RL-based Internet Congestion Control

Ashwin Shenai
Dept. of Electrical Engineering
Indian Institute of Technology Kanpur
Kanpur, India
ashwins@iitk.ac.in

Astitva Chaudhary
Dept. of Electrical Engineering
Indian Institute of Technology Kanpur
Kanpur, India
astitvac@iitk.ac.in

I. Introduction

Multiple network users contend over scarce communication resources. Data transmission rates of different traffic sources must be modulated dynamically to optimally use resources and maximise user experience. This problem is called "Congestion Control". Congestion Control is fundamental to computer networking and has a crucial impact on user experience for Internet services such as video streaming, voice-over-IP, augmented and virtual reality, the Internet of Things, edge computing, and more. Difficulties are amplified when traffic is forwarded across multiple links, since networks greatly vary in sizes, link capacities, network latency, level of competition between connections, etc. Even after three decades of research,no largely unanimous decision about the right approaches to dealing with congestion in the Internet. Network resources are limited and resource contention is fairly ubiquitous. Incoming traffic can easily exceed output bandwidth which can choke the network. Congestion leads to long delays in data delivery, wasted resources due to lost or dropped packets, and extremely poor QoS(Quality of Service). Congestion may prevent or limit useful communication which leads to congestive collapse. Congestive Collapse was first observed in 1986 where the TCP throughput from LCL to UCB (two-hop, 400 yards) dropped from 32 kbps to 40 bps.

A. Internet Congestion Control

Communication networks vary a lot - a wireless connection between a smartphone and an airport WiFi access point, shared with hundreds of other smartphones, or a data-center network operating at 100 Gb/s with very low propagation delays. Internet traffic is heavy-tailed - 80% of the traffic is carried by a few elephants while the remaining large number of connections are mice. Ultimately many factors affect the decisions that lead to resource allocation - traffic pattern, link failure, dynamic latency, packet loss, and diverse application requirements. With these many complexities, conventional rule-based methods are reduced to heuristics with no guarantee to solve the complex problem.

B. Conventional Approaches to Internet Congestion Control

1) Transmission Control Protocol(TCP): Transmission Control Protocol is what provides the abstraction of a reliable network running over an unreliable channel. TCP provides an effective abstraction of a reliable network running over an unreliable channel, hiding most of the complexity of network communication from our applications: re-transmission of lost data, in-order delivery, congestion control and avoidance, data integrity, and more. TCP embodies (i) slow start: sender can send twice the number of packets last sent each time a good ACK is received, (ii) congestion avoidance: congestion window size is capped at a threshold, and adjusted based on the size of the successfully transmitted packets, and (iii) fast re-transmission: receiver sends a dupACK on receiving an out-of-sequence segment to inform sender of packet loss.

Figure 24.10 *TCP congestion policy summary*

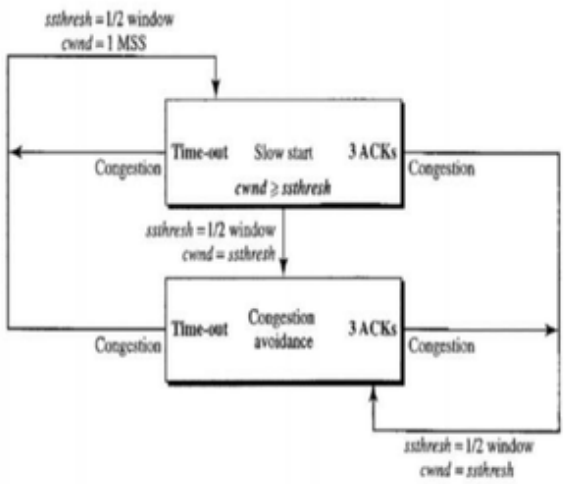

Fig. 1. Brief overview of TCP congestion policy.

2) Performance-oriented Congestion Control(PCC): Performance-oriented Congestion Control (PCC) is a congestion control architecture in which each sender continuously observes the connection between its actions and empirically experienced performance, enabling it to consistently adopt actions that result in high performance. PCC converges to a stable and fair equilibrium. Across many real-world and challenging environments, PCC shows consistent and often 10× performance improvement, with better fairness and stability than TCP. PCC requires no router hardware support or new packet format. Under PCC, send messages at some rate and monitor the network. Based on the

ACKs received, correlate the rate using a carefully designed utility function.

