efficiency	Low (All in C++)	**High**	**High**	Medium (Re-implement some ns-3 models)
Additional Resource	**None**	Low (Extra Memory)	Medium (Memory, Socket)	None
Speed	**Fast**	Medium	Slow	Slow
Ease of Use	Hard	**Easy**	**Easy**	Medium

Research & Evaluation

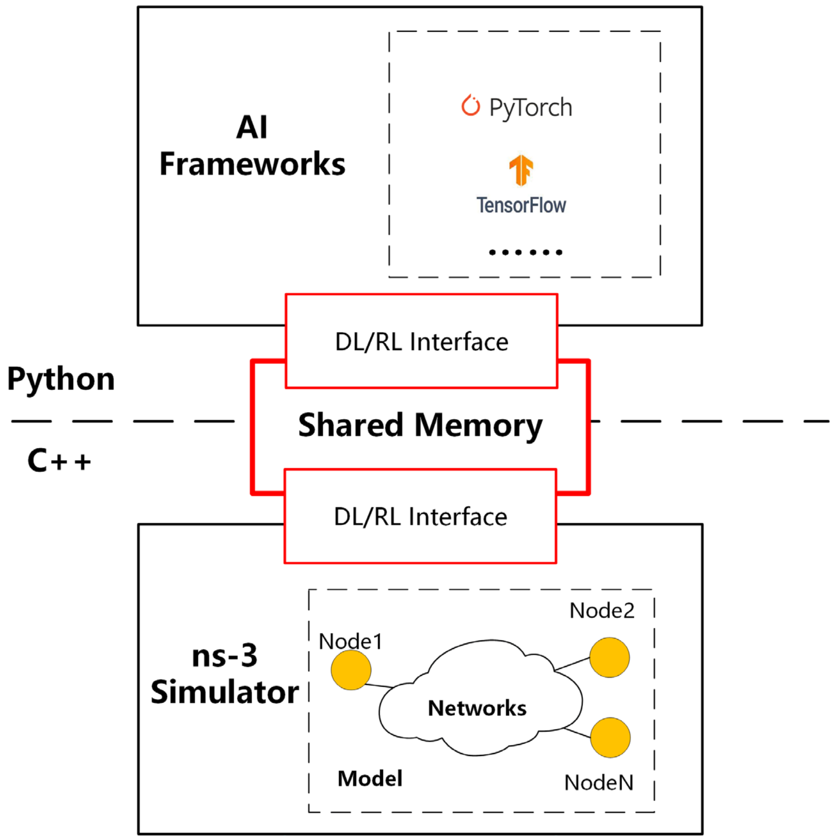
ns3-ai module

❑ Overview

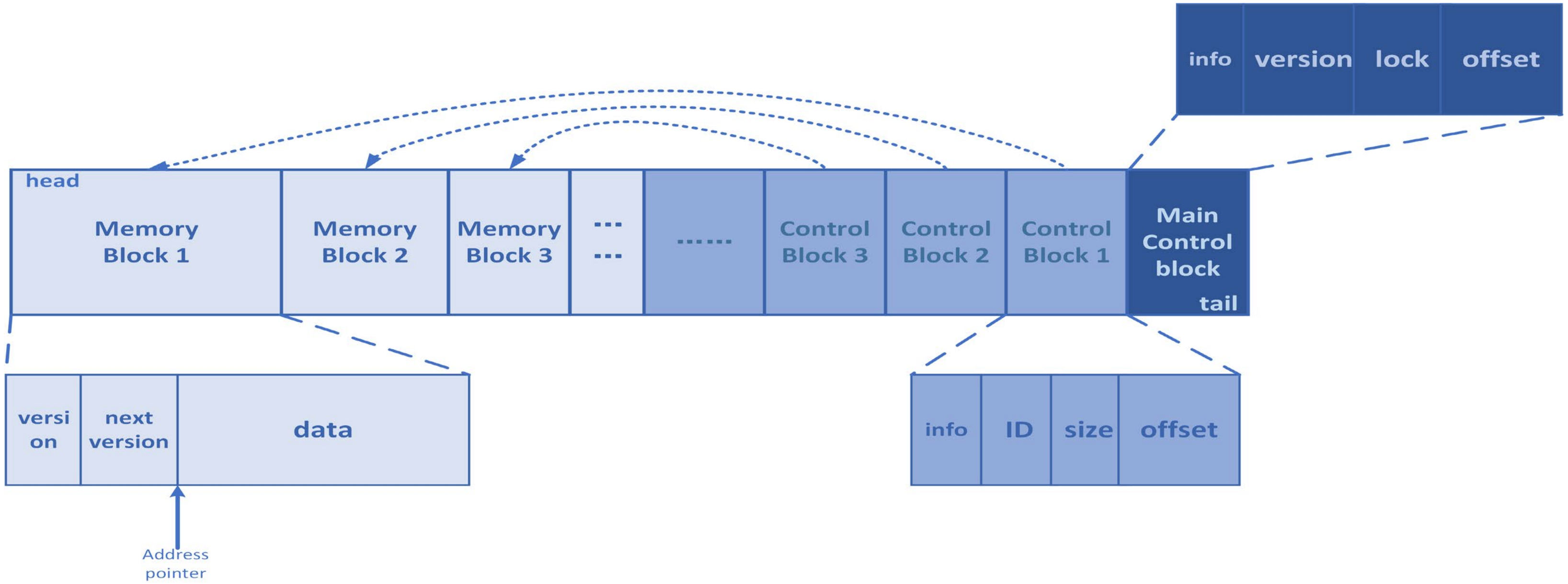
- Use shared memory as a data transmission module

