Learning to Maximize Network Bandwidth Utilization with Deep Reinforcement Learning

Hasibul Jamil, Elvis Rodrigues, Jacob Goldverg, Tevfik Kosar
Department of Computer Science and Engineering
University at Buffalo (SUNY), Amherst, NY 14260, USA
Email:{mdhasibu, elvisdav, jacobgol, tkosar}@buffalo.edu

***Abstract*—Efficiently transferring data over long-distance, high-speed networks requires optimal utilization of available network bandwidth. One effective method to achieve this is through the use of parallel TCP streams. This approach allows applications to leverage network parallelism, thereby enhancing transfer throughput. However, determining the ideal number of parallel TCP streams can be challenging due to non-deterministic background traffic sharing the network, as well as non-stationary and partially observable network signals. We present a novel learning-based approach that utilizes deep reinforcement learning (DRL) to determine the optimal number of parallel TCP streams. Our DRL-based algorithm is designed to intelligently utilize available network bandwidth while adapting to different network conditions. Unlike rule-based heuristics, which lack generalization in unknown network scenarios, our DRL-based solution can dynamically adjust the parallel TCP stream numbers to optimize network bandwidth utilization without causing network congestion and ensuring fairness among competing transfers. We conducted extensive experiments to evaluate our DRL-based algorithm's performance and compared it with several state-of-the-art online optimization algorithms. The results demonstrate that our algorithm can identify nearly optimal solutions 40% faster while achieving up to 15% higher throughput. Furthermore, we show that our solution can prevent network congestion and distribute the available network resources fairly among competing transfers, unlike a discriminatory algorithm.**

***Index Terms*—Efficient network bandwidth utilization, parallel TCP streams, deep reinforcement learning, online optimization.**

I. Introduction

With the scientific and commercial application domains generating an unprecedented amount of data, geographically distant sites need to transfer large volumes of data as part of complex analytics workflows to gain insights from the data [1]. For instance, the National Energy Research Scientific Computing Center (NERSC) generates about 1 Exabyte of data per year, which needs to be moved to Oak Ridge Leadership Computing Facility for petascale simulations [2]. However, despite high-speed networks, most data transfers cannot achieve more than a few Gbps due to inadequate network bandwidth utilization. It has been observed that single-stream TCP throughput achieved by data transfer applications is a small portion of the available end-to-end network bandwidth, and this deficiency is due to the TCP's additive increase and multiplicative decrease (AIMD) window control procedure, as reported in several studies [3]–[7].

There are two primary congestion control mechanisms: loss-based algorithms and delay-based algorithms. Loss-based algorithms, such as CUBIC [8] and NewReno [9], can achieve higher throughput, but they are prone to higher round trip time (RTT). On the other hand, delay-based algorithms, such as Copa [10] and FastTCP [11], are usually subject to acknowledgment (ACK) delay and network jitter, resulting in underutilization of network capacity. For loss-based algorithms, the deficiency in a single TCP stream's throughput is due to the window control mechanism in TCP congestion avoidance in response to packet loss. The TCP bandwidth estimation equations proposed by Mathis [12] illustrate that TCP throughput is inversely proportional to the square root of the packet losoverles rate. To achieve near-full network capacity usage on high bandwidth-delay-product (BDP) networks, a minimal packet loss rate is required for a single TCP stream. However, achieving such a low packet loss rate is impractical as different factors, like network congestion, congestion control procedure, or hardware failures, can be the underlying reason [13], [14].

The use of multiple parallel TCP streams is a common method to enhance data transfer throughput by efficiently utilizing network bandwidth. However, increasing the number of TCP streams beyond a certain point does not enhance performance but instead leads to congestion and greater packet loss. The optimal number of parallel TCP streams varies depending on dynamic conditions such as network background traffic, network route, endpoints, and network configuration. Given the non-stationary nature of the problem, it is necessary to adapt the number of parallel TCP streams dynamically. Previous work has explored different approaches to finding the optimum number of TCP streams, including real-time probing [6], heuristics [15], [16], and historical analysis models [3], [17]. However, heuristics models have been found to suffer from poor robustness, as they are designed to work with specific network conditions, which may not be present in other networks. Similarly, real-time probing can create unnecessary traffic that may overburden the network resources and cause unfairness among other contending background traffic.