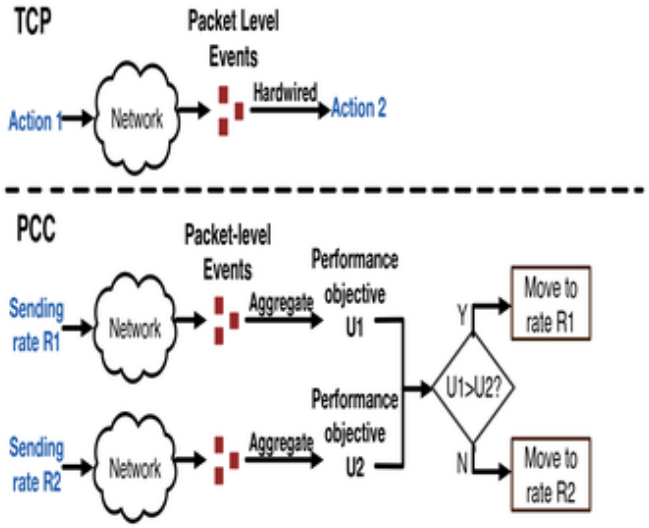

Fig. 2. PCC's decision making.

3) Remy: In this approach, the protocol designer specifies their prior knowledge or assumptions about the network and an objective that the algorithm will try to achieve, e.g., high throughput and low queueing delay. Remy then produces a distributed algorithm—the control rules for the independent endpoints—that tries to achieve this objective. Network is modeled as Markovian - decision to send only depends on packets in queue for each node. Traffic load is modeled as a stochastic process - unicast flows between node pairs are switched on/off. $\langle ack\_ewma, send\_ewma, rtt\_ratio\rangle \rightarrow \langle m, b, r\rangle$. Constructs lookup table to map states to actions such that objective is optimized.

C. Motivating RL-based Congestion Control

Exploring RL based methods in the context of congestion control is very important. If done efficiently, it can improve the performance of a crucial component of the Internet's communication infrastructure, potentially impacting the user experience of almost every performance sensitive Internet service. It can provide a many new areas of application for for RL schemes that poses novel real-world-motivated research challenges. A congestion control protocol can be regarded as mapping a locally-perceived history of feedback from the receiver, which reflects past traffic and network conditions, to the next choice of sending rate. This local history contains information about patterns in traffic and network conditions that can be exploited for better rate selection by learning the mapping from experience via a deep RL approach. Offline learning requires a dedicated offline training phase, and may not perform well if the actual network differs remarkably from the emulated one where offline training was carried out, Since RL algorithms can incorporate real-time network

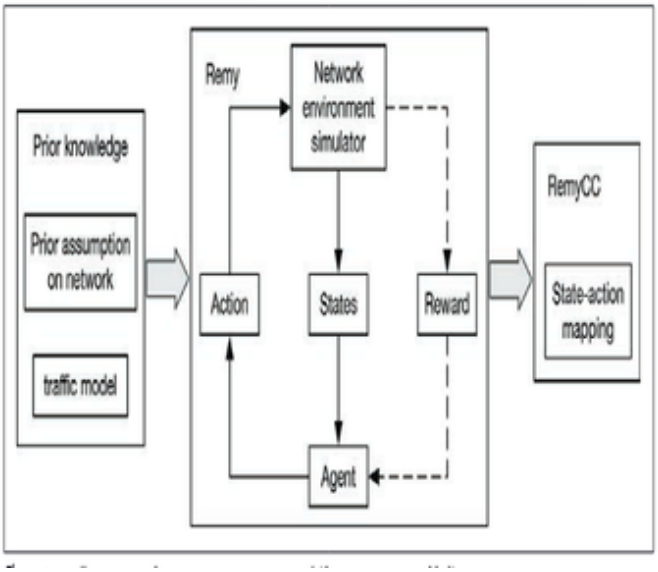

Fig. 3. Remy.

conditions and define actions accordingly, real-time control is possible in RL algorithms. Compared to supervised learning and unsupervised learning techniques, RL algorithms are more responsive to environmental changes. RL-based CC algorithms learn the CC rules directly based on different environment information.