- Divide the data transmission module into two parts
- Python side
- C++ side

- Memory Management:
  - > Shared memory pool management
  - > Dynamic memory allocation
  - > Read/Write lock



ns3-ai module



> Shared memory pool was divided into main control block, control block and memory block

> Read-write lock: version information

- Compare the version with the next version:
  > same = accessible
  > different = not accessible

ns3-ai module

Implementation: shared memory

> Use C++ to create and maintain shared memory
  - Create and release of shared memory pool
  - Inter-process read-write lock control and synchronization

> Development on the Python side
  - The same function with Python interface

Python side	C++ side	function
py_init	SharedMemoryPool	Created a shared memory pool
py_freeMemory	FreeMemory	Freed shared memory pool
py_regMemory	RegisterMemory	Got memory block and added read-write lock
py_getMemoryVersion	GetMemoryVersion	Got version information and determined the lock status
……	……	……

ns3-ai module

❑ User-friendly features

> No need to consider low-level development and management of the shared memory.

> DL/RL interface (helper) for users
> - E.g., use Env and Action for the variables directly for RL algorithms

> Directly use python to run the ns-3 examples (no need to run separately).

> Hands-on Examples:
> - Simple usage: multi-run (A+B)
> - DL model: CQI
> - RL model: RL-TCP
> - Wi-Fi Rate Control

ns3-ai module

❑ Possible Pitfalls and Difficulties

> Installation:
- Using Python3
- Environment path issues

> Shared Memory
- Don’t use the same id for different programs
- Not enough memory: change the size of shared memory to 256/512 (bytes)
- Use the right lock: Follow the multi-run example

ns3-ai module

❑ Release Timeline and Plans

2021 Feb. R1.5
1. All the components can be established in Python
2. RL multi-agent support
3. Enhance RL algorithms in TCP example

2021 Q3 R 2.0
1. Run with optimized mode in ns-3
2. More detailed documents and instructions
3. Wi-Fi Rate Control example

2020 Feb. R1.0
1. Shared Memory Operation
2. DL and RL interface
3. CQI and TCP examples

Why and When to use ns3-ai?

- Do you need Python to implement your ML algorithms?
  - C++: Multi-arm-bandit: Thompson sampling, ε-greedy
  - Python: Neural Network based algorithms like Deep Q-Learning, LSTM

- Do you need to train or execute a ML model?

- Do you already implement the code in Python?

Basic Usage

Basic Usage

- In this section, we will show how to use ns3-ai module step by step.
- Codebase: ns-3-dev (https://gitlab.com/nsnam/ns-3-dev)
- Test Machine: MacBook Pro (macOS Catalina 10.15.7)
- Installation guide: https://github.com/hust-diangroup/ns3-ai#installation
- Basic usage example: https://github.com/hust-diangroup/ns3-ai/tree/master/example/multi-run
- We now go through all the installation and example code together. (Live Demo)

