CIC0124 - REDES DE COMPUTADORES - T03 - 2022/1

Nome: Carlos Eduardo da Silva Almeida
Matrícula: 180014625

Introdução

Recentemente o streaming de vídeo tem sido usado massivamente pelos usuários de dispositivos móveis. Esse uso extensivo da internet para reprodução de vídeos tem atingido 55% da banda total desses dispositivos. Tal demanda tem feito com que o uso de algoritmos para adaptação da qualidade do vídeo seja uma área de pesquisa ativa e em crescimento.

Isso se deve ao fato que a qualidade da rede pode sofrer oscilações durante o uso pelo usuário. Dessa forma, o uso de protocolos que se adaptam a essas mudanças se faz mister uma vez que afeta diretamente a qualidade de experiência do usuário.

Uma proposta de solução é a seguinte: Pegar as qualidades recebidas e com base nelas calcular as vazões para tirar a média e com base nelas calcular a probabilidade da qualidade do vídeo aumentar ou diminuir.

Explicação do código do algoritmo ABR

Primeiramente foi criada a classe R2ADsss que foi iniciada com os atributos qi,que é uma lista das qualidades possíveis; time, que guarda o tempo em que foi feita a requisição; vazões, que guarda em uma lista às vazões da rede calculadas e a lista SS que guarda os índices das qualidades que são maiores que a última vazão na lista de vazões.

```
class R2ADsss(IR2A):

    def __init__(self, id):
        IR2A.__init__(self, id)

        self.qi = []
        self.time = 0
        self.vazoes = []
        self.SS = []

    def handle_xml_request(self, msg):

        self.time = time.time()

        self.send_down(msg)
```

Na função handle_xml_response usa-se os métodos de parsed_mpd para adquirir a lista de qualidades. Usa-se a variável t e o método time.perf_counter() para adquirir o tempo exato entre a requisição e a resposta. Com esse tempo calcula-se as vazões e as adiciona na lista vazoes. Usa-se o laço for para pegar os índices das qualidades que são menores que a última vazão registrada.

```
    def handle_xml_response(self, msg):

        parsed_mpd = parse_mpd(msg.get_payload())
        self.qi = parsed_mpd.get_qi()

        t = time.perf_counter() - self.time
        self.vazoes.append((msg.get_bit_length()) / t)

        for i in range(len(self.qi) - 1):
            if self.vazoes[-1] > self.qi[i]:
                self.SS.append(i)

        self.send_up(msg)
```

Na função handle_segment_size_request é calculada a média das vazões por meio da função mean do módulo math do python, M assume o valor do tamanho da lista de vazões. Com isso é calculado um "desvio padrão ponderado" de quanto uma vazão dista da média das vazões. Usando-se deste valor e da média das vazões se calcula a probabilidade de mudar a qualidade do segmento de vídeo. Com o valor do tamanho da lista de índices

da qualidade já passadas se calcula τ que determina a tendência em diminuir a qualidade do vídeo e Θ que é a tendência em aumentar a qualidade do vídeo.

```
def handle_segment_size_request(self, msg):

    self.time = time.perf_counter()

    media_vazoes = mean(self.vazoes)
    M = len(self.vazoes)
    weight = 0
    for i in range(M):
        weight = weight + (abs(self.vazoes[i] - media_vazoes) * i)/M

    p = media_vazoes / (media_vazoes + weight)

    print(f'P = {p}')

    max_omega = 0

    if len(self.SS) == 0:
        max_omega = 0
    else:
        if len(self.SS) == 1:
            max_omega = self.SS[-1] - 1
        else:
            max_omega = self.SS[-1] - 1

    print(f'MAX_OMEGA = {max_omega}')
```

A qualidade do vídeo a ser baixada vai ser dada pela fórmula 6 do artigo "Dynamic Segment Size Selection in HTTP Based Adaptive Video Streaming". Foi feita uma adaptação pois no artigo se trabalha com tamanho de segmento de vídeo enquanto neste trabalho se trabalha com qualidade de vídeos de tamanho de segmento iguais a 1 segundo. A adaptação é feita da seguinte forma: Se a lista de qualidades anteriores estiver vazia se escolhe a menor qualidade possível para o cálculo da qualidade que será baixado, caso contrário,

