Transmissão de Dados DASH por Seleção Dinâmica de Tamanho de Segmento

Mariana Mendanha da Cruz - 16/0136784 - mariana.mendanha.mm@gmail.com
Alexandre Ferreira - 12/0025175 - alexandre.ps1123@gmail.com
Yudi Yamane - 16/0149419 - yudiyamane@gmail.com
Engenharia de Controle e Automação
Faculdade de Tecnologia - UnB

***Abstract*—Projeto de Programação de Algoritmos Adaptativos de Streaming de Vídeo MPEG-DASH**

I. Introdução

O objetivo deste projeto é a implementação de uma técnica adaptativa de streaming (algoritmo ABR) em um cliente DASH, utilizando os conhecimentos abordados em sala de aula. Para tal implementação, foi proposta uma plataforma básica funcional chamada pyDash, cedida pelo professor, utilizando os recursos da linguagem python, na qual possui o ambiente necessário para realização do algoritmo ABR.

A arquitetura da solução DashClient apresentada tem interação com um servidor HTTP e é formada por três atributos principais, divididos em camadas, sendo essas:

- Player: Classe que implementa o comportamento do reprodutor de vídeo e o buffer. Controla os segmentos recebidos do servidor HTTP, monitora métricas de desempenho e é responsável por definir qual será o próximo segmento a ser recuperado, monta tal requisição e manda para a camada de baixo (IR2A).
- IR2A: Classe abstrata que define interface comum para algoritmos ABR. Faz a intermediação das mensagens request e response (mensagens vindas do Player e do ConnectionHandler). Define qual qualidade será utilizada na requisição dos segmentos.
- ConnectionHandler: Classe responsável pela comunicação entre a plataforma pyDash e servidor HTTP definido no arquivo .json. Converte requisições de mensagem em conexão funcional HTTP e manda resposta para camada superior. Também implementa algoritmo *traffic shaping* que controla a banda.

MPEG-DASH é um protocolo adaptativo baseado no HTTP para o streaming de mídia pela internet. Essa tecnologia é utilizada para transportar os segmentos de vídeo de servidores web para os clientes em tempo real.

Este formato de protocolo utiliza-se da taxa de bits adaptativa (ABR) que é um algoritmo de streaming de vídeo projetado para promover ao usuário uma experiência consistente e de maior qualidade em situações em que a sua largura de banda pode oscilar ou quando o usuário tem uma demanda grande pelo uso de múltiplos dispositivos conectados a rede.

O ABR soluciona tais problemas ao codificar o conteúdo em diversas variantes, cada uma com diferentes taxas de bits, tamanhos de frame e taxas de frame. Essas variantes são combinadas em um único pacote que representa o conteúdo original. O ABR alterna entre as variantes dependendo do dispositivo e largura de banda disponível, para garantir consistência e alta qualidade na reprodução.

E Como o ABR funciona? Durante a reprodução, vídeo e áudio são entregues via HTTP em pequenos segmentos, geralmente entre 3 e 10 segundos. Cada pacote contem múltiplas variantes, e tais variantes podem conter múltiplos segmentos. O reprodutor de vídeo (Player) é fornecido com um arquivo de manifesto do pacote que descreve quais variantes estão disponíveis e a localização dos segmentos para cada variante, o nome desse arquivo é MPD (Media Presentation Description) e ele é do tipo XML.

O Player solicita e baixa o segmento da variante. Enquanto o segmento é reproduzido, a velocidade de conexão é monitorada e o Player pode optar por mudar de variante, aumentando a taxa de bits do vídeo ou diminuindo-a baseado na velocidade da conexão. o Player também pode escolher variantes com diferentes taxas ou tamanhos de frame para otimizar a experiência visual do dispositivo. Este comportamento adaptativo é o que garante a reprodução consistente independentemente da velocidade.

Como visto anteriormente, tal algoritmo é executado na camada IR2A, e o método para sua implementação foi escolhido pelo grupo por meio das opções de artigos dadas nas instruções do Projeto. A solução utilizada é a "Dynamic Segment Size Selection in HTTP Based - Adaptive Video Streaming" ou Seleção dinâmica de tamanho dos segmentos em HTTP - Streaming de vídeo adaptativo.

Para este projeto, focou-se apenas no caso de streaming de um único filme de animação (Big Buck Bunny) contendo apenas vídeo sem áudio, com 596 segundos de duração. Este filme está disponível em um servidor HTTP (http://45.171.101.

167/DASHDataset/) segmentado em 6 tamanhos diferentes e codificado em 20 formatos distintos, variando de uma taxa de bits de 46980bps até 4726737bps.