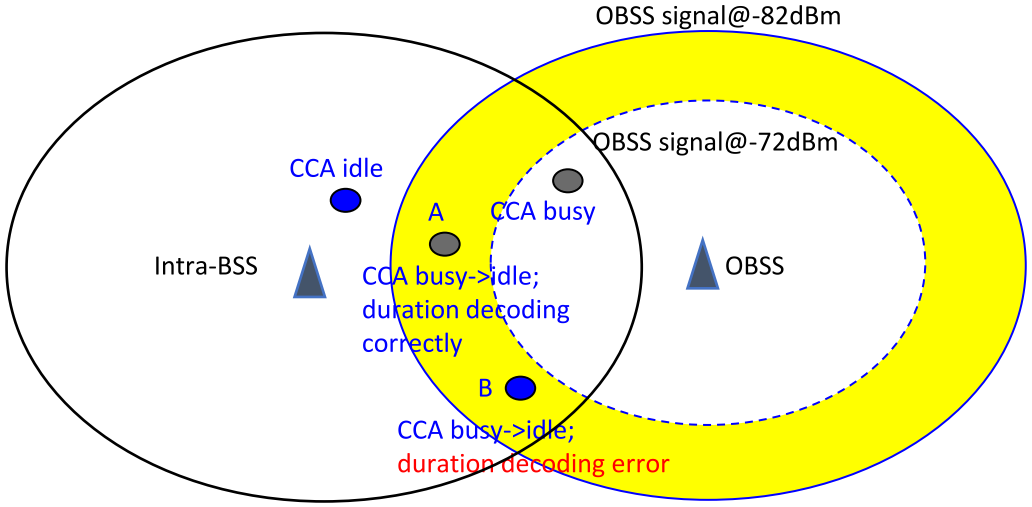
Example – Wi-Fi Rate Control

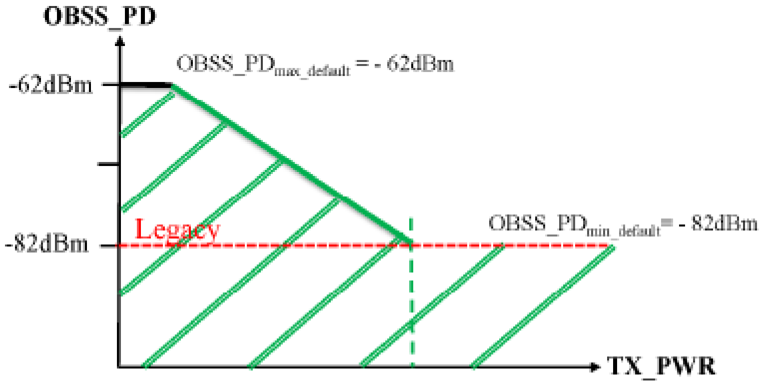
Dense and overlapping WLANs

Spatial Reuse in 802.11ax

Intra-BSS frame:
Same frame color -> have to contend for the medium as normal process.
Inter-BSS frame:
Different colors-> can ignore it and will not contend for the medium and will continue transmitting.

Enable simultaneous transmissions in overlapping networks to increase total throughput.





OBSS_PD threshold []

Benefits/Challenges from the Spatial Reuse

Benefits:

- Increase capacity
- Adaptive CCA can adjust signal level threshold
- Decrease channel contention problem
- Signals with same BSS color use a low RSSI threshold for deferral, therefor reducing collision in same BSS.
- Signals with OBSS use a higher RSSI threshold for deferral, therefor allowing more simultaneous connection

Challenges:

- Increase the interference
- The rapid changing wireless environment -> dynamically change the parameter values

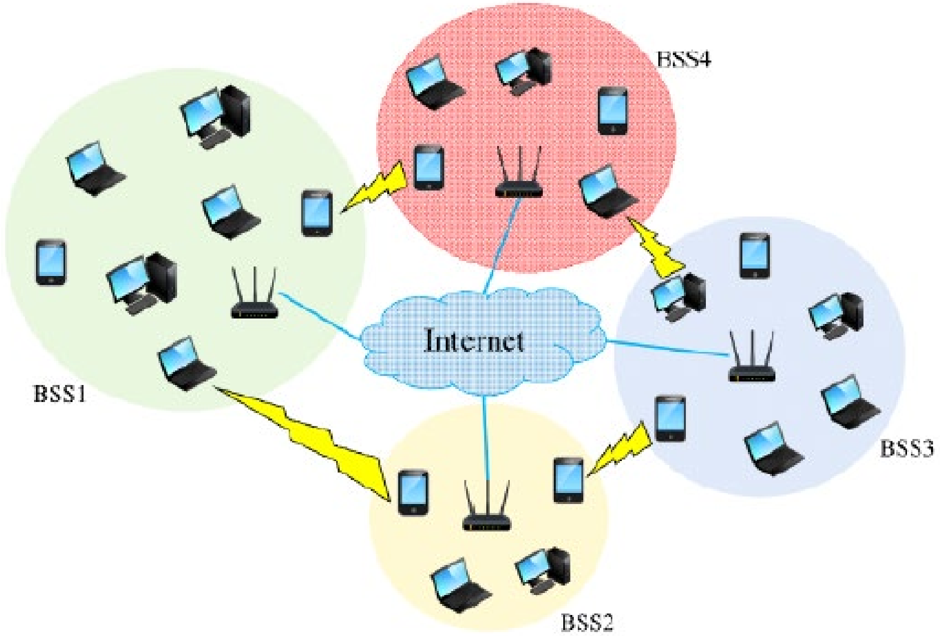
```
 STA1           STA1
   |              |
 d1|              |d2
   |      d3      |
  AP1 -----------AP2
```

```
wifi-spatial-reuse.cc simulation:
(mcs = 0, d3 = 150m, d1 = d2 =
30m)

Enable spatial reuse:
Throughput for BSS 1: 6.6468 Mbit/s
Throughput for BSS 2: 6.6672 Mbit/s
Disable spatial reuse:
Throughput for BSS 1: 5.8692 Mbit/s
Throughput for BSS 2: 5.9364 Mbit/s
```

Dense Overlapping Scenario

- Dense and overlapping WLAN networks – multiple Basic Service Set (BSS)
- Each BSS interfere with each other
- Several factors on the system performance:
  - Carrier Sense Range (CSR)
  - Interference Range (IR)
  - MCS per frame



Increase throughput and decrease delay?