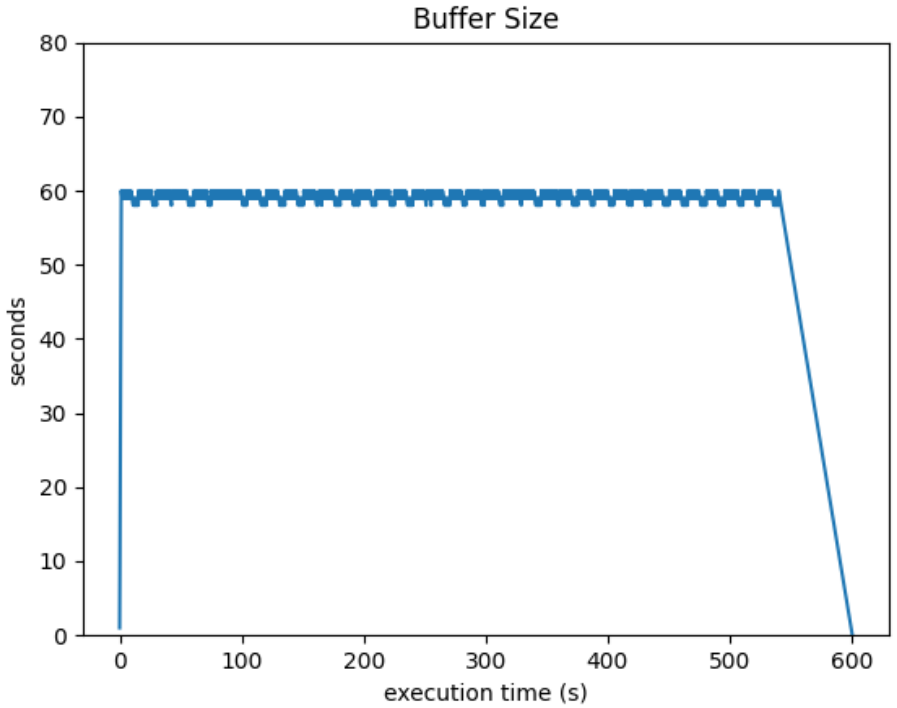
```
    t_estranho = (1 - p) * self.qi[max_omega]

    print(f'T_ESTRANHO = {t_estranho}')

    min_omega = 0

    n = len(self.qi)

    if len(self.SS) != 0:
        min_omega = n - 1
    else:
        min_omega = 1

    print(f'MIN_OMEGA = {min_omega}')

    tetha = p * self.qi[min_omega]

    ss_linha = 0

    if len(self.SS) != 0:
        ss_linha = abs(self.qi[self.SS[-1]] - t_estranho + tetha)
    else:
        ss_linha = abs(self.qi[0] - t_estranho + tetha)

    avg = ss_linha
```

Então com o resultado do cálculo da qualidade atribui-se a variável avg o mesmo. Usando um laço for seleciona-se a qualidade inferior a qualidade calculada e a adicionar a lista de id das qualidades. Na função handle_segment_size_response o que se faz é calcular o tempo entre uma requisição e sua respectiva resposta para se calcular a vazão e dps por o id da qualidade que melhor se aproxima por baixo das qualidades disponíveis na lista SS.

