A Deep Deterministic Policy Gradient Based Network Scheduler For Deadline-Driven Data Transfers

Gaurav R. Ghosal¹, Dipak Ghosal²,³, Alex Sim³, Aditya V. Thakur², and Kesheng Wu³

¹University of California, Berkeley, CA, U.S.A.
²University of California, Davis, CA, U.S.A.
³Lawrence Berkeley National Laboratory, Berkeley, CA, U.S.A.

Abstract—We consider data sources connected to a software defined network (SDN) with heterogeneous link access rates. Deadline-driven data transfer requests are made to a centralized network controller that schedules pacing rates of sources and meeting the request deadline has a pre-assigned value. The goal of the scheduler is to maximize the aggregate value. We design a scheduler (RL-Agent) based on Deep Deterministic Policy Gradient (DDPG). We compare our approach with three heuristics: (i) P_FAIR, which shares the bottleneck capacity in proportion to the access rates, (ii) V_D_Ratio, which prioritizes flows with high value-to-demand ratio, and (iii) V_B_EDF, which prioritizes flows with high value-to-deadline ratio. For equally valued requests and homogeneous access rates, P_FAIR is the same as an idealized TCP algorithm, while V_B_EDF and V_D_Ratio reduce to the Earliest Deadline First (EDF) and the Shortest Job First (SJF) algorithms, respectively. In this scenario, we show that RL-Agent performs significantly better than P_FAIR and V_D_Ratio and matches and in over-loaded scenarios out-performs V_B_EDF. When access rates are heterogeneous, we show that the RL-Agent performs as well as V_B_EDF even though the RL-Agent has no knowledge of the heterogeneity to start with. For the value maximization problems, we show that the RL-Agent out-performs the heuristics for both homogeneous and heterogeneous access networks. For the general case of heterogeneity with different values, the RL-Agent performs the best despite having no prior knowledge of the heterogeneity and the values, whereas the heuristics have full knowledge of the heterogeneity and V_D_Ratio and V_B_EDF have partial knowledge of the values through the ratios of value to demand and value to deadline, respectively.

Index Terms—Deadline-driven data transfers, Software-defined Networking (SDN), Reinforcement Learning, DDPG, Scheduling heuristics, EDF, TCP, Value maximization

I. INTRODUCTION

Large distributed science is moving to the SuperFacility model [1] in which distributed instrument facilities, HPC systems, storage, and researchers are viewed as one integrated facility. This is achieved by interconnecting these resources using a very high-speed network, such as the ESnet [2], that must provide performance guarantees on the data transfers required by complex science workflows deployed on the SuperFacility. Furthermore, due to exponential growth in the amount of data that is generated by the instruments, it is also important that the interconnecting network is run at a high utilization [3].

Next generation science workflows will require complex processing pipelines that may be processed at different HPC facilities [4]. More and more these workflows must be completed within a deadline. For example, the Large Synoptic Survey Telescope (LSST) [5] will take more than 800 panoramic images each night with its 3.2 billion-pixel camera, recording the entire visible sky twice each week. The new data will be compared with previous images to detect changes in brightness and position of objects as big as distant galaxy clusters and as small as near-by asteroids. The processing of each 30 second image must be completed within 7.5 seconds so that necessary alerts can be generated for down-stream processing. The deadlines in processing the flows impose deadlines in the data transfer from the instrument facilities (that are typically in remote locations) to the HPC facilities. Furthermore, the data transfer deadlines are becoming increasingly tight. For example, in the Linac Coherent Light Source (LCLS) [6], the location of next image is a function of the current image.

Deadline driven data transfer is not unique to large scientific workflows and is becoming important for cloud service providers with geo-distributed data centers. In order to keep the distributed locations synchronized, large data sets must be periodically transferred within a deadline [7], [8] while requiring that the network is operating at a very high utilization [3]. This requires fine grain control of the transmission rate to eliminate contention and resulting packet losses.

To support deadline-driven data transfers, ESnet sets up on-demand circuits [9] supporting packet priority allowing the circuit to be used by other traffic when the deadline flow is inactive. For cloud service providers, the deadline transfers are scheduled over private/dedicated WANs that interconnect the geo-distributed data centers. These private dedicated WANs are less noisy and more predictable compared to the general Internet. Nevertheless, the deadline transfers must be scheduled in the presence of other background and interactive flows [7], [8].

In this paper, we consider an SDN-enabled pri-
vate/dedicated WAN with a centralized controller. There is also in-network telemetry (INT) that can provide real-time fine-grained information about network state including router buffer lengths and packet drops [10]. Deadline-aware data transfer requests are made to a central network controller, which schedules the flows by setting pacing rates at the sources of the deadline flows.