Impacts on the Rate Adaptation (MCS Selection)

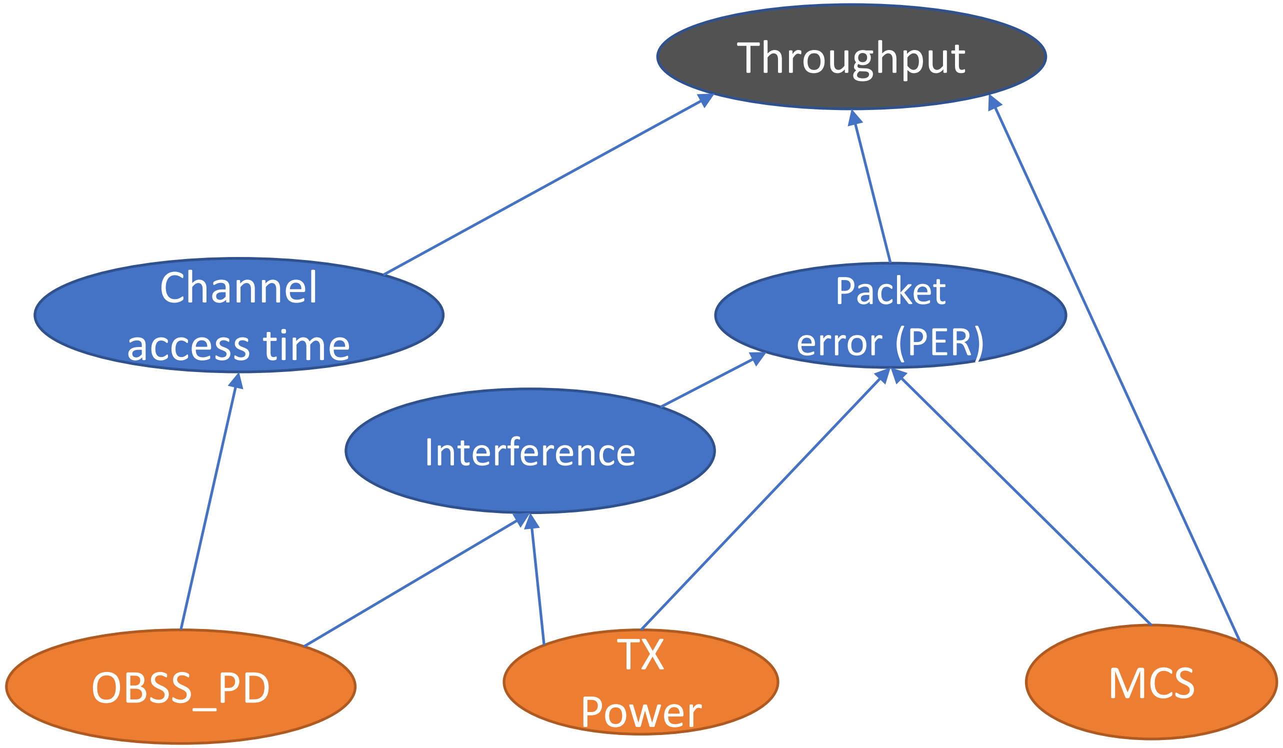
Complex dependency of different parameter

- Change one may impact others
- Hard to randomly search the optimal solutions

Dynamically changed environments

- In dense set up, more devices and more interference
- Traffic features may change very quickly

Hard to find the (sub-)optimal solution with traditional approach!



Wi-Fi Rate Control Algorithms in ns-3

Ideal: Use the SINR at the receiver and channel model to calculate the PER and then determine the data rate at the transmitter (unreal)

Minstrel:

- Goal: maximize throughput (TP)
- Trial-and-error:
  - Idea: try to send at different rates, measure their channel statistics (success probability, TP), then choose the one that can max TP to send normal packets
  - Two time period (exploration & exploitation):
    - Sampling period: send probe packets with a certain rate policy
    - Non-sampling period: send normal packets with a certain rate policy
    - Minstrel uses 10% of the traffic for sampling (measuring channel statistics)

Name	Category	Metrics	Modify Standard
Ideal	Explicit Feedback	SINR	Y
Minstrel-HT	Implicit Feedback	Throughput	N
Thompson Sampling			N

UNIVERSITY of WASHINGTON

Wi-Fi Rate Control Algorithms with RL

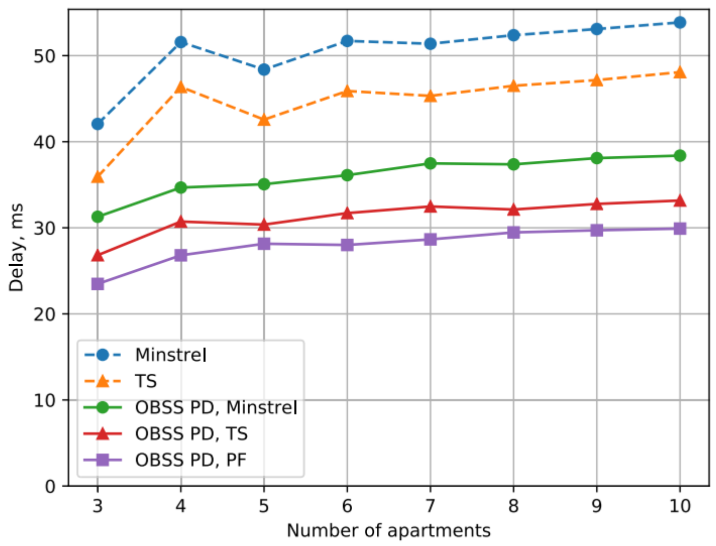


FIGURE 13. **Results for the residential building.**

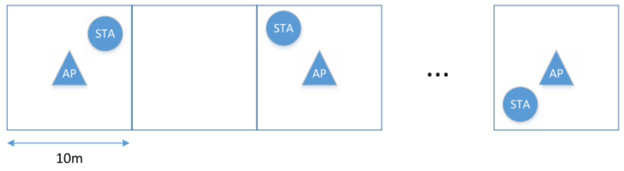


FIGURE 12. **Residential building scenario.** [5]

- **TS:** MAB algorithm, using binomial distribution to approximate the success probability and then select the MCS (arm). Using Thompson sampling (TS) approach to calculate reward.
- **PF:** Estimate the channel SINR, then using TS to approach to approximate the SINR distribution, and then select the MCS based on the SINR.
- **OBSS PD:** Using OBSS PD to enable spatial reuse setup. The same way to calculate the OBSS PD: Threshold = Average RSSI − Margin (Margin is a positive value that considers channel quality fluctuations).

