STRUCTURED REINFORCEMENT LEARNING FOR MEDIA STREAMING AT THE WIRELESS EDGE

A Thesis

by

SHREYAS RAMESHKUMAR

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Srinivas Shakkottai
Committee Members,	Dileep Manisseri Kalathil
	Natarajan Gautam
	Jean-Francois Chamberland-Tremblay
Head of Department,	Miroslav Begovic

December 2021

Major Subject: Electrical Engineering

ABSTRACT

When streaming media in an environment with less than ideal resources, simple algorithms do not perform well by not prioritizing specific clients well enough. Therefore, intelligent network sharing must occur to ensure maximum average quality of experience (QoE). We propose formulating this problem of designing intelligent policies as a Constrained Markov decision process. By relaxing the problem by using a Lagrangian formulation we reduce it down into single-client problems. Reinforcement learning can then be applied to control resource allocation to each client by implementing a threshold with respect to each client's buffer length. By developing a simulation environment and evaluating algorithms on a real world system capable of handling a variable number of clients, we demonstrate improvement in overall average QoE.

DEDICATION

To my parents and my sister who have supported me through this endeavor

ACKNOWLEDGEMENTS

I would like to first thank my advisor, Dr. Srinivas Shakkottai for his help and support over the past two years. He has always been supportive and understanding of my work, making sure my thoughts were heard. His enthusiasm and encouragement was crucial to this thesis. I would like to thank my committee members, Dr. Dileep Kalathil, Dr. Jean-Francois Chamberland-Tremblay and Dr. Natarajan Gautam for their support.

I would also like to thank my lab mate Manav Gurumoorthy and seniors Desik Rengarajan and Archana Bura. Their contributions and insights have helped me to solve numerous problems that I have faced.

Special thanks to my friends and family who have encouraged me along this journey.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of advisor Dr. Srinivas Shakkottai and Dr. Dileep Kalathil and Dr. Jean-Francois Chamberland of the Department of Electrical and Computer Engineering, and Dr. Natarajan Gautam of the Department of Industrial and Systems Engineering.

The problem formulation and analysis contained in this work was carried out by Archana Bura while the simulations were conducted in part with Desik Rengarajan and Manav Gurumoorthy. All other work conducted for the thesis was completed by the student independently

Funding Sources

Graduate study was supported by a student worker position from Dr. Srinivas Shakkottai.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION

Video streaming services are acquiring more internet traffic year by year. As of 2021, video is now over 48% of all traffic happening on a network, dominating over social networking, messaging, gaming, etc. Youtube specifically, is the number one application on mobile networks [1]. With the increase in high video quality available and the amount of data required to be transmitted, there exists a need to accurately divide available bandwidth to available clients to ensure a good quality of experience. In edge networks specifically, this available bandwidth acts as a constraint preventing clients from streaming seamlessly.

In the case of video streaming, various factors affect user experience and are not taken into account when allocating resources. If resources are shared equally among clients, it could result in all clients having a poor experience when dealing with limited resources. Some routers do prioritize some time-sensitive data packets over others, however these routers do not specifically consider the use case of media streaming. As software support for WiFi and cellular networks increases, intelligent bandwidth allocation on a finer level becomes a possibility. This software support also allows for systems to learn from the data that they transmit as well as the performance of the clients in accordance to the resources they are assigned.

The goal of this thesis is to propose a novel algorithm related to dynamically allocating bandwidth to clients specifically in the case of media streaming on Youtube. To prove the algorithm is beneficial when it comes to QoE involves computing the policy such that it is straightforward to implement on the real world system and holds up under different scenarios involving a variable number of clients and channel conditions. A diagram illustrating the specific use case of media streaming is shown in Figure 1.1, where multiple clients are connected to a common wireless access point. Each client is represented by a streaming Youtube video and has an application state in terms of its buffer length and the number of stalls it has experienced so far.

This state is relayed back to a controller through the use of a shared database. The controller then uses the states gathered from all the clients to compute a policy that determines which clients

are prioritized over others. This prioritization affects the bandwidth each client receives and consequently, their application states are impacted. This impact is measured through by calculation of QoE and once again this information is made available to the controller, completing a continuous feedback loop.

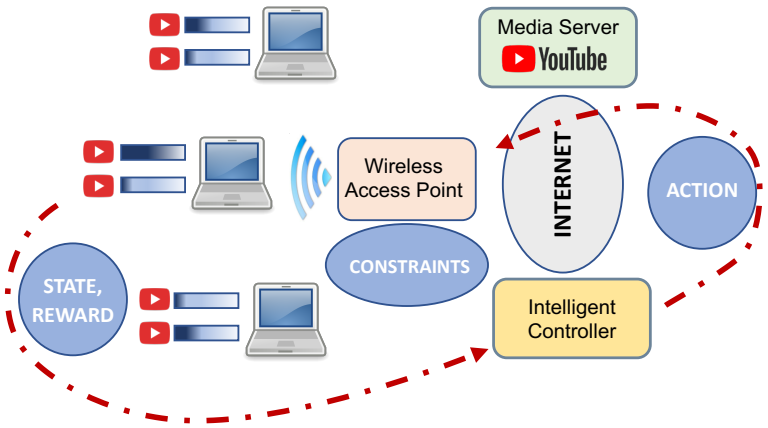

Figure 1.1: Feedback loop for reinforcement learning in a media streaming application.