This work extends the Q-table based network scheduler presented in [11], which demonstrated the feasibility of using a Reinforcement Learning based approach to schedule deadline-driven flows. In particular, the Q-table based approach cannot be applied to large networks with heterogeneous links. In this paper, we design and implement a network scheduler for deadline-driven flows based on Deep Deterministic Policy Gradient (DDPG) and an Actor-Critic model, which is able to handle larger networks with heterogeneous links. Furthermore, this paper tackles a generalization of the scheduling problem that assigns a utility value to each flow if the deadline is met. The scheduler task is to schedule the flows so as to maximize the aggregated utility value. We design a scheduler (RL-Agent) based on DDPG and compare its performance with three heuristics: (i) P_FAIR, which shares the bottleneck capacity in proportion to the access rates, (ii) V_D_Ratio, which prioritizes flows with high value-to-demand ratio, and (iii) V_B_EDF, which prioritizes flows with high value-to-deadline ratio.

The main contributions of this paper are:

1) We propose a DDPG based network controller which a) scales with respect to the size of the network (number of sources and the bottleneck link capacity), b) can be applied to heterogeneous networks, and c) can be applied to a general value maximization problem.

2) For equally valued requests and homogeneous access rates, we show that the RL-Agent performs as well as and in some cases out-performs V_B_EDF (which is the same as the Earliest Deadline First (EDF) algorithm) and significantly better than P_FAIR (same as an idealized TCP algorithm).

3) For equally valued requests and heterogeneous access rates, we show that the RL-Agent performs as well as V_B_EDF even though it has no knowledge of the heterogeneity to start with.

4) For the value maximization problem, we show that the RL-Agent out-performs the heuristics for both homogeneous and heterogeneous access networks. This is despite the fact that the RL-Agent has no prior knowledge of the heterogeneity and the values, whereas the heuristics have full knowledge of the heterogeneity and V_D_Ratio and V_B_EDF have partial knowledge of the values.

II. SYSTEM MODEL

In this study we consider a heterogeneous dumbbell network consisting of N sources and N destination nodes, as illustrated in Figure 1. The capacity of the link between router R1 and R2 is B Gbps. The access links between the sources sn_i, i = 1...N and R1 have rates r_i ∈ (0, B). However, the rates of the last-hop links from R2 to the destination nodes D_i are greater than B and, hence, are never the bottleneck.

Requests for file transfers are made to the network controller. Each request j is a five-tuple (sn_j, d_n_j, f_j, d_j, v_j) where sn_j denotes the source node, d_n_j the destination node, f_j is the filesize, d_j is the deadline, and v_j is a value awarded if the data transfer is completed within the deadline. Each request corresponds to a single TCP/IP flow. We assume only a single request from each source at any given time.

The basic unit of time is a scheduling interval; the network controller assigns a pacing rate to each active source at the start of each scheduling interval. We consider an episodic (batch) model in which the scheduler receives a request from each of the sources at the beginning of an episode [11]. Each request is to a different destination. The episode ends when all the requests have completed successfully (within the deadline) or not, after which a new episode begins. An episode consists of an integer number of scheduling intervals denoted as 1, ..., T.

At the beginning of each scheduling interval, for each request j, we define Rmin_j(t) as [11]:

\[ Rmin_j(t) = \frac{\text{remaining file size}}{\text{time until deadline}} \]  \hspace{1cm} (1)

The value Rmin_j(t) denotes the minimum rate that is required at every subsequent scheduling interval for request j to meet its deadline. We let Rmin_j(1) = f_j, the initial minimum rate for request j, be denoted by Rmin_j.

We use Sum_Rmin = \sum_{j=1}^{N} Rmin_j to denote the sum of the initial Rmin of each request. Sum_Rmin < B, Sum_Rmin = B, and Sum_Rmin > B correspond to the under-loaded, fully-loaded, and over-loaded scenarios, respectively. Note that if Sum_Rmin ≤ B, it should be possible to meet all the deadlines. However, as flows may end anytime during a
scheduling interval and pacing rate assignments are done only at the beginning of the scheduling intervals, there is some capacity waste which can result in lower than 100% success rate for the fully-loaded and the under-loaded cases. In the over-loaded case, the probability that not all the flows can be completed by their deadline increases as the difference Sum $R_{\text{min}} - B$ increases. For the subsequent discussion we introduce the term demand of a request $j$ as $D_j = \lfloor \frac{d_j}{T} \rfloor$ [3]. This demand represents the number of intervals at the bottleneck link capacity that is required to transfer the file.

