Using Machine Learning Algorithm to Provide more Robust Congestion Control

Computer Network - Final Project

Yihao Cai - Reza Saadati Fard

Dec 12, 2022

Problem Definition

Q1. Why we need network congestion control (CC) ?

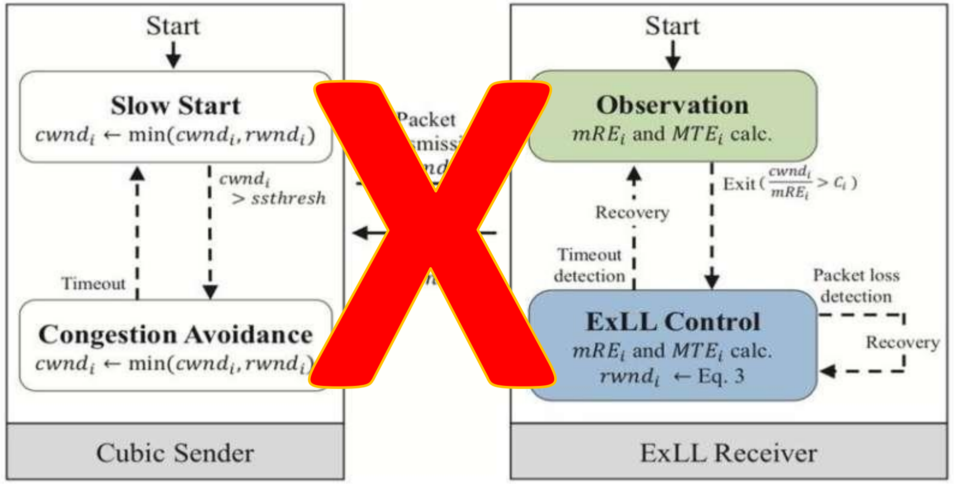
1). Fully leverage network resources (e.g. reduce transmission latency; increase data throughput)

2). Ensure the network QoS (e.g. avoid unnecessary packet-loss and provide reliable service)

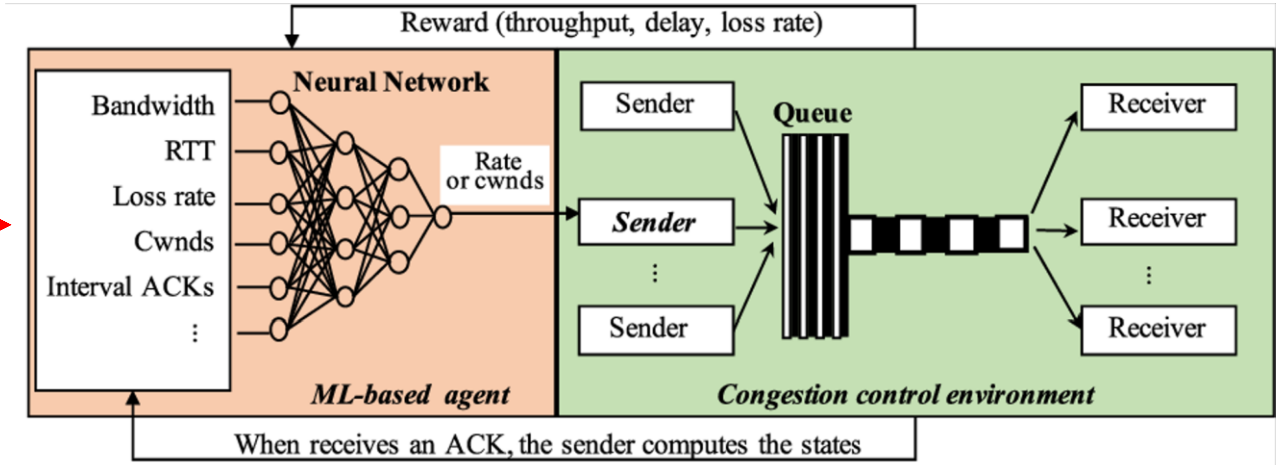
Q2. Why we use Machine-learning (ML) based approach?

