UnB Departamento de Ciência da Computação

Instituto de Ciências Exatas da Universidade de Brasília. (UnB)
Departamento de Ciência da Computação (CIC)

Redes de Computadores (CIC0124) - Turma 02

Trabalho Final de Programação

Alunos:
Felipe Perone - 200017705

14 de Fevereiro de 2025

Abstract. *This report aims to present a detailed study on the implementation of an adaptive MPEG-DASH Video Streaming algorithm, a technique that allows high-quality streaming of media content over the Internet, delivered from HTTP servers. Analyzing and verifying the proportionality between different parameters, the implemented algorithm was a quality selection based on Q-Learning, with simulations performed using fixed default learning rates = 0.3 and a discount factor = 0.95, applying the Softmax selection policy, with temperature = 1.0.*

Resumo. *Este relatório tem como intuito a apresentação de um estudo detalhado sobre a implementação de um algoritmo adaptativos de Streaming de Video MPEG-DASH, uma técnica que permite o Streaming de alta qualidade de conteúdo de mídia pela Internet, entregues a partir de servidores HTTP. Analisando e verificando a proporcionalidade entre diferentes parâmetros o algoritmo implementado foi uma seleção de qualidade do baseada em Q-Learning, com simulações realizadas utilizando taxas fixas de aprendizagem padrão = 0.3 e um fator de desconto = 0.95, aplicando a política de seleção Softmax, com temperatura = 1.0.*

1. Introdução

Nos últimos anos, os serviços multimídia ganharam muita popularidade. Esse crescimento é amplamente atribuído aos serviços de streaming de vídeo. Esses serviços geralmente podem ser divididos em Televisão por Protocolo de Internet (IPTV), oferecida por um provedor de rede e gerenciada por meio de reserva de recursos, e serviços Over-The-Top (OTT), onde o vídeo é entregue pela tradicional Internet de melhor esforço. As técnicas de Dynamic Adaptive Streaming over HTTP (DASH) estão se tornando o padrão de fato para streaming de vídeo OTT.

Clientes HAS de última geração empregam lógicas de seleção de segmentos, também chamadas de heurísticas, determinísticas para adaptar dinamicamente o nível de qualidade solicitado com base nas condições percebidas da rede e do dispositivo. No entanto, as heurísticas atuais dos clientes HAS são rigidamente configuradas para se adequar a configurações específicas de rede, tornando-as menos flexíveis para atender a uma ampla gama de cenários.

Neste artigo, é proposto um algorítmo de seleção de qualidade (ABR - Adaptive Bitrate) baseado em Q-learning para um cliente DASH, focado exclusivamente no caso de um único filme de animação: Big Buck Bunny, que está hospedado em um servidor HTTP: http://45.171.101.167/DASHDataset/. Em contraste com as heurísticas existentes, o cliente proposto aprende dinamicamente o comportamento ideal, correspondente ao ambiente de rede atual, utilizando aprendizado por reforço. A adaptabilidade é obtida considerando vários aspectos da qualidade do vídeo através de uma função de recompensa ajustável. Detalhes cerca ao desenvolvimento da função recompensa e obtenção de seus coeficientes são abordados em [Claeys 2022].

O cliente HAS proposto foi amplamente avaliado usando um simulador com um ambiente controlado e simulado de rede, na configuração do ambiente de simulação, é empregado uma distribuição de probabilidade exponencial para definir a variabilidade da

largura de banda disponível ao longo da transmissão. Isso permitirá uma avaliação mais realista do desempenho do algoritmo ABR em condições dinâmicas de rede.

O vídeo, com duração de 596 segundos e contendo apenas a componente visual (sem áudio), está segmentado de seis formas diferentes. Os segmentos estão codificados em 20 formatos distintos, variando entre taxas de bits de 46.980 bps a 4.726.737 bps, investigando múltiplas configurações de recompensa e ajustes específicos de Aprendizado por Reforço.

State element	Range	Levels
Buffer filling	$[0\ ;\ B_{max}]$sec	$\frac{B_{max}}{T_{seg}}$
Buffer filling change	$[-B_{max}\ ;\ B_{max}]$sec	$2 * \frac{B_{max}}{T_{seg}}$
Quality level	$[1\ ;\ N]$	N
Bandwidth	$[0\ ;\ BW_{max}]$bps	$N+1$
Oscillation length	[0 ; 30]segments	31
Oscillation depth	$[0\ ;\ N-1]$	N

Figura 1. Tabela que demonstra parâmetros de Estado.