Figure 1.2 shows a bare-bones methodology of the system. This problem of deciding how to best allocate resources to maximize QoE can be framed as a Markov Decision Process (MDP). Figure 1.1 indicates that resource allocation is can be performed through placing Youtube sessions in queues that have different priorities. The constraint of limited resources given to Youtube sessions makes this a constrained Markov Decision Process (CMDP).

The CMDP in case is not straightforward to resolve, due to the many random factors that play a role. Examples include Youtube playout rate, interactions between channels, and network protocols used. Due to the randomness present, the CMDP is not sufficient to derive the optimal policy. Instead, a data driven approach using constrained reinforcement learning is required. However, any reinforcement learning algorithm requires a large amount of data. Reducing the learning duration is key if the developed policy is required to be robust over different scenarios. Therefore, the first objective is to work on discovering the structure of the optimal policy. Once the structure has been

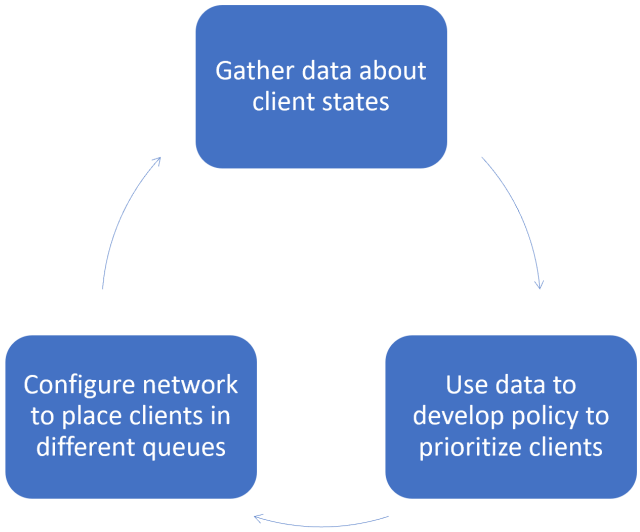

Figure 1.2: Simplified feedback loop

discovered, it can be used to expedite the learning process. In this thesis, we seek to determine a robust policy that generalizes in a variety of scenarios in a real world scenario.

2. RELATED WORK

Constrained MDPs in the area of control have been extensively resarched on previously [2, 3]. In the context of media streaming over wireless channels, several works identify structural properties of the optimal policy [4, 5, 6] that often take a threshold form. [4, 5] both consider maximizing QoE for a finite system. [4] provides optimal tradeoffs between various QoE metrics, while [5] maximizes the video QoE of several clients in a system, where the optimal policy has structural properties, and the solution can be decentralized. [5] also utilizes a dual-based decentralization approach. [6] focuses on the heavy traffic limit and provides optimal online QoE optimization in heavy traffic conditions, over fading channels.

Reinforcement learning is broadly classified into model-based methods and model-free methods to identify optimal policies for MDP/CMDP problems. In the context of model-based learning, recent works focus on characterizing the regret [7, 8] and the sample complexity [9] of learning algorithms. Model-free learning has focused on high-performance policy gradient approaches [10], such as Proximal policy gradient [11], Trust region methods [12] for the MDPs, and constrained policy optimization [13] in the CMDP regime.

A variety of systems have been proposed to improve the QoE performance of video streaming. in [14] utilizes path selection algorithms through software defined networks (SDN) to assign video streaming flows to network links.

Reinforcement learning has also been applied to a variety of video streaming applications [15, 16], which show significant improvements over existing methods. [15] proposes an algorithm Pensieve, to train adaptive bit rate (ABR) algorithms to optimize the QoE of the clients. [16] considers the complementary problem of service prioritization, and shows empirically that off-the-shelf model-based and model-free RL approaches can result in major improvements in QoE. In contrast, we provide a simple, structured RL algorithm with provable convergence guarantees, which is empirically able to attain the same performance as more complex deep learning approaches using significantly fewer data samples.

3. PROBLEM FORMULATION

Our model consists of clients providing states to the controller. The controller obtains information from each client on the state of its media stream and instructs the wireless access point to instantiate clients into two queues, referred to as a "high" and "low" priority queue. The controller periodically decides which clients are allowed into the high priority queue. While streaming, the media server sends data packets to the AP which in turn, forwards them to each client. The data packets buffers up the content allowing the video to play. Due to resource constraints on the system, not every client is assigned to the high priority queue at the same time. This turns our problem into a constrained decision making problem with an aim of maximizing overall QoE of all clients.

