Algoritmos Adaptativos de Streaming de Video MPEG-DASH - Implementação do Algoritmo ABR

Kleber Rodrigues da Costa Júnior, 20/0053680
Paulo Victor França de Souza - 20/0042548
Thais Fernanda de Castro Garcia - 20/0043722
Grupo 11

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0124 – Redes de Computadores – 2021.2, turma A.

200053680@aluno.unb.br, 200042548@aluno.unb.br, 200043722@aluno.unb.br

Universidade de Brasília
Departamento de Ciência da Computação

Abstract. *This report discusses about adaptive bitrate video transmission algorithms (ABR), where the BOLA algorithm (Buffer Occupancy Base Lyapunov Algorithm) was created and implemented, with its explanation, evaluation, results and others.*

Resumo. *Este relatório discorre a respeito de algoritmos de transmissão de vídeos com taxa de bits adaptável (ABR, Adaptive Bitrate Streaming), onde foi criado e implementado o algoritmo BOLA (Buffer Occupancy Base Lyapunov Algorithm), possuindo sua explicação, avaliação, resultados e entre outros.*

1. Introdução

Com o aprimoramento das infraestruturas de conexões banda larga e 4G foi permitido um grande crescimento do entretenimento via *streaming* de vídeo, isto é, uma tecnologia que usa internet e possibilita o envio de informações multimídia para computadores e outros dispositivos sem perder sua conexão com a Internet ou necessitar um tempo de espera para download e acesso ao conteúdo e é responsável por mais da metade do tráfego de internet no mundo. Dessa forma, é de suma importância que esses serviços tenham um bom controle da qualidade de reprodução para que não ocorra interrupção na reprodução no *streaming* e que sua qualidade não seja prejudicada pelas possíveis variações nas condições de rede ou algo parecido.

Por meio deste artigo, é relatado o processo de implementação de um algoritmo ABR (*Adaptive Bitrate Streaming*), ou seja, um algoritmo de transmissão de vídeos com taxa de bits adaptável em um cliente DASH (*Dynamic Adaptive Streaming over HTTP*), também chamado de MPEG-DASH, que é uma técnica de adaptação de *streaming* de taxa de vídeo que permite a transmissão de vídeos de alta qualidade na Internet, onde o seu funcionamento baseia-se em dividir o conteúdo em pequenos segmentos, onde cada um contém um curto intervalo de tempo de reprodução de conteúdo. Basicamente, a união desses segmentos formam um vídeo ou transmissão ao vivo. O protocolo MPEG-DASH define diversas funcionalidades essenciais para o funcionamento de conteúdo multimídia.

Para o desenvolvimento deste trabalho foi utilizado como base o filme de animação Big Buck Bunny [Blender Foundation 2008], filme lançado em 2008, que conta a história sobre um coelho gigante, tendo como vilão um esquilo voador e seus três ajudantes, sendo usado apenas a reprodução do vídeo do filme sem o áudio. O filme está disponível em um servidor HTTP Apache/2.4.41 (Ubuntu), com o link: http://45.171.101.167/DASHDataset, dividido em 6 possíveis tamanhos de segmento e codificado em 20 formatos diferentes, variando de uma taxa de bits de 46980bps até 4726737bps **(Figura 3)**. O objetivo é receber demandas de um tocador de mídia para baixar o arquivo *.mpd* relativo ao filme que será usado como base de testes, e com isso, consequentemente baixar cada um dos segmentos do filme.

Para a realização deste projeto, foi implementado o algoritmo BOLA (*Buffer Occupancy Base Lyapunov Algorithm*), um algoritmo de adaptação de bitrate quase ideal para vídeos on-line, sendo usado de base para conhecimento dessa implementação o artigo BOLA: Near-Optimal Bitrate Adaptation for Online Videos [Spiteri et al. 2020], sendo executado e testado na linguagem de programação python por meio do *framework* (estrutura) *PyDash*, liderado pelos professores Dr. Marcos Caetano e Dr. Marcelo Marotta da Universidade de Brasília (UnB), Distrito Federal, Brasil. Basicamente, se trata de uma estrutura para o desenvolvimento de algoritmos de *streaming* de vídeo adaptáveis (ABR), sendo uma ferramenta de aprendizado projetada para abstrair os detalhes da comunicação de rede, permitindo que estudantes e outras pessoas se concentrem exclusivamente no desenvolvimento e avaliação de protocolos ABR.

O algoritmo BOLA funciona majoritariamente utilizando informações sobre o *buffer* (processo de pré-carregamento de dados em uma área reservada da memória) e fornece a qualidade do próximo segmento a ser baixado, portanto, ele funciona de acordo com vazão e, ou também, pelo nível de *buffer*, selecionando automaticamente a maior qualidade possível a ser baixada sem causar travamento ou *re-buffering* (repetição do processo

de pré-carregamento de dados em uma área reservada da memória).