This paper presents a novel methodology that leverages deep reinforcement learning (DRL) to devise a policy function approximator for optimizing TCP streams based on host node network signals. Beyond the conventional methods, this study introduces a utility function with an incorporated punishment term, based on observed packet loss. This novel intervention ensures that while optimizing for maximum throughput, fairness is also maintained among competing data transfers.

The major contributions of this paper include:

- Establishing a Novel Perspective: We present compelling evidence that maximizing aggregated TCP throughput can be modeled as a DRL problem. To address this, we devise a policy gradient method that aims for optimal network bandwidth utilization.
- Innovative Reward Mechanism: Our reward function uniquely ensures both maximum throughput and fairness amidst competing background traffic. This dual-objective approach differentiates our work, tackling two pivotal concerns with a unified model.
- Comprehensive Evaluation Insights: After rigorous comparisons with various online optimization techniques, our DRL-based algorithm emerges superior. Notably, it achieves convergence 40% faster on average and delivers up to 15% enhanced aggregated throughput.

The rest of the paper is organized as follows: Section II presents the background and related works; Section III discusses our proposed method; Section IV presents the evaluation of our method; and Section V concludes the paper.

II. Background and Related Work

This section delves into the benefits of using multiple TCP streams to maximize network bandwidth utilization while also exploring the associated congestion and fairness issues that can arise when sharing network resources with background traffic. Additionally, it examines existing studies in this field and highlights the unique advantages of our model.

A single stream TCP throughput is modeled with the following Equation 1 in case the packet loss rate is less than 1/100 as shown by Mathis [12]:

$$TCP_{thr} \leq \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \tag{1}$$

Here, TCP_{thr} represents TCP throughput, MSS is the maximum segment size, RTT is the round trip time, C is a constant, and p is the packet loss ratio - calculated by dividing the number of retransmitted packets by the total number of transmitted packets. While MSS and RTT are static and depend on network characteristics, p is a dynamic signal that has a significant impact on the loss based TCP congestion avoidance algorithm. The TCP congestion avoidance algorithm treats all packet losses as a signal for network congestion and reduces the TCP sending rate (i.e., decreases the TCP window size), regardless of the reason for packet loss, whether it is due to network congestion or traffic insensitive reasons (such as hardware or link failure, TCP random early detection). We can extend the single stream TCP throughput model for multiple TCP streams by aggregating individual throughput of each stream to derive the application's aggregated TCP_{agg} as shown in Equation 2 [7]:

$$TCP_{agg} \leq \frac{C}{RTT}\left[\frac{MSS}{\sqrt{p_1}} + \cdots + \frac{MSS}{\sqrt{p_n}}\right] \tag{2}$$

where p_i is the packet loss rate of TCP connection i, MSS is the same as above and fixed for all TCP streams and RTT is the converged round trip time value of all the TCP connections. Equation 2 shows that multiple streams can achieve higher aggregated throughput than a single stream because they create a virtual combined MSS that is n times larger. However, this is only true if the proportion of packet loss is evenly distributed across all streams and there is no congestion.

The benefits of adding additional TCP streams diminish when the network gets congested due to an increase in packet loss rate and bottleneck links reaching full capacity. To achieve optimal network bandwidth utilization, it is necessary to identify the ideal number of TCP streams that do not cause congestion. However, this number is not static and varies throughout the transfer session due to fluctuations in background traffic. To address this issue, we propose a DRL-based algorithm that can learn a policy to intelligently avoid network congestion and select an optimal number of parallel TCP streams based on the network signals at the end hosts.

Several previous studies have explored the proposal of a new transport layer to enhance bandwidth utilization, such as data center congestion control [18], qtcp [19], BBR [20], and PCC-Vivace [21]. However, the widespread implementation of a new transport layer protocol is often challenging since it requires system and kernel modifications. Dong et al. [6] have suggested probing techniques to estimate available bandwidth. While this method can be effective, network probing can add to the existing traffic burden and excessive probing may result in network congestion. Several ad-hoc heuristics methods [15], [22] have utilized observation-based rules to select the number of parallel streams. However, these approaches are often not robust enough for new or previously unseen network scenarios. Other methods, such as Falcon [23] and Balaprakash et al. [24], tackle the transfer throughput maximization problem as an online convex optimization. However, these methods are stateless and do not optimize for the entire transfer, focusing instead on immediate performance improvement.

