Evaluating Fairness within Aurora, a Deep RL Congestion Control Model

Victor Sindato and Jonathan Esteban

1 Introduction

Aurora is a reinforcement-learning based congestion control protocol. Introduced in 2019, it was inspired by the recent success of reinforcement learning in domains that involve long-term decision-making. Aurora's major strength lies in its ability to quickly adapt to new network conditions such as changes in link bandwidths, queue sizes, latencies and packet losses. When evaluated alongside standard protocols such as TCP-CUBIC, that adaptability has been shown to give it a significant edge in terms of link utilization. An open challenge that remains is in implementing a sense of fairness within the protocol. The reward function that is used in training Aurora is solely focused on ensuring high throughput, low latency, and fewer packet losses. If placed alongside senders using TCP, it's been rightfully speculated that given what that reward function emphasizes, Aurora would learn to behave in a way that causes those TCP senders to always back-off in order to get higher values of the reward function. Our project's aim was to explore alternative reward functions that could be used to train Aurora alongside a TCP sender such that it not only prioritizes maximizing its own throughput but instead fairly shares the available bandwidth.

2 Prior Work

After conducting a review into the current literature, we believe that implementing TCP within Aurora's framework and optimizing for fairness is a novel problem statement. There has been work on varying trade-offs between the components of the reward function but none that we found was explicit on fairness. That said, most of our findings were made thanks to the contributions of the Aurora authors.

3 Methodology

- Implement TCP within Aurora
- Add support for TCP senders by configuring the simulated the network
- Optimize for fairness and throughput by modifying the reward function to value such goals

4 Implementing TCP within Aurora

Because Aurora is open source and very well maintained, it is very easy to use the network environment to implement TCP and reconfigure the training network to include TCP nodes.

All in all, implementing TCP's additive increase/multiplicative decrease within Aurora took about 50 lines of code which is not bad at all.

Configuring the simulated the network to support TCP senders

We then focused on structuring the network so as to support TCP senders. This was also straightforward as Aurora includes an network environment that allows us to configure any combination of nodes and links with high levels of control for congestion and send rate

Modifying the Reward Function to Optimize for Fairness and Throughput

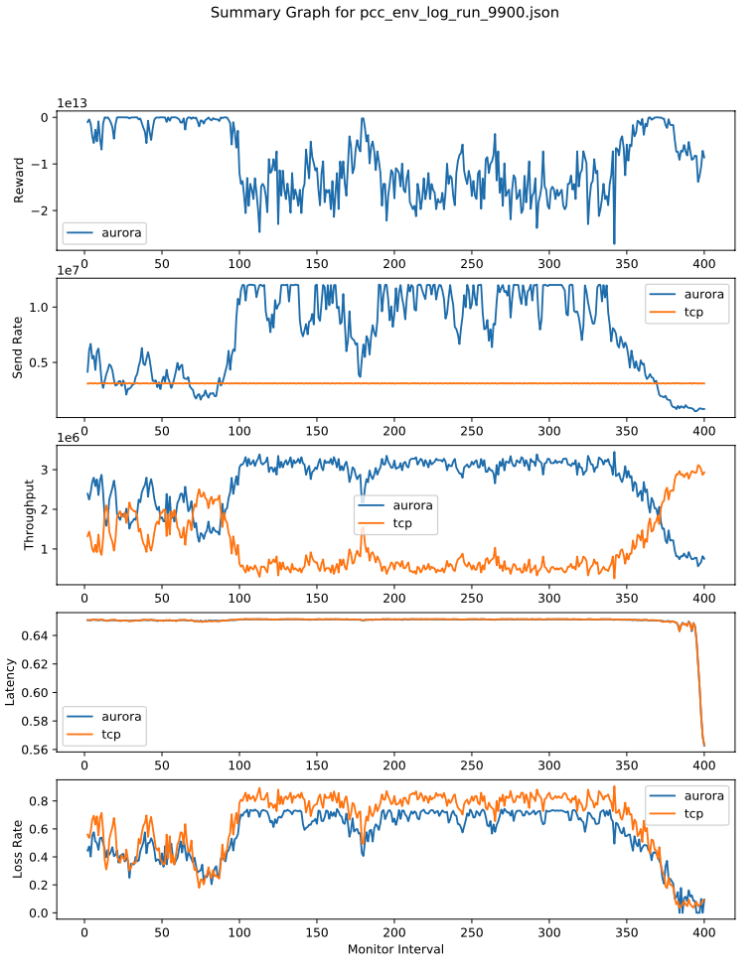
Finally we modified the reward function to optimize for fairness within the network. Aurora allowed us to use a link's throughput, latency, and loss rate as parameters for the reward function.

The original function defined by the authors only optimized throughput for one type of sender. This is what ultimately leads to unfairness in the network. We used a throughput difference heuristic to determine the difference in throughput between a normal sender and a TCP sender.

As a first pass, we found that the throughput difference did not scale properly so we cubed its weight so as to become more sensitive to small changes.

5 Results

We ran Aurora for 10,000 episodes. This plot represents the state of Aurora in the last episode. X axis represents the 400 timesteps within the last episode and the y-axis represents scaled units for each category as described in the paper.



The results were quite remarkable, we immediately saw an improvement with respect to throughput, latency, and loss rate. Particularly for latency and loss rate it seems that Aurora was able to intertwine both senders values to close margins.

6 Conclusions

We were able to quickly reproduce their results from the paper and tweak their package to meet our needs of implementing and embedding TCP within our networks.

We are very impressed with the efforts of the authors of the Aurora paper. They made an excellent job at creating a Deep Reinforcement Internet Congestion Control interface.

7 Next Steps

For next steps, we would look into creating more complex networks to test the limits of Aurora's fairness and simulate a real world network. We would add more senders and have Aurora compete with itself or with our modified version of Aurora.

Finally we would like to look into making the networks more fair with respect to loss rate as it does seem the TCP sender has a slightly higher loss rate than the normal sender.

8 Contributions

From the release of the assignment, we held weekly meetings to discuss and implement the ideas we had. In terms of the work, we worked equally on all stages of our project: the ideation phase, writing the project proposal, implementing TCP and training it with Aurora, working on the final presentation, and writing this final report.