State: We denote the state of client n at time t by $s_n(t) = (x_n(t), y_n(t))$, where $x_n(t)$ is the number of packets in the buffer, and $y_n(t)$ is the number of stalls the client n has encountered until time t. We consider a finite state buffer for every client, i.e., $x_n(t) \in [L] \triangleq \{0, 1, \ldots, L\}$. We keep track of stalls up to a finite number, and choose $y_n(t) \in [M] \triangleq \{0, 1, \ldots, M\}$. Thus, the state space of client n, denoted by $\mathcal{S}_n$ is $[L] \times [M]$ and the joint system state space is given by $\mathcal{S} \triangleq \otimes_{n=1}^{N} \mathcal{S}_n$.

Actions: The service class assigned to a client n is denoted by action $a_n(t)$, i.e., $a_n(t) \in \mathcal{A}_n \triangleq \{1, 2\}$, where 1 means the high priority service, and 2 means low priority service. Let $\mu_n(a)$ represent the service rate obtained by the n^{th} client. We assume that the packet arrival process $A_n(t)$ for every n is a Bernoulli distributed random variable, with a parameter $\mu_n(a)$, depending only upon a.

$$\mu_n(a) = \begin{cases} \mu_{n,1} & a = 1 \\ \mu_{n,2} & a = 2. \end{cases}$$

We assume that $\mu_{n,1} > \mu_{n,2}$, for each n. This means that when the client is assigned to high priority service, its service rate is higher than that of the low priority service. The joint system action space

is $\mathcal{A} \triangleq \otimes_{n=1}^{N} \mathcal{A}_n$.

Policy: The joint policy Π is a mapping from joint state space $\mathcal{S}$ to joint action space $\mathcal{A}$, $\Pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$.

State Transition Structure: The system has two state transition processes, corresponding to the media buffer evolution and the stall count evolution. We assume that the media playback process $B_n(t)$ is distributed according to Bernoulli distribution such that stall-free playout is possible if the client placed in the high priority queue. We assume a stall occurs when there is a transition from a non-zero video buffer state to a zero buffer state.

We assume that the media that is currently being played by the client may be terminated at any time with probability $\alpha > 0$ by the end-user, which causes a reset of the client state to $(0, 0)$. Hence, the possible state transitions are

$$(x_n(t+1), y_n(t+1)) = \begin{cases} (x_n(t), y_n(t)), & \text{w.p } (1-\alpha)P_{n,1}(a_n(t)), \\ (\min\{x_n(t)+1, L\}, y_n(t)), & \text{w.p } (1-\alpha)P_{n,2}(a_n(t)), \\ ((x_n(t)-1)^+, \min\{y', M\}), & \text{w.p } (1-\alpha)P_{n,3}(a_n(t)), \\ (0, 0), & \text{w.p } \alpha, \end{cases}$$

where $y' = y_n(t) + \mathbb{1}\{x_n(t) > 0, x_n(t+1) = 0\}$, and $P_{n,1}(a) = (1 - \mu_n(a))(1 - \beta_n) + \mu_n(a)\beta_n, P_{n,2}(a) = \mu_n(a)(1 - \beta_n), P_{n,3}(a) = (1 - \mu_n(a))\beta_n$, for $a \in \{1, 2\}$, and for every n. The state transition probability of the n^{th} client is denoted by $p_n(s'|s, a)$.

Reward Structure: A stall occurs when the media playback stops due to the media buffer becoming empty. Hence, we say that a *stall period* begins at time t if the media buffer becomes empty at t. A cost function is used, rather than a QoE-based reward function for consistency with a generic MDP/RL approach that focuses on cost minimization. This can be done by multiplying the instantaneous cost by -1 and maximizing the value. We use the DQS model, which was validated using 183 videos and 53 human participants [17] to gain insight into the nature of an appropriate

cost function. Here, the client starts the video with a QoE of 5. It decreases when the stall event happens, and further decreases when the stall period continues. The QoE recovers when the stall period ends.

Constrained Markov Decision Process: Our resource allocation problem takes the form of the constrained infinite horizon discounted optimization

$$\begin{aligned} &\min_{\Pi} \mathbb{E}^{\Pi}[\sum_{n=1}^{N}\sum_{t=0}^{\infty}\gamma^t c(x_n(t), x_n(t+1), y_n(t+1)))] \\ &\text{s.t} \quad \sum_{n=1}^{N} g(a_n(t)) \leq K, \quad \forall t, \end{aligned}$$

where $\gamma < 1$ is the discount factor, and $g(a_n(t)) = \mathbb{1}\{a_n(t) = 1\}$. Note that the constraint enforces the condition that only K clients may be allowed high priority service at a given time.