II. Survey of RL-based Congestion Control

A. Aurora

Congestion control is formulated as a sequential decision making problem. Agent is the sender of traffic. Time is divided into consecutive intervals called monitoring intervals. Actions are changes to sending rate at the starting of every sending interval which are then fixed for the entire interval. States are bounded(fixed-length) histories of network statistics and depends on the $latency\ gradient$ - the derivative of latency with respect to time, $latency\ ratio$ - the ratio of the current MI's mean latency to minimum observed mean latency of any MI in the connection's history, and $sending\ ratio$ - the ratio of packets sent to packets acknowledged by the receiver. The reward function is given by

$$10 * throughput - 1000 * latency - 2000 * loss$$

where throughput is measured in packets per second, latency in seconds and loss is the proportion of packages sent but not acknowledged. The model is trained with the Proximal Policy Optimization(PPO1) algorithm as the RL scheme(via stable baselines).

B. QTCP

It integrates a reinforcement-based Q-learning framework with TCP design in our approach called QTCP. QTCP enables senders to gradually learn the optimal congestion control policy in an on-line manner. QTCP does not need hard-coded

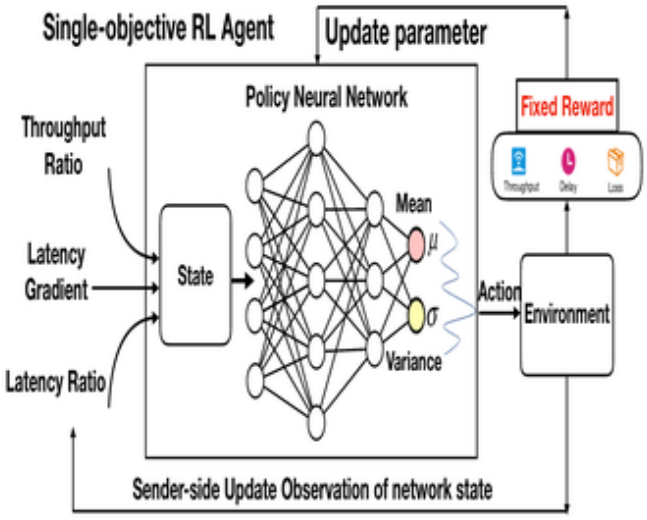

Fig. 4. Architecture for Aurora.

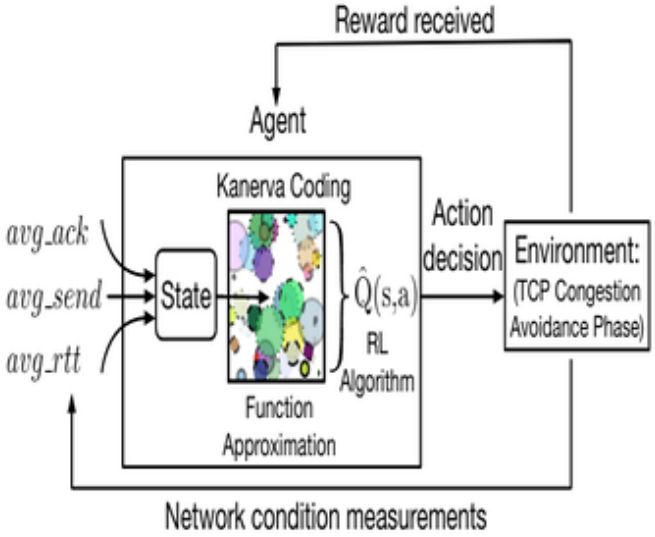

Fig. 5. QTCP Architecture.

rules, and can therefore generalize to a variety of different networking scenarios. It overcomes the limitation arising from the rule-based design principle, where the performance is linked to a pre-decided mapping between the observed state of the network to the corresponding actions. Rule-based protocols are unable to adapt their behavior in new environments or learn from experience for better performance and so do not generalize well under a wide range of network scenarios. States are unique profile of network conditions evaluated using avg_send - the average interval between sending two packets, avg_ack - the average interval between receiving two consecutive ACKs, and avg_rtt - the average RTT. Actions are the changes in the congestion window size($cwnd$) by $0,\ -1,\ +10$. The reward is based on change in the utility in consecutive intervals. The utility function is given by

$$\alpha * log(throughput) - \delta * log(RTT)$$

where α and δ control the relative weight or importance of throughput and RTT. The model is trained with Q-Learning algorithm as the RL scheme. QTCP outperforms the traditional rule-based TCP by providing 59.5 percent higher throughput while maintaining low transmission latency.