Figura 1. Pôster do filme de animação Big Buck Bunny. [Blender Foundation 2008]

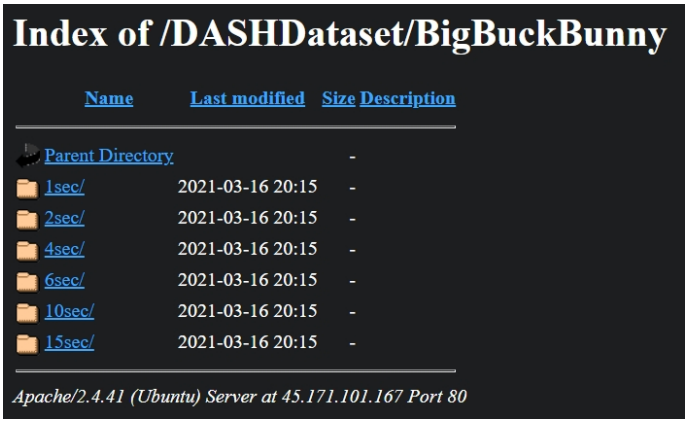

Figura 2. Filme Big Buck Bunny dividido em segmentos de tamanhos fixos.

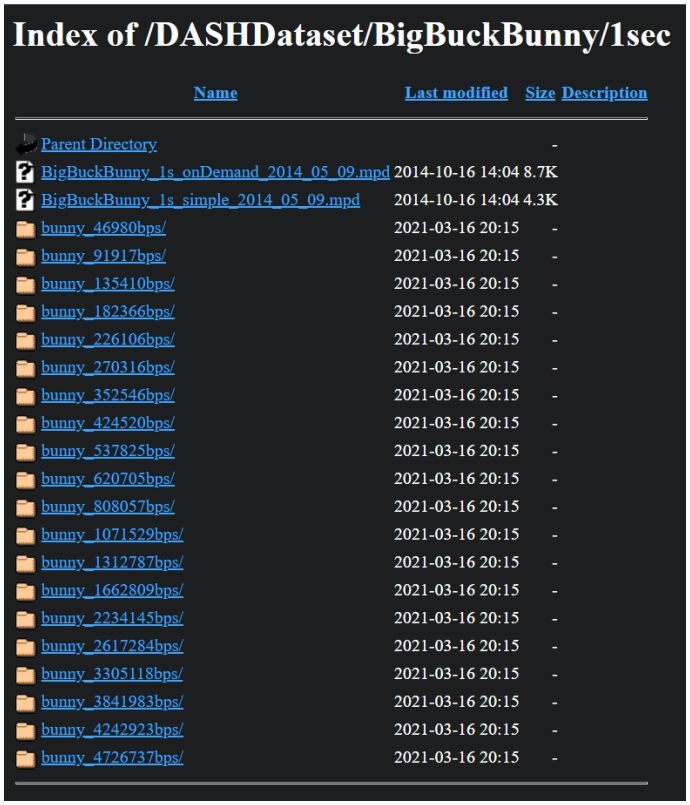

Figura 3. Variação de taxa de bits (bps) do filme Big Buck Bunny (tamanho fixo de 1 segundo).

2. O Algoritmo BOLA

O algoritmo BOLA (do inglês *Buffer Occupancy based Lyapunov Algorithm*) é um exemplo de algoritmo, dentre vários, que resolve o problema de selecionar uma qualidade adequada de um segmento de vídeo, fazendo desse problema uma questão de otimização de utilidade, valendo-se da otimização *Lyapunov* para maximizar a qualidade do vídeo e diminuir o tempo de *re-buffering*. Este algoritmo atinge uma taxa de utilidade média de tempo presente dentro de um termo aditivo O(1/V) do valor ótimo, sendo V um parâmetro de controle relacionado ao tamanho do *buffer* de vídeo.

O BOLA se caracteriza, além do seu funcionamento, por necessitar somente da necessidade de conhecimento o nível do *buffer*, não importando a largura de banda da rede. Sendo assim, este algoritmo converte o problema de otimizar as métricas de média de tempo, sujeitas à restrição, em uma série de problemas de otimização de *slots* (aberturas). O problema a ser resolvido em cada *slot* envolve a minimização de uma razão entre o valor esperado de desvio mais a penalidade naquele *slot* para o comprimento esperado do *slot*. Dessa forma, cada segmento deve ser inteiramente baixado antes que seja reproduzido, os segmentos são baixados na mesma ordem que são reproduzidos, evitando erros. Caso o *buffer* esteja cheio o *player* não pode baixar nenhum segmento novo e deve esperar o período tempo de fixo de Δ segundos antes de fazer uma nova tentativa de download, enquanto os segmentos já baixados são reproduzidos em um ritmo fixo de de $\frac{1}{p}$ segmentos por segundo. Sendo assim, a escolha do *bitrate* para um segmento impacta o seu tempo de download.