Benefits from RL (reinforcement learning):

- Explore the optimal way to search the (sub-)optimal setup <-> randomly search in traditional ways .
- Learn from the environment -> ‘remember’ similar situations.
- Capable for the optimization in large and complex scenario.

Deep RL? MAB?

UNIVERSITY *of* WASHINGTON

[5] A. Krotov, A. Kiryanov and E. Khorov, "Rate Control With Spatial Reuse for Wi-Fi 6 Dense Deployments," in IEEE Access, vol. 8, pp. 168898-168909, 2020, doi: 10.1109/ACCESS.2020.3023552.

Why and When to use ns3-ai?

- Do you need Python to implement your ML algorithms?
  - C++: Multi-arm-bandit: Thompson sampling, ε-greedy
  - Python: Neural Network based algorithms like Deep Q-Learning, LSTM
- Do you need to train or execute a ML model?
- Do you already implement the code in Python?

Name	Need Python?	Online or Model execution?	Already have code?	
Thompson Sampling	N	Y	Y	→ C++, but can ns3-ai also do the job?
DQN	Y	Y	Y	→ ns3-ai

Build TS rate control algorithm with ns3-ai

UNIVERSITY *of* WASHINGTON

Implementation of Thompson Sampling with ns3-ai

Objective:

- How to transfer data and maintain statistics using ns3-ai?
- How to add the ML algorithms using ns3-ai?
- How to build a structure to test your model?
- Benchmarking

Step by step example to build your own model with ns3-ai!

The following section would be some operation and analysis with the code.
Exp code: https://github.com/hust-diangroup/ns3-ai/tree/master/example/rate-control

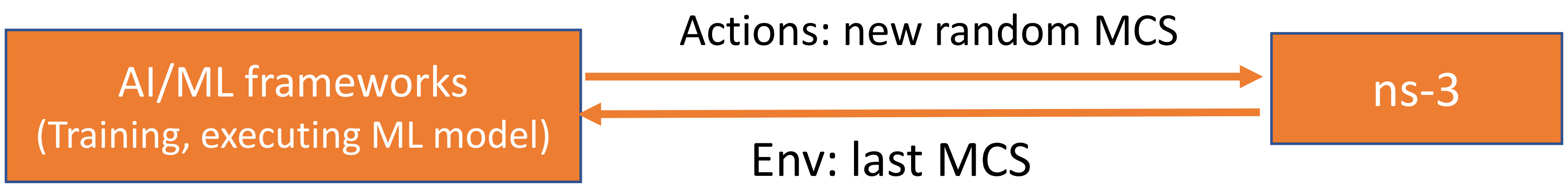
Thompson Sampling Rate Control Algorithm

- For a given MCS j (i.e., the transmission rate r_j), the number of successful transmissions follows the binomial distribution with unknown success probability p_j.
- Estimate the value of given statistics of successful transmissions -> beta distribution with parameters (1, 1)
- α_j and β_j are the numbers of successful and unsuccessful transmissions for MCS j.
- At the very beginning $\alpha_j = \beta_j = 0$
- Every time we need a value for p_j, we sample it from the following beta distribution: $p_j(x) = \frac{x^{\alpha_j}(1-x)^{\beta_j}}{\mathrm{B}(\alpha_j, \beta_j)}$
- Thompson sampling-based rate control selects an MCS: $\hat{j} = \arg\max_j p_j r_j$
- Policy improvements: use exponential smoothing after each transmission attempt (decay w and interval Δt):

$$\alpha_j(t) = \begin{cases} \alpha_j(t - \Delta t) \cdot e^{\frac{-\Delta t}{w}} + 1, & \text{in case of success} \\ \alpha_j(t - \Delta t) \cdot e^{\frac{-\Delta t}{w}}, & \text{otherwise} \end{cases}$$