A. Justification of the Heterogeneous Dumbbell Network

The heterogeneous dumbbell network serves as a good abstraction of a large complex network such as the ESNet [2] due to the following reasons. First, many distributed instruments facilities are located at remote locations and it is the access links that connect these facilities to the core network that typically have low capacity. For example, the LSST cameras, which are located on top of a mountain in the Andes [5], have low capacity links connecting to the ESNet. Second, most of the ESNet links have very high capacity. There are very few links that are bottleneck links. For example, in ESNet the trans-continental links between the US and Europe is typically the only congested links. Finally, the destination nodes representing HPC systems or Data Transfer (DTN) nodes [12] are typically connected using very high speed networks and are not bottleneck.

B. Heuristic Scheduling Algorithms

We consider the following heuristics for comparison:

- **Proportional Fair Share Allocation (P_FAIR):** The P_FAIR heuristic shares the bottleneck link capacity fairly among the active sources in proportion to their access link rates. If there are $k$ active sources, then the pacing rate $a_i$ to source $i$ is:

$$a_i = \min \left( r_i, \frac{r_i}{\sum_{j=1}^{k} r_j} \times B \right)$$

(2)

If all the access rates are equal, then this would correspond to an equal partition. The P_FAIR heuristic simulates an idealized version of the TCP protocol and does not take into account the value of each flow.

- **Value-Demand Ratio Based Allocation (V_D_Ratio):** The V_D_Ratio heuristic is an adaptation of the Greedy RT algorithm outlined in Jain et al. [13]. The problem considered by Jain et al. was to schedule deadline driven jobs with different maximum parallelism (analogous to the heterogeneous access link rates) in large computing cluster of a fixed processing capacity $B$ so as to maximize the total value. The jobs are malleable implying that processing time scales with amount of capacity that is allocated (up to the maximum parallelism). The algorithm is greedy because it allocates high priority in allocating capacity to short jobs that have high values. It takes the deadline into account only indirectly by ignoring jobs that cannot meet the deadline. The key result is that the GreedyRT algorithm is $O\left(\frac{n}{m-k} \cdot \frac{s}{s-1}\right)$ approximation of the optimal, where $k$ is the maximum parallelism and $s$ is a slackness factor that denotes the degree of flexibility the scheduler has in scheduling a job. Consequently, as long as $s$ is large and $k \ll B$, this heuristic is near-optimal.

To the best of our knowledge, there is no prior research on V_B_EDF heuristic for the value maximization problem for deadline-driven jobs. This heuristic directly accounts for the deadline by prioritizing jobs that have high value and early deadline. As such it is the same as EDF when all the values are equal. EDF has been widely studied in scheduling deadline-sensitive jobs on multi-processor systems [14], [15] and for packet scheduling in multihop networks with hard deadlines [16]. For many of the above applications under certain constraints EDF achieves the same performance as the optimal offline algorithm [16]. For deadline driven data transfers in a heterogeneous network with multiple paths [8] and for the over-loaded cases [11], EDF is not optimal. Nevertheless, as shown in Section V, it out-performs the V_D_Ratio heuristic.

C. Notes about the Heuristics

The V_D_Ratio heuristic is adapted from the Greedy RTL algorithm outlined in Jain et al. [13]. The problem considered by Jain et al. was to schedule deadline driven jobs with different maximum parallelism (analogous to the heterogeneous access link rates) in large computing cluster of a fixed processing capacity $B$ so as to maximize the total value. The jobs are malleable implying that processing time scales with amount of capacity that is allocated (up to the maximum parallelism). The algorithm is greedy because it allocates high priority in allocating capacity to short jobs that have high values. It takes the deadline into account only indirectly by ignoring jobs that cannot meet the deadline. The key result is that the GreedyRT algorithm is $O\left(\frac{n}{m-k} \cdot \frac{s}{s-1}\right)$ approximation of the optimal, where $k$ is the maximum parallelism and $s$ is a slackness factor that denotes the degree of flexibility the scheduler has in scheduling a job. Consequently, as long as $s$ is large and $k \ll B$, this heuristic is near-optimal.

To the best of our knowledge, there is no prior research on V_B_EDF heuristic for the value maximization problem for deadline-driven jobs. This heuristic directly accounts for the deadline by prioritizing jobs that have high value and early deadline. As such it is the same as EDF when all the values are equal. EDF has been widely studied in scheduling deadline-sensitive jobs on multi-processor systems [14], [15] and for packet scheduling in multihop networks with hard deadlines [16]. For many of the above applications under certain constraints EDF achieves the same performance as the optimal offline algorithm [16]. For deadline driven data transfers in a heterogeneous network with multiple paths [8] and for the over-loaded cases [11], EDF is not optimal. Nevertheless, as shown in Section V, it out-performs the V_D_Ratio heuristic.