Relaxed CMDP: A commonly used approach to solving CMDPs is to relax hard constraints and making them hold in expectation to obtain a near-optimal policy [18]. The hard constraint can then be re-enforced by an indexing approach wherein the actions leading to the highest value are chosen, while subjected to the constraint. We follow this approach to soften the constraint to obtain the following problem:

$$\begin{aligned} &\min_{\Pi} \mathbb{E}^{\Pi}[\sum_{n=1}^{N}\sum_{t=0}^{\infty}\gamma^t c(x_n(t), x_n(t+1), y_n(t+1))] \\ &\text{s.t} \quad \mathbb{E}^{\Pi}[\sum_{n=1}^{N}\sum_{t=0}^{\infty}\gamma^t g(a_n(t))] \leq \bar{K}, \end{aligned} \tag{3.1}$$

where, $\bar{K} = K/(1-\gamma)$, $a(t) \sim \Pi(s(t))$, $a(t) = (a_n(t))_{n=1}^{N}$, $s(t) = (s_n(t))_{n=1}^{N}$. The constraint is now the discounted total number of times the high priority service is allocated to all the clients. The policy here depends on the joint state of the system as a whole. Hence, a centralized controller is needed to compute the policy, adding to complexity.

By using the assumption that clients' transitions are independent of each other given the action, the centralized problem can be decomposed in to N smaller problems [5].

Let $\pi_n : \mathcal{S}_n \times \mathcal{A}_n \to [0,1]$ be policy of client n. Define

$$J^{\pi_n}(x,y;\lambda) = \mathbb{E}^{\pi_n}[\sum_{t=0}^{\infty} \gamma^t [c(x_n(t), x_n(t+1), y_n(t+1)) \\ + \lambda g(a_n(t))] | (x_n(0), y_n(0)) = (x,y)], \tag{3.2}$$

where $a_n(t) \sim \pi_n(x_n(t), y_n(t))$, $(x_n(t+1), y_n(t+1)) \sim p_n(((x_n(t), y_n(t)), a_n(t), \cdot)$. λ is basically the 'price' imposed by the AP for violating the constraint imposed. The optimal client policy and the optimal client value function are then given as

$$\pi_n^* = \arg\min_{\pi_n} J^{\pi_n}, \quad J^n = J^{\pi_n^*}. \tag{3.3}$$

4. ANALYSIS OF THE OPTIMAL POLICY

Our analysis begins by stating two important properties of the optimal policy and value function. In particular, we state that the optimal policy has a threshold structure and the value function has a unimodal structure with respect to the threshold parameter.

4.1 Threshold Structure

We first state that the optimal policy for each client has a simple threshold structure, following an approach similar to [5]. We present this result as a theorem below.

Theorem 1. *The optimal policy π_n^* for any single client $n \in [N]$ has a threshold structure. More precisely, for any given stall count $y \in [M]$, there exists an integer $\theta^*(y)$ such that*

$$\pi_n^*(x,y) = \begin{cases} 1 & \text{if } x \leq \theta^*(y), \\ 2 & \text{if } x > \theta^*(y). \end{cases}$$

4.2 Unimodality

A threshold policy, denoted as π_θ, can be completely specified by its threshold parameter vector $\theta = \{\theta(0), \theta(1), \ldots, \theta(M)\}$. Let $J(\theta;\lambda)(x,y) = J^{\pi_\theta}(x,y,\lambda)$. In this subsection, we will show that $J(\theta;\lambda)$ has a unimodal structure with respect to θ. Since this result is true for any arbitrary λ, we omit λ from the notation for convenience and denote $J(\theta;\lambda)$ simply as $J(\theta)$.

Definition 1. *$J(\theta)$ is unimodal in θ if and only if the following are true for every $y \in [M]$:*
(i). For $\theta(y) \leq \theta^*(y)$, $J(\theta)$ is monotonically non-increasing in $\theta(y)$, and (ii) for $\theta(y) > \theta^*(y)$, $J(\theta)$ is monotonically non-decreasing in $\theta(y)$, while the rest of dimensions of θ remain fixed.

We now present the unimodality property of $J(\theta)$ formally as a theorem below

Theorem 2. *$J(\theta)$ is unimodal in θ.*

5. RL FOR LEARNING THE OPTIMAL THRESHOLD

Our proposed algorithm follows the methodology of actor-critic algorithms with function approximation and slightly modifies our hard-threshold policy to a soft-threshold policy. Denoting $\pi_\theta(x, y, a)$ as the probability of taking action a, when state is (x, y), the soft-threshold policy corresponding to the threshold parameter vector θ is defined as

$$\pi_\theta(x, y, a) = \begin{cases} 1 - \frac{1}{1+e^{-x+\theta(y)}} & \text{if } a = 1, \\ \frac{1}{1+e^{-x+\theta(y)}} & \text{if } a = 2. \end{cases}$$

The threshold parameter vector θ can be a real-valued vector now, instead of the integer-valued vector.

Instead of following a stochastic gradient descent approach, we follow a more rigorous stochastic approximation approach [19] to show the almost sure convergence our proposed algorithm. Our Decentralized Threshold algorithm, given as Algorithm 1, is a three time scale stochastic approximation algorithm which iteratively updates the value function, policy, and the Lagrange multiplier. At each time step, it calls N instances of ACSUBROUTINE, that runs an actor-critic step, one for each client. After each client updates its value function and its policy, the Algorithm 1 retrieves the experienced actions from the clients, and updates the Lagrange multiplier by running a gradient ascent in the slowest time scale. According to the step sizes defined earlier, we run the value update at a faster time scale, the policy at a slower time scale, and the Lagrangian update at the slowest time scale.