O artigo BOLA: Near-Optimal Bitrate Adaptation for Online Videos [Spiteri et al. 2020] descreve de forma mais aprofundada os aspectos matemáticos envolvidos a respeito da definição do problema de otimização, contudo, o BOLA se demonstrou bastante ágil comparado a outros algoritmos adaptativos de streaming de vídeo MPEG-DASH, podendo ter grande utilidade e aplicação.

2.1. Implementação do Algoritmo BOLA

Posteriormente será apresentado o processo da implementação do algoritmo BOLA, explicitando com trechos de código todo o procedimento, sendo usado um modelo de programação orientada a objetos (POO). Primeiramente foi utilizada uma classe que auxilia na implementação do algoritmo, onde se inicia com oito atributos, sendo alguns deles de valor fixo para uso nas fórmulas futuras **(Figura 4)**.

```
class R2A_BOLA(IR2A): # Classe do algoritmo BOLA
                      # Utiliza informações sobre o buffer e fornece a qualidade do próximo segmento a ser baixado.
                      # De acordo com a vazão e/ou nível de buffer, seleciona automaticamente a maior qualidade possível a ser baixada.
                      # Evita travamentos ou re-buffering na transmissão.

    def __init__(self, id): # Método de inicialização.
        IR2A.__init__(self, id)

        self.whiteboard = Whiteboard.get_instance() # Estatísticas em tempo real geradas pela plataforma
        self.parsed_mpd = '' # mpd é a extensão do arquivo de vídeo.
        self.lista_qi = [] # Quality index (endereço da qualidade).
        self.tempo_requisição = 0 # Tempo de requisição.
        self.lista_vazões = [] # Lista de vazões.
        self.gamma = 5
        self.qd_max = 0
        self.r_agora = 0
```

Figura 4. Método construtor da classe.

Para se lidar com as requisições XML (eXtensible Markup Language, formato de arquivo universal usado para criar documentos com dados organizados), foi usada uma função chamada *handle_xml_request*, onde no final é enviado uma mensagem para a camada de baixo **(Figura 5)**.

```
def handle_xml_request(self, msg): # Trata a requisição do xml
    self.tempo_requisição = perf_counter() # Tempo de requisição (tempo de desempenho, em segundos).

    self.send_down(msg) # Requisição é de cima para baixo (deve-se enviar a mensagem para camada de baixo).
```

Figura 5. Método de tratamento de requisições XML.

Além disso, é necessário também um método que lide com as respostas XML, isto é, foi criado uma função chamada de *handle_xml_response*, este método calcula a vazão, coloca ela na lista de vazões e ao final de tudo envia a mensagem para a camada de cima **(Figura 6)**.

```
def handle_xml_response(self, msg): # Trata a resposta do xml.
    self.parsed_mpd = parse_mpd(msg.get_payload())
    self.lista_qi = self.parsed_mpd.get_qi() # Lista de qualidades (self.lista_qi) é criada através do parser feito no conteúdo do arquivo mpd

    self.lista_vazões.append(msg.get_bit_length() / (perf_counter() - self.tempo_requisição)) # Calcula a vazão e coloca ela na lista de vazões.

    self.send_up(msg) # Resposta é de baixo para cima (deve-se enviar a mensagem para camada de cima).
```

Figura 6. Método de tratamento de respostas XML.