To address these limitations, our work presents a novel learning-based approach that utilizes deep reinforcement learning. We propose a policy function approximator to generalize the relationship between observed network signals in the end host and the network congestion state. Our approach discovers the optimal number of parallel TCP streams to operate on the sweet spot, maximizing available network bandwidth while avoiding network congestion.

III. Methodology

Our design principles are as follows: (1) no prior knowledge of the network or end hosts is required; (2) only observations obtainable at the end host are used; (3) a DRL approach is used to learn the relationship between observed network signals and network congestion state; and (4) the problem is formulated as a partially observable Markov decision process (POMDP). The first principle ensures that the proposed method is applicable to a wide range of networks and end hosts. The second principle allows the method to be implemented without requiring any changes to the network or end hosts. The third principle allows the method to learn the optimal number of parallel TCP

streams to utilize available network bandwidth. The fourth principle allows the method to handle the fact that the network condition is only partially observable.

Our introduced DRL agent employs a policy gradient method, specifically proximal policy optimization (PPO) [25]. The POMDP framework enables the DRL agent to utilize observed network signals as the state of the environment, adjusting the number of parallel TCP streams to optimize network bandwidth. Feedback in the form of performance metrics is used with a utility function to derive a reward for the chosen action, allowing the DRL agent to optimize its decisions over time. The environment consists of the end hosts, data transfer application, and network links. The DRL policy agent uses a fully connected deep neural network (DNN) model to select an action based on present and previous n steps of network signals (i.e., state space). The feedback (i.e., achieved throughput, packet loss rate (plr), and round trip time (rtt)) from the environment are used with a utility function to derive a reward for the chosen action of DRL agent. The cycle repeats until the transfer process concludes, with each step considered a monitoring interval (MIs) [21]. Our model's DNN and PPO hyperparameters are described in table I.

TABLE I: DNN and PPO Hyper-parameters

Hyperparameter	Value	Hyperparameter	Value
Model type	fully-connected	PPO entropy coefficient	0.01
Model nonlinearity	tanh	PPO clip param	0.3
Model hidden layers	[512, 512]	PPO VF clip param	10
Weight sharing between θ, θ_v	true	PPO KL target	0.01
Learning rate	0.00005	SGD iterations per batch	30
Discount factor γ	1	SGD minibatch size	1000

A. DRL Architecture

1) State Space: Minimizing the number of state variables or attributes that are included in the environment is desirable to reduce the size of the exploration space, and only those attributes that are closely related to the DRL agent's goal should be included. Since network link characteristics such as loss rate, available bandwidth, and latency vary across different networks, state variables in terms of network signals should avoid signals with high variability (e.g., selecting the change in RTT over time rather than the current RTT value). The sender selects an action a_t at a particular measurement interval (MI) t and receives observation feedback consisting of performance metrics after a preset time. From this feedback, a signal vector x_t is computed, which includes state variables such as the RTT gradient, RTT ratio [26], packet loss rate (plr), and average throughput of the current MI. The DRL agent's next action is determined by a fixed-length history of these signal vectors. So, at time t, if the state is s_t for our agent, s_t could be defined as: $s_t = \left(x_{t-(n)}, \ldots, x_t\right)$. Using a bounded-length history of signal vectors rather than the most recent one helps the agent effectively detect patterns in network conditions changes, allowing more accurate reactions.

2) Actions: The DRL agent's action space is limited to five actions to accommodate network condition changes. The actions include adding five, one, or no TCP streams, and removing one or five TCP streams. The chosen actions encourage the agent to increase the number of streams quickly to use available bandwidth, while allowing reducing the number of TCP streams when necessary, such as in a congested network.

3) Utility Function and Reward: In this study, a utility function is introduced to guide the DRL agent in maximizing its cumulative value during a transfer session. The objective of achieving optimal utilization of network bandwidth is divided into two parts: maximizing throughput and minimizing packet loss. The latter is chosen because it is a strong signal of network congestion. Additionally, incorporating packet loss in the utility function promotes fairness among concurrent flows, as detecting an increase in packet loss rate allows the agent to take action to decrease the number of TCP streams. Our utility function has the following non-linear form:

$$U\left(n_i, T_i, L_i\right) = \frac{T_i}{K^{n_i}} - T_i L_i \times B \tag{3}$$

The variables in the utility function are n_i for the number of parallel TCP streams, T_i for average throughput, and L_i for the aggregate packet loss rate. The constants K and B can be adjusted to control the cost-benefit, performance, and punishment terms. It is important to note that the improvement in throughput is not linearly related to the increase in the number of parallel TCP streams, and the value of K can be tuned to ensure that the utility function will increase as long as there is a noticeable (e.g., 1%) throughput gain for a number of parallel TCP streams. As a result, this non-linear utility function guarantees that the transfer throughput will converge and bandwidth resources will be allocated fairly among transfers originating from multiple users. We utilize the difference between the utility values of two consecutive MIs to formulate the reward signal. The utility function specified earlier establishes the linkage between the state space and the reward. We choose to employ the difference between the utility values of successive MIs rather than the precise value of the utility function. This decision is rooted in the DRL agent's goal, which mandates that a rise in the utility value indicates an increase in the throughput and decrese in packet loss rate; therefore, the corresponding action should be incentivized. Consequently, we define the reward as follows [19]:

- x, if $U_t - U_{t-1} > \epsilon$
- y, if $U_t - U_{t-1} < -\epsilon$
- 0, otherwise

The variables x and y indicate positive and negative reward values correspondingly. Tunable thresholds ϵ set the sensitivity of the DRL agent's selected action to choose the appropriate action after finding the current optimal point. However, setting a low value for ϵ may cause oscillation in the agent's action, and a high value for ϵ may cause the agent to learn slowly.

B. Training Algorithm

PPO [25] uses a clipped surrogate objective and actor-critic loss to enhance exploration and sample efficiency. Algorithm 1 outlines the training process for our agent, which learns a policy function $\pi(a|\mathbf{s};\theta)$ for the actor. The algorithm runs for N episodes, during which the agent chooses actions based

Algorithm 1: Actor-critic (PPO) algorithm for learning a policy to optimize the available network bandwidth utilization

Input : Receiver IP and $PORT$, file directory DIR, $transferEnvironment$,ϵ

Output : A stochastic policy function denoted as $\pi(a \mid \mathbf{s}; \theta)$ that determines the number of TCP streams to use based on the state of the system s and a value function denoted as $V(\ \mathrm{s}; \theta_v)$ that estimates the value of a particular state s. The stochastic policy function outputs the action a from the set of possible actions $\mathcal{A}$, given the state s, and policy function's parameters are denoted by θ. On the other hand, the value function's parameters are denoted by θ_v

1 #**Initialization**
2 Randomly initialize the model parameters θ, θ_v
3 Maximum number of episodes N
4 #**Training**
5 $n \leftarrow 0$
6 **while** $n < N$ **do**
7 $s \leftarrow \mathrm{Reset}\ (transferEnvironment)$
8 $done \leftarrow False$
9 **while** $done \neq$ *True* **do**
10 $s \leftarrow State(transferEnvironment)$
11 $a \leftarrow \pi(a|s;\theta)$
12 $S, R \leftarrow \mathrm{Transfer}(s, a)$
13 #**added state, action and reward until the episode finishes** $S = (s, a, R)$ **with** $s \subseteq Statespace, a \subseteq actionspace$ **and** r **is reward.** $done$ **is** $True$ **once episode ends**
14 Reset the gradients to zero for the policy function and value function: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$
15 **for** $(s, a, R) \in$ *StateIterator* (S) **do**
16 #**Compute the ratio of new to old policy:**
17 ratio $\leftarrow \frac{\pi(a|s;\theta)}{\pi(a|s;\theta_{\text{old}})}$
18 #**Compute the PPO surrogate objective:**
19 $L(\theta) \leftarrow \min(\text{ratio} \cdot A_t, \text{clip}(\text{ratio}, 1-\epsilon, 1+\epsilon) \cdot A_t)$
20 #**Compute the gradients of the policy function:**
21 $d\theta \leftarrow d\theta + \nabla_\theta L(\theta)$
22 #**Compute the gradients of the value function:**
23 $d\theta_v \leftarrow d\theta_v + \partial(R - V(s;\theta_v))^2/\partial\theta_v$
24 Perform update of θ using $d\theta$ and θ_v using $d\theta_v$ and store old values.
25 $n \leftarrow n + 1$