Theorem 3. *Let $a(m)$, $b(m)$ and $c(m)$ be the standard stochastic approximation step size sequences satisfying the following properties: (i) $\sum_{m=1}^{\infty} a(m) = \infty, \sum_{m=1}^{\infty} b(m) = \infty, \sum_{m=1}^{\infty} c(m) = \infty$, (ii) $\sum_{m=1}^{\infty}[(a(m))^2 + (b(m))^2 + (c(m))^2] < \infty$, (iii) $\lim_{m\to\infty} b(m)/a(m) \to 0$, and (iv) $\lim_{m\to\infty} c(m)/b(m) \to 0$. Then the Algorithm 1 converges to the optimal values of MDP for every client n and the optimal Lagrange multiplier. i.e., $(J_m^{(n)}, \theta_m^{(n)}) \to (J^n, \theta^n)$, $\lambda_m \to \lambda^*$.*

Algorithm 1 Decentralized Threshold Algorithm

Initialize Value function $J_0^{(n)}(x,y) = 0, \forall (x,y) \in \mathcal{S} \times [M]$, and $\forall n \in [N]$. Threshold $\theta_0^{(n)} = 0$, state $(x_n(0), y_n(0)) = (0,0)$, $G_n = 0$, counts $\eta_n(x,y) = 0$ for every (x,y), for every client n.
for $m = 0, 1, \ldots,$ **do**
    for Each client $n = 0, 1, \ldots, N$ **do**
        $G_n \leftarrow$ ACSUBROUTINE(m, λ_m).
    end for
    Update the Lagrange multiplier
    $\lambda_{m+1} = (\lambda_m + c(m)[\sum_{n=1}^{N} G_n - K])^+$.
end for

Algorithm 2 ACSUBROUTINE(m, λ_m)

Take the action $A \sim \pi_{\theta_m}(x, y, .)$.
Obtain the next state $\zeta_m(x,y)$, and costs $c(x, \zeta_m(x,y))$, $g(A)$ from the environment.
Update the count $\eta(x,y) \leftarrow \eta(x,y) + 1$.
Update the value function according to

$$J_{m+1}(x,y) = J_m(x,y) + a(\eta(x,y)) \left[\lambda_m g(A) + \quad [c(x, \zeta_m(x,y)) + \beta J_m(\zeta_m(x,y))] - \ J_m(x,y)\right],$$
$$J_{m+1}(x',y') = J_m(x',y'), \forall (x',y') \neq (x,y).$$

Update the threshold

$$\begin{aligned}\theta_{m+1}(y) = \Lambda(\theta_m(y) + b(\eta_(x,y))[\lambda_m g(A) + [c(x, \zeta_m(x,y)) + \beta J_m(\zeta_m(x,y))] \\ - J_m(x,y)] \nabla \ln \pi_{\theta_m}(x,y,A), \\ \theta_{m+1}(y') = \theta_m(y'), \text{ when } y' \neq y.\end{aligned}$$

Update the state $(x,y) \leftarrow \zeta_m(x,y)$.
Return $g(A)$.

The updates in our algorithm correspond to stochastic approximation updates. Stochastic approximation theory then guarantees convergence if certain assumptions are satisfied. Our algorithm satisfies these assumptions and our unimodality result guarantees global almost sure convergence.

6. SIMULATION SYSTEM

Developing a simulator is necessary as developing policies on the real system would take huge amounts of data. Collecting the data required would take very long, making the system impractical. Therefore, all the policy training occurs on the simulator, developed on Python using OpenAI Gym. The simulator consists of a wireless access point (AP) and two queues which are set to have a fixed bandwidth to which clients are put in. The simulator is meant to closely mimic the real system where clients then share the bandwidth assigned to the queue equally among them. For example, if there were 4 clients in the 4Mbps queue, then each client would stream video with 1Mbps bandwidth.The number of clients can be variable and is set to 6 as default. The simulator instantiates each client with a buffer, and models the evolution of the buffer based on queue assignment and video bitrate. The video bitrate (in Mbps) is sampled from a normal distribution with mean 2.9 and standard deviation 1. Each client plays videos sequentially and stalls occur when the buffer runs out. The video lengths are drawn from a normal distribution with mean 600 and standard deviation 50.

Each client possesses a buffer state which includes the number of times the video has stalled and the number of seconds it has been playing since the last stall or buffering. Using this data, an intelligent controller calculates its QoE using the DQS model. We use "reward" using the QoE, rather than "cost" as used in the analytical model, since most RL implementations use this approach (by we multiplying "cost" by -1 and maximizing instead of minimizing).