Em linhas gerais, será abordado o funcionamento detalhado do algoritmo implementado pelo grupo, juntamente com um exemplo específico do funcionamento do protocolo, com dados e figuras ilustrativas do caso. Em seguida, haverá uma avaliação dos resultados do algorítimo comparando-o com casos exemplo mais simples e não muito eficientes para a averiguação da eficácia do ABR implementado. Por fim serão discutidos os resultados gerais e dificuldades encontradas pelo grupo.

II. Algoritmo ABR

Nessa seção será abordada a implementação do Algorítimo enunciado anteriormente:

A. Resumo do algoritmo

A proposta geral do algoritmo em execução é o uso estático do tamanho de segmento (SS) para reprodução. É monitorado a estabilidade de conexão em função da vazão (throughput) e as condições da rede são adaptadas dinamicamente alterando o valor de SS. O algorítimo original foi adaptado pelo grupo de tal forma a substituir o uso do SS pelos valores da lista de qualidades.

Sabe-se que o número de pausas tem um impacto maior negativamente à qualidade da experiência do usuário do que o tamanhos das pausas, então o objetivo estabelecido será diminuir ao máximo o número de pausas, e, portanto, se uma pausa vier a acontecer, ela ocorrerá por tempo suficiente para que a chance de novas pausas acontecerem seja mínima.

O vídeo utilizado conta com 6 tamanhos de segmento diferentes e 20 qualidades. A figura (1) mostra o tempo necessário para começar a reprodução do vídeo com relação a cada qualidade para cada tamanho de segmento.

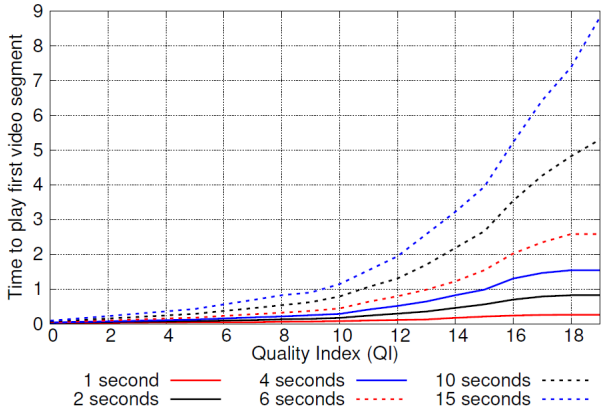


Fig. 1: Tempo para começar reprodução

Aqui é possível visualizar que grandes tamanhos de segmento demandam mais tempo para reprodução, já que é necessário mais tempo para o download do primeiro pacote de vídeo. Da mesma forma que uma maior qualidade demanda mais pois um quadro de maior qualidade é mais pesado que o mesmo de menor. Portanto, a qualidade menor tem a vantagem de ser menor, porém a solução de, por exemplo, deixar a qualidade sempre em zero não é cabível. Pois apesar de o mínimo de pausas possíveis, perde-se na experiência do usuário, que tem mais impacto nessa equação.

Tal gráfico pode ser levado em consideração para o tempo inicial de reprodução, onde iremos utilizar uma qualidade menor para então avaliação dos dados devolvidos da camada de Player.

Para este projeto, o tempo máximo de buffer está definido em 60 segundos. Após alguns testes para diferentes formatos de tráfego, mostrou-se necessário fazer, também, um controle da qualidade de vídeo de acordo com o tempo de buffer. A abordagem escolhida foi de definir q_0 (pior qualidade) para um intervalo de tamanho 0 segundos até 10 segundos. Para um tamanho de buffer maior que 50 segundos, escolhe-se a melhor qualidade (q_{19}). E, para o intervalo de buffer maior que 10 segundos até 50 segundos a qualidade de vídeo é escolhida pelo algoritmo adaptado.

B. Explicação do algoritmo

Nessa seção será descrito a proposta de seleção dinâmica da qualidade adaptada do artigo de referência baseada na performance da conexão de rede. Utilizando de molde a classe abstrata IR2A fornecida, implementando os mesmos processos passados por meio do roteiro, foi criado então um arquivo na mesma pasta com o nome *"r2adynamicsegmentsizeselection.py"*, este arquivo é o nosso ABR.