Traditional network CC schemes are **rule-based** → Static and Rigid

Machine-learning CC schemes are **data-based** → Dynamic and Flexible



Rule-based CC Approach

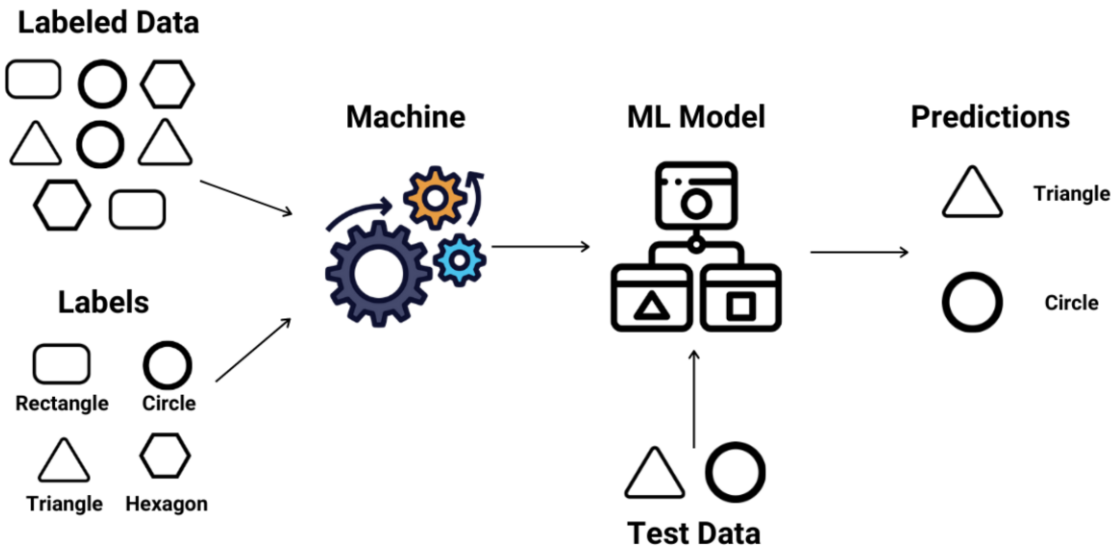


Data-based CC Approach

Introduction – DNN net

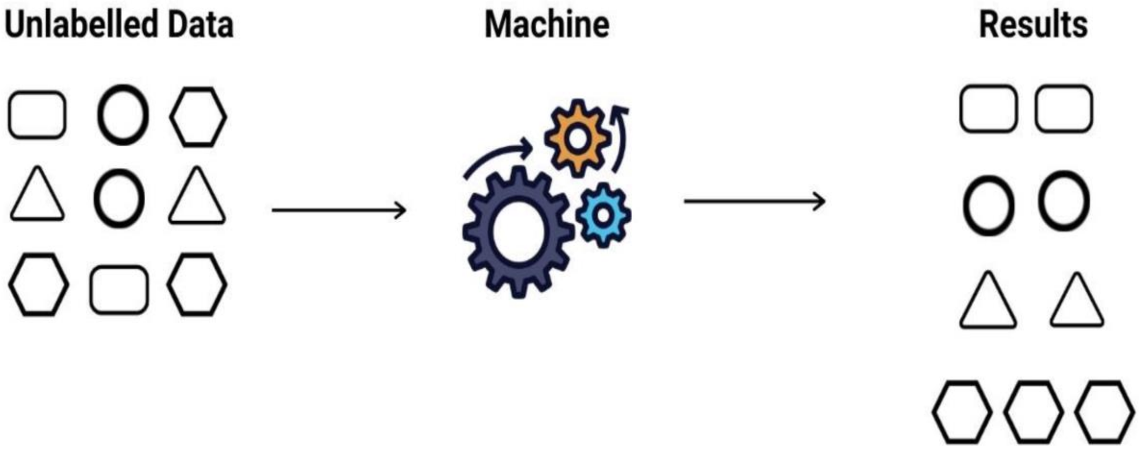
Supervised Learning

Make a mapping between Input and Output as much precise as possible



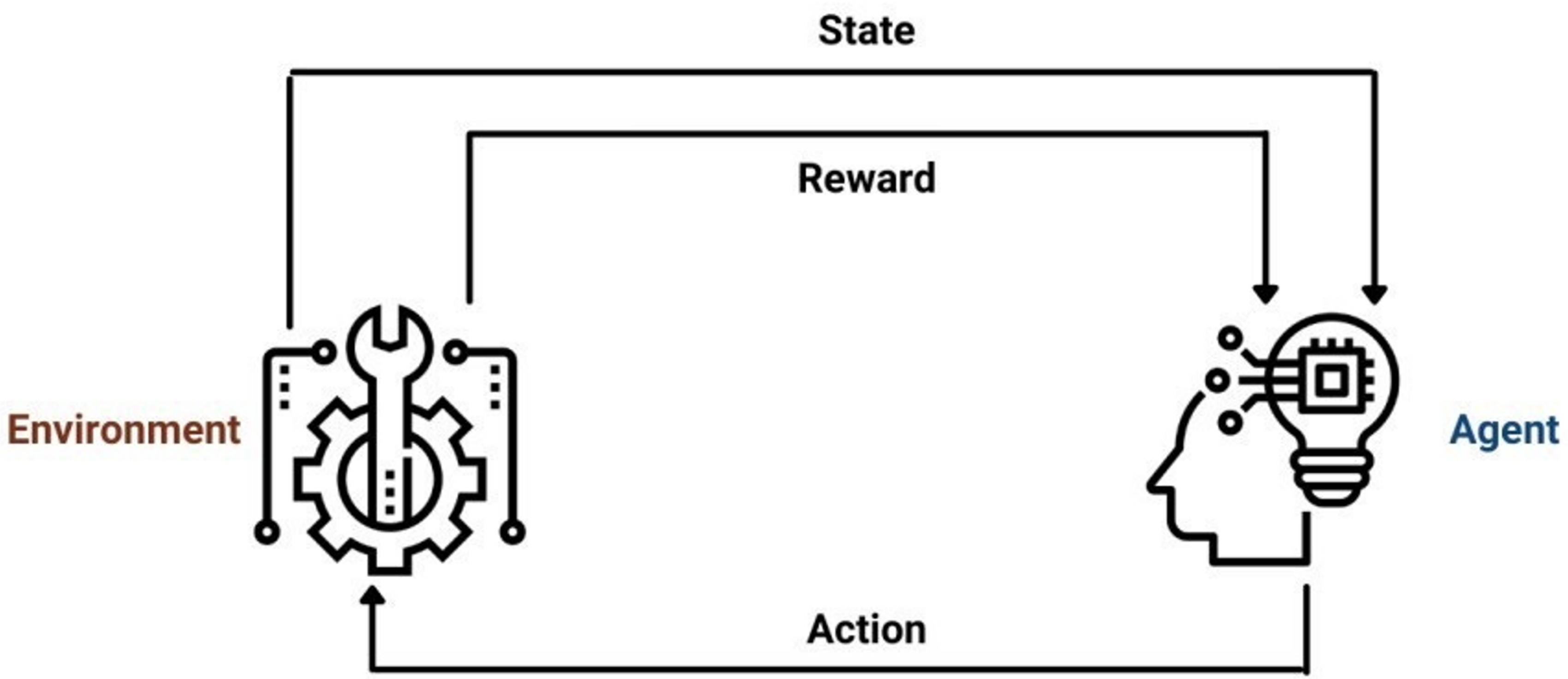
Unsupervised Learning

Classify data with respect to their similarities (diverse dimensionality)



Introduction – DNN net

Reinforcement Learning (RL) → Help agent decide action to take that could help it gain the most rewards during interaction with environment



RL in CC

Why Reinforcement-Learning ?

- Less sample/input data required
- Provide long-term benefit strategy
- Suitable for environment interaction

Proposed Method:

- **Main idea comes from =** *"Classic Meets Modern: a Pragmatic Learning-Based Congestion Control for the Internet" (https://doi.org/10.1145/3387514.3405892)*

- **Project Goal =**

  Build a robust RL-based framework for End-to-end network congestion control

  Extend its availability by introducing various network Topos using **Mininet** interface