The state of the system as a whole is the combined state for all clients that are present in the system. Therefore, the total number of states as a whole is infinite (as the video buffer and number of stalls have no theoretical limit). The action is the a permutation of clients that end up in the high priority queue and the low priority queue. This results in a total of 2^6 possible actions. As mentioned before, when the number of clients in each queue is fixed, the system can be treated as N independent single-client systems, each having appropriate shared bandwidth. Intuitively, training on this system should be faster, since we obtain N [state, action, next-state, reward] samples

per time step here, as opposed to just one in the joint system case. Furthermore, solving using the dual approach introduces a Lagrange multiplier λ, which also has to be learned by performing a hyperparameter search to ensure that when the controller follows the optimal policy the number of clients in each service class satisfies the service class constraints. Given the structure of the simulator, we implement two categories of algorithms, centralized and decentralized algorithms. Centralized algorithms train on the joint states and action spaces of all 6 clients, whilst the decentralized algorithms train on a single agent system.

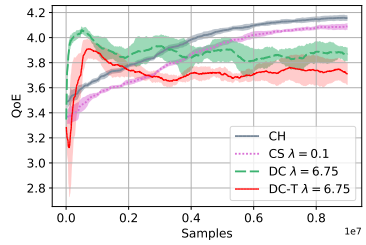

Figure 7.1: Training in simulation

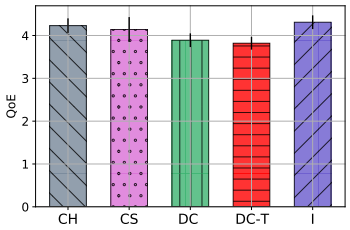

Figure 7.2: QoE in simulation

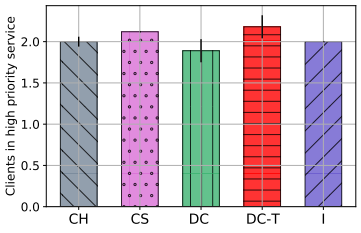

Figure 7.3: Clients assigned to high priority service

7 different algorithms are simulated and compared on the real system. Intuitively, we expect some algorithms to perform better than others, however results are verified on the simulation and real world system.

Vanilla (V): This is a simple policy that merges all available throughput into a single service class and shares it equally. We will use this as a base policy in the real-world system, but do not simulate it.

Greedy Buffer (GB): This algorithm awards high priority service to the clients with the lowest video buffer state, subject to resource constraints. This algorithm does not consider stall count as a factor in calculating QoE and is used as a well-established algorithm with a decent performance in the real-world evaluation.

Centralized Hard PPO (CH): QFlow [16] shows that unstructured Deep Q-Learning (DQN) performs really well in a context much like ours but requires a long training duration. A more recent policy gradient algorithm (specifically Proximal Policy Optimization (PPO) [11]). It is implemented with a hard constraint of 2 clients with in the high priority queue and 4 in the low priority queue.

Centralized Soft PPO (CS): This algorithm implements the Lagrangian relaxation by adding

a penalty λ for accessing "high" service. We use the same PPO algorithm, but do not impose a hard constraint as in CH. Rather, we impose a soft constraint on the joint action space, by penalizing the agent for placing more than two clients in the high priority queue. The algorithm is trained using a hyperparameter search over λ such that we have an average of 2 clients in the "high" service class.

Decentralized PPO (DC): This algorithm takes advantage of the conditional independence property that enables division into N independent systems, along with a Lagrangian relaxation of the constraints. The PPO algorithm is now used on an individual client basis, but since we obtain 6 samples per step, it can train much faster. However, we do not impose any structure on the policy class. Again, we need to find the appropriate penalty λ to enforce the constraint.

Decentralized Threshold (DC-T) This algorithm is similar to DC in utilizing division into N systems for faster learning, but also imposes a threshold structure of the optimal algorithm. It requires only one neuron (logistic function) to represent a threshold policy, which makes for simple implementation. Like DC, it too can train faster than the centralized approaches, and also needs an appropriate penalty term to enforce constraints.

Index (I): This algorithm follows the philosophy behind the Whittle Index [18] in converting a soft-constrained into a hard-constrained and robust policy. We calculate the index for each client as a function of buffer length, number of stalls, and QoE. However, we digitize these values and also place an upper limit. We order the states of clients based on their values obtained from DC-T, and provide "high" service the two with the highest value. We will see that this can be used regardless of the number of clients, or the quality of their channels.

7.1 Training and Evaluation

Figure 7.1 shows the evaluation of the algorithms on a joint system during training (averaged over 5 random seeds). We observe that decentralized algorithms converge over four times faster than centralized algorithms as they exploit the structure of the environment. The performance difference between the trained centralized and decentralized algorithms is negligible as seen in Figure 7.2 (averaged over 100 runs). The performance of DC-T is on par with the best performing algorithms, which confirms our hypothesis that the optimal policy is indeed a threshold policy.

The index policy, I that is hard-constrained version of DC-T shows similarly high performance. Figure 7.3 (averaged over 100 runs) shows the number of clients in the high priority queue during evaluation of the soft constrained algorithms is near 2, and so confirms our choice of λ.

