TOWARDS AI-ACCELERATED VOLUMETRIC HUMAN TELEPRESENCE

by

Ezra Brooks

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Computer Science

Charlotte

2023

Approved by:

____
Dr. Pu Wang

____
Dr. Minwoo Lee

____
Dr. Weichao Wang

ABSTRACT

EZRA BROOKS. Towards AI-Accelerated Volumetric Human Telepresence. (Under the direction of DR. PU WANG)

Real-Time video solutions such as volumetric human telepresence require a degree of quality than cannot be guaranteed by existing solutions. In order to implement reinforcement learning in this context, programmable platforms are required for fast prototyping, evaluation and testing. We present a platform that can be used to develop and test more robust models with performance parameters that closely resemble real-world scenarios. We explore the components required to make such a platform viable, and we demonstrate the performance of the overall system.

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Dr. Pu Wang, for his guidance and support throughout my master's program. I would also like to thank Dr. Minwoo Lee and Dr. Weichao Wang for serving on my thesis committee and providing helpful feedback. I am grateful to Intel for providing me with the opportunity to conduct my research at UNCC. I would also like to thank Pinyarash Pinyoanuntapong, Houston Huff, and Ayman Ali for their guidance and support. Finally, I would like to thank Jessica Schoenherr for her love and support during this process. Without her, this would not have been possible.

TABLE OF CONTENTS

List of figures

Chapter 1: Introduction

1.1 Motivation

Current research in real-time video performance suggests that reinforcement learning can be used to improve video quality of experience. [1][2][3] In order to develop reinforcement learning in the context of real-time video it is necessary to evaluate models embedded in a network environment that mimics real-world scenarios. To achieve this goal, we present a reconfigurable framework for fast model development and evaluation that combines real-time video, reinforcement learning, network architecture, and simulation of real-world network congestion.

The motivation for this work comes out of the larger project of real-time human telepresence. The success of this technology depends on improvements in real-time video performance. This framework provides functionality to evaluate RL models in a real-world scenario, so that they can be incorporated into a larger real-time human telepresence system.

1.2 Proposed Solution

RL-WebRTC is a framework composed of multiple components that act together in a unified pipeline. The first component provides the environment for developing reinforcement learning (RL) models that can be used for improving performance in real-time video transmission. Specifically, Stable-Baselines3 implements state-of-the-art reinforcement learning algorithms such as Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradient (DDPG). [4] Other custom algorithms can also be implemented as needed.

The second component implements the RL algorithm in the context of a network environment. The AI-Gym toolkit provides a training environment suitable for use in a real-time video application, and the algorithm is implemented in the context of WebRTC, an open-source framework for real-time communication. This process allows for customized algorithms, and it is a well-documented and widely used standard. [5]

The third component provides the environment for deploying the application in various platforms such as NS3 (simulation), Mininet (emulation) and a testbed (real world implementation). Together, these components provide the full pipeline from algorithm development to training and deployment, that can be used to implement reinforcement learning within a real-time video application. The framework is flexible in its configuration and can be implemented in a variety of platforms and contexts.

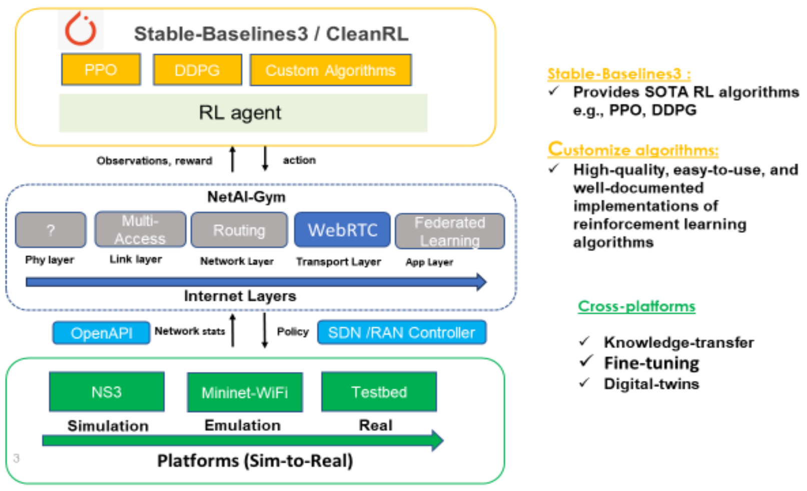


Figure 1. RL-WebRTC components

Chapter 2: Background

2.1 WebRTC

WebRTC is the gold standard for real-time peer-to-peer networking. It supports Real-Time Transport Protocol (RTP) and uses server configuration to isolates tasks such as establishing connections and media streaming. In addition, it is open source and supported by all major platforms. [5] As a widely accepted and well-documented framework, WebRTC is a strong candidate to provide core functionality in any real-time video application.