O algoritmo foi esquematizado de tal forma a medir a estabilidade da rede e sua vazão média. Para isso, no início de cada reprodução, escolhemos a menor qualidade (0) já que até então não temos conhecimento da performance da conexão, e também para começar a reprodução mais cedo do que escolhendo uma qualidade maior, como mostrado na figura 1. Inicialmente definimos uma lista *qi*, que possui as qualidades possíveis de 0 a 19, também definimos uma lista *throughputs* que contém as últimas (M) vazões medidas, da mais antiga para a mais recente. Desta forma a vazão média é facilmente computada por:

$$\mu = \frac{\sum_{i=0}^{M} \lambda_i}{M} \tag{1}$$

Em que μ é a variável *media*, λ a lista *throughputs* e *M* o número de vazões medidas até então, esta média foi computada por meio da função *mean* do módulo statistics e nos dá uma indicação da média das vazões experimentadas durante a execução. E, para o cálculo desta média simples é utilizado, no máximo, a amostra das últimas 50 vazões medidas. Esta abordagem foi escolhida para que seja possível a média convergir para um certo valor quando a rede estiver estável, e, para que amostras muito antigas de vazão não interfiram durante a escolha da qualidade de reprodução do vídeo. Esta operação pode ser vista a seguir em código:

```
# calcula a media simples de vazao
# das ultimas M amostras
media = statistics.mean(
  self.throughputs[-50:])
```

Sabendo que a conexão pode ser instável, o valor da média da vazão não é suficiente para chegar na qualidade de reprodução mais adequada. Para lidar com essas variações e agir de forma mais rápida à mudanças, vazões mais recentes têm mais peso que as antigas. Este ponderamento é chamado de σ^2_{weight} e calculado da seguinte maneira:

$$\sigma^2_{weight} = \sum_{i=1}^{M} \frac{i}{M} |\lambda_i - \mu| \tag{2}$$

Em Python, a equação anterior é calculada assim:

```
# calculo do sigma^2.
sigma_quadrado = 0.0
m = len(self.throughputs[-50:])
aux = enumerate(self.throughputs[-50:])
for i, data in aux:
 sigma_quadrado+=((i+1)/m)*abs(data-media)
```

em que *m* recebe o número de vazões medidas, *i* é o contador da iteração e assume o papel de peso de cada medição de vazão, finalmente, é feito o somatório por meio de um laço, obtendo o valor do σ^2_{weight}.

Calculadas a média de vazão (μ) e σ^2_{weight}, é definida a probabilidade p que pertence ao intervalo [0,1], estima a tendência de mudança da qualidade e é dada por:

$$p = \frac{\mu}{\mu + \sigma^2_{weight}} \tag{3}$$

A equação anterior é calculada da seguinte forma:

```
# Calculo da probabilidade p
p = media/(media+sigma_quadrado)
```

utilizando as variáveis *media* e *sigma_quadrado* já calculadas.

Para conexões estáveis, isto é, sem variação brusca, p se aproxima de 1. p é utilizado para estimar a possibilidade de aumentar ou diminuir a qualidade:

$$\tau = (1 - p) \cdot qi[max(0, q_i - 1)] \tag{4}$$

Em que τ define a tendência de diminuir a qualidade, lembrando que qi é a lista de qualidades e q_i é a última qualidade experimentada. Implementou-se como:

```
# posicao da ultima qualidade selecionada
last_qi_index = 0
x=self.selected_qi[len(self.selected_qi)-1]
for i in range(len(self.qi)):
 if self.qi[i] == x:
  last_qi_index = i
# calculo de tau
# tau=(1 - p)*omega[max(0,last_qi_index-1)]
if last_qi_index-1 < 0:
 tau=(1-p)*max(0, self.qi[last_qi_index])
else:
 tau=(1-p)*max(0, self.qi[last_qi_index-1])
```

Da mesma forma, temos:

$$\theta = p \cdot qi[min(N, q_i + 1)] \tag{5}$$

Em que θ define a tendência de aumentar a qualidade. Como definido anteriormente, p próximo de 1 (conexão estável) resultará em τ perto de 0 e θ perto do próximo maior valor de qualidade dado o valor atual e para p próximo de 0 (conexão instável) resultará em τ perto do valor de qualidade anterior dado o valor atual e θ perto de 0. Implementou-se θ da seguinte forma:

```
if last_qi_index+1 > 19:
  teta = p*min(self.qi[19],
  self.qi[last_qi_index])
else:
  teta = p*min(self.qi[19],
  self.qi[last_qi_index+1])
```

A nova qualidade é computada como:

$$qi_{new} = argmin(q_i - \tau + \theta) \tag{6}$$

No qual é escolhido o valor mais próximo de qualidade calculado por $\tau + \theta$. É importante notar que a qualidade pode continuar a mesma. Segue código implementando a lista de novas qualidades:

```
# calculo da nova qualidade
# new_qi = argmin(omega[i] - tau + teta)
new_qi = []
for i in self.qi:
 if (i - tau + teta) > 0:
  new_qi.append(i - tau + teta)
```