III. RL-AGENT BASED SCHEDULER

The overall architecture of the RL-agent is shown in Figure 2. In this section, we discuss the various components of
the architecture and how the scheduling task is represented for reinforcement learning.

A. Reinforcement Learning Formulation

In the following paragraphs we define the state and action spaces and the reward function.

a) State: The state $s_t$ of the system at the beginning of the scheduling interval $t$, is defined as

$$s_t = [\text{Rmin}_1(t), \text{Rmin}_2(t), \ldots, \text{Rmin}_N(t)],$$

where $\text{Rmin}_i(t)$ is defined in Eq. 1. When a flow $i$ completes or its deadline has expired, the $i$th component in the state vector is set to 0. For each flow $i$, the state gives the minimum pacing rate that the agent needs to allocate at every scheduling interval until the deadline for the flow $i$ to finish. This choice of the state gives the agent direct information about the urgency with which each flow must be allocated capacity in the bottleneck link.

b) Action: Action $a_t$ for the interval $t$ is defined as

$$a_t = [a_1(t), a_2(t), \ldots, a_N(t)],$$

where $a_i$ is the pacing rate allocated to the $i$-th flow. This definition of action is similar to the work in [11]. However, the key difference is that we consider $a_t \in \mathbb{R}^N$, where $N$ is the number of flows in the network. As such, this work considers the problem in a continuous action space. The use of a continuous action space formulation here, as opposed to the discrete formulation in [11] holds several key advantages. In the Q-table approach, each state-action pair had to be individually enumerated and visited by the agent in order to generate an estimate of the value. With an increasing network size, the training of the agent could become infeasible. The continuous formulation allows for similar state-action pairs to be generalized over, and thus efficient learning over a larger state-action space [17].

An action is valid if it satisfies the following constraints:

$$a_i(t) \geq 0, \quad i = 1 \ldots N$$

$$\sum_{i=1}^{N} a_i \leq B.$$  

The second constraint implies that at every scheduling interval the scheduler uses all of the capacity of the bottleneck link. In order to generate such actions from our reinforcement learning agent, we apply the softmax function to the output layer of the actor network and scale the result to the bottleneck capacity $B$.

c) Reward: In this reinforcement learning problem, we ultimately wish to maximize the number of flows that the agent is able to complete before their deadlines. This suggests that a flow completion reward could be appropriate, where the agent receives a positive reward every time a flow is completed within its deadline and a zero reward otherwise. In this case, however, the reward will be nonzero infrequently, resulting in slow training. As a result, here we consider reward functions that give the agent information at every time step.

The reward function utilized in [11] measured the extent to which the action prioritized the completion of flows with the earliest deadlines. This prior reward function is defined as:

$$R(s_t, a_t) = \sum_{i=1}^{N} (d_{max} - d_i) * a_i,$$  

where $d_i$ denotes the time until the deadline for the $i$th flow, $d_{max}$ is the maximum time until deadline for any flow and $a_i$ is the $i$th component of the action vector, or the amount of bottleneck capacity allocated to the $i$th flow. Under this reward function, for example, any allocation to the flow with the latest deadline contributes zero to the reward because $d_i = d_{max}$ and so $(d_{max} - d_i) = 0$.

However, the reward function defined in Eq. 7 depends only on the action that is specified by the agent and not on variables from the state itself. This means that the reward function does not depend on the effect that the action had in the environment. For heterogeneous access link rates, for example, not all actions specified by the agent may be possible to implement.

In this paper, we consider the following reward function:

$$R(s_t, a_t) = \sum_{i=1}^{N} (d_{max} - d_i) * \Delta f_i,$$  

where $d_{max}$ and $d_i$ have the same significance as the previous reward function. $\Delta f_i$ represents the difference between the file size remaining for flow $i$ before the action is taken and the file size remaining for flow $i$ after the action is taken. Thus, the $\Delta f_i$ term measures how much progress the action taken made toward finishing flow $i$. If the action taken by the agent was fully implemented over the unit time interval, then $\Delta f_i = a_i$ so Eq. 7 is a special case of this reward function.

