Technical Notes on
Implementing Online Deep Reinforcement Learning for Real-Time Media Access Control Scheduling in Radio Access Networks

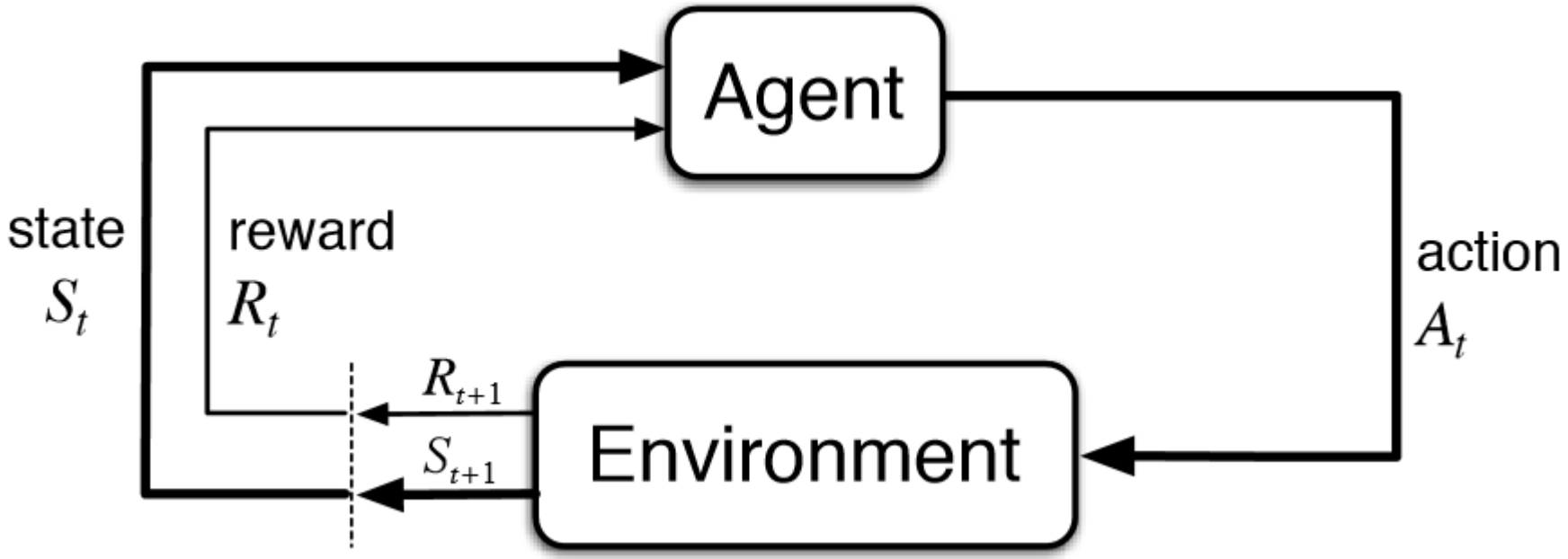
Presented by Dr. Zhouyou Gu, research fellow with Prof. Jihong Park.

This presentation explain the underlying technologies establishing the paper:

Z. Gu et al., “Knowledge-assisted deep reinforcement learning in 5G scheduler design: From theoretical framework to implementation,” IEEE Journal on Selected Areas in Communications, vol. 39, no. 7, pp. 2014–2028, 2021.

Source codes available at https://github.com/zhouyou-gu/drl-5g-scheduler

Background:
Deep Reinforcement Learning (DRL)



Background: Markov Decision Process

- $s_1, a_1, r_1, \dots, s_t, a_t, r_t, s_{t+1}, \dots$
- State-action-reward sequence
  - Observe a state
  - Take an action using policy μ: $a_t = \mu(s_t)$,
  - Get a reward, and adjust the policy
  - then go the next state
- Markovian property: Future state/reward only depends on current state/action.
  - No need to look back to predict the future/No hidden state that affecting the future.
- A transition: $\mathcal{T}_t =< s_t, a_t, r_t, s_{t+1} >$
- Goal find the best policy μ that
  - Task: $\max_{\mu} \mathrm{E}\left[\sum_{i=0}^{\infty} \gamma^i r_i\right]$
  - γ is the discount factor on future reward.

Background on Q learning:
μ is based on a table, $\max_a Q(s_t, a)$

	A1	A2
S1	Q(S1,A1)	..
S2	..	..
S3	..	Q(S3,A2)
S4	..	..
S5	..	..
S6	..	..

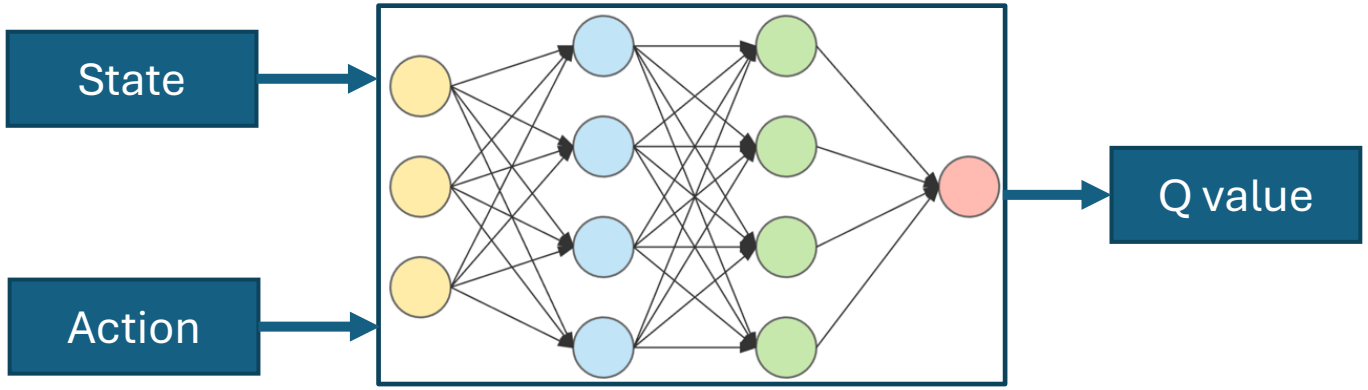
```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
   Initialize s
   Repeat (for each step of episode):
      Choose a from s using policy derived from Q
         (e.g., ε-greedy)
      Take action a, observe r, s'
      Q(s, a) <-- Q(s, a) + α [r + γ max_α Q(s', a') - Q(s, a)]
      s <-- s';
   until s is terminal
```

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Tabular approaches suffer from the curse of dimensionality.