C. RL-TCP

It designs a learning-based TCP CC schemes for wired networks with under-buffered bottleneck links, a reinforcement learning (RL) based TCP CC (RL-TCP). RL-TCP tailors the design of states and action space towards networks with under-buffered bottleneck links. Also, RL-TCP treats the temporal credit assignment of reward according to TCP dynamics. The states are defined using five variables, $ewma\_send$, $ewma\_ack$, avg_rtt, $ssthresh$, $cwnd$. Actions are the changes in the congestion window size($cwnd$) by $-1,\ 0,\ +1,\ +3$. The reward is based on the utility in consecutive timesteps. The utility function is given by

$$log(throughput) - log(B) - \delta_1 * log(RTT - RTT_{min})$$

$$+\delta_2 * log(1 - loss\_rate)$$

where B is the bottleneck bandwidth, and δ_1 and δ_2 are adjustable coefficients. To learn the Q-function Q(s, a), it uses SARSA, a popular on-policy temporal difference (TD) learning algorithm for value-based RL. It achieves a better tradeoff between throughput and delay, under various simulated network scenarios.

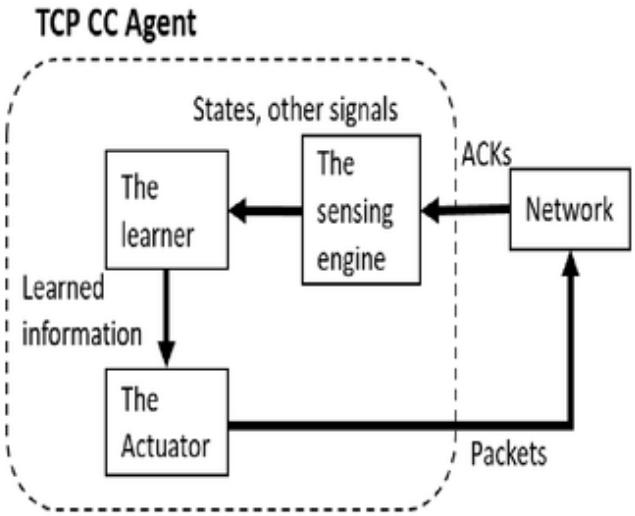

Fig. 6. TCP CC Agent.

D. RAX

It proposes Reactive Adaptive eXperience based congestion control (RAX), a method of congestion control that uses online reinforcement learning to maintain an optimum congestion window with respect to a given reward function and based on current network conditions. It uses a neural network based approach that can be initialized either with random weights or with a previously trained neural network to improve stability and convergence time. It proposes Partial Action Learning(PAL), a formulation of Deep RL that supports delayed and partial rewards. PAL can handle delay in rewards. One action generates several partial actions. These generate partial rewards on interacting with the environment. Agent performs a new action on receiving a partial reward. When all partial rewards of an action are received, the RL algorithm proceeds to train the agent.The states are defined using the EWMA of Time between last two ACKs received, RTT of last received packet, indicator for if the last packet was lost, current cwnd. Actions are real numbers that represents the change to the cwnd. Reward is based on sum of all sent bytes, received with ACK bytes, time between previous and current ACK. The RL algorithm used is Actor Critic (not explicitly specified), and qt. Rax converges to a stable, close-to-optimum solution.

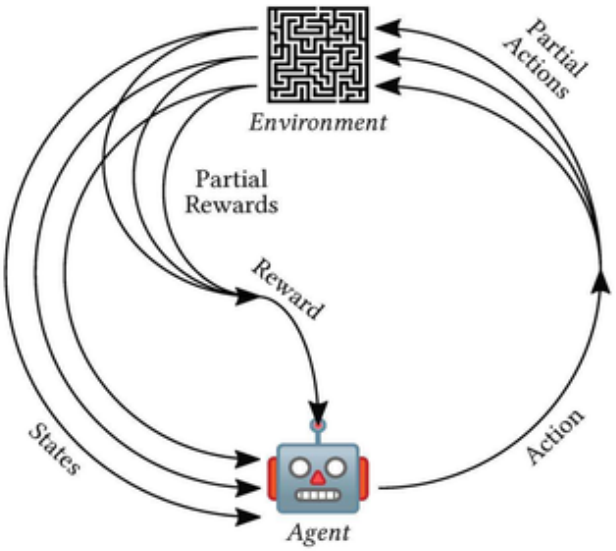

Fig. 7. Partial Action Learning.