Por fim, é tratado as requisições e respostas relacionadas ao tamanho do segmento com os métodos *handl_segment_size_request* e *handle_segmentsize_response* **(Figuras 7 e 8)**. Ao finalizar todo procedimento é permitido a transmissão de alta qualidade de conteúdo de mídia pela Internet com um desempenho relativamente rápido.

```python
def handle_segment_size_request(self, msg): # Recebe como parâmetro uma msg do tipo base.SSMessage (requisição de a um segmento de vídeo (ss)).
    self.tempo_requisição = perf_counter() # Tempo de requisição.

    t = min(msg.get_segment_id(), 596 - msg.get_segment_id()) # 596 é a duração total do filme em segundos.
    t_linha = max(t/2, 3*msg.get_segment_size())
    qd_max = min(self.whiteboard.get_max_buffer_size(), t_linha)
    self.V = (qd_max - 1)/(log(self.lista_qi[-1]/self.lista_qi[0]) + self.gamma)

    buffer = self.whiteboard.get_playback_buffer_size()

    if len(buffer) > 0:
        bufferSize = buffer[-1][1]
    else:
        bufferSize = 0

    prev = -math.inf
    m_line = 0
    idx = 0

    i = 0
    while i < len(self.lista_qi):
        if (self.V*log(self.lista_qi[i]/self.lista_qi[0])  + self.V*self.gamma - bufferSize) / self.lista_qi[i] >= prev:
            prev = (self.V*log(self.lista_qi[i]/self.lista_qi[0]) + self.V*self.gamma - bufferSize) / self.lista_qi[i]
            idx = i
        if self.lista_qi[i] <= max(self.lista_vazões[-1], self.lista_qi[0]):
            m_line = i
        i+=1

    if idx >= self.r_agora:
        if m_line >= idx:
            m_line = idx
        elif m_line < self.r_agora:
            m_line = self.r_agora
        else:
            m_line += 1

        idx = m_line

    self.r_agora = idx
    msg.add_quality_id(self.lista_qi[self.r_agora])

    self.send_down(msg) # Requisição é de cima para baixo (deve-se enviar a mensagem para camada de baixo).
```

Figura 7. Método de tratamento de requisições de tamanho do segmento.

```python
def handle_segment_size_response(self, msg): # Recebe como parâmetro uma msg do tipo base.SSMessage (resposta para a requisição de um segmento de vídeo específico).
    self.lista_vazões.append(msg.get_bit_length() / (perf_counter() - self.tempo_requisição)) # Calcula a vazão e coloca ela na lista de vazões
    self.send_up(msg) # Resposta é de baixo para cima (deve-se enviar a mensagem para camada de cima).
```

Figura 8. Método de tratamento de respostas de tamanho do segmento.

2.2. Avaliação do Algoritmo BOLA

Foram avaliados os resultados provenientes dos testes realizados com o algoritmo BOLA na plataforma *PyDash*, primeiramente, no caso HHHHHH **(Figura 9)**, onde foi observado bons resultados no que se diz respeito ao tamanho do *buffer* em relação ao tempo em segundos, mesmo que em certo momento o tamanho do *buffer* tenha crescido bastante.

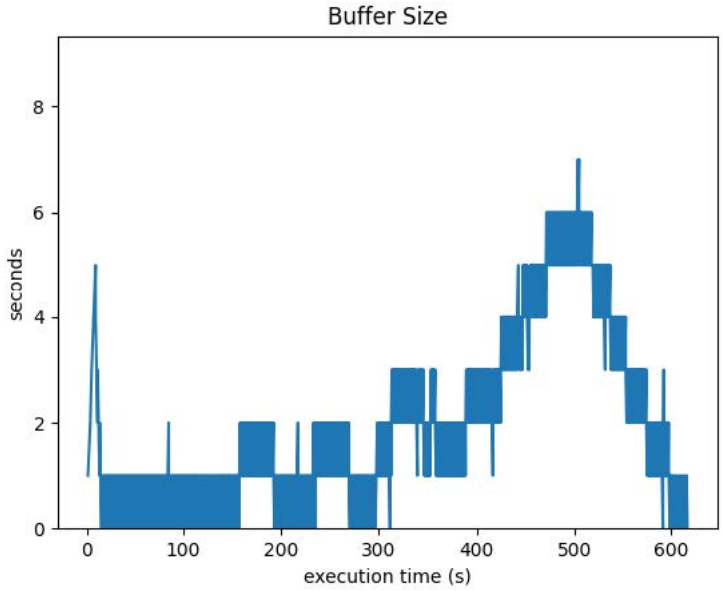

Figura 9. Gráfico do tamanho do buffer em relação ao tempo do algoritmo BOLA no caso HHHHHH.

Dessa forma, depois foi analisado também um teste no caso LLLLLL **(Figura 10)** e MMMMMM **(Figura 11)**, onde o algoritmo se comportou muito bem, apesar de apresentar algumas irregularidades em seu comportamento, possuindo um desempenho mais estável do que no caso HHHHHH, onde no começo do vídeo há um aumento rápido do tamanho do *buffer* e no final há um decaimento rápido também, pelo fato do vídeo ter chegado ao final.

