Evaluation of Reinforcement-Learning Queue Management Algorithm for Undersea Acoustic Networks Using ns-3

Peng Zhang, Pedro A. Forero, Daniel Yap, Dusan Radošević
{peng.zhang13.civ; pedro.a.forero.civ; dusan.radosevic.civ}@us.navy.mil
Naval Information Warfare Center, Pacific (NIWC Pacific)

Workshop on ns-3

June 24, 2022
Authors introduction

Mr. Peng Zhang
§ Machine learning engineer and algorithm developer

Mr. Daniel Yap
§ Software and networking engineer

Dr. Pedro Forero
§ Signal Processing, machine learning and communication systems

Mr. Dusan Radojevic
§ Physical layer integration and experimentation lead
Undersea Acoustic Networking

- Characterized by limited and variable bandwidth availability
- Need to provide quality-of-service (QoS) guarantees to different traffic types
- Queue management enables 'shaping' traffic based on QoS requirements and bandwidth availability

Some known queue management policies:

§ Priority scheduling
  - Does not guarantee that all traffic types are served

§ Fair queuing (FQ) scheduling
  - Suboptimal since it does not adapt to the data traffic requirements and dynamics

§ Weighted-fair queuing (WFQ)
  - Using fixed weights can be problematic as the traffic pattern and the state of the links change
Our approach

▼Learn WFQ weights via reinforcement learning (RL)

§ Agent acquires state information and acts on the environment according to RL-based policy at each decision epoch.
  - Point in time when agent makes decision.

§ RL agent seeks a policy to select WFQ weights that maximizes network throughput and minimize packet delay.

States (S):
- queue size
- queue size delta
- violation (\(v\))
- # of marked packets (\(\lambda\))
- mean delay
- mean delay delta

Action:
- WFQ weights (\(w\))

\[
r^{(i+1)} = \sum_{d=1}^{D} \kappa_d \left[ C_0 w_d^{(i)} B^{(i)} - C_1 1 \left( |v_d^{(i)}| > 0 \right) - C_2 \lambda_d^{(i)} \right]
\]

where \(\kappa_d\) is the priority of the d-th queue, \(B^{(i)}\) is total bandwidth, \(C_0, C_1,\) and \(C_2\) are hyperparameters.
Our approach (Cont.)

▼RL agent is trained using OpenAI Gym environment in Python

§ Compatible with external training algorithm libraries, such as RLLlib and Stable-Baselines 3
§ Trained with Soft Actor-Critic (SAC) algorithm

▼The WFQ policy is deployed using ns3gym

§ Enables communications between ns-3 simulation and Python application
WFQ module in ns-3

We developed two modules

§ WfqQueueDisc - Enqueue(), Dequeue()
  - Modified CoDel (mCoDel) is used as child class for congestion identification

§ ProtocolPacketFilter - Classify()
Aquasim-NG is an ns-3 library that provides several underwater acoustic networking protocols and acoustic propagation models.

Packets start with source node application and are consumed in destination routing layer.

WFQ queuedisc in TrafficControlLayer dequeues a portion of packet based on queue weight allocation at each decision epoch.

mCoDel (as part of WFQ queuedisc) marks packets that may cause delay and notifies application to reduce data rate.

Figure 3: ns-3 node structure
WFQ agent training

Trained in OpenAI Gym environment

- Parameters are identical to ns-3 simulation
  - Data rate
  - Delay requirement
  - Queue priority
  - Modified CoDel parameters

- Trained with Stable-Baselines3 SAC algorithm with entropy term $\alpha = 20$

- Used 4 hidden layers with 64 neurons per layer for actor network and 128 neurons per layer for critic network

Figure 4: Training result. Shaded region represents 95% confidence interval
Model deployment

▼ WFQ policy model is loaded in Python
▼ We use ns3gym library to connect the WFQ policy to the ns-3 simulation, enabling data exchange
  § ns-3 callback in C++
  § OpenAI Gym API in Python
▼ WfqQueueDisc public functions to set weight and obtain state information for WFQ policy
▼ Action synchronization to avoid dequeuing more packets than policy assigns
  § ReleasePackets()
    - Calls Wake() in NetDeviceQueue
  § StopPackets()
    - Calls Stop() in NetDeviceQueue

Figure 5: ns-3 and OpenAI Gym interaction
Queue management tests

- Conducted 10 tests
  - each test runs 200 decision epochs, or 1100 simulation second with 5.5 seconds per decision epoch interval

- 2-node setup
  - Source node with 3 OnOffApplication - 800 bps, 1600 bps, 2400 bps
    - Each application corresponds to one type of traffic
  - OnOffApplication always on (OnTime = 1.0)

- Three queue management strategies
  - Static WFQ algorithm
  - Random WFQ algorithm
  - Dynamic WFQ algorithm
    - Proportional to queue size at each decision epoch
  - Our RL WFQ algorithm
Queue management tests

Figure 6: RL agent evaluation against three other WFQ policies: random action (top right), static weights (bottom left), and dynamic weights (bottom right) in each subfigure.
Summary

▼Developed RL-based algorithm that maximizes queue data throughput while satisfying QoS requirements
  § State: queue size, mean queue delay, bandwidth, etc
  § Action: WFQ weights
▼Learned policy for dynamically choosing WFQ policy weights
  § Update actor network through SAC
▼Developed new ns-3 modules: WfqQueueDisc, ProtocolPacketFilter
▼ns3gym enables data exchange between RL model in Python and ns-3 simulation in C++
▼Plan to train RL agent directly in ns-3
Questions?