In the case that each flow has a value, $v_i$ that represents how much preference is given to completing the flow, it is appropriate to scale the reward corresponding to a given flow.
by the value for that flow. This gives the most general case of the
reward function used in this paper:

\[ R(s_t, a_t) = \sum_{i=1}^{N} (d_{max} - d_i) \ast \Delta f_i \ast v_i \]  

(9)

When flows are equally valued, the \( v_i \) term can be ignored; thus, reward functions in Eqs. 7 and 8 are special cases of
this reward function.

**B. The DDPG Architecture**

The deep deterministic policy gradient algorithm utilizes an actor
critic architecture [17]. We construct an actor network
that takes as input the state vector described by Eq. 3 and
outputs an action vector described by Eq. 4 that satisfies Eqs. 5
and 6. Additionally, we construct a critic network that takes
in both the action and state vectors and returns an estimate
of \( Q(s_t, a_t) \), the value of taking action \( a_t \) at state \( s_t \). The
networks are trained using \((s_t, a_t, r_t, s_{t+1})\) tuples sampled
randomly from a store termed the replay buffer.

**a) Actor Network:** The actor network aims to learn a
policy function \( \pi_{\theta_{\text{actor}}} : S \rightarrow A \) that maximizes the total
expected discounted reward given by:

\[ \mathbb{E}[\sum_{t=0}^{T} \gamma^{t-t} \ast R(s_t, a_t)] \],  

(10)

where \( S \) is the set of vectors described by Eq. 3 and \( A \) is
the set of vectors described by Eq. 4, obeying restrictions in
Eqs. 5 and 6. This expectation computes the total reward that
is expected to be received by following a given policy, where
rewards occurring later are discounted by a constant hyper
parameter, \( \gamma \), between 0 and 1. A gradient ascent process
is utilized to learn the parameters \( \theta_{\text{actor}} \) that maximize this
expectation, by following the gradient given by

\[ \mathbb{E}[\nabla_{a_t} Q(s_t, a_t) \ast \nabla_{\theta_{\text{actor}}} \pi_{\theta_{\text{actor}}}(s_t)] \]  

(11)

In this formula, the \( \nabla_{a_t} Q_{\theta_{\text{critic}}}(s_t, a_t) \) is based off of a
learned approximation of the \( Q_{\theta_{\text{critic}}}(s_t, a_t) \) function in
the critic network. The parameters of the actor network are
updated using the following equations:

\[ \theta'_{\text{actor}} = \theta_{\text{actor}} + \alpha \ast \mathbb{E}[\nabla_{a_t} Q(s_t, a_t) \ast \nabla_{\theta_{\text{actor}}} \pi_{\theta_{\text{actor}}}(s_t)] \],  

(12)

where \( \alpha \) represents a constant positive learning rate.

**b) Critic Network:** The critic network is trained to learn
the function \( Q(s_t, a_t) \), which is the expected reward from
taking action \( a_t \) during the state \( s_t \) and then following the
policy \( \pi_{\theta_{\text{actor}}} \). In the context of the reinforcement learning
formulation presented here, the critic network aims to learn
the extent to which an action taken at a given state would con-
tribute to the earliest deadline flow being completed first. The
critic network was trained to approximate a target \( Q(s_t, a_t) \)
value given by the Recursive Bellman Equation, as follows:

\[ Q_{\text{Bellman}}(s_t, a_t) = R(s_t, a_t) + \gamma \ast Q_{\theta_{\text{critic}}}(s_{t+1}, a_{t+1}) \]  

(13)

where \( Q_{\theta_{\text{critic}}} \) indicates the value that is taken from the critic
network parameterized by parameters \( \theta_{\text{critic}} \). These param-
ters, \( \theta_{\text{critic}} \), are updated to minimize the squared error between
\( Q_{\theta_{\text{critic}}}(s_t, a_t) \) and \( Q_{\text{Bellman}}(s_t, a_t) \). Thus, the parameters
are updated as follows:

\[ \theta_{\text{critic}}^{\text{updated}} = \theta_{\text{critic}} - \nabla_{\theta_{\text{critic}}} \alpha + \mathbb{E}[(Q_{\theta_{\text{critic}}}(s_t, a_t) \]  

(14)

\[ - Q_{\text{Bellman}}(s_t, a_t))^2] \]

c) Target Networks: As seen in the update equations
for the actor and critic networks, the networks are updated
directly and indirectly based off of their current values. For
example, the update to the critic network depends on the value
of the critic network itself. This has been identified as being
susceptible to divergence and instability [17]. One innovation
is to create separate target networks for both the actor and
critic networks that are used to compute values for the network
updates. The parameters of the target networks slowly trail
those of the actual networks through the following update:

\[ \theta = \tau \ast \theta + (1 - \tau) \ast \theta'^T \]  

(15)

where \( \tau \) is a positive constant between 0 and 1. By utilizing
this separate network for calculating the updates for the
network parameters, we ensure the stability of the network
updates and avoid divergence [17].