RL in CC

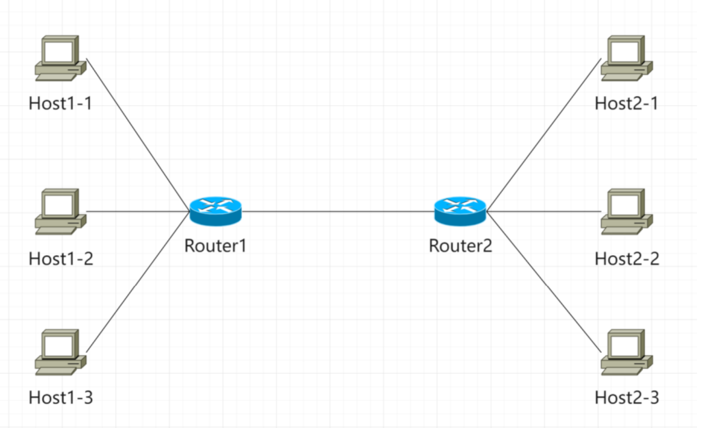
- Evaluation Metrics:

  Network latency
  Throughput
  Robustness
  Fairness
  Real-time Efficiency
  TCP Friendiness (Not penalizing other flows)

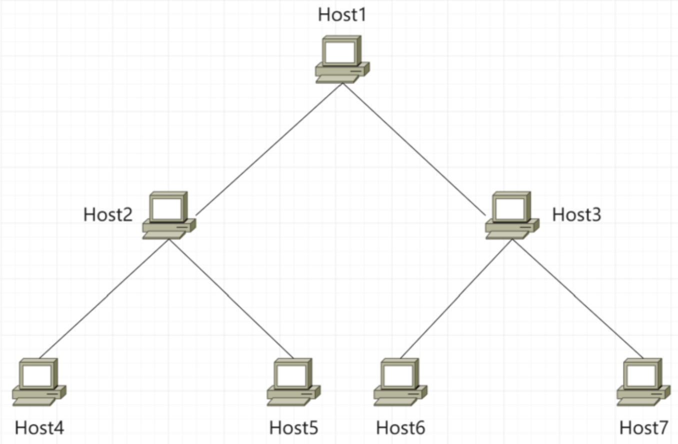
- **Potential Challenges = (Intrinsic RL Problems)**
  - Improper feature/reward function selection would cause bad performance/behavior
  - Large action space makes model hard to converge, impacting the real-time efficiency