1.1. Q-Learning

O aprendizado por reforço é uma técnica de aprendizado de máquina onde um agente interage com um ambiente para aprender a tomar decisões que maximizem uma recompensa cumulativa ao longo do tempo. Dentro desse contexto, o Q-learning é um algoritmo baseado em tabelas que permite ao agente aprender uma política desejada por meio da exploração e exploração do ambiente.

O Q-learning utiliza uma tabela chamada Q-table, onde os valores Q representam estimativas da recompensa esperada para cada par estado-ação. Esses valores são atualizados iterativamente com base na equação de Bellman, representada na figura 2.

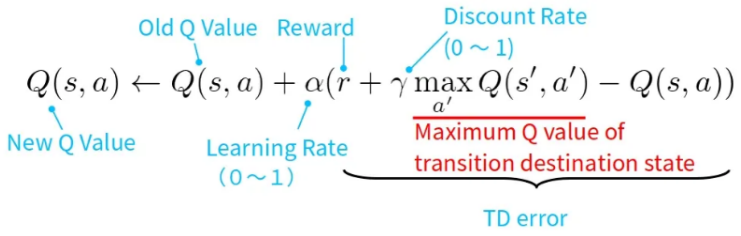


Figura 2. Equação de Bellman.

Na figura acima, Q(s, α) é o valor Q atual para o estado s e ação α, α é a taxa de aprendizado, que controla a influência das novas informações, r é a recompensa recebida após executar a ação, γ é o fator de desconto, que pondera a relevância das recompensas futuras, Q(s',α') representa a melhor estimativa de valor qualidade Q futura para o próximo estado.

Com objetivo de permitir que o agente escolha, de forma autônoma, a melhor qualidade para cada segmento de vídeo transmitido, levando em consideração fatores e

parâmetros do estado em que está situado, o agente obtém um estado inicial dentro do ambiente, determinado na primeira requisição de segmentos de vídeo, ainda dentro do método de requisição handle segment size request(self, msg) escolhe uma ação com base em uma estratégia de decisão softmax. Enquanto a resposta do servidor, utilizando o método handle segment size response(self, msg), é aonde o agente irá receber uma recompensa ou penalização com base na ação tomada, em seguida a Q-table é atualizada, com base na experiência adquirida, concluindo o ciclo. A figura 3 faz alusão as etapas do processo de aprendizagem do Q-learning. [Awan 2024]

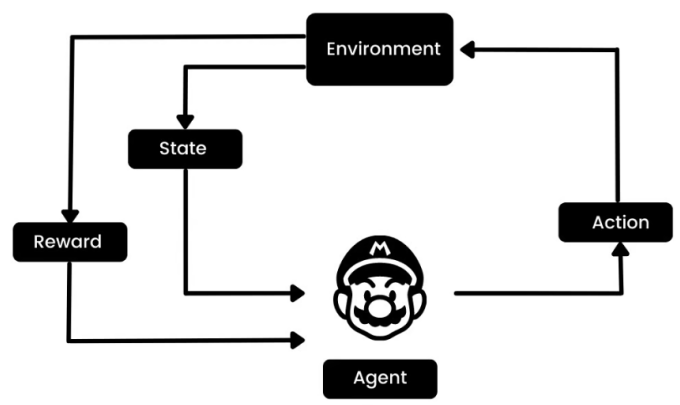


Figura 3. Etapas aprendizado por reforço.

1.2. Função Softmax

O dilema Exploração vs. Exploração é um dos desafios centrais em problemas de tomada de decisão e aprendizado por reforço. Ele ocorre quando um agente precisa decidir entre: *Exploration*: Buscar novas estratégias ou opções, testando alternativas que podem ser melhores no longo prazo, mas que podem gerar retornos abaixo do esperado no curto prazo, e *Exploitation*: Utilizar o conhecimento já adquirido, escolhendo a ação que maximiza o retorno imediato, sem investir recursos na busca por novas alternativas.

O equilíbrio entre esses dois aspectos é essencial para garantir um aprendizado eficiente. Muita exploração pode reduzir a recompensa no curto prazo, enquanto a priorização excessiva da exploração pode impedir a descoberta de estratégias ótimas em ambientes incertos, esse balanceamento influencia diretamente a convergência do agente para uma solução ótima.