**d) Replay Buffer:** When training the critic network on
samples directly taken from the agent’s trajectory, there exist
temporal correlations between the training samples that could
induce a higher variance in the critic network’s approximation
of the true \( Q(s_t, a_t) \) function. This can lead to divergence of
the critic network. An approach for addressing this issue is
the use of a replay buffer in which \((s_t, a_t, r_t, s_{t+1})\) tuples
are stored [17]. At each training step, a mini-batch of these
tuples are sampled with uniform probability from the replay
buffer and utilized for the update. This breaks the temporal
correlations between the samples and upholds the independent
and identically distributed (i.i.d) assumption expected by the
stochastic gradient ascent algorithms utilized [18].

**e) Exploration Noise:** An important tool employed in re-
forcement learning applications is exploration noise, where
the action generated by the reinforcement learning agent is
perturbed by random noise prior to being implemented. Here,
the effect of exploration noise is to vary the pacing rates that
are assigned to each flow in a random fashion. Exploration
noise ensures that the reinforcement learning agent is able
to explore different regions within the state-action space,
preventing the agent from being trapped in a local optima.
Like other DDPG applications, we make use of an additive
Ornstein-Uhlenbeck temporally-correlated noise process [17].
Following the addition of exploration noise, the action was
re-scaled to meet the restrictions in Eqs. 5 and 6.

**IV. METHODOLOGY**

**A. Tool**

A prototype RL-Agent was implemented using the Tensor-
Flow machine learning library [19]. We generated actor and
critic networks consisting of an input layer, followed by a fully-connected hidden layer with 400 units, followed by a fully connected hidden layer of 300 units. A network simulator was developed using the NumPy numerical library which, given a value of the desired Sum_Rmin of all of the requests, generated request file sizes and deadlines and simulated an episode by generating the state at each scheduling interval, taking and implementing the action, and returning the reward signal to the agent.

In this work, the RL-Agent was trained on a single network configuration. The RL-Agent was trained in an iterative manner with the environment initialized with randomly generated requests of a given Sum_Rmin workload level. The agent would step through the scheduling intervals in the environment until every flow was completed or expired and the episode terminated. Afterwards, a new episode was generated and training continued. In this work, we allowed training to continue up to 25,000 training episodes. However, the learning curves demonstrate that the performance converges well before then (in approximately 5000 episodes).

### B. Metrics

We have used the following metrics in this study

1) **Evaluation Run**: Periodically during the RL-agent training, an evaluation run of 100 episodes was conducted. During these 100 episodes, no exploration noise was added to the actions and no network updates were performed.

2) **Success Rate**: This is defined as the fraction of the number of requests that meet the deadline. For the RL-Agent, this was reported as an average over the 100 episode evaluation run.

3) **Total Episode Value**: This is defined as the sum of the values of all completed flows. For the RL-Agent, this is averaged over the 100 episode evaluation run.

### C. Parameters

Table I gives a summary of all the hyper-parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Typical values</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_rb</td>
<td>Replay Buffer Size</td>
<td>$10^6$</td>
</tr>
<tr>
<td>size_mb</td>
<td>Mini-batch Size</td>
<td>64</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Target Network Update Weight</td>
<td>0.001</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Bellman Equation Discount Factor</td>
<td>0.99</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Learning Rate</td>
<td>0.001</td>
</tr>
</tbody>
</table>

In the following section unless otherwise stated the file size for each request is sampled from Uniform distribution of range 2 to 50 Gbs (Gigabits).

### V. Results and Discussion

#### A. Homogeneous Link Rates with Equally Valued Flows

We first considered the case of five sources with homogeneous access link rates of 20 Gbps and the bottleneck link of 20 Gbps capacity. Workloads with different Sum_Rmin values corresponding to under-loaded, fully-loaded, and over-loaded cases were generated. The learning curves for three cases 1) Sum_Rmin = 15 Gbps, 2) Sum_Rmin = 20 Gbps, and 3) Sum_Rmin = 25 Gbps are shown in Figure 3. The results show that the RL-Agent’s performance stabilizes quickly and as expected to different values. The Success Rate performance of the RL-Agent compared to the three heuristics are shown in Figure 4. The RL-Agent achieves superior performance to P_FAIR and V_D_Ratio heuristics. Moreover, it is able to achieve comparable performance to the V_B_EDF heuristic for under-loaded and fully-loaded cases and out-perform for over-loaded cases. These results replicate the work done in [11], utilizing a DDPG architecture as opposed to a Q-table. Note that the inferior performance of V_D_Ratio in
this and the subsequent cases is because the requirement 
\[ k(= \max\{r_1, \ldots, r_N\}) \ll B \] 
is not met.