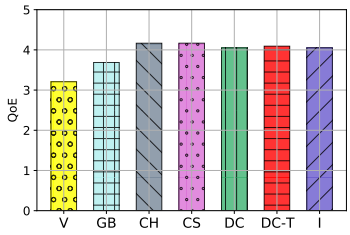

Figure 8.1: Comparison of average QoE

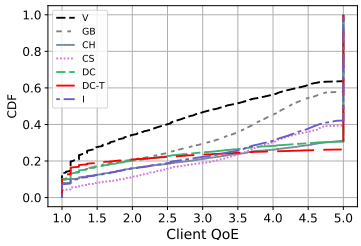

Figure 8.2: Comparison of QoE CDF

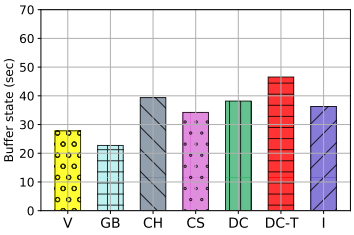

Figure 8.3: Comparison of average buffer state

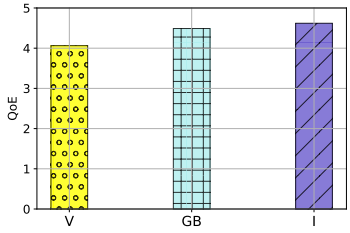

Figure 8.4: Comparison of average QoE with varying number clients

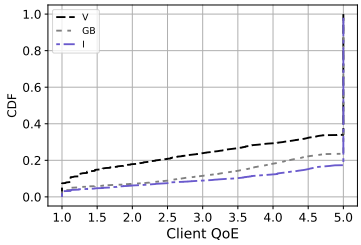

Figure 8.5: Comparison of QoE CDF with varying number of clients

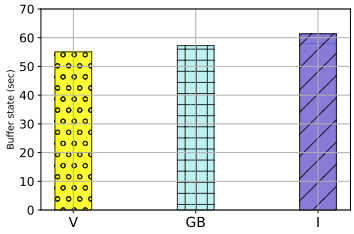

Figure 8.6: Comparison of average buffer state with varying number of clients

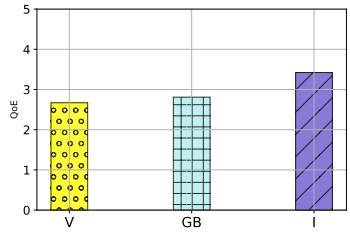

Figure 8.7: Comparison of average QoE in a poor channel

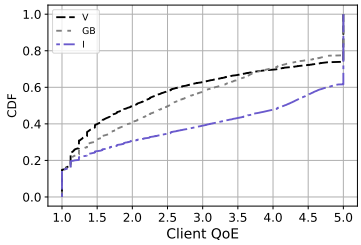

Figure 8.8: Comparison of QoE CDF in a poor channel

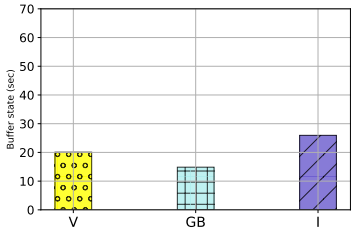

Figure 8.9: Comparison of average buffer state in a poor channel

The real system is set up similar to [16] with 4 Intel NUCs, with 3 of the NUCs acting as clients and the last as the controller. The client NUCs have Youtube sessions running and relevant data is gathered. The 4 NUCs are all connected to a WiFi router with OpenWrt installed on it. The OpenWrt operating system allows for various queues to be created where packets are transmitted.

We create a high priority and low priority queue for the purpose of our experiments. Every 10 seconds, the controller receives a new action from the policy that determines which clients enter the high priority queue and the low priority queue. Each evaluation has a runtime of three hours, to ensure sufficient data collection. The controller NUC runs various policies and accordingly sends clients into the two queues using Linux Traffic Control. We compare the 7 policies mentioned in Chapter 7 answer three different questions accordingly.

8.1 Does structured RL provide high-performance?

In the first scenario we look at, we have two queues, a high priority and a low priority queue and 6 clients. The 6 clients are distributed among the two queues depending on the state of clients and the policy running. We first determine if structured RL approaches are near-optimal. Figure 8.1 shows the average QoE across three hours of YouTube sessions for all 7 policies. As expected, the centralized policies CH and CS do the best, with the decentralized versions DC, DC-T and I following close behind. GB does reasonably well, but is unable to account for stalls influencing QoE, hence causing performance loss. The vanilla approach shows why intelligent control is necessary, as it lags behind the RL approaches by full QoE unit, i.e., it is over 30% worse. Figure 8.2 shows the CDF of QoE samples, where the value of the RL approach is even clearer from the fact that about 60-70% of samples have a perfect score of QoE 5, while the other approaches are about 20% worse. Finally, Figure 8.1 shows the average video buffer length, and appears to indicate that Greedy tries to equalize buffers, letting them go too low before prioritizing. The RL-based policies all have higher average buffers, consistent with higher QoE and fewer stalls.