Dentro desse contexto, a função Softmax é um método matemático probabilístico amplamente utilizado em aprendizado por reforço para a seleção de ações. Baseando-se na distribuição de Boltzmann, ela atribui probabilidades às ações disponíveis de acordo com os valores Q aprendidos, permitindo uma escolha que favorece ações de maior recompensa esperada, sem eliminar completamente a possibilidade de *exploration*. A figura 4 representa a equação mátemática da função softmax e um exemplo de sua atribuição de probabilidades baseado em magnitude valores.

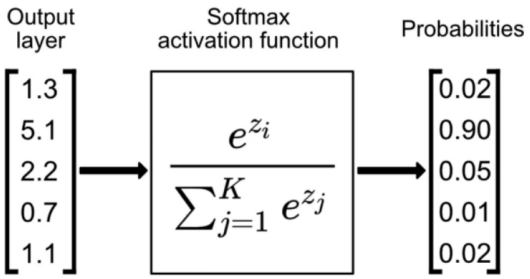


Figura 4. função Softmax.

Na figura acima zi é o vetor de input, que no caso deste trabalho vai ser um array de 20 valores Q, cada valor representa uma ação e uma qualidade. E a variavel zj representa o vetor de saída, que é um array das probabilidades de seleção de cada ação, gerado por meio da entrada.

2. Procedimentos

Nessa sessão será demonstrado os procedimentos, através da linguagem de programação Python, implementados para o desenvolvimento do Cliente Dash baseado em Q-learning obtido. A fim de consumarmos esse objetivo foi definida a classe R2Qlearning, e todos os métodos definidos em seu escopo, serão alvos de analise dessa sessão do trabalho.

2.1. funções init, handle xml request e handle xml response

```python
class R2AQlearning(IR2A):
  def __init__(self, id):
    IR2A.__init__(self, id)
    self.throughputs = []
    self.qi = []
    self.request_time = []
    self.quality_lista_1 = []
    self.seg_num = 0
    self.state_space = [] #passa valores de state de ss_request p/ ss_response
    self.action_space = [] #passa valores de action de ss_request p/ ss_response
    self.Q_table = {}  # Dicionário para armazenar os valores Q
    self.osc_list = []
    pass
```

Figura 5. Definição função Init.

O método init serve como um construtor para inicializar os atributos de um objeto quando uma instância de classe é criada. Portanto é nela que são definidas as listas e contadores que serão empregados em operações em outras funções, mantendo seu valor ao longo do código, não somente no método cujo está declarado. Portanto aqui devemos definir nossa Q-table, que vai consistir de um dicionário que tem como chave uma tupla *state* que contém os parâmetros do *environment* atual, e tem como valor de cada chave um array de 20 valores Q, um valor pra cada ação (lembrando que a ação nesse trabalho é a seleção de uma qualidade, então a ação 0 é a seleção da qualidade self.qi[0]).

```python
def handle_xml_request(self, msg):
    self.request_time = time.perf_counter() #inicia contagem de tempo da requisição xml
    self.send_down(msg)
```

Figura 6. Função handle xml request.

```python
def handle_xml_response(self, msg):
    parsed_mpd = parse_mpd(msg.get_payload())
    self.qi = parsed_mpd.get_qi()
    #print(f" chegou aqui na lista self.qi handle_xml_response {self.qi}") #tirar dps
    t = (time.perf_counter() - self.request_time)/2
    self.throughputs.append(msg.get_bit_length()/t)
    self.send_up(msg)
```

Figura 7. Função handle xml response.

A função representada na figura 6 é a responsável por realizar a requisição do arquivo xml, durante a passagem nela é oportuno obter uma medição do tempo a fim de obter a primeira vazão de rede ao calcular com a medição obtida na função da figura 7. A qual é a responsável por receber no ambiente IR2A a resposta da requisição do arquivo xml.

Durante a única passagem nessa função deve-se extrair, por meio do parser disponível na plataforma pydash, a lista de qualidades de segmentos disponíveis do video desejado, cujo endereço está especificado no arquivo de entrada json.

2.2. Handle segment size request

É na metodologia de requisição de segmentos aonde o protocolo de seleção de qualidade ABR está definido. Antes de estabelecer o cálculo dos parâmetros de estado e das recompensas, devemos primeiro estabelecer duas alternativas que o código deve optar seguir ao chegar nessa função. Se o código estiver requisitando o primeiro segmento de qualidade do video ou não.