One key advantage of the continuous action formulation 
of the DDPG model is that the RL-Agent scales well to 
networks with more sources and larger bottleneck capacities. 
We considered a homogeneous network with 7 sources 
and a bottleneck capacity of 40 Gbps. This configuration 
was infeasible with the simple Q-table approach in [11]. 
As before we considered a range of Sum_Rmin values corre-
spanding to the under-loaded, fully-loaded and overloaded 
cases. The results shown in Figure 5 demonstrate that the 
scheduler is able to achieve a Success Rate similar to the 
5 source case. The RL-Agent achieves performance close 
to the V_B_EDF heuristics, for the under-loaded and fully-
loaded cases and out-performs it in the over-loaded cases 
(Sum_Rmin > 40). It is notable that only minor changes 
had to be made to allow the agent to accommodate the 
network with 7 sources. Specifically, the input and output 
layers of the (neural) network were modified to match the 
number of sources and the exploration noise variance was 
scaled proportionally to the bottleneck capacity B. However, 
the training time for the network remained the same as in 
the case of the 5 source network. Hence, the RL-Agent can be 
applied in even larger networks.

B. Heterogeneous Link Rates with Equally Valued Flows

In the case of networks with heterogeneous access rates, 
the pacing rate assigned to a source may be larger than its access rate. 
As a result, the heterogeneity must be taken into account 
to avoid wasting the bottleneck capacity. To investigate this, 
we trained RL-Agent for a heterogeneous network with 5 
sources with access rates (20, 20, 20, 20, 0). This is an extreme 
case as requests from the source with 0 access rate can never 
be completed. This is particularly disadvantageous to the RL-
Agent since it does not have direct information about the 
heterogeneity. This case would give a sharp contrast of the 
performance of the RL-Agent in comparison to the heuristics 
if it did not adapt to the heterogeneity.

We considered both the reward function in Eq. 7 [11] as 
well as the new generalized reward function given in Eq. 8. 
This new reward function implicitly gives the agent information 
about the heterogeneity. We examined the correlation be-
tween the total reward that an agent earned over an evaluation 
episode and the number of flows that it completed. The total 
episode reward when utilizing the reward function in Eq. 7 had 
a weak correlation (correlation coefficient \( r = 0.362 \)) with 
the number of flows that the agent successfully completed. 
Utilizing the new reward function in Eq. 8, on the other hand, 
resulted in a strong positive correlation (\( r = 0.857 \)) 
between episode reward and number of completed flows. The 
low correlation value indicates that the reward function in 
Eq. 7 is unsuitable for use in maximizing the number of 
flows completed. This finding motivated the modification of 
the reward function to that shown in Eq. 8. In Figure 6, the 
learning curve of the RL-Agent trained with reward function 
in Eq. 8 is shown in comparison to the learning curve for the 
homogeneous network for the fully-loaded case. The Episode 
Flow Completion Rate for the heterogeneous case is corrected 
to account for the fact that it could at most complete 4 flows. 
The similarity of the learning trends establishes that the reward 
function in Eq. 8 is appropriate for training the RL-Agent for 
a heterogeneous network.

We next considered a network with 5 sources with \( B = 20 \) Gbps 
and access rates of (20,20,20,20,0) and compared the 
performance of the RL-Agent to the heuristics for different 
Sum_Rmin values. Results, displayed in Figure 7, indicate 
that the RL-Agent attains superior performance to the P_FAIR
and V_D_Ratio heuristics and comparable performance to V_B_EDF. When comparing the results, it is important to note that while the heuristics had full knowledge of the heterogeneity, the RL-Agent could only learn this through the reward signal.

Fig. 7: Success ratio as a function of Sum_Rmin for the RL-Agent and three heuristics. Parameters: \( B = 20 \) Gbps and \( N = 5 \). Access rates = \([20, 20, 20, 20, 0]\).

C. Flows with Different Values

In the case of differently valued flows, the total episode value metric is used. We considered the same network of 5 sources with homogeneous access link rates of 20 Gbps and a 20 Gbps bottleneck link. Sources were each assigned a unique integer value between 1 and 5. During the training process, the value associated with each source remained constant. The performance of the RL-Agent in comparison to the heuristics is displayed in Figure 8. The RL-agent surpasses P_FAIR, V_D_Ratio, and the V_B_EDF although RL-agent had to learn about the different values through the reward.