E. Eagle

Eagle is a new congestion control algorithm to refine existing heuristics. Eagle takes advantage of expert knowledge from an existing algorithm, and uses deep reinforcement learning (DRL) to train a generalized model with the hope of learning from an expert. Learning by trial-and-error may not be as efficient as imitating a teacher; by the same token, DRL alone is not enough to guarantee good performance. Eagle seeks help from an expert congestion control algorithm, BBR, to help train a long-short term memory (LSTM) neural network in the DRL agent, with the hope of making decisions that can be as good as or even better than the expert. With an extensive array of experiments, it is discovered that Eagle is able to match and even outperform the performance of its teacher, and outperformed a large number of recent congestion control algorithms by a considerable margin. The states are defined using the summary of the last four timesteps (three RTTs per timestep) - whether agent faced delay, ewma of loss rate, ewma of latency ratio, difference between number of increases and decreases in sending rate from the point of delay. The actions are defined as: increase sending rate by 2.89 and cwnd by 2, decrease sending rate by 1.25 and cwnd by 1.25, or do nothing. The rewards are based on change in delay and other state parameters. The Cross-entropy method is used as the RL algorithm.

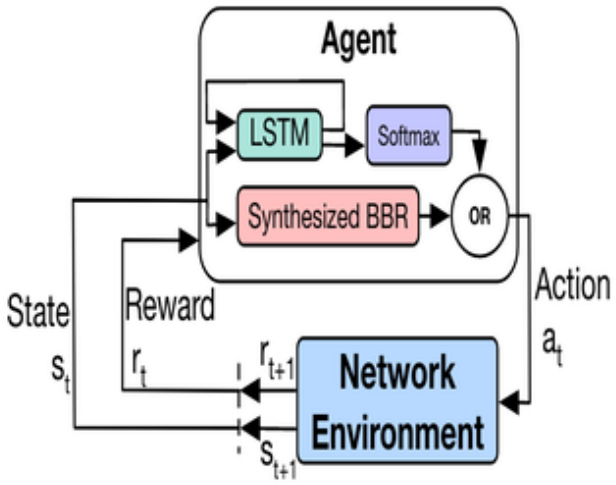

Fig. 8. Eagle Architecture Overview.

F. MVSFT-RL

It proposes to formulate congestion control with an asynchronous RL agent that handles delayed actions. MVFST-RL is a scalable framework for congestion control in the QUIC transport protocol that leverages state-of-the-art in asynchronous RL training with off-policy correction. It analyzes modeling improvements to mitigate the deviation from Markovian dynamics, and evaluate the method on emulated networks from the Pantheon benchmark platform. The states are defined as 100ms summary of network statistics with history of actions taken - 20 statistics per ACK. The actions are defined as

$$cwnd \rightarrow \{cwnd, cwnd/2, cwnd * 2, cwnd - 10, cwnd + 10\}$$

The rewards are defined as

$$\log(t + \epsilon) - \beta \times \log(d + \epsilon)$$

where t is average throughput, d is average max delay during the window, the parameter β trades-off between the t and d, and ϵ is a small value to ensure numerical stability. The RL algorithm used is IMPALA.

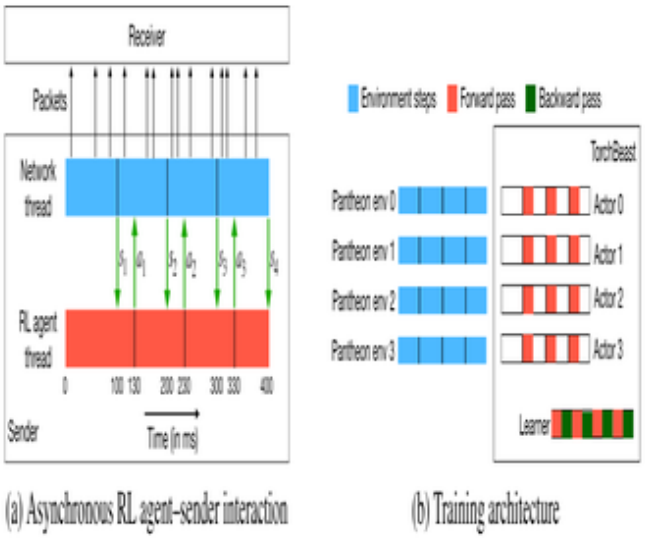

Fig. 9. MVSFT Architecture Overview.