Caso o primeiro segmento de video esteja sendo requisitado nessa passagem do código, antes de fazer a seleção de qualidade com parâmetros reais o agente deve simular episódios para treinamento e obtenção de conhecimento cerca de estados. Para isso é implementado esse looping *for*, representado na figura 8, que realiza a simulação de 200000 episódios.

```python
if self.seg_num == 0:
  # Loop de treinamento (simulação)
  for episode in range(100000):  # Número de iterações de aprendizado
      # Obter estado inicial do ambiente
      bufferfilling = random.uniform(0, Bmax)  # Simulação de preenchimento do buffer
      buffer_change = random.uniform(-Bmax + bufferfilling, Bmax - bufferfilling)  # Simulação de variação do buffer
      quality = random.randint(0,num_qualities-1) # Qualidade atual ESTRATEGIA RELACIONAR À BANDWIDTH
      bandwidth = random.choice(self.qi)  # Simulação de largura de banda
      osc_length = random.randint(0, 60)  # Comprimento da oscilação
      osc_depth = random.randint(0, 19)  # Profundidade da oscilação
      state = (round(bufferfilling, 1), round(buffer_change, 1), quality, round(bandwidth, 1), osc_length, osc_depth)
      # Selecionar ação usando Softmax
      #aqui
      if state not in self.Q_table:
        self.Q_table[state] = np.zeros(num_qualities)  # Inicializa Q(s, a) se não existir, cria um array de 20 zeros no dicionário Q_table com chave = state.

      q_values = self.Q_table[state] #passa o array de 20 valores Q (um p/ cada ação) pra variavel q_values
      exp_q = np.exp(q_values / tau)
      probabilities = exp_q / np.sum(exp_q)
      action = np.random.choice(range(num_qualities), p=probabilities) #seleciona através da política softmax
```

Figura 8. Implementação do looping For.

Dentro do looping são determinados aleatoriamente, entre os valores possíveis de limite, os valores de cada uma das variáveis de estado com auxilio da biblioteca *numpy*,

em seguida é tomada uma ação através da politica softmax com base nos valores da Q-table. Com base na figura 9, ainda dentro do looping, é calculada recompensa da ação tomada e após isso é gerado um novo estado, de maneira análoga à anterior, e a q-table é atualizada.

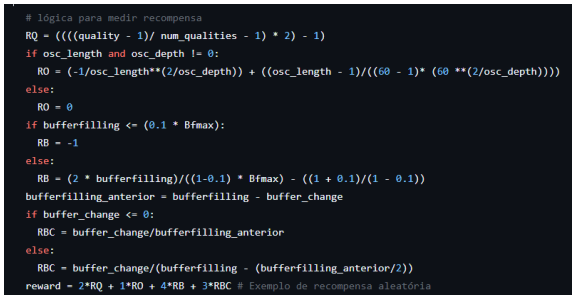


Figura 9. Função recompensa dentro do looping For.

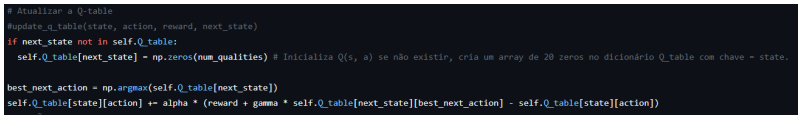


Figura 10. Atualização da Q-table dentro do looping For.

Supondo ainda estar requisitando o primeiro segmento de video, mas agora já ter percorrido todo este looping duzentas mil vezes, deve-se realizar a obtenção de valores de estado reais e a seleção de qualidade através da politica softmax maneira análoga ao looping *for* anterior. Porém como ainda estamos requisitando o primeiro segmento de video, não temos valores de ambiente suficiente para obter dados sobre buffers, oscilações e qualidades passadas, devido a ausência de dados passados. Portanto é determinado manualmente o estado inicial, conforme a figura 11 representada abaixo demonstra.

Figura 11. Estado inicial estipulado.