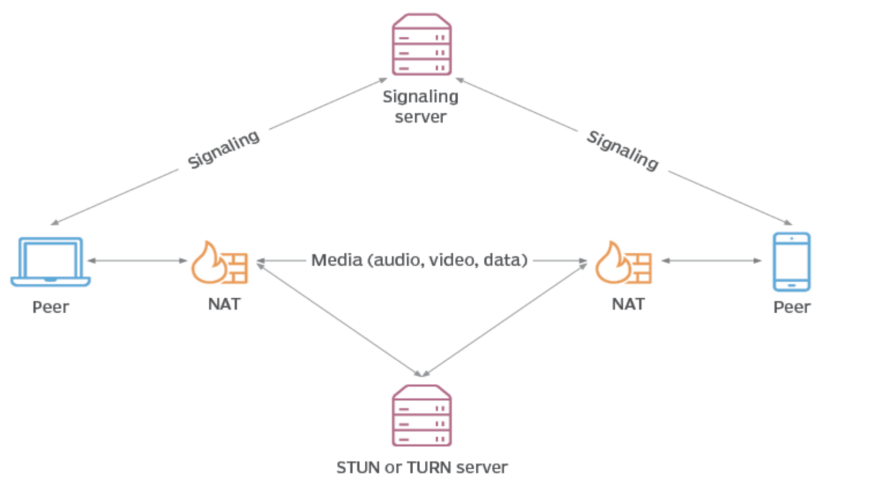


Figure 2. WebRTC server architecture [5]

2.2 Google Congestion Control (GCC)

WebRTC implements the Google Congestion Control (GCC) algorithm, which was designed to work with RTP/RTCP protocols and is based on the idea of using delay gradient to infer congestion. This algorithm is heuristic based and provides a set of rules to govern bitrate based on measured delay. [6] While WebRTC is the current standard, current research suggests that the performance of WebRTC could be improved with the inclusion of reinforcement learning to augment or even replace the rule-based approach taken by GCC.

$$A_r(t_i) = \begin{cases} \eta A_r(t_{i-1}) & \sigma\text{=Increase} \\ \alpha R_r(t_i) & \sigma\text{=Decrease} \\ A_r(t_{i-1}) & \sigma\text{=Hold} \end{cases}$$

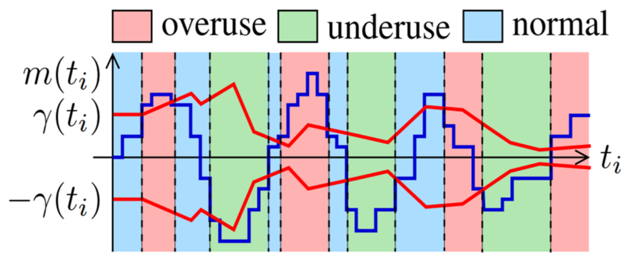


Figure 3. GCC heuristic rules [6]

2.3 Hybrid Receiver-Side Congestion Control (HRCC)

HRCC is a framework that adapts WebRTC and adds a reinforcement learning component. The HRCC framework combines a heuristic congestion control scheme with an RL agent to dynamically tune the values of GCC parameters depending on the network variability. This framework demonstrates a creative approach to RL-based real-time video, and it promises improved performance over the original WebRTC framework. [7]

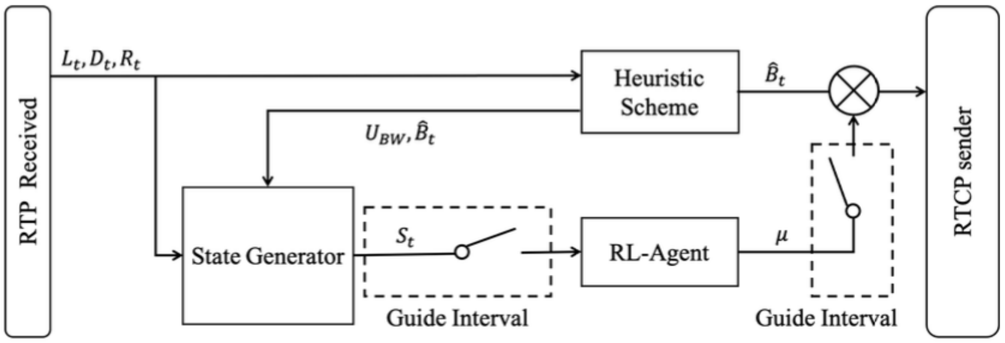


Figure 4. HRCC decision model [7]