Subroutines: Reset $(transferEnvironment)$: Return the $transferenvironment$ to its original state
Subroutines: State $(transferEnvironment)$: Get the $transferEnvironment$'s current state
Subroutines: $\mathrm{Transfer}(s, a)$: Apply action a to State s, and return the reward achieved
Subroutines: $StateIteratoor$ (S): Iterate over all the $\langle state, action, reward\rangle$ tuple stored in S after an episode

on the current policy and receives feedback performance signals to calculate rewards (lines 7-13). It then stores the $\langle state, action, reward\rangle$ tuple for each episode in object S. After each rollout, the algorithm aggregates gradients to update the actor and critic network parameters. Lines (17-19) calculate the policy ratio and the PPO objective, which constrains policy updates within a specified boundary to ensure stability. The policy gradient loss in line 21 directs the modification of θ to enhance the expected reward, while in line 23, the difference between the rollout reward R and the state value $V(s;\theta_v)$ is subtracted to reduce gradient variance [27]. The value V is concurrently trained to decrease its prediction error. PPO's computational demands primarily arise from processing episode data and subsequent optimization steps. The time complexity of the algorithm is $O(N)$, where N is the number of episodes as during training, the average length of each episode is constant.

IV. Evaluation

This section presents evaluation results for our DRL agent's instantaneous throughput dynamics, throughput performance, fairness metrics, and convergence* dynamics of multiple transfers. We compare the performance of our solution with two other online optimization algorithms: Gradient Descent (GD) and Bayesian Optimizer (BO). For an in-depth discussion of the GD and BO configurations, please refer to [23].

A. Experimental Setup

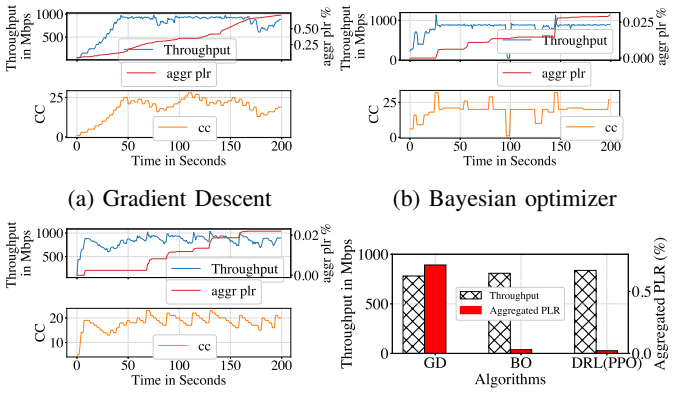
Our learning-based model was trained and tested in two wide-area network testbeds: (1) Chameleon Cloud [28] with the server at the University of Chicago and the client at the Texas Advanced Computing Center ; and (2) CloudLab [29], with the server at the University of Wisconsin and the client at the University of Utah. Chameleon nodes operate on Dell PowerEdge R740 with Intel Xeon Skylake CPUs, 192 GiB RAM and 10 Gbps network interface card(NIC). The CloudLab client uses an HPE ProLiant XL170r server with an Intel E5-2640v4 processor and 64 GiB RAM with 1 Gbps NIC. Transfers were conducted using $500 \times 1GB$ memory-to-memory files.

B. Instantaneous Throughput Dynamic of Single Transfer

To assess the optimization algorithms' performance in finding the optimal point and their behavior afterward, we limited the achievable throughput of each TCP stream to 50 Mbps (i.e., throttling), and restricted the network to have only the sender and receiver nodes (i.e., no-background traffic). In a 1000 Mbps network, twenty parallel TCP streams (i.e., CC) are required to fully utilize the link. Running more than twenty streams will result in a lower utility value due to an increase in packet loss rate and the divisor term in equation 3 (i.e., K^{n_i}). The algorithms were run for 200 seconds each. It's noteworthy that we initially experimented with the deep Q-network (DQN) algorithm. However, due to the network's inherent stochasticity, partial observability, and extensive state space, DQN struggled to adapt and learn effectively. Contrarily, the PPO algorithm learned well under these conditions, proving more adept at managing the environment's complexities, hence it became our algorithm of choice.