Background on Deep Q-learning: Using a neural network (NN) to approximate the table

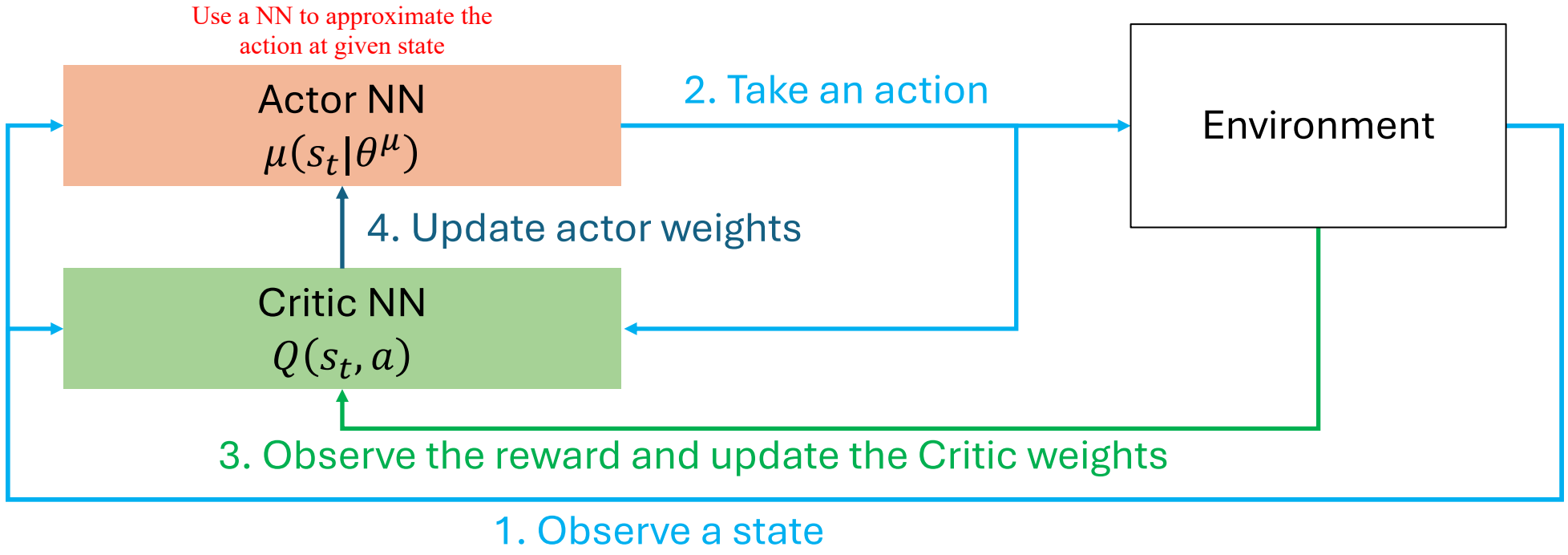
Use a NN to approximate Q values at the given state and action



- Based on Bellman equation, the Q values should be
  - $Q(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a)$.
- Then, the loss function of the NN is
  - $L = \left[Q(s_t, a_t|\theta^Q) - \left(r_t + \gamma \max_a Q(s_{t+1}, a|\theta^Q)\right)\right]^2$
- Still, selecting the best action need to solve $\max_a Q(s_t, a|\theta^Q)$;
  - Typically, exhaustive search is used to find the best action.

It still suffers from the curse of dimensionality when find the best action.

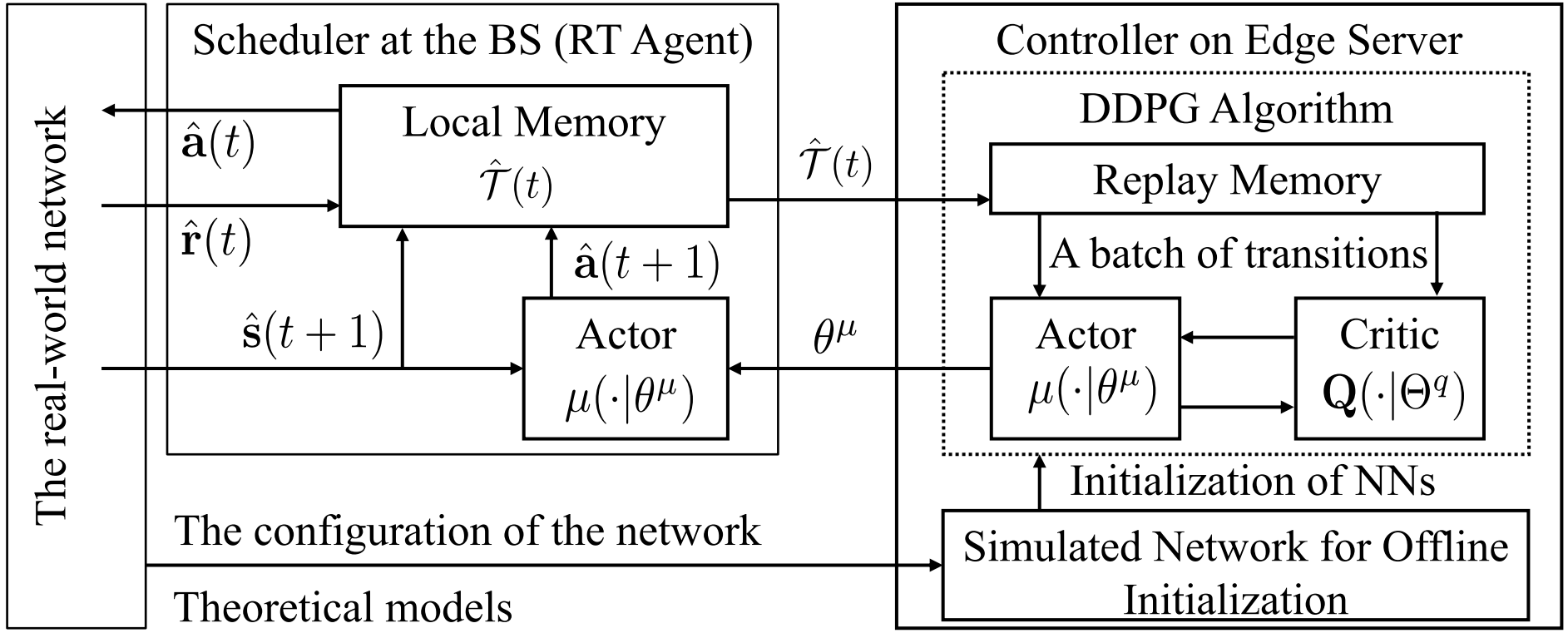
Background on Actor-Critic DRL: Deep Deterministic Policy Gradient (DDPG)



- Loss of critic $L^Q = [Q(s_t, a_t|\theta^Q) - (r_t + \gamma Q(s_{t+1}, \mu(s_{t+1})))]^2$.
- Loss of actor $L^\mu = -Q(s_t, \mu(s_t|\theta^\mu))$.