Após o cálculo da nova qualidade adicionamos ela a uma lista de qualidades selecionadas e recuperamos seu índice, como pode ser visto a seguir:

```
# adiciona a qualidade a lista
for i in self.qi:
 if min(new_qi) > i:
  self.selected_qi.append(i)

# pega o indice da qualidade selecionada
next_qi_index = 0
for i in range(len(self.qi)):
 aux=self.selected_qi[len(self.selected_qi)-1]
 if self.qi[i] == aux:
  next_qi_index = i
```

Por fim, temos um tratamento do algoritmo de acordo com o valor do buffer em tempo real utilizando o whiteboard. Este tratamento consiste no que foi mencionado na seção anterior e ocorrem decisões de qualidade de acordo com o tamanho do buffer. Cabe ressaltar que o algoritmo mencionado para um buffer se encontra entre 10 e 50 segundos é exatamente o explicado anteriormente na seção “Explicação do algorítimo” e sua implementação foi feita da seguinte maneira:

```
# resgata o tamanho do buffer
buffer_size=self.whiteboard
  .get_playback_buffer_size()

# buffer_size[0, 10]
if len(buffer_size) > 5 and
  0 <= buffer_size[len(buffer_size)-1][1]
  <= 10:
    self.selected_qi.append(self.qi[0])

# buffer_size[11, 50]
elif len(buffer_size) > 5 and
  10 < buffer_size[len(buffer_size)-1][1]
  <= 50:
    self.selected_qi.append(self.qi[
      next_qi_index])

# buffer_size[51, 60]
elif len(buffer_size) > 5 and
  buffer_size[len(buffer_size)-1][1] > 50:
    self.selected_qi.append(self.qi[19])

# len(buffer_size) <= 5
else:
    self.selected_qi.append(self.qi[
      next_qi_index])
```

Todo esse processo exemplificado se encontra no método *handle_segment_size_request* que tem o objetivo de definir a qualidade passada para o connection_handler, portanto, passamos a mensagem codificada contendo a qualidade calculada para a camada debaixo.

quando o IR2A recebe a resposta da camada debaixo *handle_segment_size_response* passa a ser executado, e é nele que calcularemos os valores da latência (rtt) e da vazão para serem utilizados nos próximos segmentos. Por fim, mandamos a mensagem codificada para cima (camada do player).

III. Resultados

A. Exemplo de funcionamento do algoritmo

Vamos exemplificar o funcionamento do algoritmo para o caso em que arquivo *dash_client.json* possui as seguintes configurações:

O valor de buffer inicial antes da reprodução é 5 segundos, o tamanho máximo de buffer é 60 segundos, o passo na reprodução de vídeo é 1 segundo, o intervalo do perfil de tráfego é 5 segundos, a semente de tráfego é 1, o url_mpd escolhido foi o vídeo *"BigBuckBunny"* como já foi mencionado anteriormente e por último o nome da classe referente ao algoritmo implementado é *"R2ADynamicSegmentSizeSelection"*.

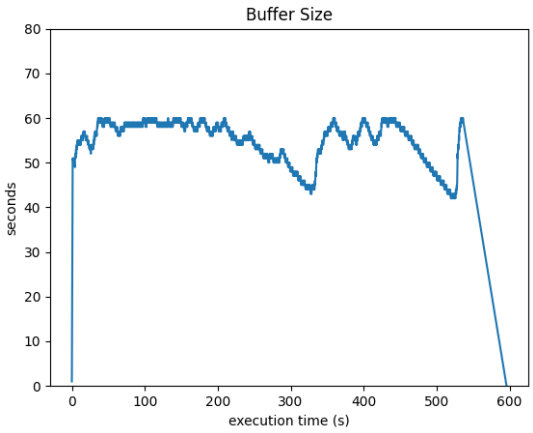
Para exemplificar a execução do algoritmo foram feitos testes em três formatos de tráfego diferentes: L (low), M (medium) e H (high).

Para o formato de tráfego do tipo L, obtemos os gráficos presentes na figura (2). Para este tipo de tráfego (L), que simula o melhor caso para a vazão, podemos observar como que o controle do tamanho do buffer se mostrou eficiente quando a vazão da rede é alta. Vale ressaltar que o a ação do controle do tamanho do buffer começa após um número de 5 amostras de tamanho do buffer. O algoritmo manteve a qualidade do vídeo no mínimo durante certo tempo o que fez armazenar buffer e quando o buffer atingiu 51 segundos a qualidade do vídeo saltou do valor mínimo para o valor máximo, mantendo a mesma qualidade durante quase toda a reprodução, atingindo uma média de qualidade de 16.95, de acordo com os índices da lista de qualidades disponíveis. E, sendo executado sem pausas. A qualidade da reprodução diminuiu em alguns pontos devido a rede não ter conseguido manter constante a vazão, o que fez o buffer diminuir e consequentemente selecionar uma qualidade inferior.