Gradient Descent (GD) peaks in 50 seconds from CC=2 (Figure 1a), constantly probing for network condition changes. Bayesian Optimizer (BO) attains the optimum at 30 seconds (Figure 1b), quickly zoning into a concurrency of 20 after some random samplings. It occasionally revisits the search space, constrained by its surrogate model's past observations. The DRL agent, trained over 5,000 episodes in-total 28 hours, pinpoints the optimal point in 12 seconds (Figure 1a), adjusting CC around 20 to account for potential network shifts. Figure 1d shows the DRL agent achieving a peak throughput of 837 Mbps during the $500 \times 1GB$ transfer, surpassing BO's 807 Mbps and GD's 780 Mbps, while also maintaining the lowest packet loss.

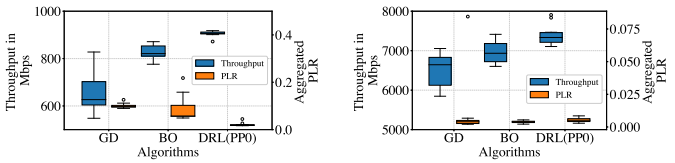
*Convergence refers to the point where each transfer's throughput stabilizes and becomes relatively constant. This happens when the transfers find a fair allocation of the available bandwidth and share it equally.



(a) Gradient Descent (b) Bayesian optimizer

(c) DRL(PPO) (d) PLR and Throughput

Fig. 1: Performance (i.e., throughput and aggregated packet loss rate) comparison of Gradient Descent, Bayesian Optimization, and Reinforcement Learning (PPO) where the optimal number of the parallel stream (i.e., CC) is 20.

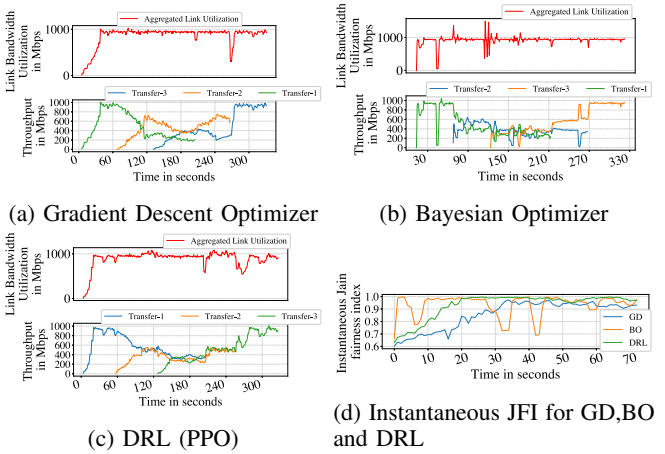


(a) Cloudlab (b) Chameleon

Fig. 2: The aggregated packet loss rate and throughput for various algorithms in two different testbeds - Cloudlab with 1Gbps link capacity and Chameleon with 10Gbps link capacity. Each algorithm was tested for 10 runs, transferring a total of $500 \times 1GB$ files. The results indicate that the DRL agent outperformed other algorithms, achieving approximately 15% higher mean throughput while maintaining a lower aggregated packet loss rate in both testbeds.

The reported results favor the DRL agent since it has the capability to learn and generalize patterns in network conditions based on the network signal vectors and adjust the number of CC effectively. The non-linear form of the utility function enables the DRL agent to learn how to maximize the use of the available network bandwidth while avoiding actions that can lead to network congestion. We retrained the same DRL agent for an additional 1000 episodes for cloudlab and chameleon testbeds, respectively, while removing the TCP stream's bandwidth limitation (i.e., throttling). This retraining fine-tuned the DRL agent's policy to adapt to the new testbed environment and changes in bandwidth capacity (e.g., cloudlab has a 1Gbps link capacity, while Chameleon has 10 Gbps). Figure 2 displays the cloudlab and chameleon testbed performance for ten runs for each algorithm, where each run transfers a total of $500 \times 1GB$ files. As shown in the box plots, the DRL agent achieved around 15% more mean throughput while maintaining a lower aggregated packet loss rate in both testbeds.

While the PPO agent requires retraining for each environment, indicating limited generalization, it exhibits improved data efficiency. Specifically, adaptation to a new environment necessitates only 1000 training episodes, a marked reduction



(a) Gradient Descent Optimizer (b) Bayesian Optimizer

(c) DRL (PPO) (d) Instantaneous JFI for GD,BO and DRL

Fig. 3: Dynamic throughput while multiple transfers share the same network resource. In Figure 3d, the instantaneous Jain fairness index is depicted for all three algorithms when all three transfers for each algorithm occur simultaneously on the shared network.

from the initial 5000 episodes. The balance between data efficiency and generalization capability of our DRL agent is a point of ongoing exploration for our group.