RL in CC – Implemented Topo in Mininet

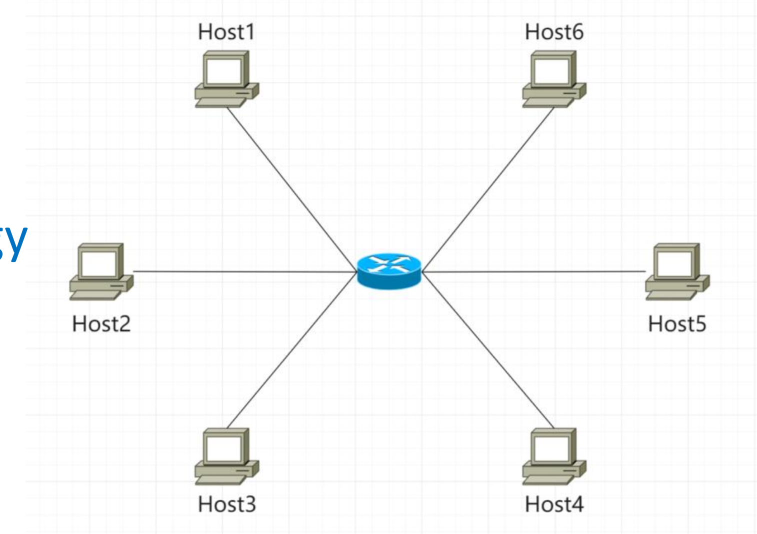
1). Dumbbell Topology



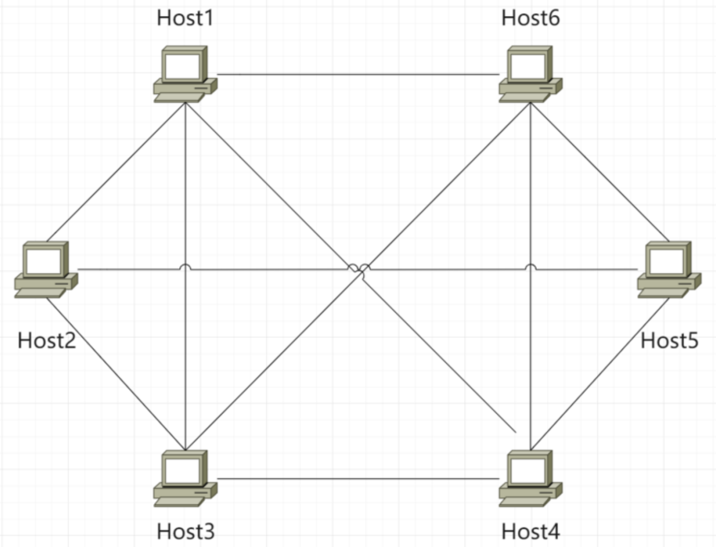
2). Tree Topology



3). Star Topology



4). Mesh Topology



RL in CC – Dumbbell Topo

- **Scenario**

  Web fetching (Inbound/outbound traffic)

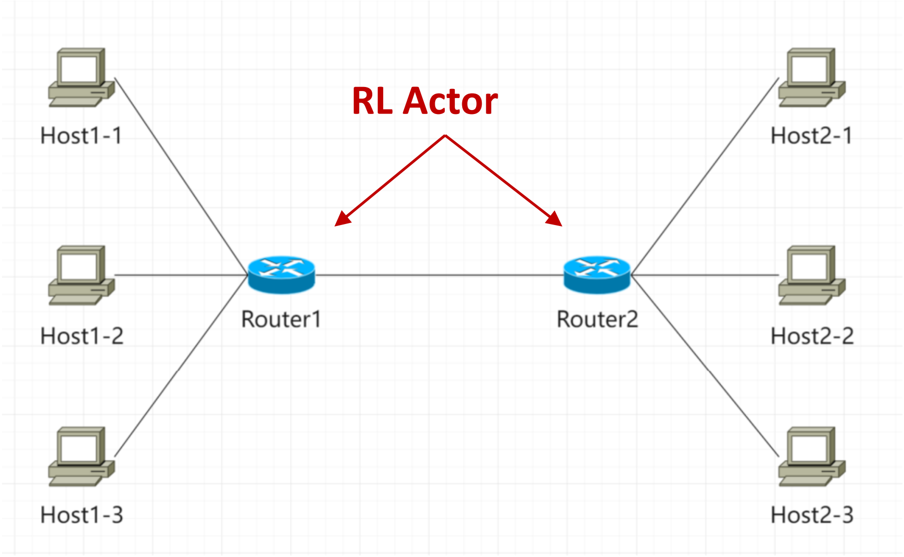
- **Characteristics**

  Commonly-used network topology

  Centralized management and control

- **Experiment**

  make two gateway devices the RL actors to see whether they outperform traditional CC method through some metrics (RTT, Throughput, Fairness, etc.)