Caso deseje-se obter um segmento de vídeo que não seja o primeiro, deve-se obter através de técnicas e funções em disponibilidade na plataforma Pydash, os valores reais dos parâmetros que compõem um estado.

```
else: #self.seg_num > 0
  buffer_filling_lista = self.whiteboard.get_playback_buffer_size()
  bufferfilling = buffer_filling_lista[-1][1]
  if len(buffer_filling_lista) >= 2:
    buffer_change = bufferfilling - buffer_filling_lista[-2][1]
  else:
    buffer_change = 0
  quality_lista = self.whiteboard.get_playback_qi() #n usa
  quality = self.quality_lista_1[-1] #verificar DPS self.qi.index(quality_lista[-1][1])
  bandwidth_referencial = self.throughputs[self.seg_num]
  if bandwidth_referencial <= self.qi[0]:
    bandwidth = self.qi[0]
  elif bandwidth_referencial >= self.qi[len(self.qi)-1]:
    bandwidth = self.qi[len(self.qi)-1]
  else:
    for i in range(len(self.qi)):
      if bandwidth_referencial >= self.qi[i]:
        pass
      else:
        bandwidth = self.qi[i-1]
        break
```

Figura 12. Calculo de obtenção dos parâmetros de estado.

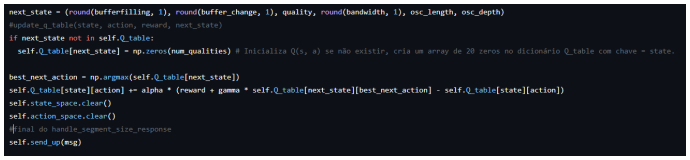
```
if len(self.quality_lista_1) < 2:
  #caso a lista tenha menos de 2 elementos não tem oscilação!
  osc_length = 0  # Comprimento da oscilação
  osc_depth = 0  # Profundidade da oscilação
else:
  if self.quality_lista_1[-1] < self.quality_lista_1[-2]:
    self.osc_list.append(self.seg_num)
    if len(self.osc_list) < 2:
      #caso a lista tenha menos de 2 elementos não tem oscilação!
      osc_length = 0  # Comprimento da oscilação
      osc_depth = 0  # Profundidade da oscilação
    else:
      o1 = self.osc_list[-1] - self.osc_list[-2]
      if o1 >= 60:
        osc_length = 0
        osc_depth = 0
      else:
        osc_length = o1
        osc_depth = self.quality_lista_1[-2] - self.quality_lista_1[-1]
  else:
    osc_length = 0
    osc_depth = 0
```

Figura 13. Continuação Calculo de obtenção dos parâmetros (oscilação) do estado.

As figura 12 e 13 acima demonstram as diferentes técnicas implementadas para obtenção de valores reais de estado. A partir desses valores a seleção de ação Softmax é executada, de maneira idêntica a exposta acima, o caso da recursividade de treinamento *for*.

2.3. Handle segment size response

Iremos utilizar a função que recebe na plataforma IR2A a resposta da requisição de segmento, para calcularmos a função de recompensa e determinar o próximo estado dessa forma obtendo todos os argumentos necessários para atualizar a Q-table, finalizando a implementação do nosso protocolo ABR. A partir dos valores de estado, transpostos pela lista state space do campo request para response, é calculado a função recompensa de maneira idêntica a demonstrada anteriormente. O próximo estado é calculado obtendo os valores mais recentes de cada parâmetro de estado, da mesma maneira que era calculado anteriormente, e definindo a tupla next state, que é argumento da atualização do dicionário Q-table, que é realizada utilizando a equação de Bellman representada na figura 14, o procedimento atualização do dicionário pode ser observado na figura abaixo:



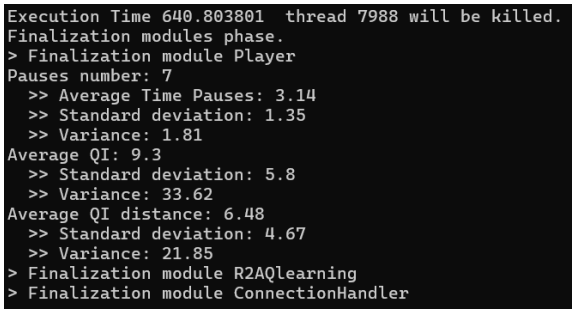
```
next_state = (round(bufferfilling, 1), round(buffer_change, 1), quality, round(bandwidth, 1), osc_length, osc_depth)
#update_q_table(state, action, reward, next_state)
if next_state not in self.Q_table:
  self.Q_table[next_state] = np.zeros(num_qualities) # Inicializa Q(s, a) se não existir, cria um array de 20 zeros no dicionário Q_table com chave = state.

best_next_action = np.argmax(self.Q_table[next_state])
self.Q_table[state][action] += alpha * (reward + gamma * self.Q_table[next_state][best_next_action] - self.Q_table[state][action])
self.state_space.clear()
self.action_space.clear()
#final do handle_segment_size_response
self.send_up(msg)
```

Figura 14. Atualização do dicionário Q-table de acordo com a equação 1.