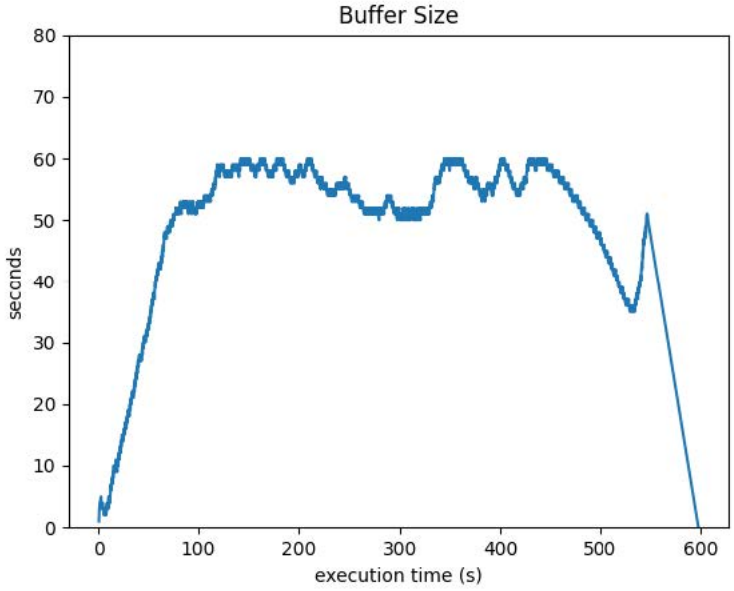

Figura 10. Gráfico do tamanho do buffer em relação ao tempo do algoritmo BOLA no caso LLLLLL.

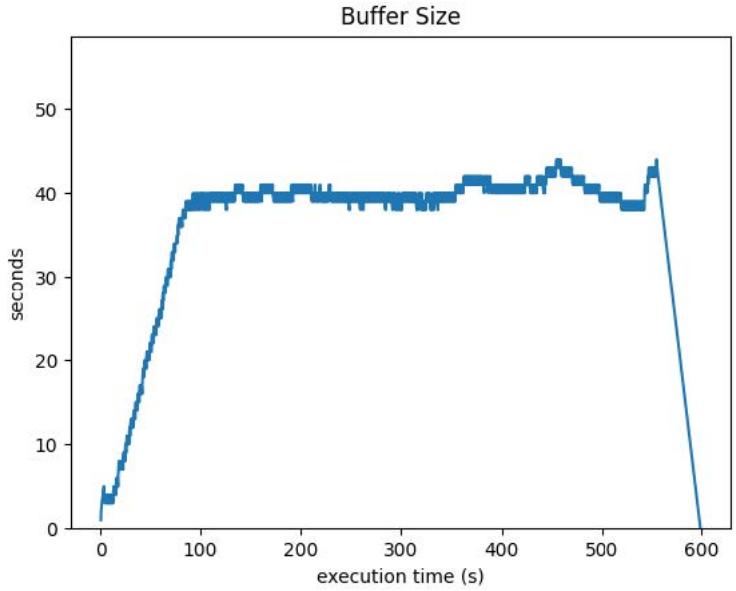

Figura 11. Gráfico do tamanho do buffer em relação ao tempo do algoritmo BOLA no caso MMMMMM.

Foi oberservado que a plataforma *PyDash* gerou taxas de transferências estáveis em relação ao tempo, independente do caso, isto é, os casos HHHHHH, LLLLLL e MMMMMM (**Figuras 12, 13 e 14**).

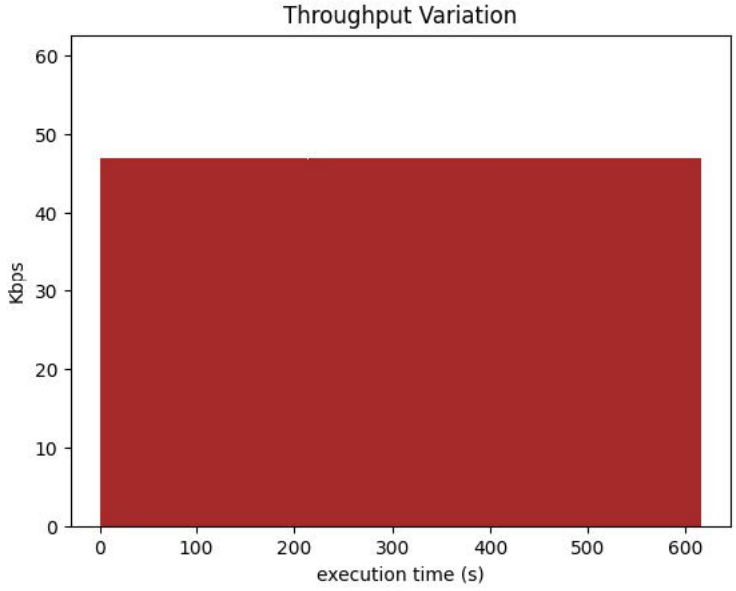

Figura 12. Gráfico da taxa de transferência em relação ao tempo do algoritmo BOLA no caso HHHHHH.