Note that there are many other actor-critic DRL formulations that are beyond the scope of this presentation.

Overview on Online DRL For Real-Time Scheduler in Radio Access Networks

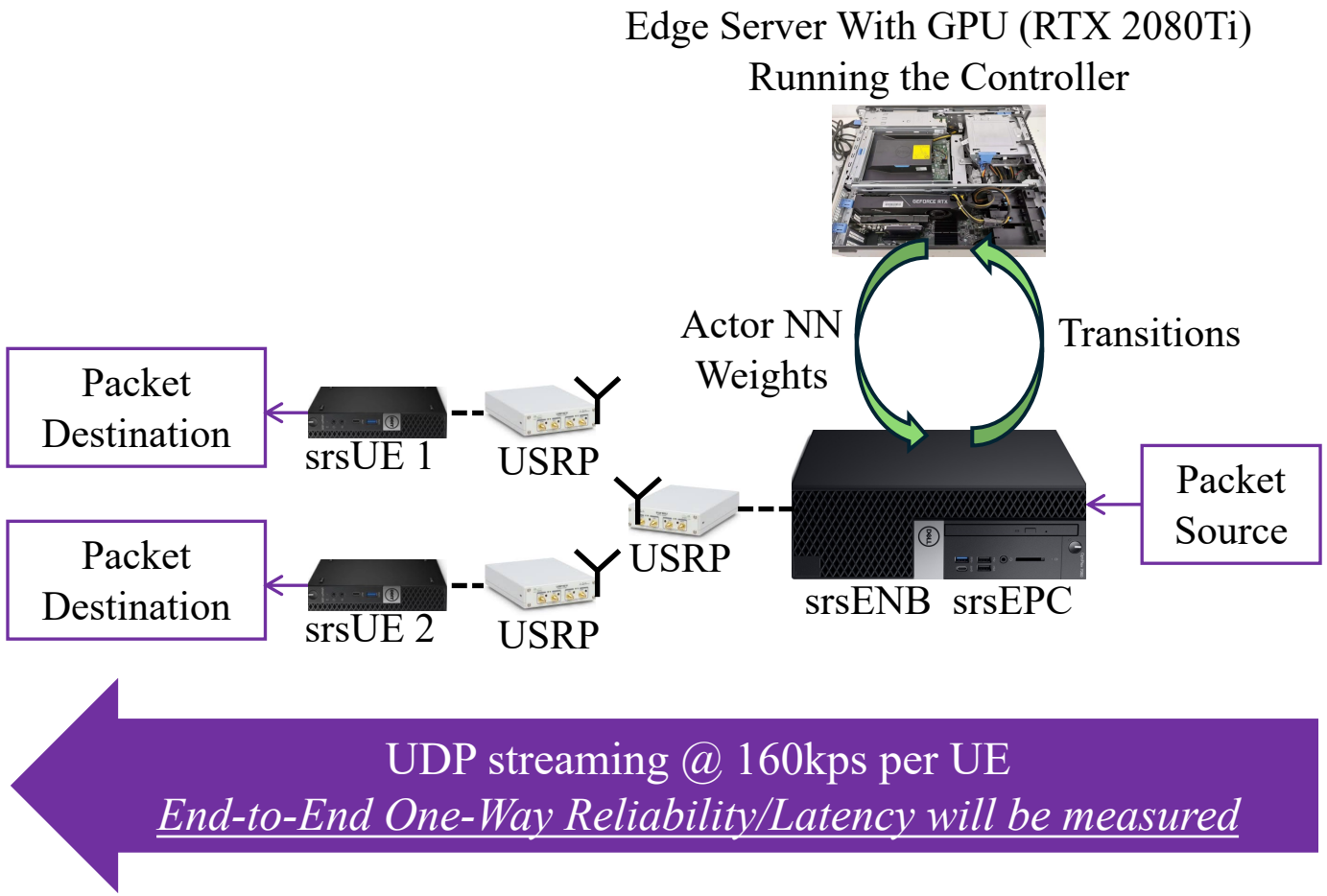


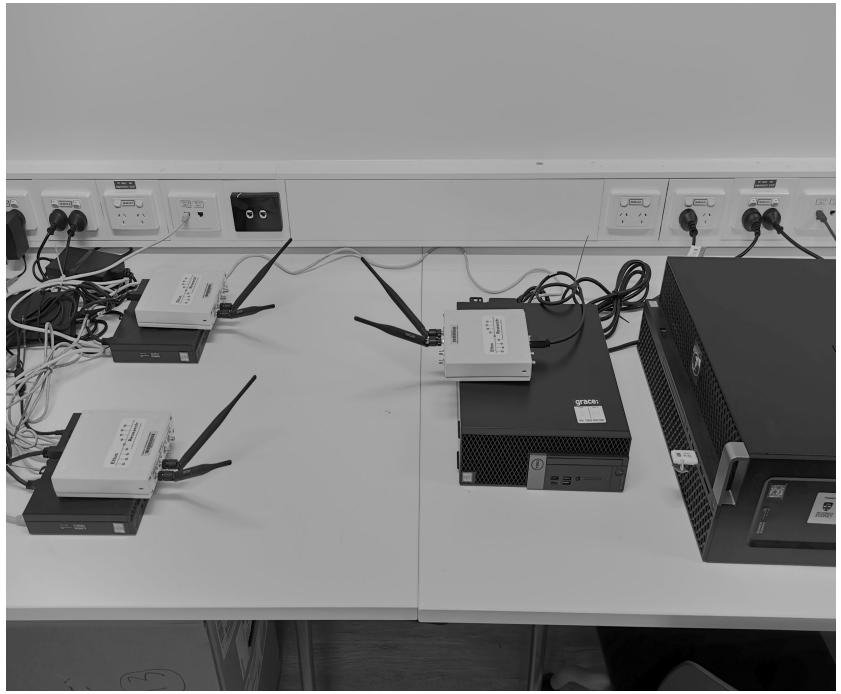
State: Channel Qualities, Head-of-line Queueing Delay;
Action: User transmissions decisions;
Reward: User QoS, e.g., reliability and delay.