2.4 Gemini

Gemini is another recent project that also takes a unique approach to improving WebRTC performance. This framework provides a hybrid model that switches between GCC and an RL algorithm. [8] Unlike HRCC, which focuses on GCC, Gemini addresses the overall WebRTC architecture, establishing logic to govern the choice between a rule-based and learning-based approach.

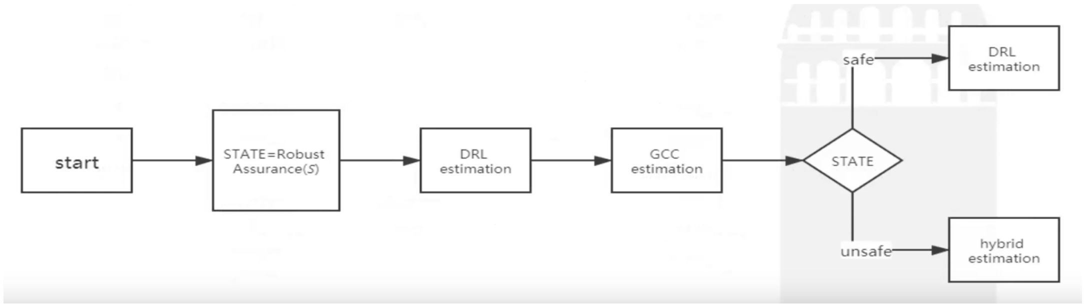


Figure 5. Gemini decision model [8]

2.5 Bang-on-Bandwidth (BoB)

The Bang-on-Bandwidth (BoB) framework builds on Gemini and HRCC to provide an alternate approach to implementing reinforcement learning with WebRTC. [9] BoB starts with a heuristic-based approach, and then switches to a learning-based approach. BoB promises improved performance over both Gemini and HRCC. In addition to providing a new approach to real-time video performance, BoB adds a configuration layer on top of WebRTC to run multiple models, as well as integrated scripts for visualizing the results. The project is especially useful for research since it provides an open-source implementation for public testing.

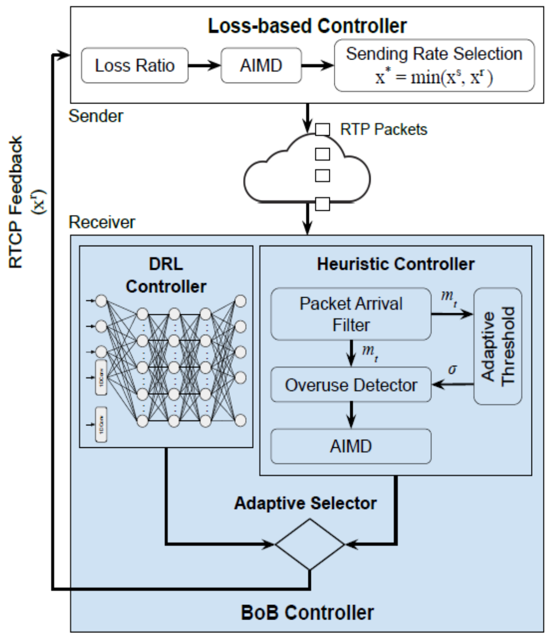


Figure 6. BoB architecture [9]

Chapter 3: RL-WebRTC

3.1 RL-Based Adaptive Rate Control

While traditional WebRTC uses a heuristic method to control bitrate, the goal of RL-WebRTC is to implement adaptive rate control using reinforcement learning in order to achieve the optimal encoding rate for higher quality of service (QoS) requirements. An algorithm such as PPO can be trained on network criteria such as packet round trip time (RTT), jitter, bitrate, and packet loss. It can then be implemented in the application layer as part of a WebRTC-based framework.