TABLE II: Average Episode Value for heterogeneous access links rate given in \( R \) and different value assignments given in \( V \). Parameters: \( B = 20 \), \( N = 5 \). Sum_Rmin = 20.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>P_FAIR</th>
<th>V_D_Ratio</th>
<th>V_B_EDF</th>
<th>RL-Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R=(20,20,20,20,0) ) ( V=(5,4,3,2,1) )</td>
<td>10.14</td>
<td>11.53</td>
<td>12.62</td>
<td>12.93</td>
</tr>
<tr>
<td>( R=(20,20,20,20,0) ) ( V=(1,2,3,4,5) )</td>
<td>7.24</td>
<td>8.23</td>
<td>8.96</td>
<td>9.14</td>
</tr>
</tbody>
</table>

We examined the value maximization problem utilizing the heterogeneous network with access rates \([20,20,20,20,0]\). This heterogeneity was chosen as an extreme case, as failure of the agent to learn about the heterogeneity would result in a definite drop in performance. We considered two value assignments, one in which the source with 0 access rate had the highest value (5) and the other in which it had the lowest value (1). The average episode value for the RL-Agent as well as the heuristics is presented in Table II. The RL-Agent is able to out-perform all the heuristics. This demonstrates the significant robustness of the RL-Agent as it had no prior knowledge of both the heterogeneity and the values, whereas the heuristics had full knowledge of the heterogeneity. V_D_Ratio and V_B_EDF had partial knowledge of the value (through the ratios of value to demand and value to deadline, respectively) while P_FAIR did not take into account value.

VI. RELATED WORK

Deadline-aware flow scheduling has been investigated for inter-datacenter traffic [7], [8], [20], for science workflows [21], and for data center networks [22]. The study in Tempus [8] considered a mix packing and covering problem to develop an online traffic engineering framework to efficiently schedule and route long flows to maximize the fraction of transfer delivered before deadline. The study in [7] formulated an optimization problem and proposed a framework called Ameoba for flexible bandwidth allocation to meet deadlines. The study in [21] considered a multihop network with deadline-driven requests and compared two classes of heuristics - one that optimized globally and one that optimized locally at each link - and compared the success rate in meeting deadlines and network utilization. The study illustrated the complexity of the scheduling problem for a general network.

Deadline-based packet transfers over multi-hop networks have been studied [16]. Finding optimal policies for deadline-driven transfers is a complex problem with multi-hop networks where contention can occur at multiple links [16]. Scheduling deadline-driven malleable job on a compute cluster has been investigated [3]. As mentioned in Section II, this problem is similar to the deadline-driven data transfers problem considered in this paper. However, the proposed heuristic is for a special case and hence does not perform very well when applied to the problem considered in this paper.

There has been significant recent work in applying RL-Agent based approaches for resource allocation in wireless
networks and job scheduling in computing systems. The study in [23] demonstrated that an RL-Agent can be trained to optimize scheduling jobs with processor and memory demands. The study demonstrated that the RL-Agent can perform as well as and sometimes out-perform known optimal heuristics.

This paper builds on [11] and significantly extends it in multiple dimensions. First, using a DDPG (as opposed to a simple Q-table in [11]) addressed a critical problem of scalability. The framework now allows investigating more general network with heterogeneous links with a large number of sources and higher bottleneck link capacity as has been demonstrated in this paper. Second, the generalization of the reward function has allowed formulating the more general value maximization problem. Finally, the results have been compared with both new and known near-optimal heuristics for a heterogeneous access network.

VII. CONCLUSIONS & FUTURE WORK

This paper describes a DDPG based reinforcement learning agent that is capable of scheduling deadline driven flows and achieves performance that matches or outperforms well-known and near-optimal heuristics. In particular, the DDPG method consistently attains superior performance in the over-loaded cases. There are multiple avenues for further work with regards to this specific scheduling problem. One possible area of investigation is identifying exploration noise processes that allow for the state-action space to be efficiently covered, as well as the optimal settings for the parameters of these processes. Although the Ornstein Uhlenbeck noise used in this paper has been shown as effective in physical control tasks [17], it remains unclear whether it is best suited to scheduling tasks, such as this one. Further investigation can also be conducted in determining the sensitivity of the agent’s performance to a wider range of hyper parameter settings. These more general investigations could potentially help in the use of DDPG in other scheduling tasks, particularly an extension of the problem to more complex network settings that include routing. This would require scalable representations for the states and actions and a further investigation into the flow completion reward.

VIII. ACKNOWLEDGEMENT

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES