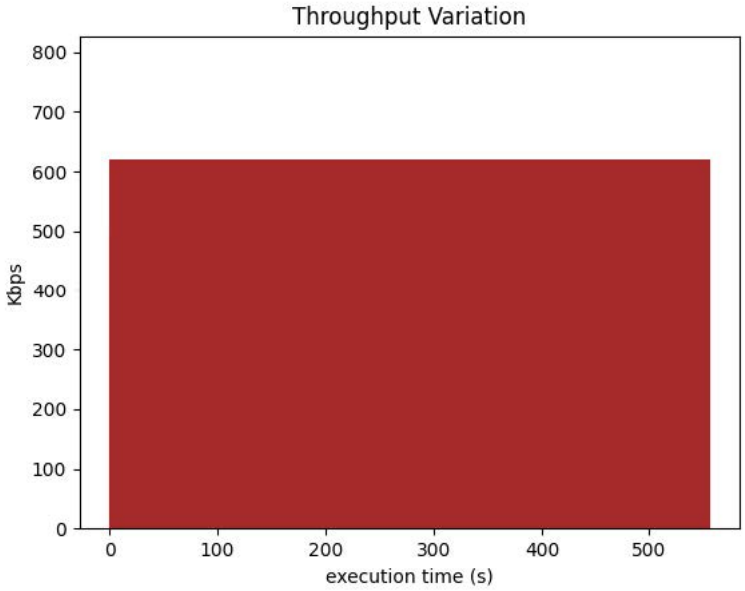

Figura 13. Gráfico da taxa de transferência em relação ao tempo do algoritmo BOLA no caso MMMMMM.

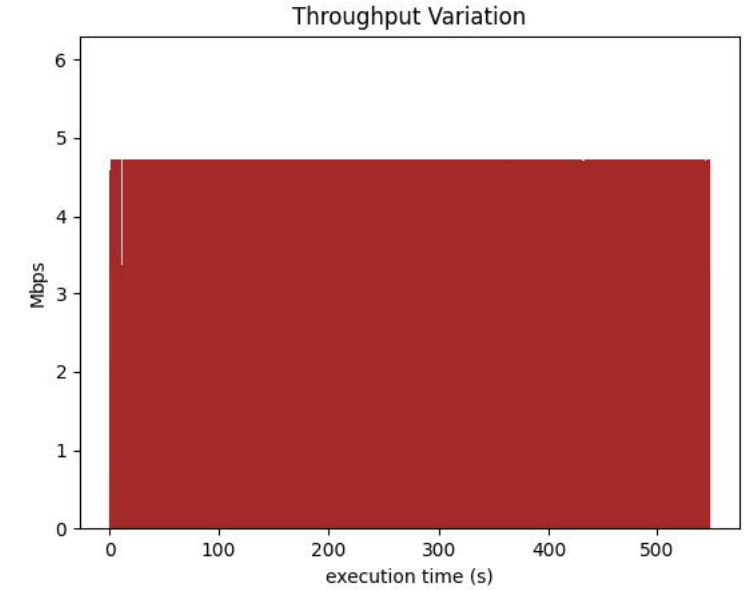

Figura 14. Gráfico da taxa de transferência em relação ao tempo do algoritmo BOLA no caso LLLLLL.

Foi avaliado também, o índice de qualidade **(Figuras 15 e 16)**, onde é possível observar que houve um crescimento bem acentuado no início do gráfico do algoritmo BOLA, porém, posteriormente acaba apresentando certa instabilidade para conseguir manter a qualidade.

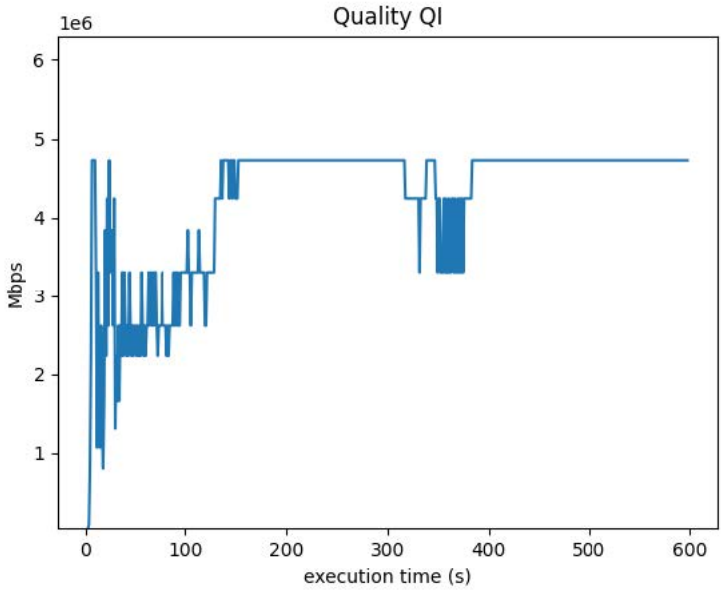

Figura 15. Gráfico do índice de qualidade do algoritmo BOLA no caso LLLLLL.