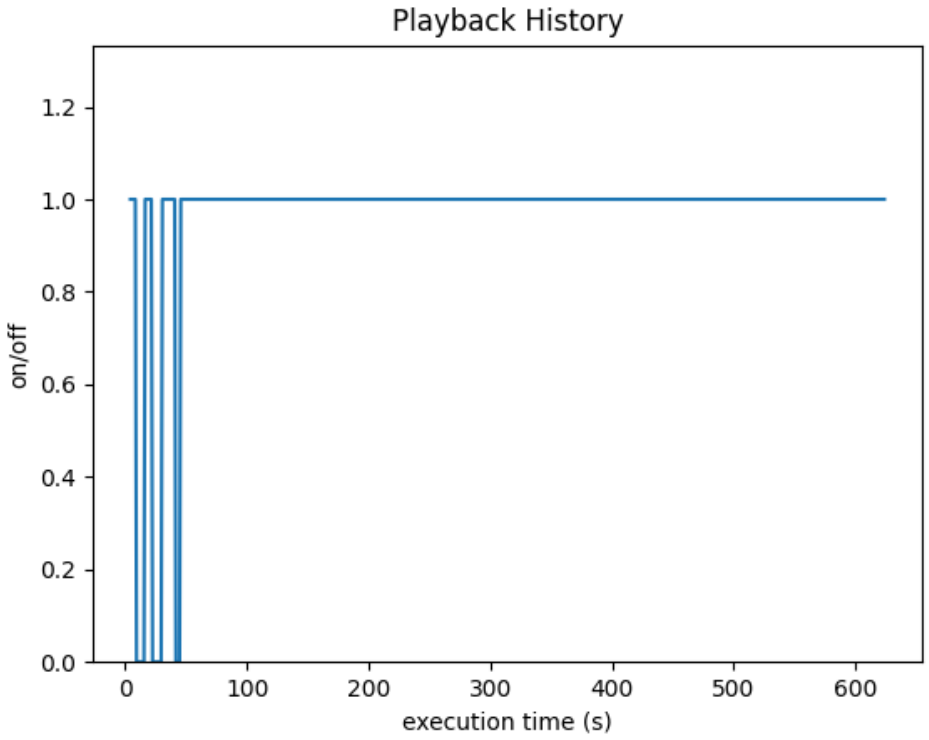
```
        print(f'SS_LINHA = {avg}')

        selected_qi = 0
        for i in range(len(self.qi)):
            if avg > self.qi[i]:
                selected_qi = i

        msg.add_quality_id(self.qi[selected_qi])
        self.send_down(msg)

    def handle_segment_size_response(self, msg):

        t = time.perf_counter() - self.time
        self.vazoes.append((msg.get_bit_length()) / t)
        for i in range(len(self.qi)):
            if self.vazoes[-1] > self.qi[i]:
                self.SS.append(i)
        self.send_up(msg)

    def initialize(self):

        pass

    def finalization(self):
        pass
```

Gráficos

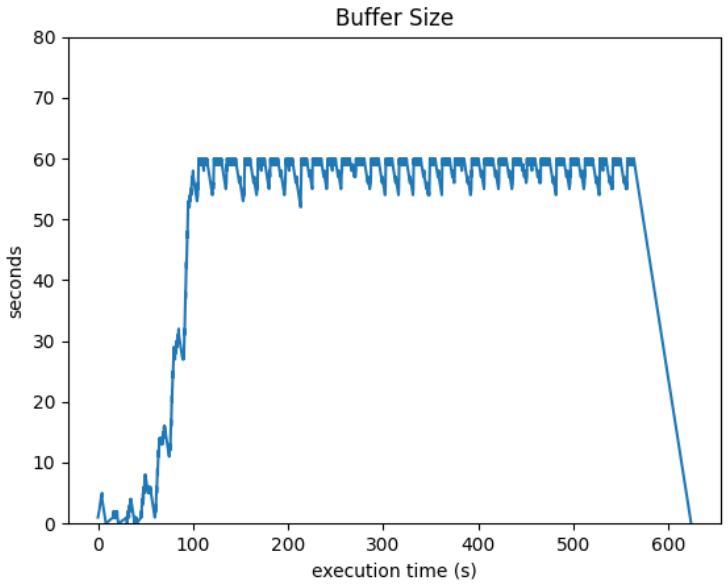
Pode-se ver pela estabilidade do tamanho do Buffer que o algoritmo implementado faz as escolhas certas de qualidade na maioria das vezes.