3. Análise dos Resultados

Ao executar o código exposto no trabalho foram obtidos o seguintes comportamentos, valores e gráficos.



```
Execution Time 640.803801  thread 7988 will be killed.
Finalization modules phase.
> Finalization module Player
Pauses number: 7
  >> Average Time Pauses: 3.14
  >> Standard deviation: 1.35
  >> Variance: 1.81
Average QI: 9.3
  >> Standard deviation: 5.8
  >> Variance: 33.62
Average QI distance: 6.48
  >> Standard deviation: 4.67
  >> Variance: 21.85
> Finalization module R2AQlearning
> Finalization module ConnectionHandler
```

Figura 15. Feedback de execução do código.

A partir da figura acima depreende-se diferentes parâmetro obtidos na execução do video, como os valores de interrupções e o tempo médio de cada interrupção, qualidade média entre outros. Apesar de não ter obtido uma qualidade media muito elevada, é inferido que tiveram diversos momentos com qualidades satisfatórias devido a variância estar alta, porém isso também abre discussão a respeito da frequente de oscilações presentes. Apesar disso, foram apenas, em média, 21 segundos de pausa durante a total execução do video de 10 minutos, resultado satisfatório tendo em vista que a função recompensa prioriza os parâmetros relacionados a manutenção do buffer. A figura 16 representa o gráfico de playback do video ao longo de sua duração.

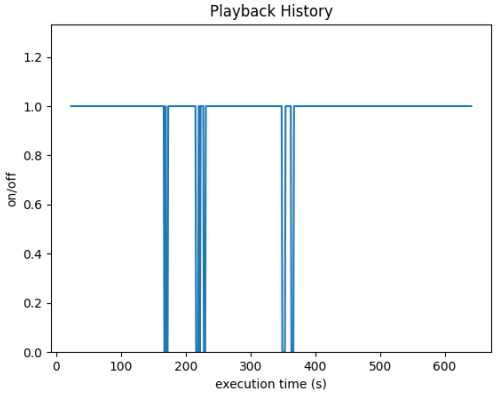


Figura 16. Gráfico que mostra o histórico de playback da execução do código proposto.

As figuras 17 e 18 capturam a tomada de decisão do agente de reduzir a qualidade solicitada de um segmento ao perceber que o buffer estava se esvaindo, sinal de flutuação na largura de banda, e a decisão de aumentar a qualidade ao perceber que o buffer estava com alto preenchimento, dando o player o luxo de buscar um segmento de melhor qualidade pela segurança de ausência de pausas graças a quantidade elevada de segmentos ja baixados.

```
Execution Time 615.790838 > selected QI: 12
Execution Time 616.758114 > buffer size: 8
Execution Time 617.758813 > buffer size: 7
Execution Time 618.759596 > buffer size: 6
Execution Time 619.760492 > buffer size: 5
Execution Time 620.760773 > buffer size: 4
Execution Time 620.92669 > received: 581, 1312787, 453048, True, http://45.171.101.167/DASHDataset/BigBuckBunny/1sec, bunny_$Bandwidth$bps/BigBuckBunny_1s$N
umber$.m4s, 45.171.101.167
Execution Time 620.926894 > measured throughput: 88204.6130214018
Execution Time 620.927043 > buffer size: 5
Execution Time 620.927183 > request: 582, 0, 0, True, http://45.171.101.167/DASHDataset/BigBuckBunny/1sec, bunny_$Bandwidth$bps/BigBuckBunny_1s$Number$.m4s,
 45.171.101.167
Execution Time 620.927502 > selected QI: 0
Execution Time 620.969684 > received: 582, 46980, 31944, True, http://45.171.101.167/DASHDataset/BigBuckBunny/1sec, bunny_$Bandwidth$bps/BigBuckBunny_1s$Num
ber$.m4s, 45.171.101.167
Execution Time 620.969927 > measured throughput: 747170.3938610854
Execution Time 620.97009 > buffer size: 6
Execution Time 620.970215 > request: 583, 0, 0, True, http://45.171.101.167/DASHDataset/BigBuckBunny/1sec, bunny_$Bandwidth$bps/BigBuckBunny_1s$Number$.m4s,
```