8.2 Is indexing robust to a variable number of clients?

We next address whether the index policy I can be applied unaltered to the case where the number of clients is less than 6, which means that there is no need to train over a variable number of clients. So we perform an experiment where we decrease number of clients from 6 to 4 over three hours. As expected, the QoE of all three policies increases, with all three policies having an average QoE above 4 in Fugure 8.4. However, the overall trend is maintained as the Index policy

does the best as seen in Figures 8.4, 8.5, and 8.6.

8.3 Does the index approach generalize to variable channels?

Depending on our surroundings, our traffic control queues could face loss and delay. The question arises as to whether the RL approach needs to be trained separately over many channel realizations? Since indexing simply orders clients based on their states, we wish to determine if it can be used unchanged even when the channel quality is low. Hence, we introduce packet losses and delays using a network emulator to create channels with disturbances. We first group clients with similar channels into clusters and divide the total available resources in a proportionally fair manner across clusters. We then apply our service differentiation policies across members of each cluster. We show results for a cluster seeing 10% packet loss and 20 ms delay, and, as expected, QoE drops as seen in Figure 8.7. However, the overall order still holds as Vanilla and Greedy Buffer have QoEs over 20% worse than Index. Further, the client QoE CDF and average buffer state indicate same trend as seen in 8.8 and 8.9, with Indexing still performing significantly better than the base approaches.

9. CONCLUSION

In this work we explored the value of structured reinforcement learning for solving constrained MDPs in the context of media streaming systems. The key observation was that by first relaxing the constraint, we obtain a simpler CMDP whose optimal policy both has a threshold structure, and value function has a unimodal behavior with respect to the threshold parameter. These two properties enable us to develop a 3-timescale policy gradient algorithm that provably converges to the global optimum, rather than the local optimum that is usually guaranteed by the policy gradient approach. We empirically verify that exploiting structure enables much faster and simpler approaches to learning a near-optimal policy, which can be made robust to variations in the setting via indexing. Finally, we verify using real-world experiments our hypothesis regarding structured RL being able to robustly outperform traditional approaches in a variety of scenarios.

Future work involves translating our system to act as a RAN intelligent controller (RIC) at a cellular base station to test out other policies with a finer level of control.

REFERENCES

[1] Sandvine, "The Mobile Internet Phenomena Report." https://www.sandvine.com/phenomena, 2021.

[2] E. Altman, *Constrained Markov decision processes*, vol. 7. CRC Press, 1999.

[3] E. Altman, "Applications of Markov decision processes in communication networks," in *Handbook of Markov decision processes*, pp. 489–536, Springer, 2002.

[4] A. ParandehGheibi, M. Médard, A. Ozdaglar, and S. Shakkottai, "Avoiding interruptions—a qoe reliability function for streaming media applications," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 5, pp. 1064–1074, 2011.

[5] R. Singh and P. Kumar, "Optimal decentralized dynamic policies for video streaming over wireless channels," *arXiv preprint arXiv:1902.07418*, 2019.

[6] P.-C. Hsieh and I.-H. Hou, "Heavy-traffic analysis of qoe optimality for on-demand video streams over fading channels," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1768–1781, 2018.

[7] Y. Efroni, S. Mannor, and M. Pirotta, "Exploration-exploitation in constrained mdps," *arXiv preprint arXiv:2003.02189*, 2020.

[8] D. Ding, X. Wei, Z. Yang, Z. Wang, and M. Jovanovic, "Provably efficient safe exploration via primal-dual policy optimization," in *International Conference on Artificial Intelligence and Statistics*, pp. 3304–3312, PMLR, 2021.

[9] A. HasanzadeZonuzy, A. Bura, D. Kalathil, and S. Shakkottai, "Learning with safety constraints: Sample complexity of reinforcement learning for constrained mdps," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 7667–7674, 2021.

[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[12] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[13] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *International Conference on Machine Learning*, pp. 22–31, PMLR, 2017.

[14] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "SDN-based application-aware networking on the example of YouTube video streaming," in *Proceedings of EWSDN*, 2013.

[15] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 197–210, 2017.

[16] R. Bhattacharyya, A. Bura, D. Rengarajan, M. Rumuly, S. Shakkottai, D. Kalathil, R. K. Mok, and A. Dhamdhere, "Qflow: A reinforcement learning approach to high qoe video streaming over wireless networks," in *Proceedings of the twentieth ACM international symposium on mobile ad hoc networking and computing*, pp. 251–260, 2019.

[17] H. Yeganeh, R. Kordasiewicz, M. Gallant, D. Ghadiyaram, and A. C. Bovik, "Delivery quality score model for Internet video," in *Proceedings of IEEE ICIP*, 2014.

[18] P. Whittle, "Restless bandits: Activity allocation in a changing world," *Journal of applied probability*, vol. 25, no. A, pp. 287–298, 1988.

[19] V. S. Borkar, *Stochastic approximation: a dynamical systems viewpoint*, vol. 48. Springer, 2009.