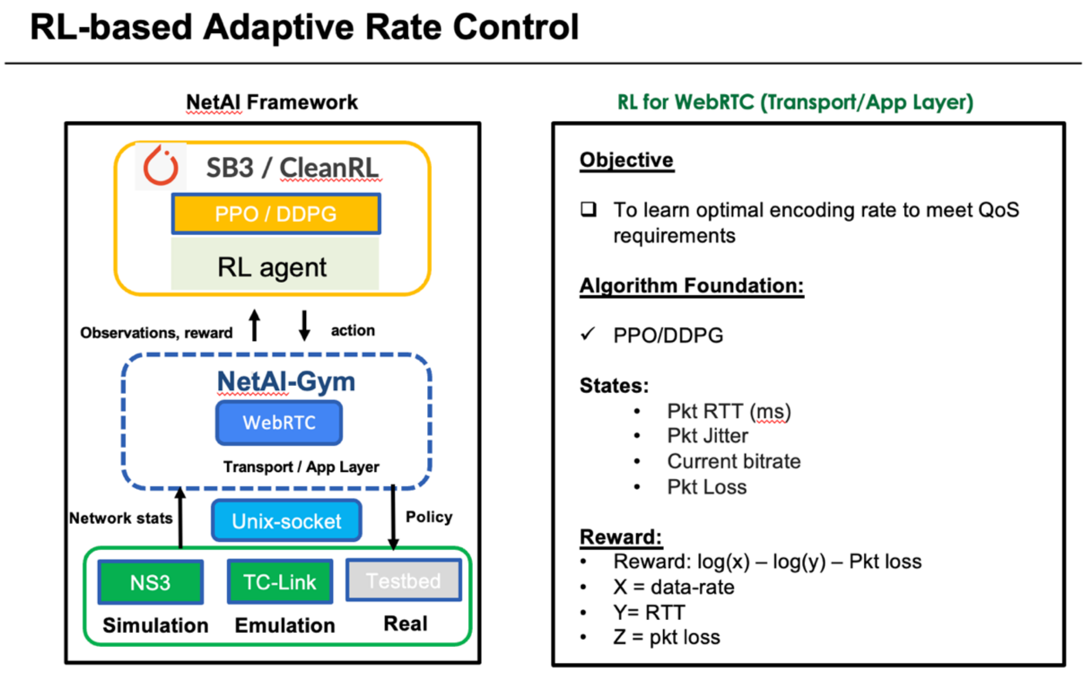


Figure 7. RL-based adaptive rate control training module

3.2 RL-WebRTC Pipeline

The RL-WebRTC pipeline consists of training, evaluation, and emulation components. The training component uses real-life network traces for state observations, and the WebRTC Gym environment for NS3 simulation. The model can then be evaluated by running it in an application based on AlphaRTC, an open-source version of WebRTC that provides an interface for custom built RL models. [10] Finally, the RL-WebRTC emulator provides a network environment where the application runs. The network configurations can be modified if needed, but a user can also simply train and run a model in a network scenario without needing to focus on the implementation details. In this way, more focus can be spent on improving the algorithm performance, rather than designing and managing a test environment.