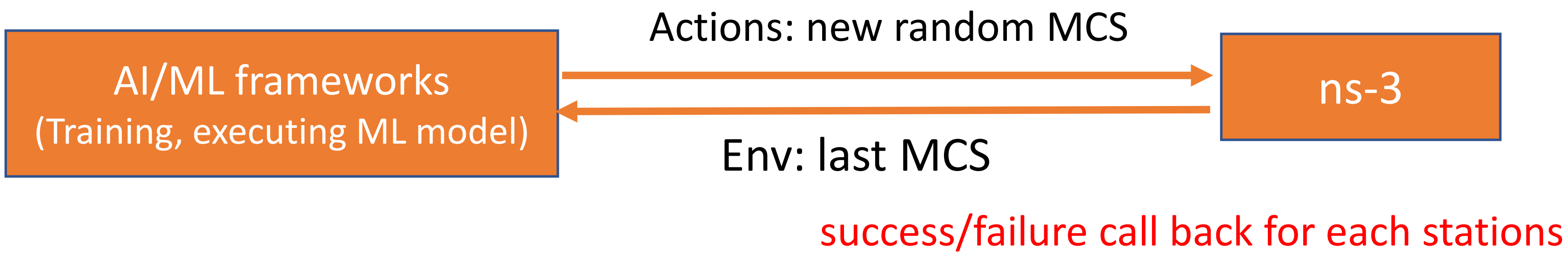
$$\beta_j(t) = \begin{cases} \beta_j(t - \Delta t) \cdot e^{\frac{-\Delta t}{w}} + 1, & \text{in case of failure} \\ \beta_j(t - \Delta t) \cdot e^{\frac{-\Delta t}{w}}, & \text{otherwise} \end{cases}$$

Step 1: Change MCS using ns3-ai



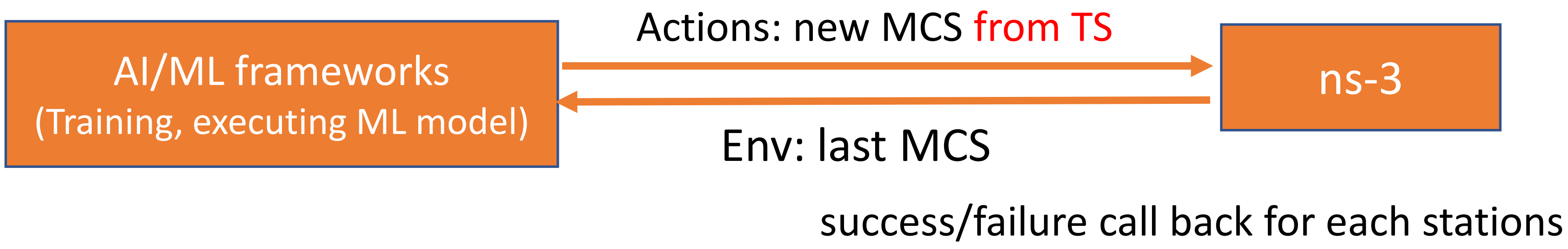
- Add a new rate controller class
- Send the old MCS to Python using ns3-ai
- Put a new MCS back to ns-3

Step 2: Get success/failure statistics from ns-3



> Define call back to transfer the statistics of transmission failures/success
> Transfer the data to Python
> Maintain the statistics for different nodes

Step 3: Add Thompson sampling to obtain new MCS



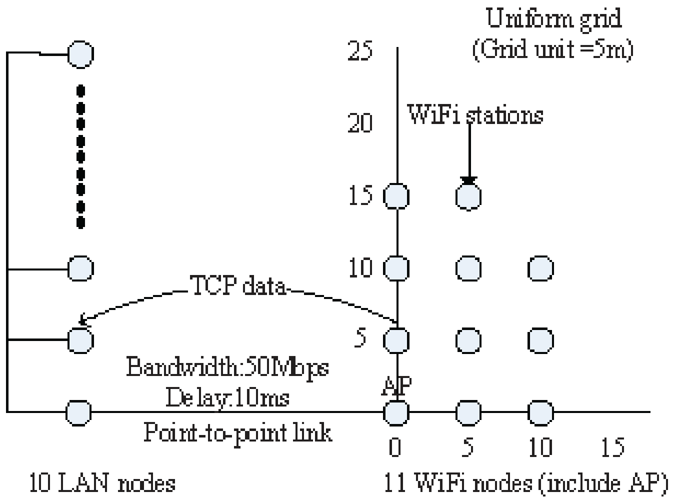
- Implement the TS algorithm in Python
- Using the TS to get the new MCS from the statistics gathered
- Transfer back the action using ns3-ai

Now you have a new version of TS using Python and ns3-ai!

Simulation and Benchmarking

Simulation Scenario

- Created by modifying the file “ examples/tutorials/third.cc” in ns-3.
- The topology contains 10 wired LAN nodes connected to each other and one of the nodes is connected to the stationary Access Point(AP) of the Wireless Network using a point to point link with 50Mbps bandwidth and 10ms delay.
- Reference code: https://github.com/DodiyaParth/802.11ac_compatible_RAAs_Performance_Analysis_in_NS3



Simulation Scenario [6]

[6] Huang, Tingpei, et al. "A comparative simulation study of rate adaptation algorithms in wireless LANs." *International Journal of Sensor Networks* 14.1 (2013): 9-21.

Simulation Setup

Error Rate Model	**NistErrorRateModel**
Channel Delay Model	ConstantSpeedPropagationDelay Model
Channel Loss Model	LogDistancePropagationLossMode l
MAC(Station/AP) Type	Sta WifiMac/ ApWifiMac
Application Data Rate	1 Mbps
Packet Size	1024 bytes
Mobility Model	RandomDirectional2dMobilityMo del
Mobility Speed	Random Variable : U(15.0 mps, 20.0 mps)
Simulation Topology of Wifi nodes	Grid, rectangle range: (-100m, 100m, -100m, 100m)

- Calculate the throughput every second with different rate control algorithms.
- Change the total node numbers and simulation duration to compare the results.