Dumbbell Topology

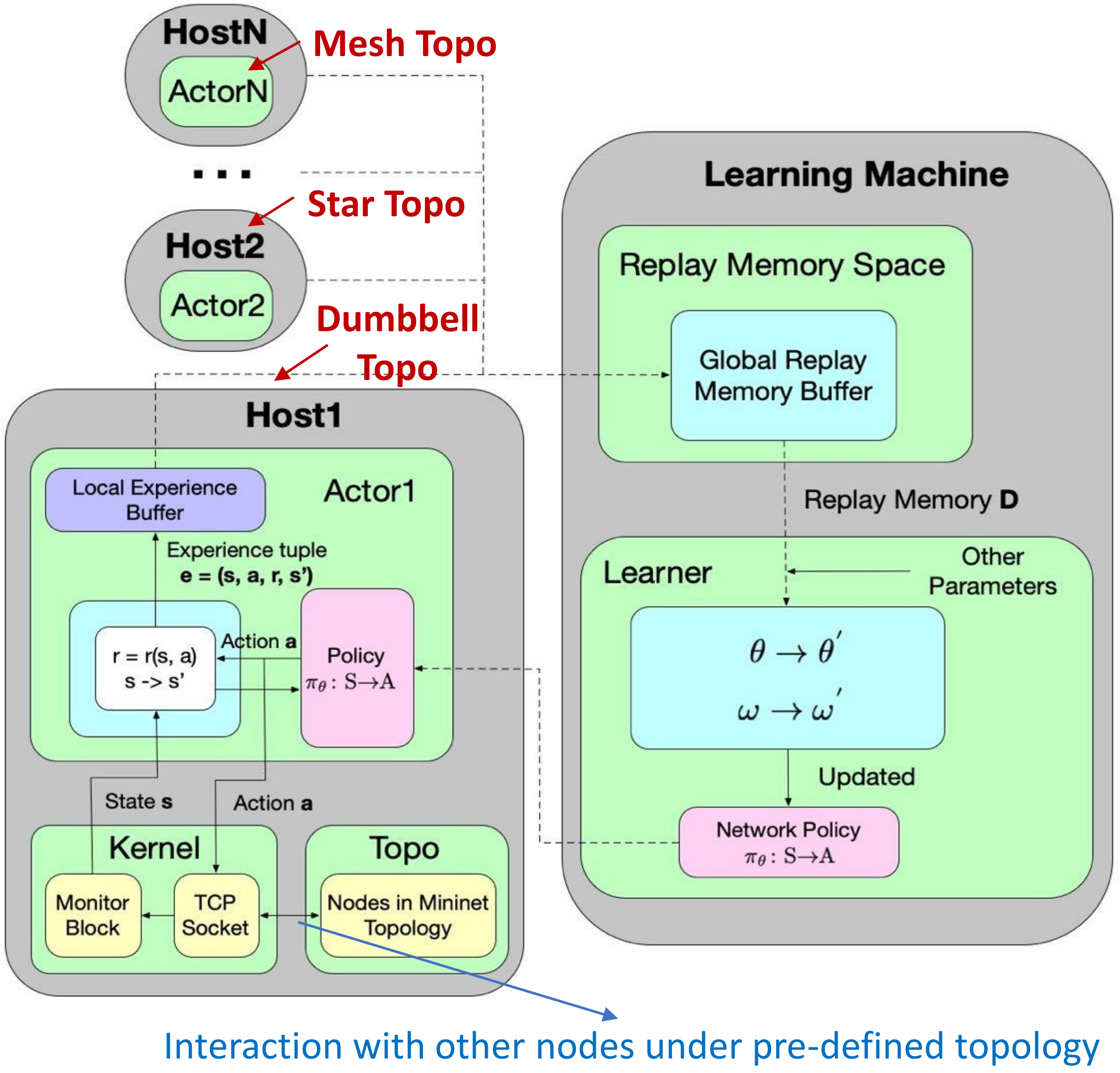
RL in CC – Model Architecture

Host = Actor + Traffic in Kernel

- **Actor =** Action Generator
  - Sync policy from learning agent
  - Update experience tuple **e** (param)
- **Kernel**
  - TCP socket for executing actions (change cwnd/slowing sending rate per time)
  - Monitor block for state observation

Learning Agent = RMS + Learner

- **RMS =** Replay Memory Space
  - Store tuples for all RL-based actors
  - Split data/send to learner for training
- **Learner =**
  - Generate policies for distributed host



Model Evaluation – Train config

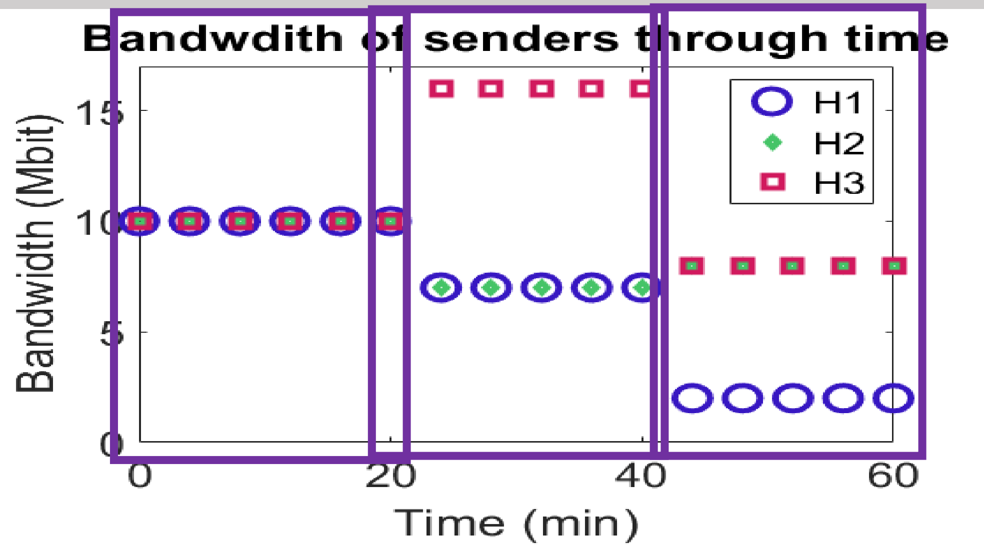
- Training:
  - 5 actors in RL
  - Using Dumbbell topology in the training
- Model Input:
  - Throughput
  - loss packet
  - Delay
  - Number of valid ACK
  - RTT
  - CWND