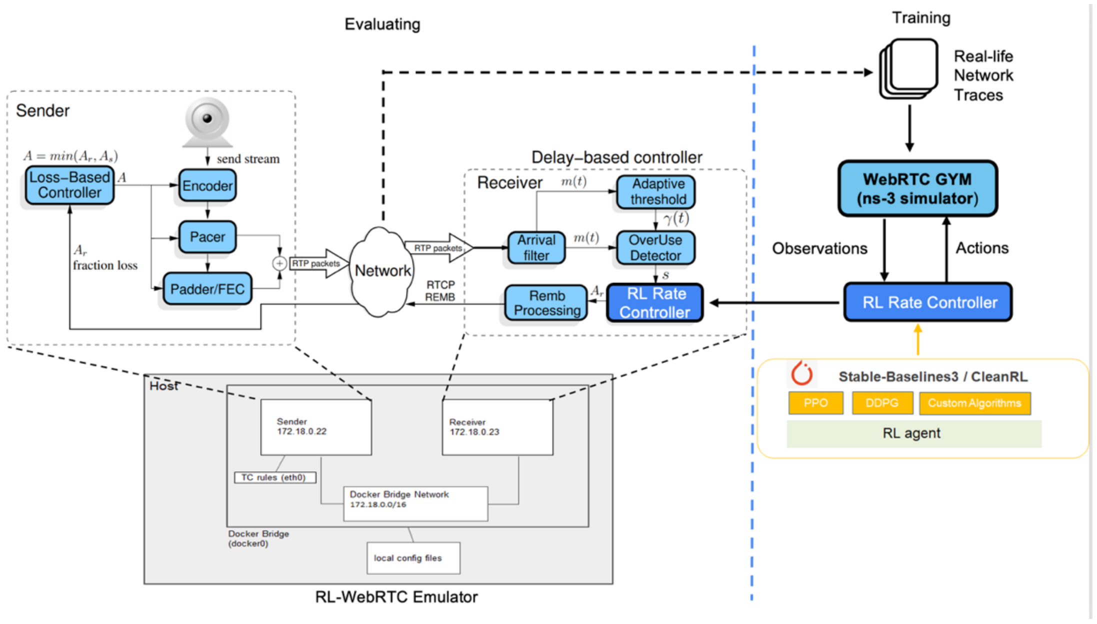


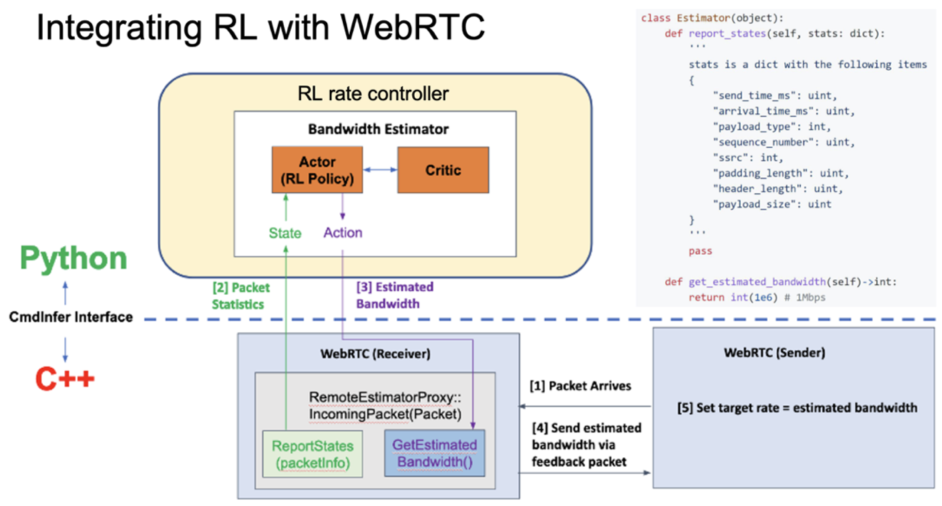
Figure 8. RL-WebRTC architecture

3.3 Reinforcement Learning

There are two main approaches used for implementing the RL component of RL-WebRTC. CleanRL is a high-quality single file implementation of deep reinforcement learning algorithms. [11] It is easy to extend and customize and it implements everything in a single file. Stable-Baseline3 (SB3) is less flexible, but it provides more state-of-the-art RL algorithms, and is easy to use and well-documented. [4] For testing purposes, SB3 is ideal for prototyping a simple model. For long-term research, CleanRL appears to be a better solution for more complex models.

3.4 Integrating RL with WebRTC

To implement the RL model within a WebRTC app, we build the app around AlphaRTC. This is an open-source project that provides the original WebRTC source code, along with an interface for running a custom RL model. The interface sends RTP packet data from the WebRTC bitrate estimator module and passes it to a customizable Python template. The RTP parameters are then provided to the model and a bitrate estimate is returned. The interface then sends the estimate back to the WebRTC congestion controller module, where it is processed accordingly.



```python
class Estimator(object):
    def report_states(self, stats: dict):
        '''
        stats is a dict with the following items
        {
            "send_time_ms": uint,
            "arrival_time_ms": uint,
            "payload_type": int,
            "sequence_number": uint,
            "ssrc": int,
            "padding_length": uint,
            "header_length": uint,
            "payload_size": uint
        }
        '''
        pass

    def get_estimated_bandwidth(self)->int:
        return int(1e6) # 1Mbps
```

Figure 9. AlphaRTC RL interface for WebRTC

3.5 RL-WebRTC Emulator: Docker Network

WebRTC runs on the local host by default. To evaluate our model, it is essential to provide an environment for the app that emulates a real-world network scenario. To achieve this, we provide a Docker network with a defined subnet, and launch the sender and receiver components of the application on two containers within that network. This provides an isolated environment in which the two containers are on the same network, but otherwise behave just like two separate machines. The advantage of Docker is that the application environments are replicable, and the entire evaluation stage can be executed on a single machine.