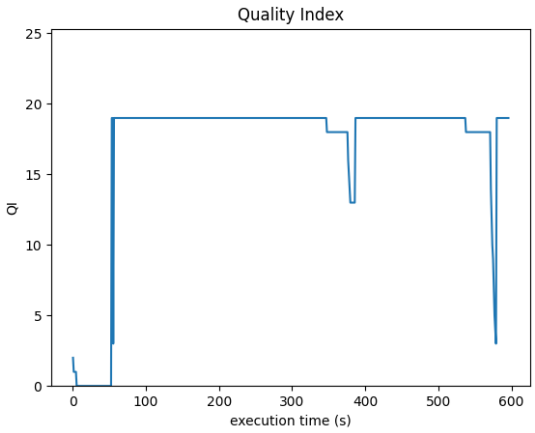
Para o formato de tráfego do tipo M, temos os gráficos presentes na figura (3). Para este caso médio de vazão, é o caso em que é melhor visto algoritmo performando. No início da execução do vídeo o é armazenado um buffer de 51 segundos com qualidade mínima e posteriormente ocorre o salto para a qualidade máxima, porém a rede não possui vazão suficiente para manter o buffer superior a 50 segundos durante toda execução. Então, passa a performar o algoritmo adaptado que faz com que a qualidade do vídeo oscile conforme vai se adaptando a vazão obtida a cada requisição. O comportamento do gráfico (3c) possui este formato devido ao modelo de ação do algoritmo que tende a aumentar a qualidade selecionada de acordo com a estabilidade da rede, então conforme o histórico de vazões são mais próximos, a qualidade tende a aumentar em relação a qualidade selecionada anterior. E, quando ocorre oscilações na rede o algoritmo tende a escolher qualidades mais próximas da última amostra de vazão, já que esta possui maior peso no cálculo da média das vazões.

E, para o formato de tráfego do tipo H, em que simula o pior caso. Temos o comportamento presente na figura (4a). Como já foi mencionado o controle de tamanho do buffer começa após um histórico de 5 amostras de tamanho de buffer. Isso explica o salto presente no início da execução da figura (4d). Porém como a variação de vazão mostra - figura (4f). A vazão está limitada a um valor menor que 50kbps. O que faz com que o buffer nunca seja superior a 10 segundos e, portanto, a qualidade fica limitada ao valor mais baixo disponível. Durante a execução ocorreram duas pausas que mesmo com a qualidade selecionada sendo mínima, salva a exceção presente no início da execução,não foram possíveis de serem evitadas.

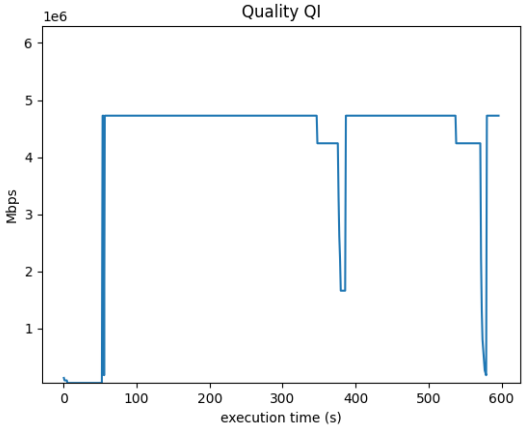
Após a execução do algoritmo, obtemos diversos gráficos com informações sobre o seu funcionamento que serão analisados em mais detalhes na próxima seção.



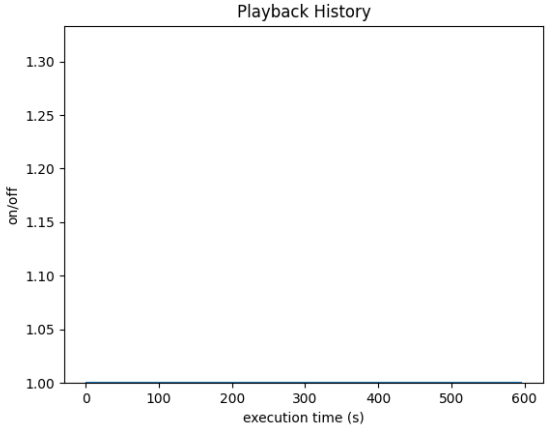
(a)