Figura 17. Comportamento de redução da qualidade selecionada pelo agente, devido a esvaziamento do buffer.

```
Execution Time 547.674205 > selected QI: 12
Execution Time 547.982882 > received: 531, 1312787, 1456296, True, http://45.171.101.167/DASHDataset/BigBuckBunny/1sec, bunny_$Bandwidth$bps/BigBuckBunny_1s
$Number$.m4s, 45.171.101.167
Execution Time 547.983033 > measured throughput: 4709132.728831521
Execution Time 547.983178 > buffer size: 28
Execution Time 547.983293 > request: 532, 0, 0, True, http://45.171.101.167/DASHDataset/BigBuckBunny/1sec, bunny_$Bandwidth$bps/BigBuckBunny_1s$Number$.m4s,
 45.171.101.167
Execution Time 547.983651 > selected QI: 17
Execution Time 548.657479 > buffer size: 27
Execution Time 548.958622 > received: 532, 3841983, 4604544, True, http://45.171.101.167/DASHDataset/BigBuckBunny/1sec, bunny_$Bandwidth$bps/BigBuckBunny_1s
$Number$.m4s, 45.171.101.167
Execution Time 548.958854 > measured throughput: 4719826.71137895
```

Figura 18. Outro exemplo de comportamento de aumento de qualidade selecionada do agente, devido a tamanho satisfatório do buffer.

As figuras abaixo representam, respectivamente, o gráfico da variação do preenchimento do buffer e da variação de qualidade de segmentos ao longo do *playout* do vídeo.

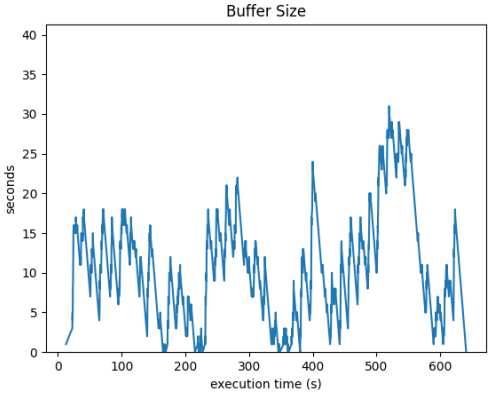


Figura 19. Feedback de execução do código.

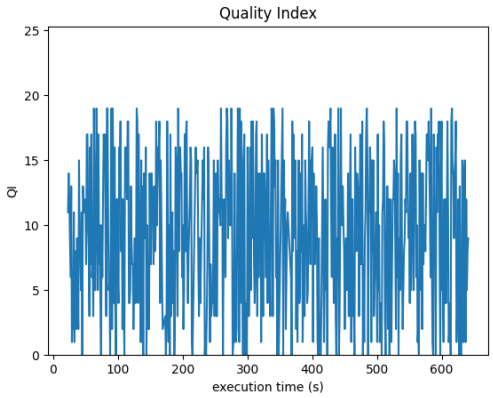


Figura 20. Feedback de execução do código.

4. Conclusão

Em suma foi desenvolvido uma implementação de um algoritmo ABR, por meio da linguagem de programação em Python, que utiliza um algoritmo de aprendizado de reforço pra criar um agente capaz de tomar decisões com base no ambiente em que ele está inserido a fim de maximizar desempenho . Por meio deste trabalho foram compreendidos variados conceitos cerca do funcionamento do processo de streaming de video e do protocolo HTTP. Os resultados obtidos demonstraram que a abordagem baseada em *reinforcement learning* tem grande potencial para aprimorar protocolos de adaptação dinâmica de streaming de video por HTTP, reforçando a viabilidade do uso desse tipo de algoritmo em aplicações de entrega de conteúdo multimídia OTT.

Referências

[Awan 2024] Awan, A. A. (2024). Ondas estacionárias em corda. `https://www.datacamp.com/pt/tutorial/introduction-q-learning-beginner-tutorial`. [Online; accessed 24-Abril-2024].

[Claeys 2022] Claeys, M. (2022). Design of a q-learning-based client quality selection algorithm for http adaptive video streaming. In Department of Information Technology, G. U.-i., editor, *New Trends in Animation and Visualization*. Alcatel-Lucent Bell Labs, Copernicuslaan 50, B-2018 Antwerpen, Belgium.