C. Fairness and Convergence of Multiple Transfer

The three algorithms' fair resource sharing and convergence dynamics were evaluated in the cloudlab testbed while multiple transfers were initiated over a single bottleneck link, with each transfer controlled by a separate instance of a different optimizer, namely, GD, BO, or DRL. Jain's fairness index(JFI) [30] quantifies fairness among competing transfers, measuring resource allocation (e.g., bandwidth). Calculated as the ratio of squared total resource allocation to the sum of squared individual allocations, its value ranges from 0 (indicating inequality) to 1 (representing perfect fairness where each user receives an equal resource share). Additionally, each TCP stream's achievable throughput was limited to 50 Mbps.

In Figure 3a, three transfers employing the GD optimizer were initiated approximately 50 seconds apart. The start of each subsequent transfer affected the throughput of the previous ones, but they never stabilized. Specifically, over 200 seconds, transfers 1, 2, and 3 achieved average throughput of 471 Mbps, 465 Mbps, and 479 Mbps respectively. The overall link bandwidth usage averaged 866 Mbps, with a fairness (JFI) value of 0.90.[†] In contrast, Figure 3b reveals that BO is optimized for better fairness and convergence compared to GD. When subsequent transfers were introduced, they nearly matched the previous transfer's throughput with occasional fluctuations. Their 200-second averages were 492 Mbps, 369 Mbps, and 566 Mbps for transfers 1, 2, and 3, respectively. The link's average bandwidth usage was 921 Mbps, with a JFI value of 0.99. Notably, at timestamp 150, BO's aggressive approach caused noticeable network congestion.

[†]Ideally, transfer 2, always sharing the link, should have the lowest average throughput. Transfers 1 and 3, having sole link access for roughly 50 seconds each, should have similar and higher throughput.

Meanwhile, as depicted in Figure 3c, when three independent DRL agents shared the link, they converged to the fair share of the available link bandwidth. Their respective throughputs over 200 seconds were 554 Mbps, 364 Mbps, and 524 Mbps, resulting in a link usage of 893 Mbps and a JFI of 0.97. The reason DRL achieved better convergence and fair share is due to the incorporation of the bounded historical network signal vectors in the DRL agent's state. This allowed the DRL agent to quickly detect patterns in network changes and adjust the number of TCP streams to avoid network congestion. In Figure 3d, the instantaneous JFI is depicted for all three algorithms during the period when all three transfers are competing for the available network link. This specific event takes place after roughly at 150-second, as shown in Figures 3a, 3b, and 3c. A noteworthy observation is that the DRL agent demonstrates a more stable and consistent instantaneous JFI in comparison to the GD and BO algorithms.

V. Conclusion

Parallel TCP streams can significantly increase application throughput by leveraging a larger virtual message segment size (MSS). However, using too many TCP streams without proper consideration can lead to network congestion and unfairness. Moreover, the optimal number of parallel TCP streams varies dynamically and depends on unpredictable background traffic conditions. In this study, we demonstrate how deep reinforcement learning (DRL) can determine the ideal number of parallel TCP streams for any transfer, without relying on rule-based heuristics. Our approach leverages historical network signal vectors to adapt to various network conditions and optimize network bandwidth utilization. Our DRL policy-gradient method outperforms other online optimization algorithms, achieving near-optimal points 40% faster and up to 15% better throughput. Additionally, we introduce a punishment term in the utility function to prevent network congestion and promote fairness among concurrent transfers. As a future direction, we aim to reduce the DRL agent's initial training time and explore the possibility of multi-parameter optimization. Our goal is to extend the DRL agent's capabilities beyond parallel TCP streams to support other file transfer performance parameters such as the level of pipelining, the number of CPU cores involved, and the CPU frequency level.

Acknowledgements

This project is in part sponsored by the National Science Foundation (NSF) under award number CCF-2007829.

References

[1] E. Deelman, T. Kosar, C. Kesselman, and M. Livny, "What makes workflows work in an opportunistic environment?" *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1187–1199, 2006.

[2] Y. Kim, S. Atchley, G. R. Vallée, and G. M. Shipman, "{LADS}: Optimizing data transfers using {Layout-Aware} data scheduling," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015, pp. 67–80.