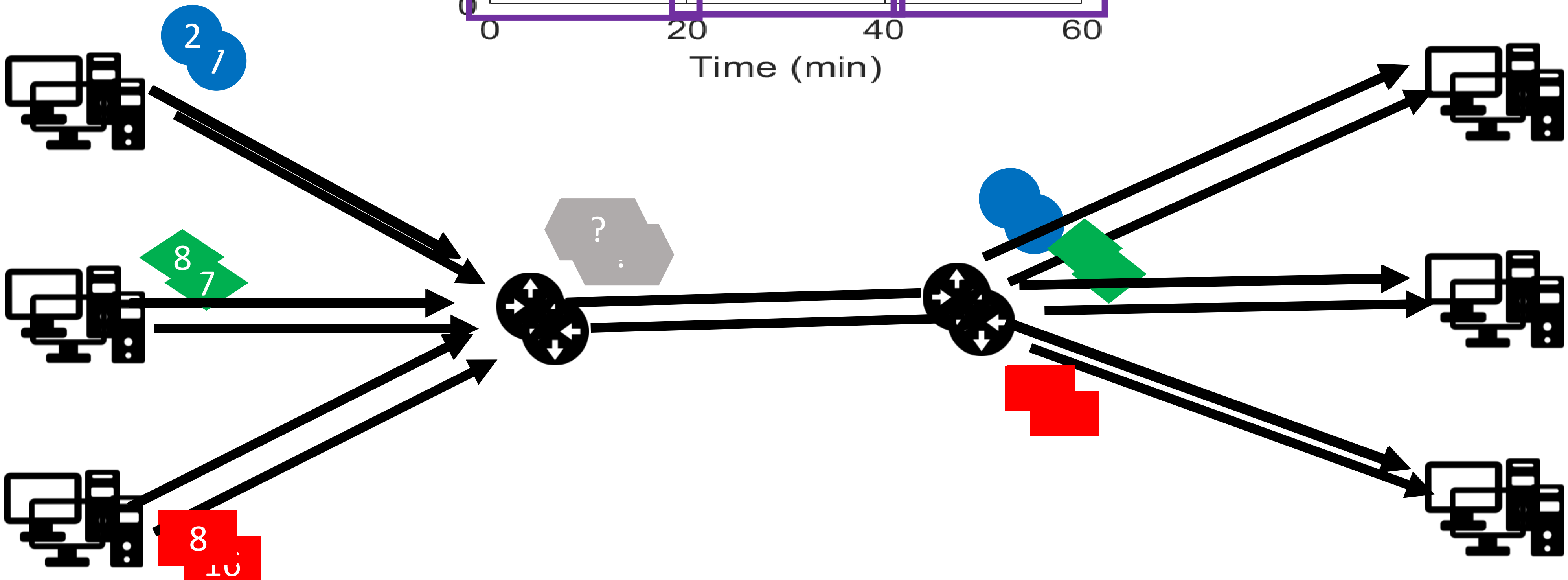
$$R = \sum_i throughput_i - delay_i$$

- Hardware:

OS	CPU	GPU	RAM
Linux 22.0.4	Intel Corei7-7700	Geforce 1060	16 GB

Model Evaluation - Testing Scenario





Model Evaluation - Metric

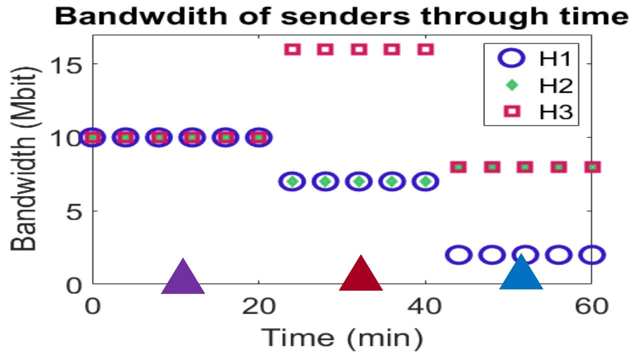
- Three metrics for evaluation:
  1. Bandwidth
  2. Buffer usage
  3. Robustness

- Model buttle neck is 15 Mbit/s
- The worst case situation, senders send 30 Mbit/s

Goal is to maximize bandwidth and in the meanwhile minimize the delay (lower usage of the buffer capacity)

- Underlying method:
  TCP Cubic

Model Evaluation – Results



- Alll results are normalized between 0 and 1 (0 is empty and 1 is full)

- Throughput Results:

	0-20 min Mean±std	20-40 min Mean±std	40-60 min Mean±std	Total Mean±std
TCP Cubic	0.309 ± 0.23	0.352 ± 0.19	0.482± 0.12	0.381±0.18
TCP RL	0.212± 0.461	0.209±0.51	0.482±0.39	0.218± 0.472

- Buffer usage Results:

	0-20 min Mean±std	20-40 min Mean±std	40-60 min Mean±std	Total Mean±std
TCP Cubic	0.981±0.02	0.975 ± 0.015	0.923± 0.061	0.956± 0.032
TCP RL	0.824±0.132	0.761± 0.144	0.729± 0.105	0.771± 0.127

Future Steps

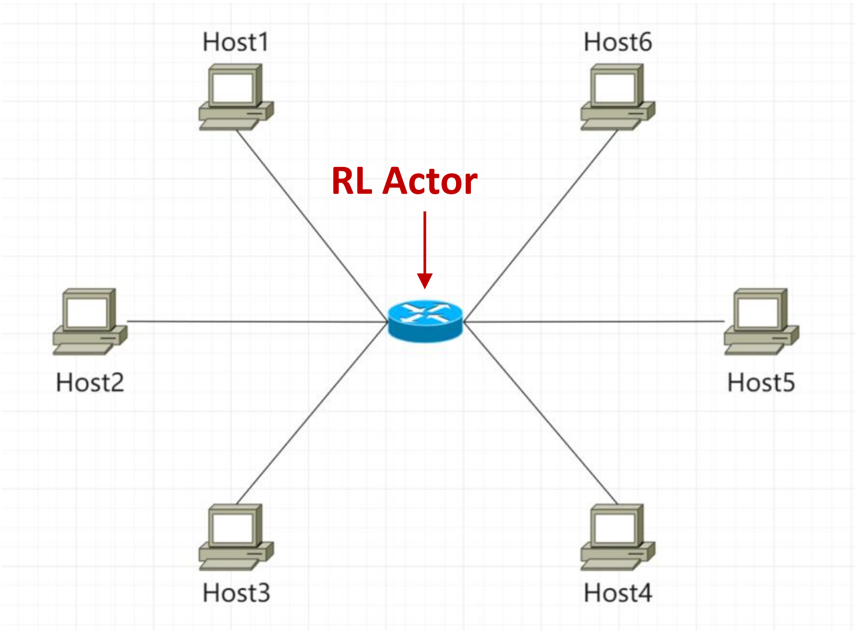
Some of the following suggestions can help to deal with lack of robustness in current RL CC model

1. Benefiting Multi-actor advantages of RL
2. Exploring other Reward Function
3. Hyper-parameter tuning
4. Comparison with other DNN CC methods
5. Using other emulators
6. More sophisticated RL method

Thank You!

Star Topology =

- **<u>Scenario =</u>** Reverse Proxy/Gateway (e.g. Nginx)
- **<u>Advantages =</u>** 1). Easy fault detection; 2). Scalable and extensible; 3). Less expensive(one I/O port per host)
- **<u>Potential Issue =</u>** 1). SPOF (Single Point of Failure) Issue; 2). Network High Concurrency Problem
- **<u>Experiment =</u>** set central node to be the RL actor. High concurrency should be mitigated by cutting down cwnd (and/or sending rate) using RL algorithm



Star Topology