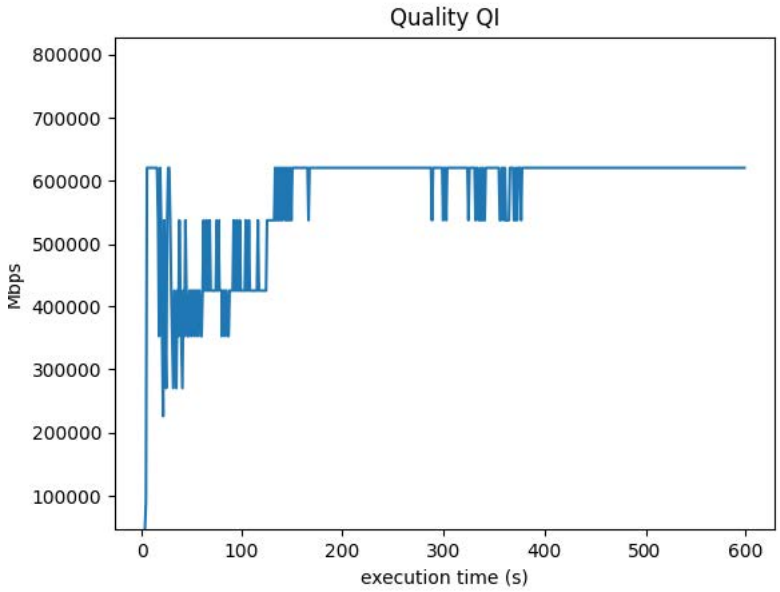

Figura 16. Gráfico do índice de qualidade do algoritmo BOLA no caso MMMMMM.

Em seguida **(Figura 17)**, pode-se notar que o algoritmo BOLA fez uma quantidade significativa de pausas, porém isso é justificado com o fato do algoritmo ter feito escolhas priorizando qualidades maiores, sendo o motivo dessas pausas.

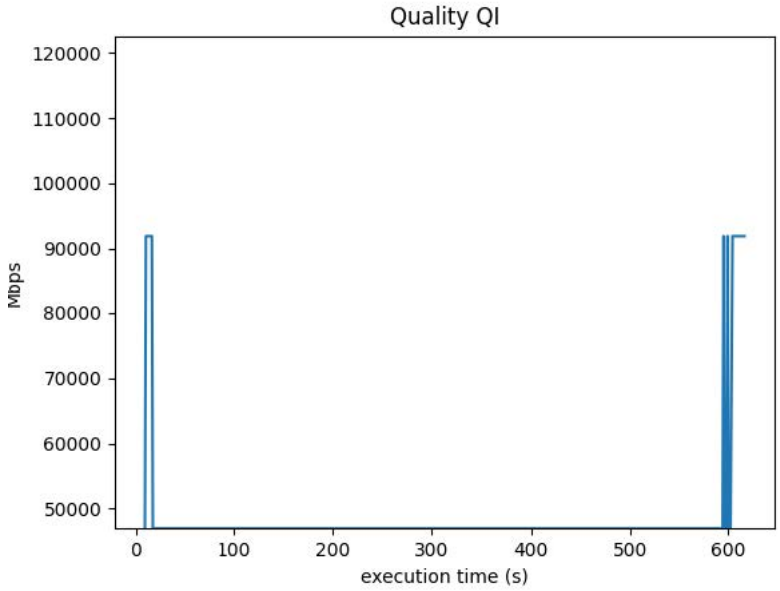

Figura 17. Gráfico do índice de qualidade do algoritmo BOLA no caso HHHHHH.