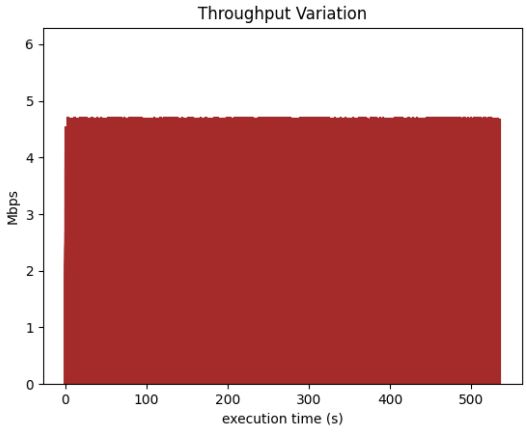
(b)



(c)

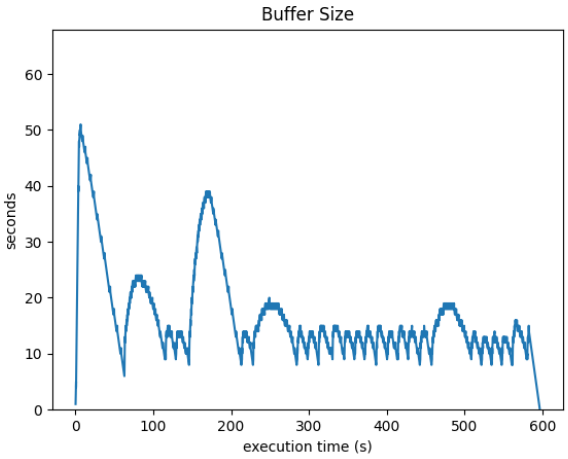


(d)

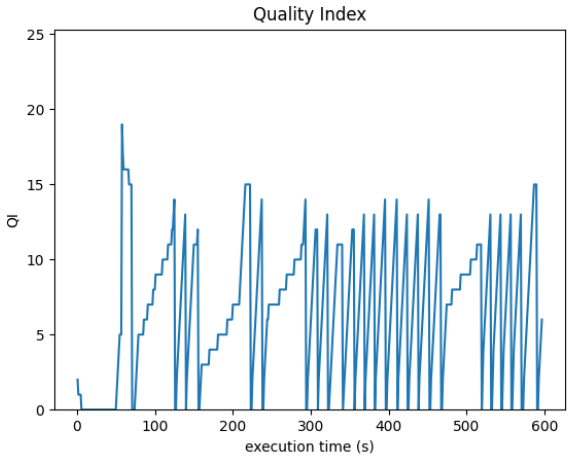


(e)

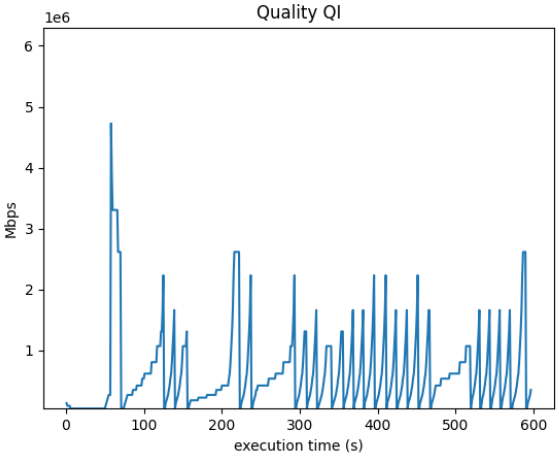
Fig. 2: Formato do tráfego selecionado = L



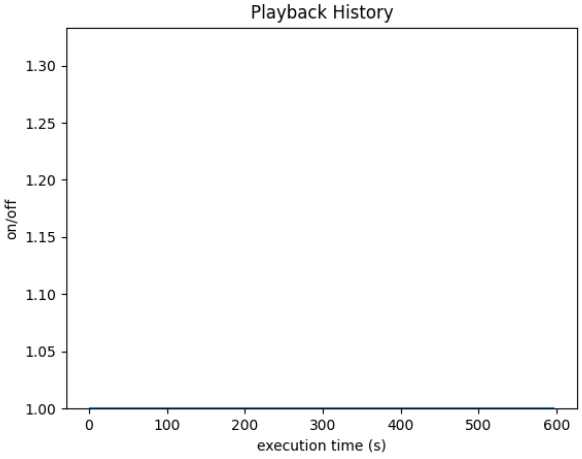
(a)