Expert knowledge on the system is used to
- Better formulate state, action and reward,
- Design critic structure, transition sampling, reward shaping,

to improve the convergence of the DDPG.
Please refer to the paper for more details,
while they are not the focus of this presentation.

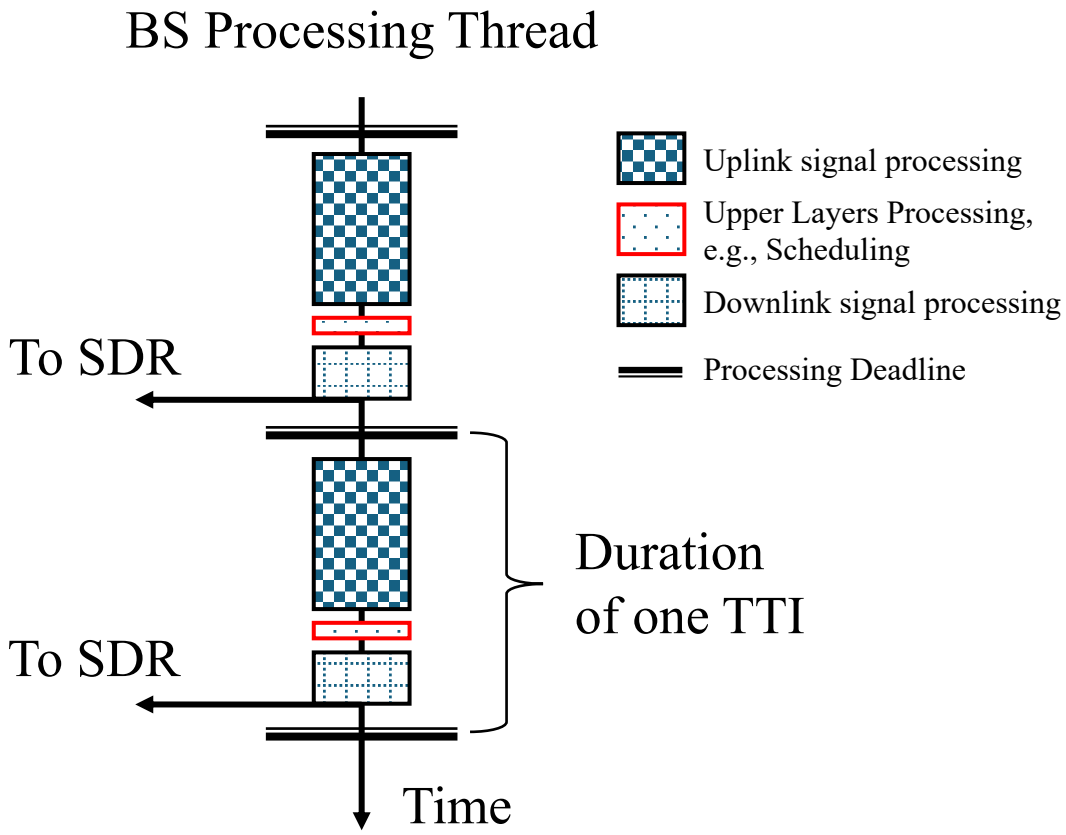
Overview on Real-World Implementation





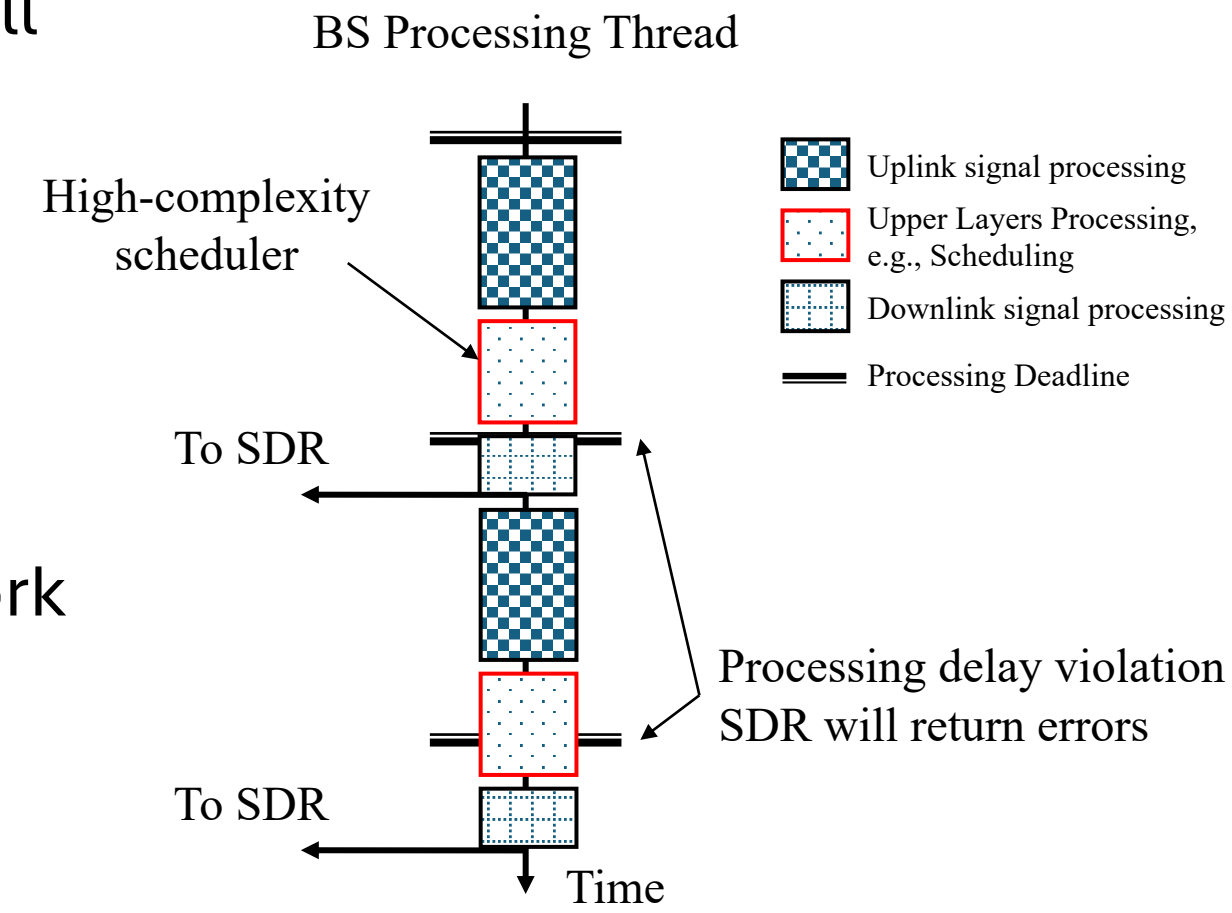
Challenge: Limited Time Budget for NN Inference in Processing Thread of srsRAN

- In each transmission time interval (TTI), a thread in the BS will
  - Decode uplink signals;
  - Process the upper layers,
    - Including MAC scheduling;
  - Encode downlink signals.
- Deadline of processing is
  - To pass the downlink signals on time to the software-defined radio (SDR),
  - So that the signals can be transmitted on time in the corresponding TTI.