Simulation Results

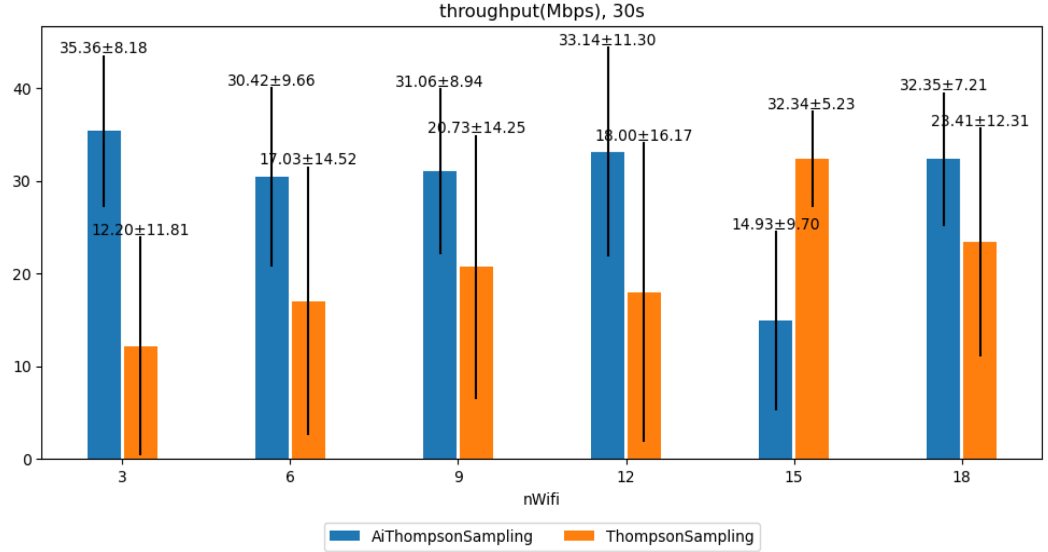


Fig. 1 Average throughput for all 20 random seeds

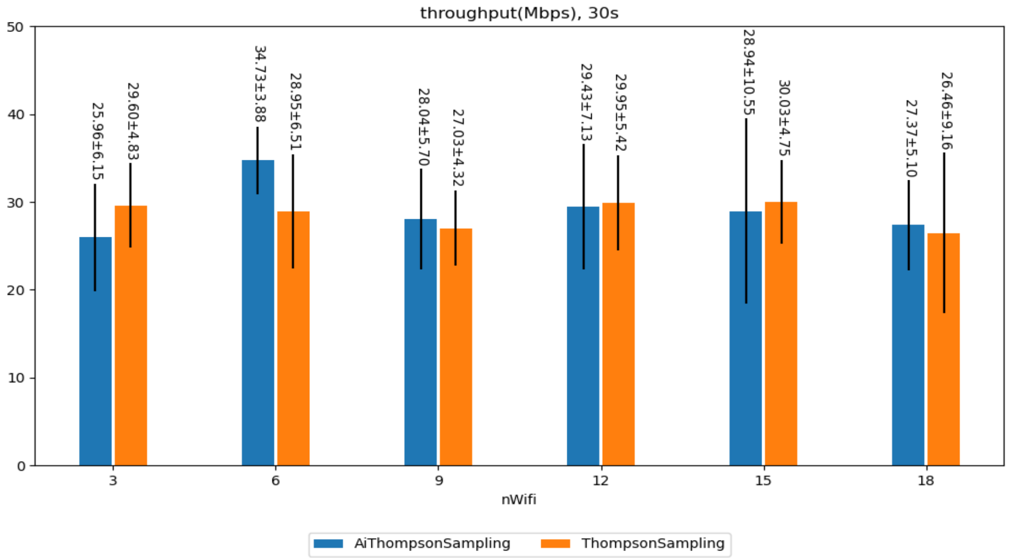
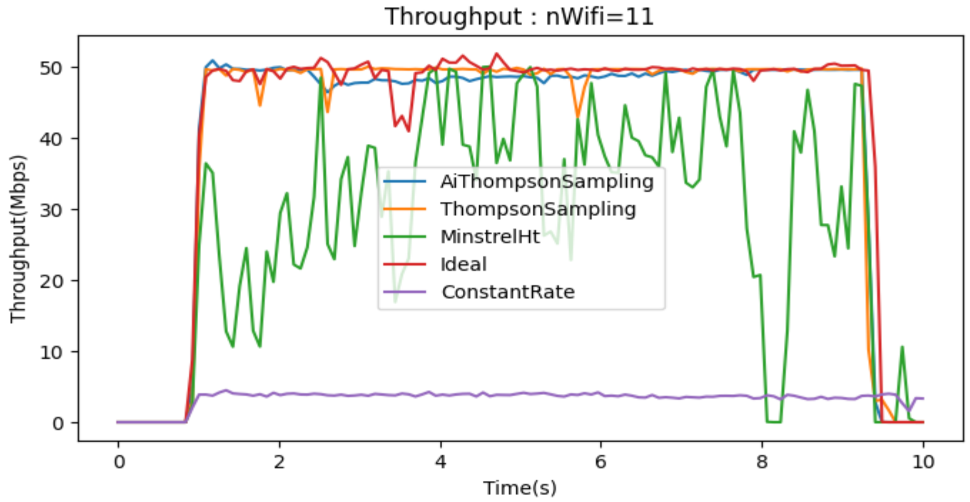


Fig. 2 Average throughput after removing bad cases

- Simulation duration: 30 s
- Change the random seed for 20 different seeds and test the performance

- Fig 1: Average throughput for all 20 cases
- Fig 2: Remove the bad case (less than 2 Mbps) and then run average

- In most cases, the ns3-ai based TS has similar results compared with the pure ns-3 implementation
- By changing the random variable, the variance of the TS algorithm is large, and the pure ns-3 version may have more bad cases.
- Both algorithms outperform the minstrel algorithm.