G. DeepCC

DeepCC leverages advanced deep reinforcement learning (DRL) techniques to let machines automatically learn how to steer throughput-oriented TCP algorithms toward achieving applications' desired delays in a highly dynamic network such as the cellular network.DeepCC plug-in is used to boost the performance of various old and new TCP schemes including TCP Cubic, Google's BBR, TCP Westwood, and TCP Illinois in cellular networks. Through both extensive trace-based evaluations and real-world experiments, it is shown that not only DeepCC can significantly improve the performance of TCP schemes, but also after accompanied by DeepCC, these schemes can outperform state-of-the-art TCP protocols including new clean-slate machine learning-based designs and the ones designed solely for cellular networks. The states are defined as the fixed-length history of network statistics - average throughput, average packet delay, number of samples for averaging, cwnd. The action is to determine α to set $cwnd = 2^{\alpha} \times cwnd_{tcp}$. The reward is defined as the delay-throughput product, negative if delay is above target else positive. The RL algorithm used is the Deep Deterministic Policy Gradient(DDPG).

H. Orca

A pragmatic and evolutionary approach combining classic congestion control strategies and advanced modern deep reinforcement learning (DRL) techniques and introduce a novel hybrid congestion control for the Internet named Orca. Through extensive experiments done over global testbeds on

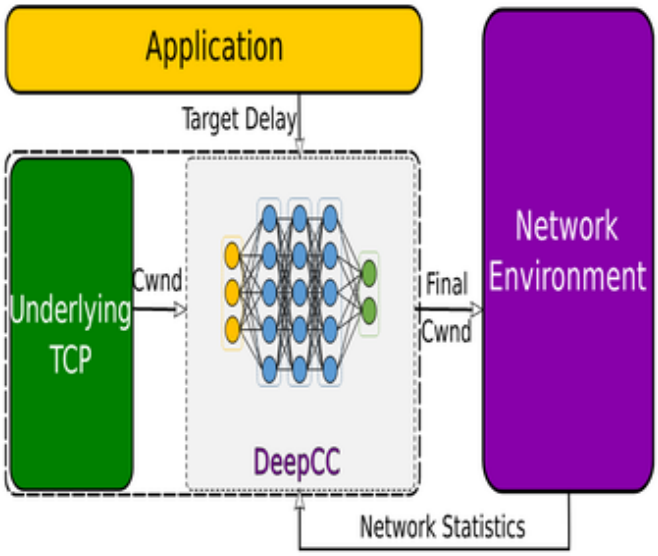

Fig. 10. DeepCC high-level overview.

the Internet and various locally emulated network conditions, it is demonstrated that Orca is adaptive and achieves consistent high performance in different network conditions, while it can significantly alleviate the issues and problems of its clean-slate learning-based counterparts. The states are defined as the fixed-length history of network statistics - 9 per timestep. The action is to determine α to set $cwnd = 2^{\alpha} \times cwnd_{tcp}$. The reward is defined as

$$\frac{\text{throughput} - \zeta \times \text{loss/clip}(d_{min}, \text{delay})}{tp_{max}/d_{min}}$$

where ζ is a coefficient determining the relative impact of the loss rate compared to the throughput rate.

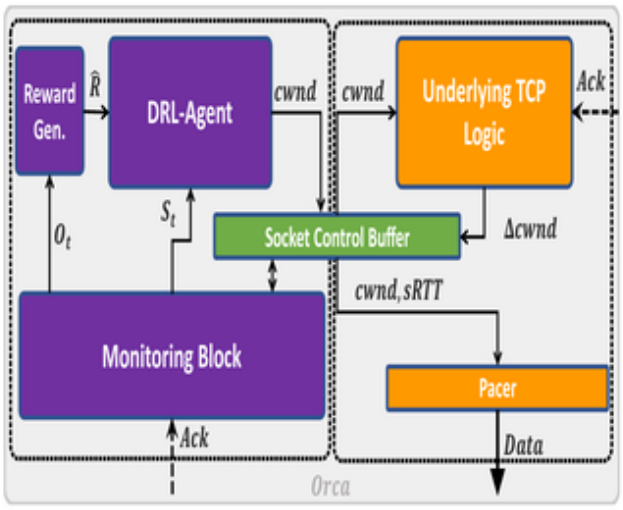

Fig. 11. Orca architecture and high-level overview.