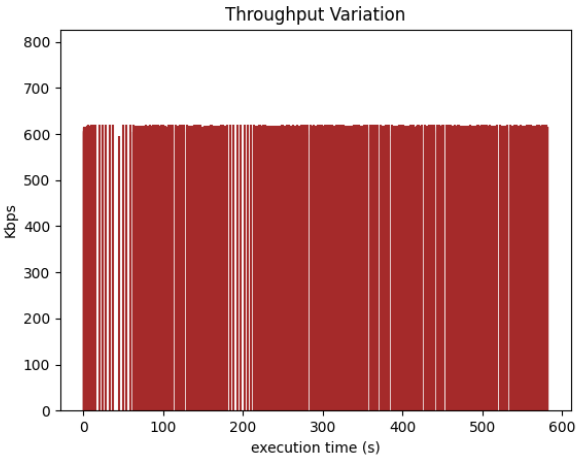
(b)



(c)

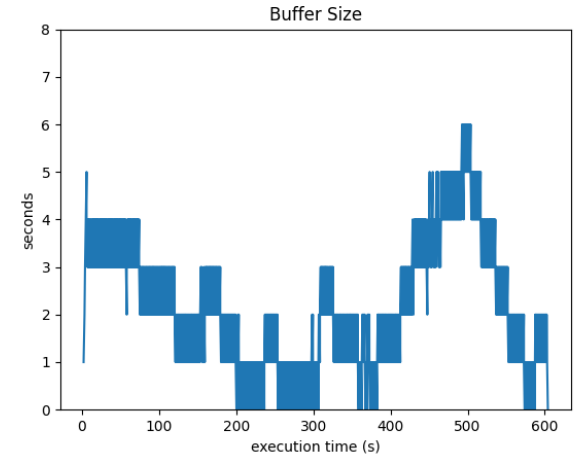


(d)

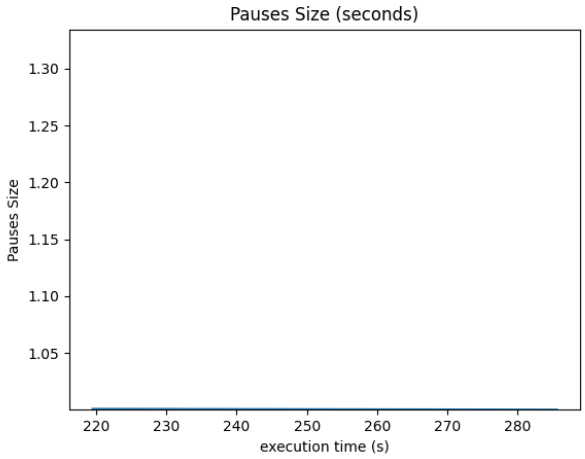


(e) o

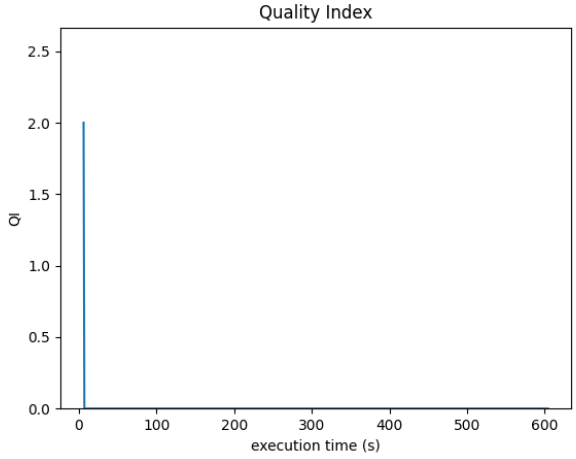
Fig. 3: Formato do tráfego selecionado = M



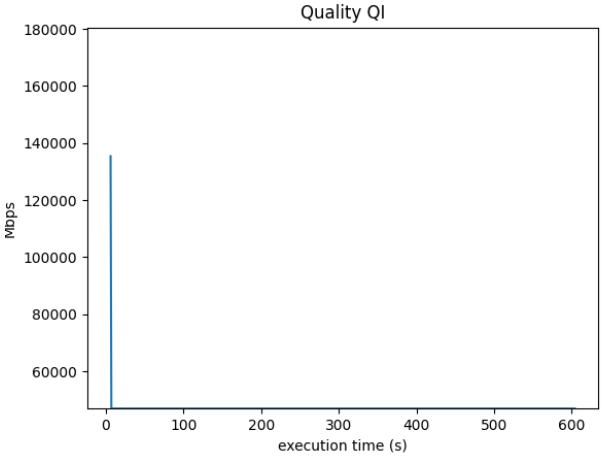
(a)