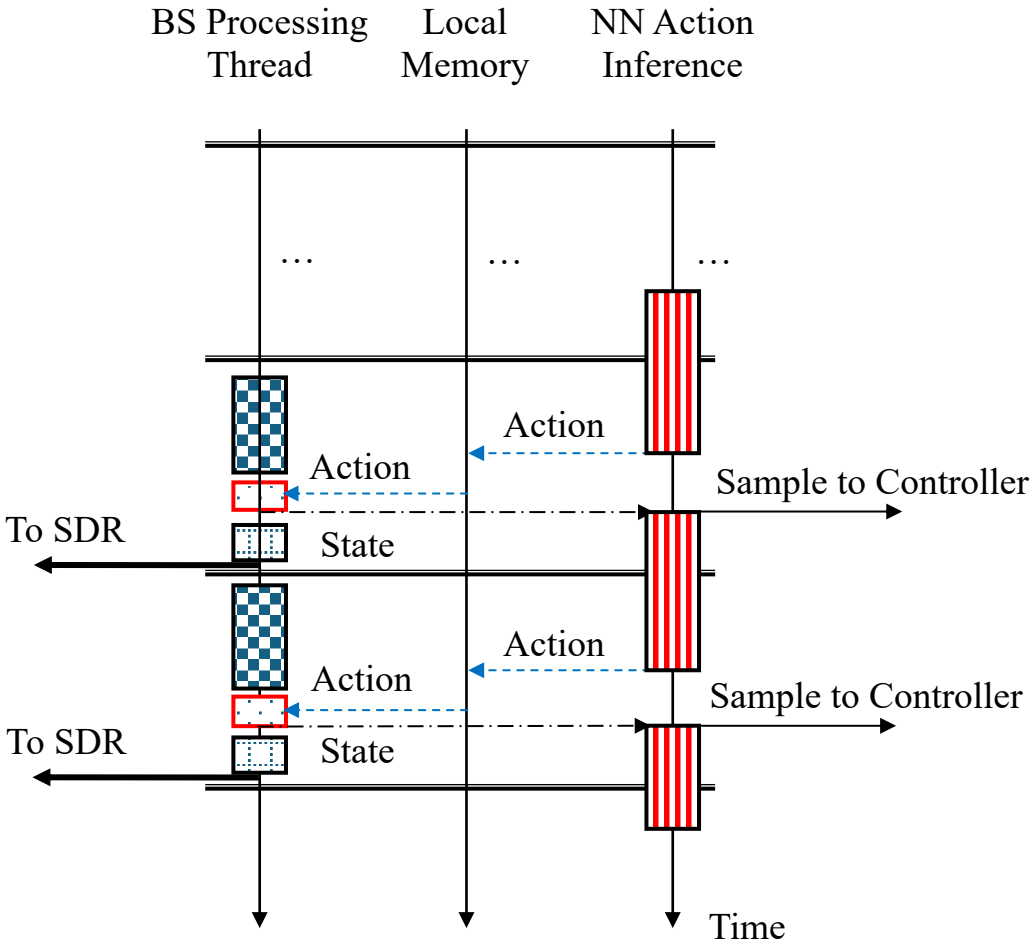
Challenge: Limited Time Budget for NN Inference in Processing Thread of srsRAN

- High complexity schedulers will cause additional time in scheduling,
  - Causing the delayed signal processing before deadline.
- The BS will be out of synchronization and the network will crash.



Asynchronized NN Inference In Parallel

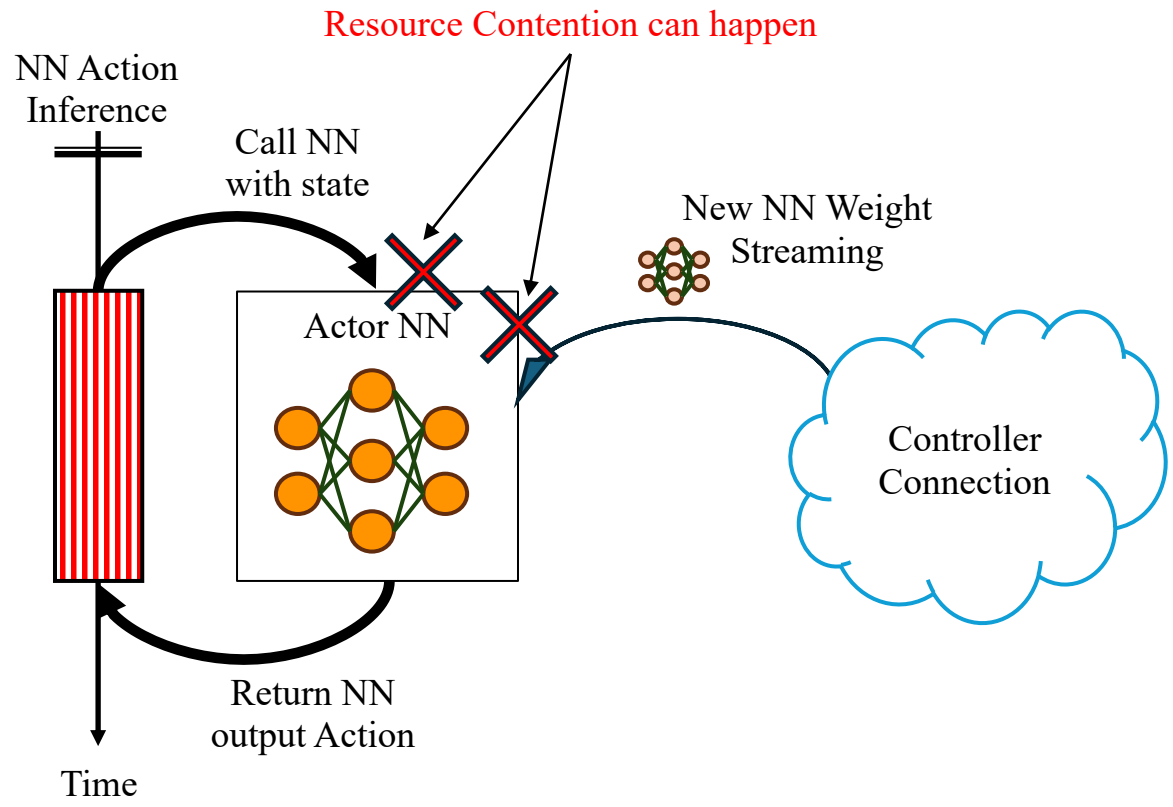
- The NN inference is Implemented in a parallel thread,
  - I.e., an asynchronized agent.
- The agent repeats the following:
  - Takes state passed by the BS thread;
  - Computes the action as NN inference;
  - Saves the action in a local memory.
- The agent's NN inference does not affect the processing of the BS.
- It also manages training sample uploading to the central controller.