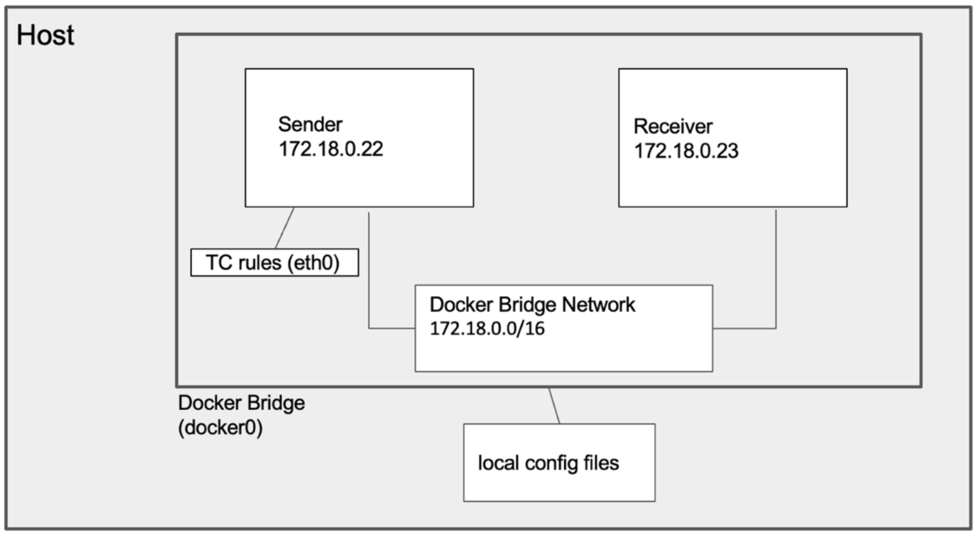


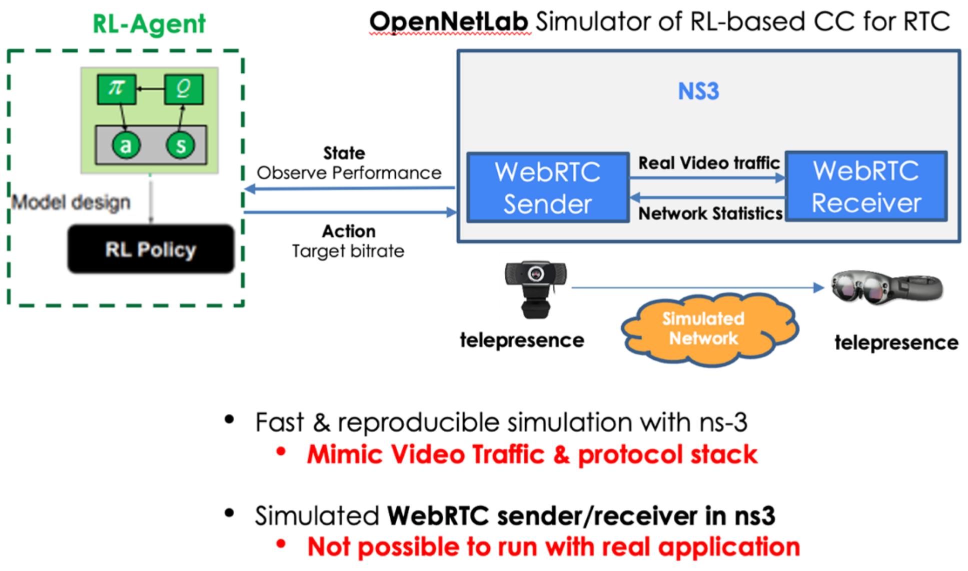
Figure 10. RL-WebRTC network architecture

3.6 Traffic Shaping

The final part of model evaluation involves traffic shaping, specifically limiting bandwidth on a given interface to simulate real world congestion. RL-WebRTC implements Linux Traffic Control (TC), a standard utility for traffic shaping in the Linux environment. The setup allows configurations of various model and network trace combinations, and the TC rules are executed on a defined interface. In the example shown in Fig. 10 the bandwidth limitation is applied to traffic leaving the sender via the standard eth0 interface.