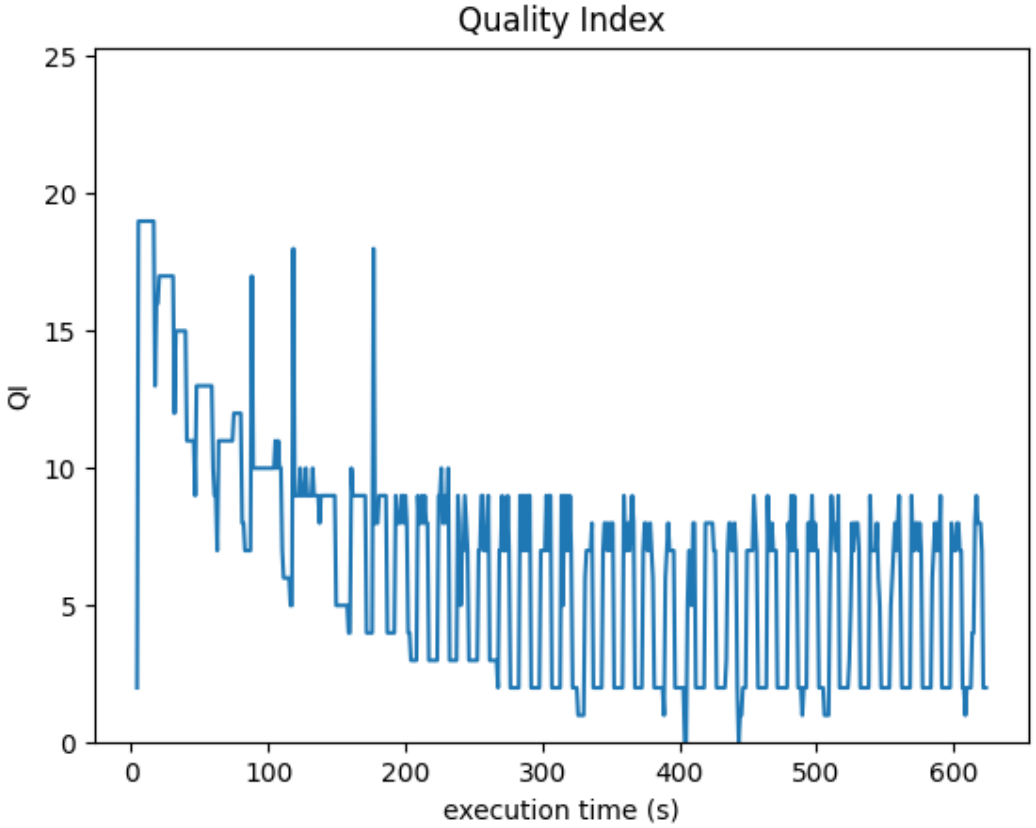


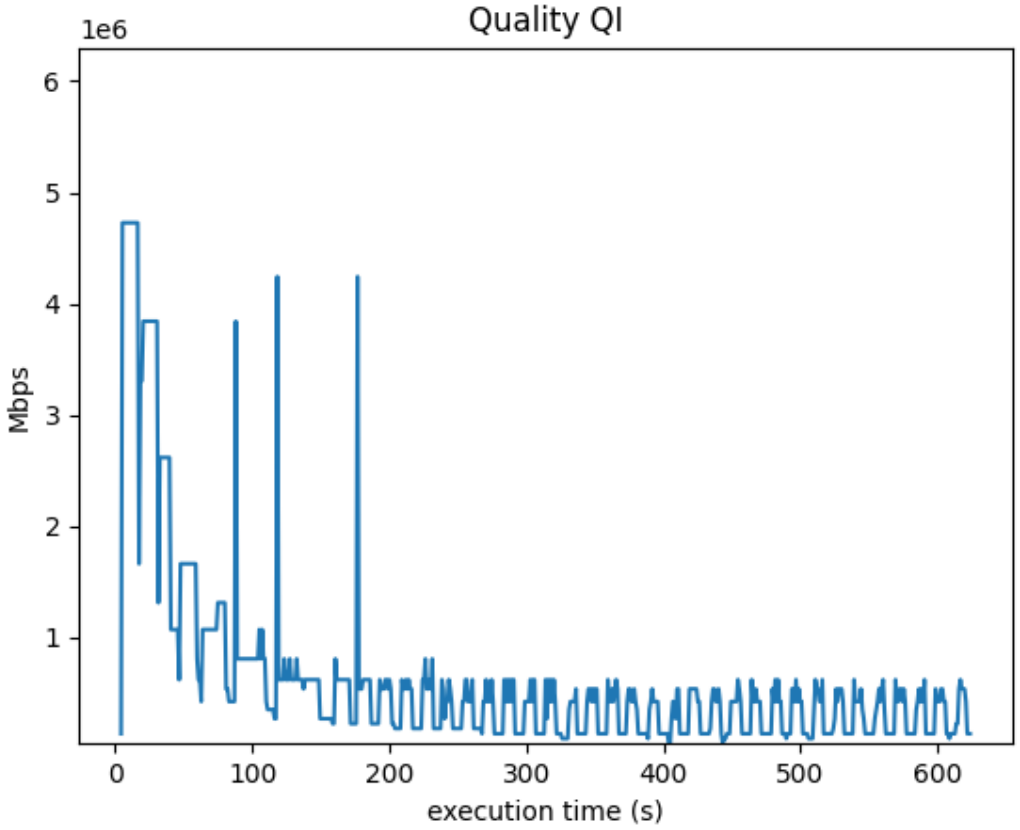
Analisando o histórico de reprodução temos que o algoritmo executou o vídeo com poucas pausas, sendo elas no total 3.