Challenge: Thread Safety in Continuous NN Weight Update

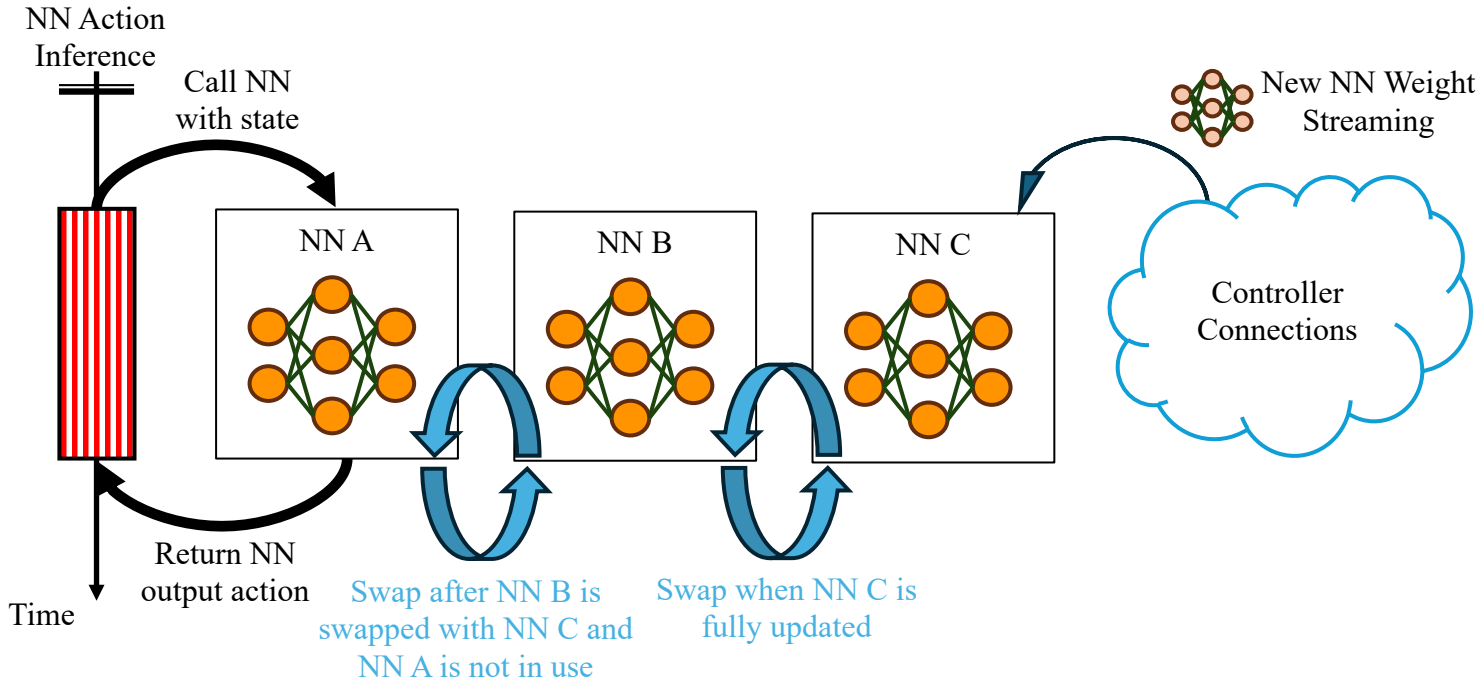
- Resource contention can happen when the NN is used for
  - State-action inference at the BS;
  - NN weight update based on new weights sent by the controller.

- Causing unpredictable behaviors in the system.