3.7 RL Training in WebRTC Gym

To train specifically for AlphaRTC, we use the OpenNetLab Gym. This gym uses NS3 and WebRTC for training RL models that will be used to build a bandwidth estimator for WebRTC. This setup provides fast and reproducible simulations that mimic video traffic and the protocol stack, and it uses a simulated WebRTC sender and receiver in NS3.



- Fast & reproducible simulation with ns-3
  - **Mimic Video Traffic & protocol stack**
- Simulated **WebRTC sender/receiver in ns3**
  - **Not possible to run with real application**

Figure 11. OpenNetLab AI Gym for WebRTC

3.8 Initial Results

While the focus of this project is the network implementation itself, and the integration of the various components at the application layer, it is helpful to look at some outputs to see how a basic model performs. In the training stage we can compare the performance of various SB3 algorithms using the NS3 gym. Figures 12 and 13 show an example from a baseline model with no fine tuning. In the case, it is a comparison of PPO and DDPG algorithms. We can see from the initial outputs that the models perform similarly in simulation regarding overall reward, and for specific network parameters such as data receive rate. Having set an initial baseline, the user is now free to focus on tasks such as tuning the model, or performing further analysis with the emulator, because the training and testing environments are already established and available as part of the platform.

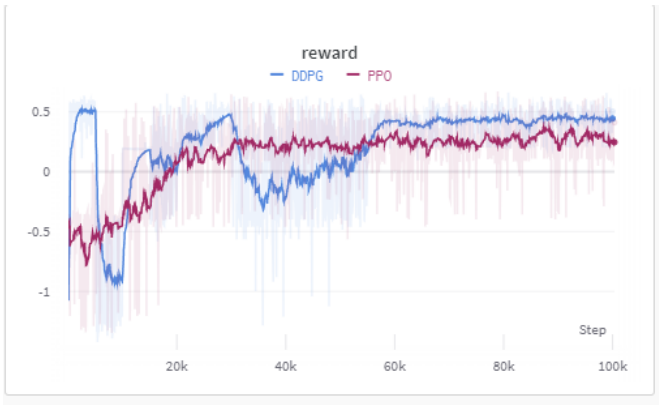


Figure 12. Initial training: reward

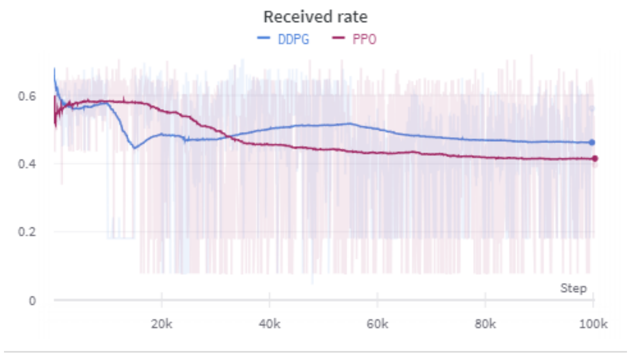


Figure 13. Initial training: receive rate

For testing we run RL-WebRTC in evaluation mode using various Stable-Baselines algorithms, and the 4G 500kbps trace from training. Figures 14 and 15 show that the models perform well against the original trace, and predicted bandwidth does not exceed the actual available bandwidth, except in a few instances. At this point, the researcher could use these results to benchmark performance, and improve the model with further tuning.