I. Multi-Objective Congestion Control(MOCC)

MOCC is the first multi-objective congestion control algorithm that attempts to address this question. The core of MOCC is a novel multi-objective reinforcement learning framework for CC to automatically learn the correlations between different application requirements and the corresponding optimal control policies. Under this framework, MOCC further applies transfer learning to transfer the knowledge from past experience to new applications, quickly adapting itself to a new objective even if it is unforeseen. It provides both user-space and kernel-space implementation of MOCC. Real-world Internet experiments and extensive simulations show that MOCC supports well multi-objective, competing or outperforming the best existing CC algorithms on each individual objectives, and quickly adapting to new application objectives in 288 seconds (14.2× faster than prior work) without compromising old ones. The states are defined as the fixed-length history of network statistics - sending ratio, latency ratio, latency gradient. The actions are defined as change in sending rate, which are multiplicative. The rewards are defined as dynamically parameterized with multi-objective weights and normalized performance measures on throughput, latency and packet loss rate. The RL algorithm used is Proximal Policy Optimization(PPO).

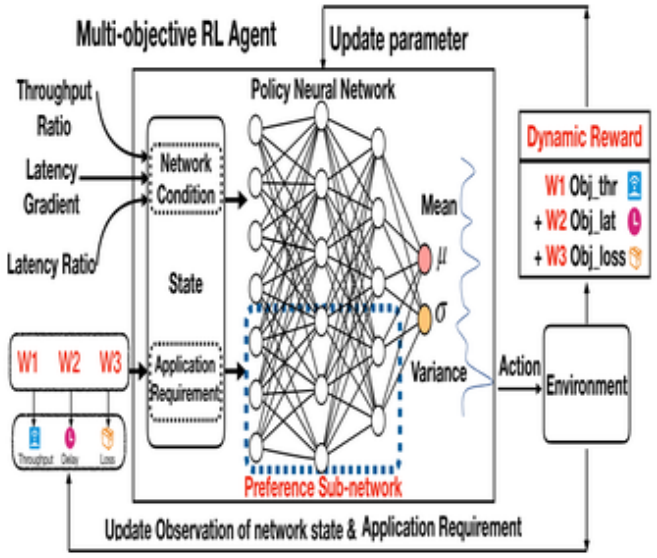

Fig. 12. MOCC architecture.

III. EXPERIMENTS

We present various experiments on Aurora, which uses deep neural networks as function approximators for RL-based congestion control. Tests were run using Pantheon [3], which uses Mahimahi shells to emulate network links. All tests were run locally via emulation on an Ubuntu 18.04 VM. For the brevity of this report, we only detail the nature of the tests run for each model variation and the details of the experiments run. The results can be found in the final presentation slides submitted along with this report (and also hosted here). The code used/written in the project is hosted at https://github.com/ashwin2802/EE698V

A. Testing Scenarios

All tests are run on a single emulated link for a fixed duration of 30s.

- **Test 1:** Target throughput is kept constant at 12 Mbps.
- **Test 2:** Target throughput is kept constant at 12 Mbps. Results are averaged over 5 runs
- **Test 3:** Target throughput is kept constant at 12 Mbps. 3 competing flows are simulated in the same link
- **Test 4:** Target throughput is modeled as a Poisson distributed RV. The average throughput is around 2-3 Mbps.
- **Test 5:** Target throughput is kept constant at 60 Mbps.

B. Model Architecture

To experiment with the effect of changing the model architecture, we trained the following models with PPO. Note that the actor and critic network architectures are kept the same.

- **Experiment 1:** [64, 32] - This is the default architecture used.
- **Experiment 2:** [128. 8] - This architecture was used to test the effect of widening the network layers. Performance improvement in Test 4 was observed.
- **Experiment 3:** [64, 32, 64] - This architecture was used to test the effect using more network layers. No significant performance improvement was observed.