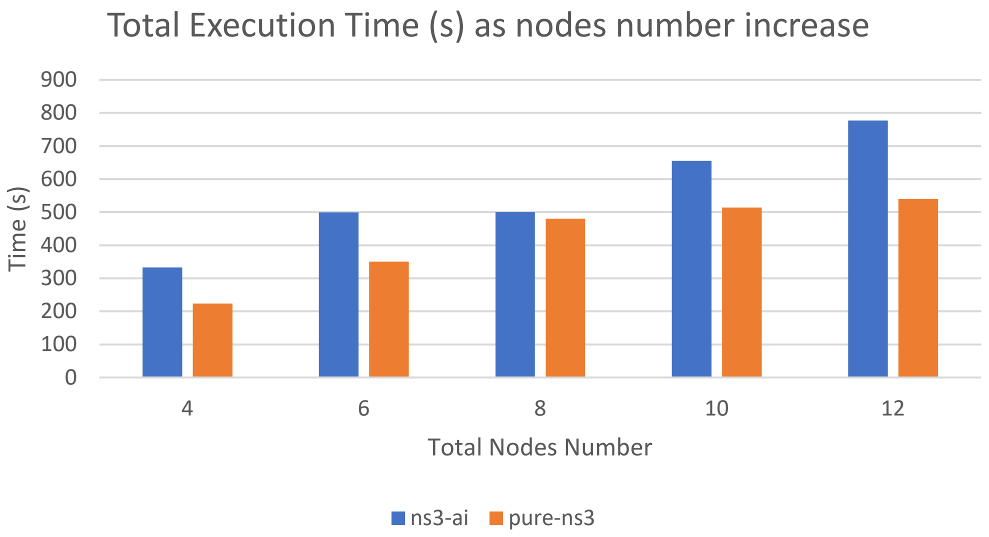


UNIVERSITY of WASHINGTON

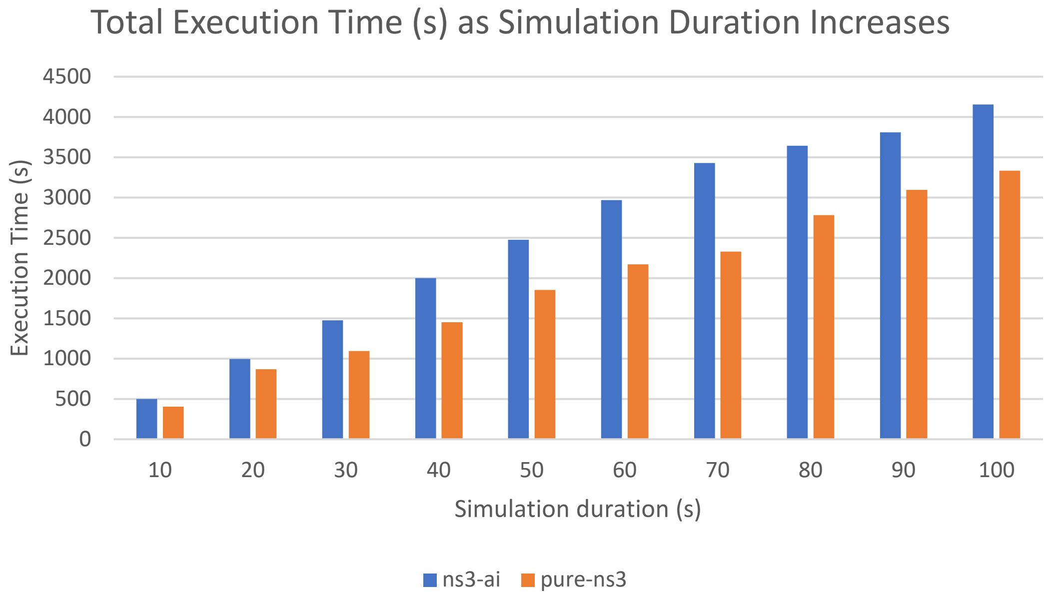
https://gitlab.com/nsnam/ns-3-dev/-/issues/414#note_602000119

Benchmarking

Test Server:
CPU: Intel Xeon Gold 6242 2.8 GHz
Memory: 128 GB
OS: Centos 7



Simulation duration is 10 s



Total STAs number is 8

- Using ns3-ai has around 20% increase on the program execution time in average.
  - ns3-ai introduces extra time that transfers data with shard memory.
  - The difference of the random seed may also contribute to the execution time.

UNIVERSITY *of* WASHINGTON

Thank you!

Backup