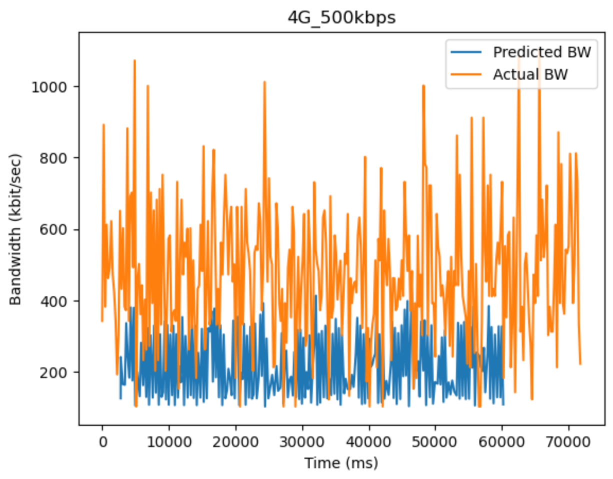


Figure 14. Emulation of SB-DDPG model

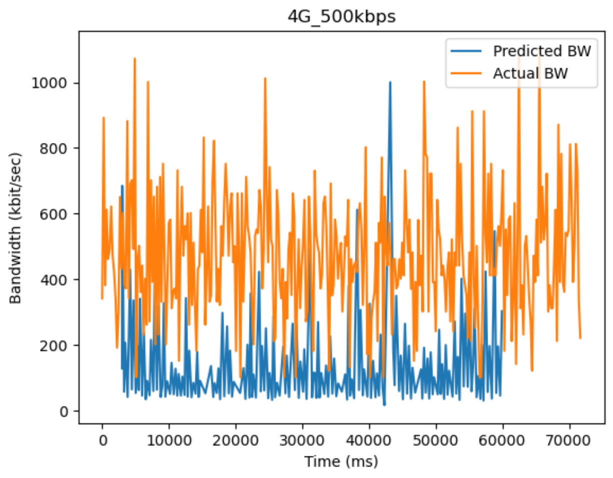


Figure 15. Emulation of SB-SAC model

Ultimately, the goal of this platform is for researchers to be able to load a variety of models and traces and generate results for different parameters without needing to be concerned with the details of network implementation, environment setup, and other administrative details. The results shown here depict the first step of such a project. Using the docker setup and network configurations provided, it is possible to run the emulation in a network environment with one or more models, and then use the results to inform changes to the model training, and the corresponding bandwidth estimation code itself.

Chapter 4: Conclusion

4.1 Contribution

Real-Time video solutions such as volumetric human telepresence require a degree of quality than cannot be guaranteed by existing solutions. In order to develop a strong RL-based system, programmable platforms are required for fast prototyping, evaluation and testing. The platform presented here can be used to develop and test more robust models with performance parameters that closely resemble real-world scenarios.

4.2 Future Work

While Docker is a useful tool for maintaining identical environments, and achieving network isolation, it might also be interesting to evaluate performance with more complex custom networks, such as those provided by Mininet. One area of future work might involve implementing this framework with Containernet. a network emulator that provides Mininet capabilities along with Docker container support. [12]

Another area for future work would be to provide a unified platform, rather than splitting the training and evaluation stages into separate components. While we have seen initial success running the project in stages, there would be greater benefit from Integrating the RL training and evaluation into a single environment.

Finally, there is the possibility of extracting more information from the evaluation process. To fully support more advanced real-time video requirements, it might be beneficial to provide additional Quality of Service (QoS) and Quality of Experience. (QoE) metrics for more nuanced feedback.

References

[1] S. Huang, J. Xie. DAVE: Dynamic Adaptive Video Encoding for Real-time Video Streaming Applications. In IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON), 2021

[2] H. Mao, R. Netravali, M. Alizadeh. Neural Adaptive Video Streaming with Pensieve. In Proceedings of SIGCOMM '17, August 21-25, 2017.

[3] T. Huang, R. Zhang, L. Sun. Self-play Reinforcement Learning for Video Transmission. In 30th Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'20), June 10–11, 2020, Istanbul, Turkey.

[4] Stable-Baselines3 [Online] https://stable-baselines3.readthedocs.io/en/master/

[5] WebRTC [Online] https://webrtc.org/

[6] G. Carlucci, L. De Cicco, S. Holmer, S. Mascolo. Analysis and Design of the Google Congestion Control for Web Real-time Communication (WebRTC). In *MMSys'16, May 10-13, 2016*

[7] B. Wang, Y. Zhang, S. Qian, Z. Pan, and Y. Xie. A hybrid receiver-side congestion control scheme for web real-time communication. In *ACM MMSys*, 2021.

[8] Y. Tianrun, W. Hongyu, H. Runyu, Y. Shushu, L. Dingwei, and Z. Jiaqi. Gemini: An ensemble framework for bandwidth estimation in web real-time communications. In ACM MMSys, 2021.

[9] A. Bentaleb, M. Akcay, M. Lim, A. Begen, R. Zimmerman. BoB: Bandwidth Prediction for Real-Time Communications Using Heuristic and Reinforcement Learning. In IEEE Transactions on Multimedia, 2022

[10] OpenNetLab AlphaRTC [Online] https://github.com/OpenNetLab/AlphaRTC

[11] CleanRL [Online] https://github.com/vwxyzjn/cleanrl

[12] Containernet [Online] https://containernet.github.io/