C. RL Algorithm

To compare the performance of various RL algorithms with this problem, we trained models with fixed architecture of [64, 32] with the following algorithms:

- **Experiment 1:** PPO - This is the algorithm used in the paper, and the default setting.
- **Experiment 2:** TD3
- **Experiment 3:** SAC

We observed that TD3 and SAC required fewer timesteps as compared to PPO to achieve the same level of convergence. However the convergence achieved in both these algorithms was not reliable, as we observed significiant levels of divergence upon further training. The overall performance of the models trained with TD3 and SAC was worser than the models trained with PPO. Hence, we concluded that PPO is the preferred algorithm to use for this problem.

D. Reward Function

The following reward functions were experimented with.

- **Experiment 1:**

$$10 * throughput - 1000 * latency - 2000 * loss$$

This is the default reward function suggested in the paper

- **Experiment 2:**

$$20 * throughput - 1000 * latency - 2000 * loss$$

We expected the model with this reward function to perform better on throughput. However, no significant performance improvements were observed.

- **Experiment 3:**

$$5 * throughput - 1000 * latency - 2000 * loss$$

We expected the model with this reward function to perform better on latency. However, no significant performance improvements were observed.

IV. ROADBLOCKS

1) Code/libraries used are outdated. Python2 used in place of Python3 and Tensorflow 1.14 used in place of Tensorflow 2+.
2) Outdated kernel issues with Orca and DeepCC.
3) All models are trained on a CPU.

V. FURTHER AVENUES

1) Model based approaches - Integrating a model of the equivalent MDP along with the model.
2) Meta Learning - Training the model with parametrizable weights to allow it adapt to changes in network conditions without retraining.
3) Reward Engineering - looking at rewards other than just linear combinations of the metrics
4) Competitive Learning - using other models/CC algorithms to help the model benchmark its performance while learning
5) Redeveloping the frameworks to support newer software versions

REFERENCES

[1] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, Aviv Tamar. A Deep Reinforcement Learning Perspective on Internet Congestion Control. Proceedings of the 36th International Conference on Machine Learning, PMLR 97:3050-3059, 2019.

[2] Yiqing Ma, Han Tian, Xudong Liao, Junxue Zhang, Weiyan Wang, Kai Chen, and Xin Jin. 2022. Multi-objective congestion control. In Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys '22). Association for Computing Machinery, New York, NY, USA, 218–235. https://doi.org/10.1145/3492321.3519593

[3] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for internet congestion-control research. In Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '18). USENIX Association, USA, 731–743.

[4] Abbasloo, S., Yen, C.-Y., and Chao, H. J. (2020). Classic meets modern: A pragmatic learning-based congestion control for the internet. SIGCOMM '20, page 632–647, New York, NY, USA. Association for Computing Machinery.

[5] Abbasloo, S., Yen, C.-Y., and Chao, H. J. (2021). Wanna make your tcp scheme great for cellular networks? let machines do it for you! IEEE Journal on Selected Areas in Communications, 39(1):265–279.

[6] Bachl, M., Zseby, T., and Fabini, J. (2019). Rax: Deep reinforcement learning for congestion control. In ICC 2019 - 2019 IEEE International Conference on Communications (ICC), pages 1–6.

[7] Dong, M., Li, Q., Zarchy, D., Godfrey, P. B., and Schapira, M. (2015). PCC: Re-architecting congestion control for consistent high performance. In 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), pages 395–408, Oakland, CA. USENIX Association.

[8] Emara, S., Li, B., and Chen, Y. (2020). Eagle: Refining congestion control by learning from the experts. In IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, pages 676–685.

[9] Jiang, H., Li, Q., Jiang, Y., Shen, G., Sinnott, R., Tian, C., and Xu, M. (2021). When machine learning meets congestion control: A survey and comparison. Computer Networks, 192:108033.

[10] Li, W., Zhou, F., Chowdhury, K. R., and Meleis, W. (2019). Qtcp: Adaptive congestion control with reinforcement learning. IEEE Transactions on Network Science and Engineering, 6(3):445–458.

[11] Sivakumar, V., Delalleau, O., Rockt aschel, T., Miller, A. H., K uttler, H., Nardelli, N., Rabbat, M., Pineau, J., and Riedel, S. (2019). Mvfst-rl: An asynchronous rl framework for congestion control with delayed actions. arXiv preprint arXiv:1910.04054.

[12] Winstein, K. and Balakrishnan, H. (2013). Tcp ex machina: Computer-generated congestion control. 43(4).

[13] https://stable-baselines.readthedocs.io/en/master/index.html

[14] https://spinningup.openai.com/en/latest/index.html