[3] H. Jamil, L. Rodolph, J. Goldverg, and T. Kosar, "Energy-efficient data transfer optimization via decision-tree based uncertainty reduction," in *ICCCN*, 2022, pp. 1–10.

[4] J. Kim, E. Yildirim, and T. Kosar, "A highly-accurate and low-overhead prediction model for transfer throughput optimization," *Cluster Computing*, vol. 18, pp. 41–59, 2015.

[5] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic, "Parallel tcp sockets: Simple model, throughput and validation," in *INFOCOM 2006*, 2006.

[6] D. Lu, Y. Qiao, P. Dinda, and F. Bustamante, "Modeling and taming parallel tcp on the wide area network," in *IPDPS*, 2005.

[7] T. Hacker, B. Athey, and B. Noble, "The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network," in *IPDPS*, 2002.

[8] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," vol. 42, no. 5. New York, NY, USA: Association for Computing Machinery, jul 2008, p. 64–74.

[9] A. Gurtov, T. Henderson, and S. Floyd, "The NewReno Modification to TCP's Fast Recovery Algorithm," ser. Request for Comments, no. 3782. RFC Editor, Apr. 2004.

[10] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based congestion control for the internet," in *NSDI 18*. Renton, WA: USENIX Association, Apr. 2018, pp. 329–342.

[11] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "Fast tcp: Motivation, architecture, algorithms, performance," vol. 14, no. 6, 2006.

[12] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the tcp congestion avoidance algorithm," vol. 27, no. 3. New York, NY, USA: Association for Computing Machinery, jul 1997, p. 67–82.

[13] T. Lakshman and U. Madhow, "The performance of tcp/ip for networks with high bandwidth-delay products and random loss," 1997.

[14] T. Kosar, *Data placement in widely distributed systems*. The University of Wisconsin-Madison, 2005.

[15] L. Di Tacchio, M. S. Q. Z. Nine, T. Kosar, M. F. Bulut, and J. Hwang, "Cross-layer optimization of big data transfer throughput and energy consumption," in *2019 IEEE CLOUD*, 2019.

[16] M. Balman and T. Kosar, "Data scheduling for large scale distributed applications," in *the 5th ICEIS Doctoral Consortium, In conjunction with the International Conference on Enterprise Information Systems (ICEIS'07). Funchal, Madeira-Portugal*, 2007.

[17] L. Rodolph, M. S. Q. Zulkar Nine, L. Di Tacchio, and T. Kosar, "Energy-saving cross-layer optimization of big data transfer based on historical log analysis," in *IEEE ICC 2021*, 2021, pp. 1–7.

[18] C. Tessler, Y. Shpigelman, G. Dalal, A. Mandelbaum, D. H. Kazakov, B. Fuhrer, G. Chechik, and S. Mannor, "Reinforcement learning for datacenter congestion control," *CoRR*, vol. abs/2102.09337, 2021.

[19] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, "Qtcp: Adaptive congestion control with reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 445–458, 2019.

[20] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, oct 2016.

[21] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC vivace: Online-Learning congestion control," in *NSDI 18*, Renton, WA, 2018.

[22] M. Balman and T. Kosar, "Dynamic adaptation of parallelism level in data transfer scheduling," in *2009 International Conference on Complex, Intelligent and Software Intensive Systems*. IEEE, 2009, pp. 872–877.

[23] M. Arifuzzaman and E. Arslan, "Online optimization of file transfers in high-speed networks," in *Proceeding of Super*. ACM, 2021.

[24] P. Balaprakash, V. Morozov, R. Kettimuthu, K. Kumaran, and I. Foster, "Improving data transfer throughput with direct search optimization," in *ICPP*, 2016.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms." arXiv, 2017.

[26] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proceedings of ICML*, vol. 97, 09–15 Jun 2019, pp. 3050–3059.

[27] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 1999.

[28] K. Keahey, J. Anderson, Z. Zhen, and et al., "Lessons learned from the chameleon testbed," in *Proceedings of USENIX ATC'20*, July 2020.

[29] D. Duplyakin, R. Ricci, A. Maricq, and et al., "The design and operation of CloudLab," in *Proceedings of the USENIX ATC*, Jul. 2019.

[30] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," 1998.