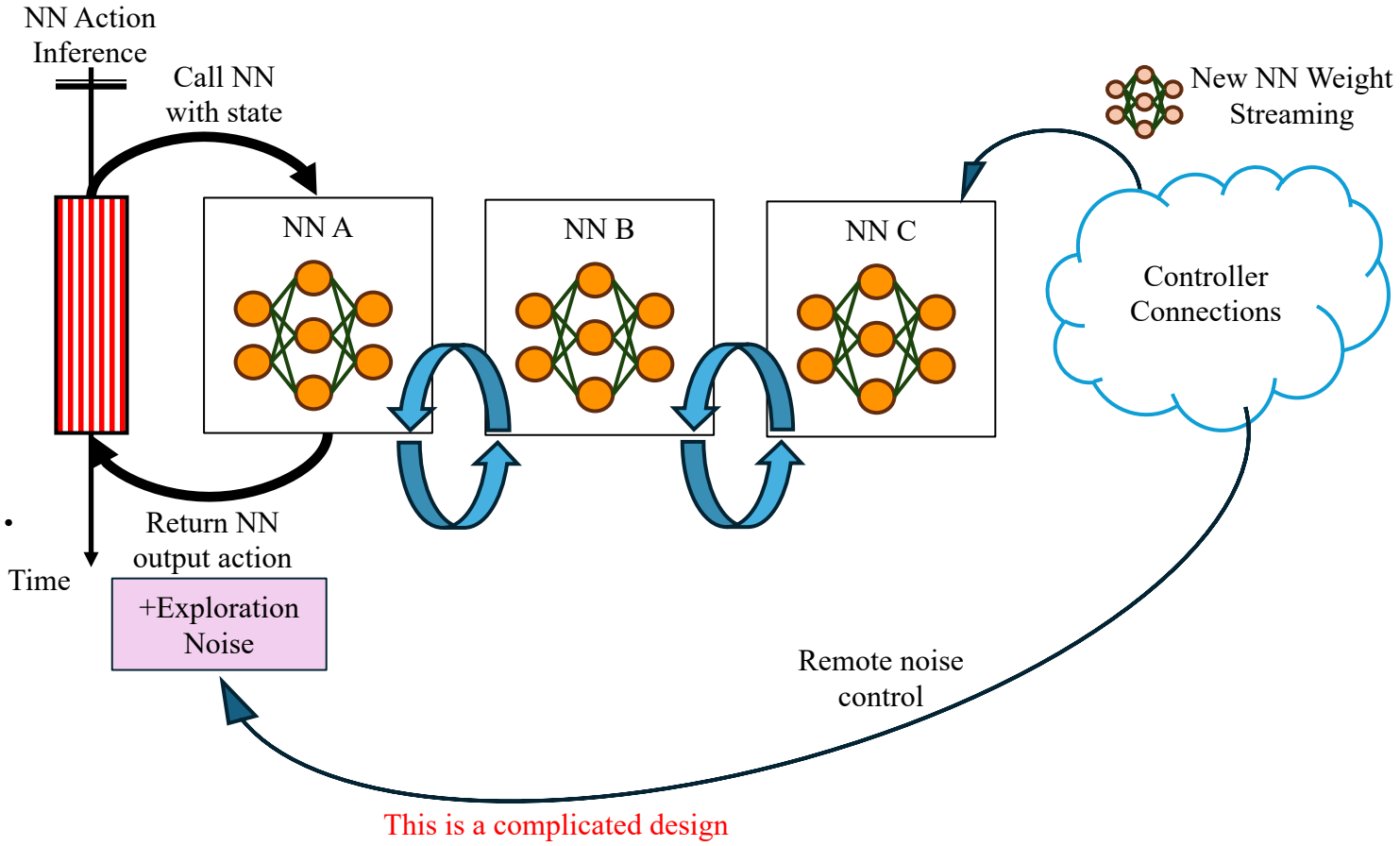
Hot NN Update Using Multiple Copies

- Swap B-C when
  - C is fully updated.
- Swap A-B when
  - B-C is newly swapped;
  - A is not in use.
- Simply swap the NNs' pointers.



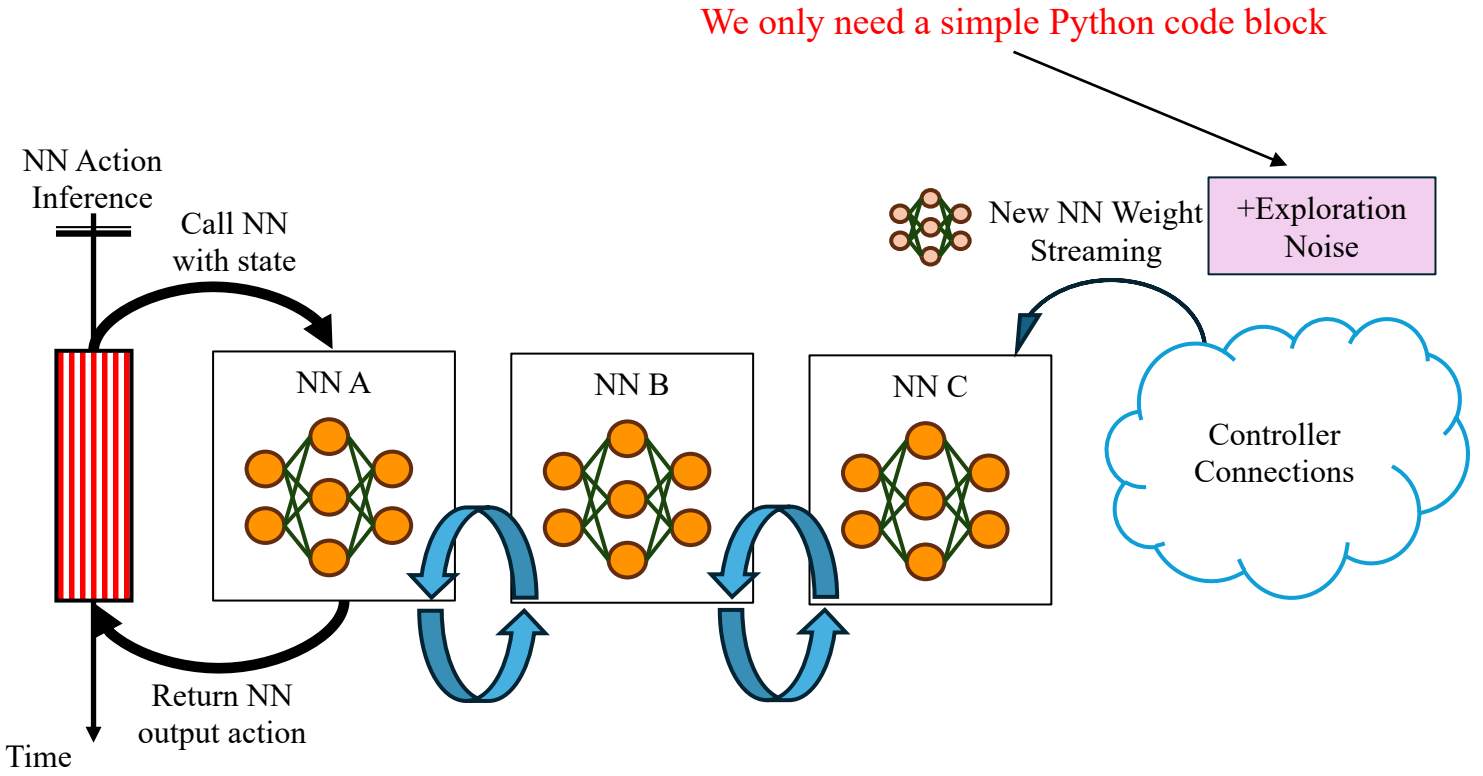
Challenge: Remote Action Exploration Control

- Classic DRL using action space exploration.
- This requires additional implementation, e.g.,
  - exploration mechanisms in the agent;
  - Controller connections in agent to control explorations.
- These are too complicated and not flexible.