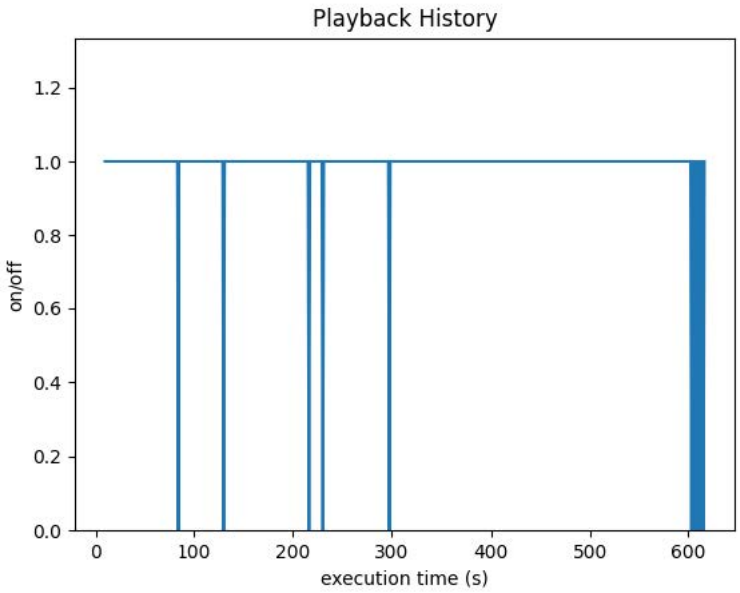

Figura 18. Gráfico do *playback* do algoritmo BOLA no caso HHHHHH

Em relação ao número de pausas, ao analisar o caso HHHHHH, tendo em vista sua dura limitação na largura de banda. Foi observado **(Figura 18)** que o algoritmo se saiu bem e obteve doze pausas no total, possuindo uma qualidade média foi de 0,03. Já nos casos LLLLLL e MMMMM, foram apresentadas zero pausas.

3. Conclusão

Neste trabalho foi explorado o algoritmo BOLA (Buffer Occupancy Base Lyapunov Algorithm) para escolher dinamicamente a qualidade do vídeo de forma que se adaptasse às variações de rede. Foi criado e implementado o algoritmo de forma satisfatória, garantindo uma boa qualidade e apresentando bons resultados no que se propõe.

Foi entendido também o funcionamento do mecanismo de *stream* e do protocolo *MPEG-DASH*, sendo de imprescindível ajuda para a implementação de maneira correta e satisfatório do algoritmo escolhido dentre vários existentes. Dessa forma, conclui-se que este protocolo é essencial para o serviço de *streaming* de vídeos, o qual representa uma parcela mais que majoritária do trafego da Internet atualmente, sendo de suma importância um bom desempenho neste quesito.

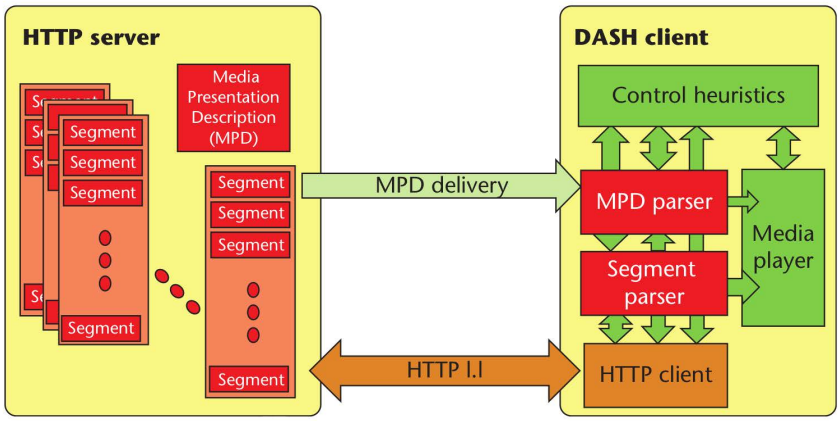

Figura 19. Padrão MPEG-DASH.

4. Referências

[Sodagar 2011] Sodagar, I. (2011). The MPEG-DASH Standard for Multimedia Streaming Over The Internet. IEEE multimedia, 18(4):62–67. (ieeexplore.ieee.org/document/6077864).

[Spiteri et al. 2020] Spiteri, K., Urgaonkar, R., and Sitaraman, R. K. (2020). BOLA: Near-Optimal Bitrate Adaptation For Online Videos. IEEE/ACM Transactions on Networking, 28(4):1698–1711. (ieeexplore.ieee.org/abstract/document/9110784).

[Blender Foundation 2008] Blender Foundation (2008), Big Buck Bunny. Wikipédia, a enciclopédia livre. (pt.wikipedia.org/wiki/Big$_B uck_B unny$

[UnB 2022] UnB, R. C. (2022). Arquivo "Especificacao_pydash.pdf"na disciplina CIC0124 – Redes de Computadores – 2021.2. (aprender3.unb.br/mod/resource/view.php?id=580236).

[M. A. Marotta, G. C. Souza, M. Holanda, M. F. Caetano 2021] M. A. Marotta, G. C. Souza, M. Holanda and M. F. Caetano. "PyDash - A Framework Based Educational Tool for Adaptive Streaming Video Algorithms Study", 2021 IEEE Frontiers in Education Conference (FIE), 2021, pp. 1-8, doi: 10.1109/FIE49875.2021.9637335. (ieeexplore.ieee.org/document/9637335).