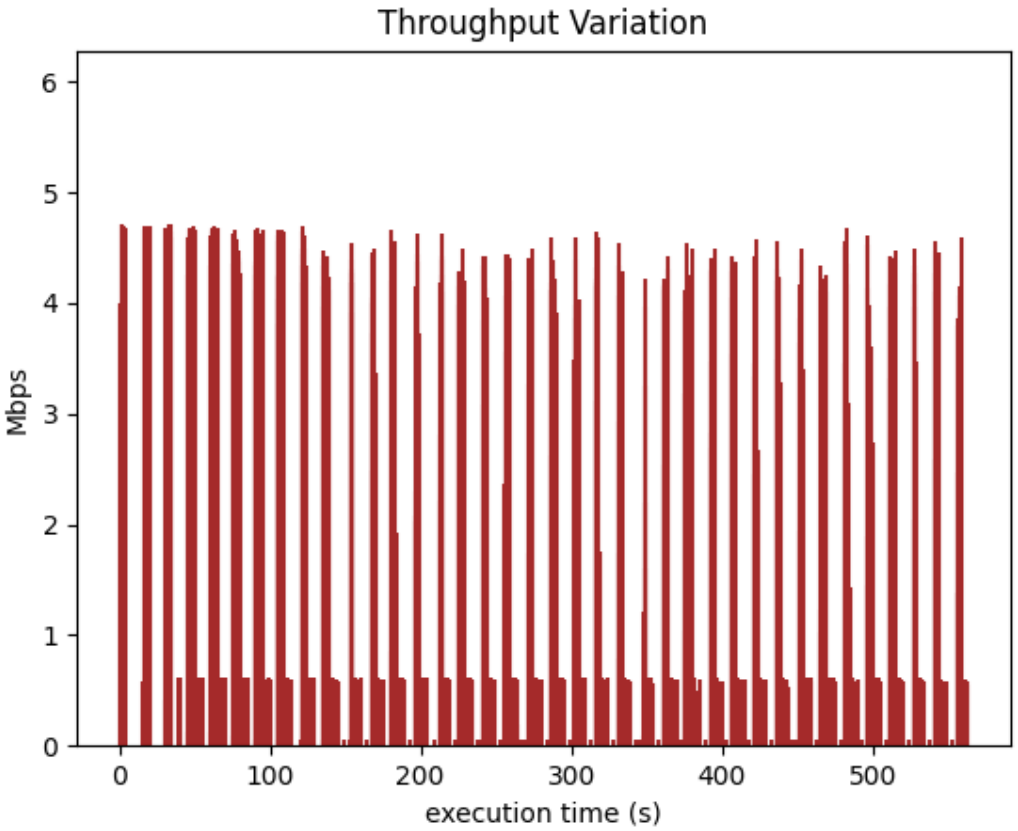


A regularidade do tamanho do Buffer indica que o algoritmo escolheu qualidades boas para a rede mesmo havendo variações.




Quality Index
QI
execution time (s)
0
5
10
15
20
25
0
100
200
300
400
500
600


Quality QI
1e6
Mbps
execution time (s)
1
2
3
4
5
6
0
100
200
300
400
500
600



Conclusão

De acordo com as estatísticas mostradas no final do vídeo houve 3 pausas com média de 6,4 segundos. Julgando que o vídeo tem 596 segundos temos que a porcentagem de 0,3% de tempo de pausa, o que é bem aceitável. E levando em conta que os gráficos apontam para uma adaptação da qualidade creio que os objetivos do trabalho foram atingidos.