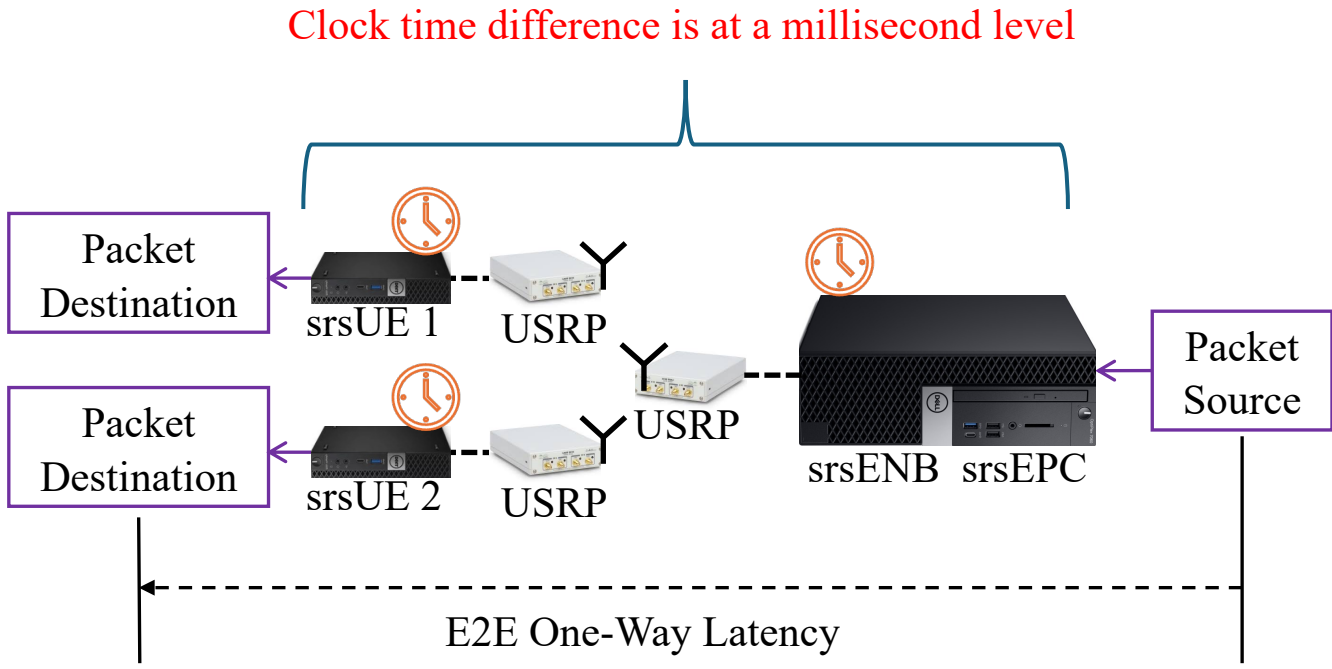
Parameter Space Exploration With Centralized Control

- Simply add noise in parameter space when streaming the weights.
- No need on additional mechanism at the agent and interference between the agent and the controller.
- Noise can be easily controlled in the controller using Python.



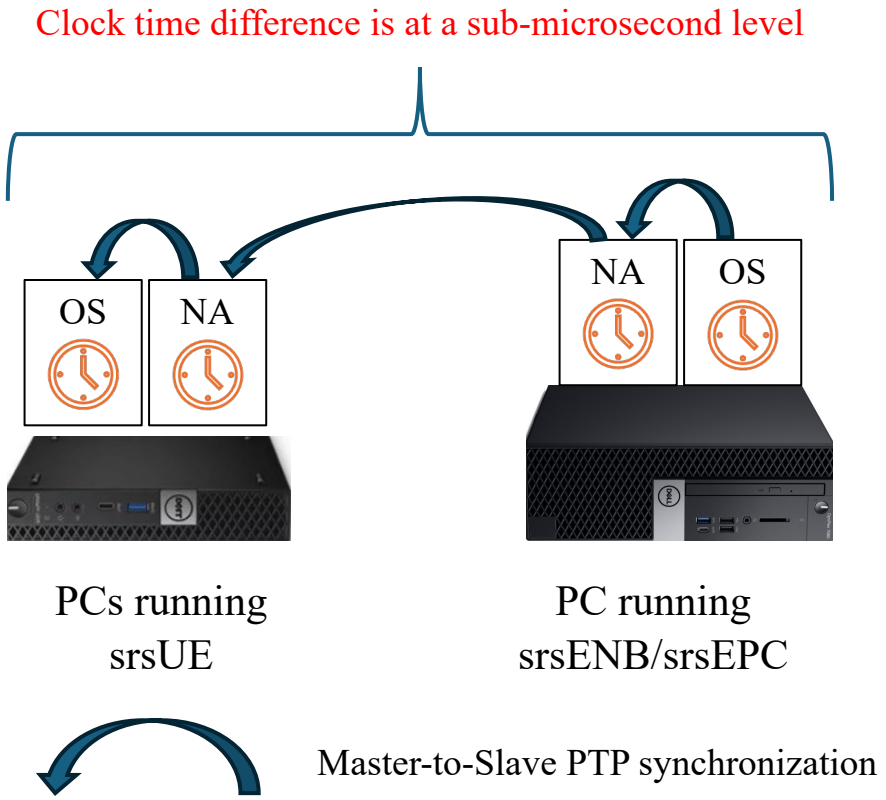
Challenge: One-Way Latency Measurement for Performance Profiling

- Typical computers are network time protocol,
  - With a time synchronization error at few milliseconds.
- With this time error,
  - One can hardly measure the one-way latency accurately,
  - Which is also few milliseconds.
- Higher time synchronization accuracy is needed in computers.



Precision Time Protocol (PTP) Synchronization on Computers in Experiments

- PTP Synchronization is implemented in the following flow
  - PC-srsENB/EPC-Operating System (OS) =>
  - PC-srsENB/EPC-Network Adaptor (NA)=>
  - PC-srsUE-Network Adaptor=>
  - PC-srsUE-Operating System
- The NAs used in the experiments have hardware PTP capabilities.
- Synchronization error < 1 microsecond,
  - With sufficiently high accuracy to one-way delay measurement.



Extensibility/Limitation of Implementations

- The architecture can be easily extended to multiple BS case
  - The current-implementation of central controller can handle multiple agent connections from multiple BSs (not tested though).
- DRL algorithms in the central controller can be easily changed,
  - Without affecting the online architecture.
- Only MLPs with few activation function options can be constructed as the NN at the agent in the BS.
  - Because the NN is initialized at the agent using a hard-coded protocol.
- Real-time NN inference at BS agent is made using LibTorch on CPU,
  - Inference using potentially available GPUs at the BSs are not studied.

Overall Source Code Structure