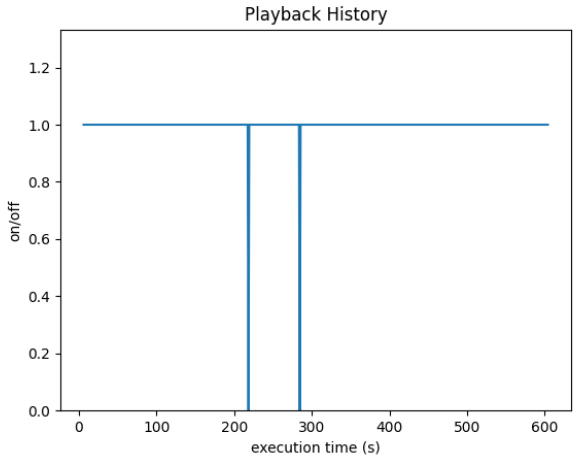
(b)



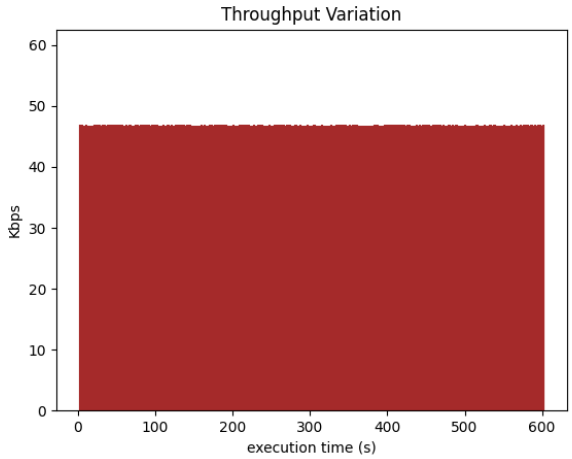
(c)



(d)



(e)

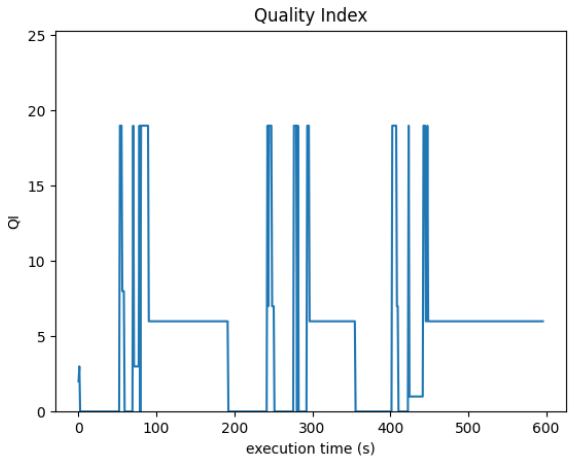


(f)

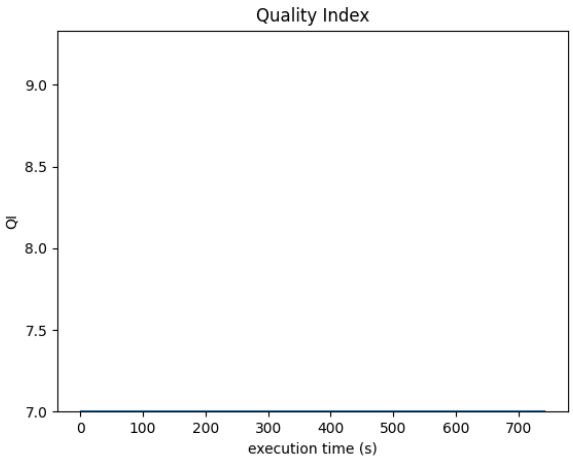
Fig. 4: Formato do tráfego selecionado = H

B. Avaliação do algoritmo

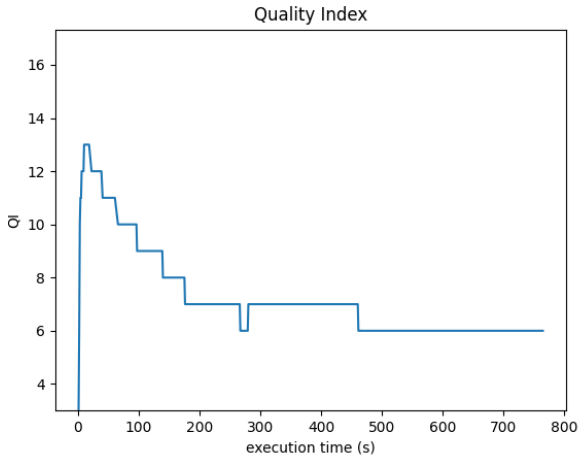
Com os gráficos obtidos anteriormente do algoritmo implementado - a sequência do perfil de tráfego escolhido foi "LMH" (ou low-medium-high) - iremos fazer uma comparação com os resultados dos 3 algoritmos exemplo passados, os quais, sabemos que não são eficientes. A seguir temos os 4 gráficos da figura 5 resultantes da qualidade de reprodução de vídeo para o mesmo caso nos diferentes algoritmos:



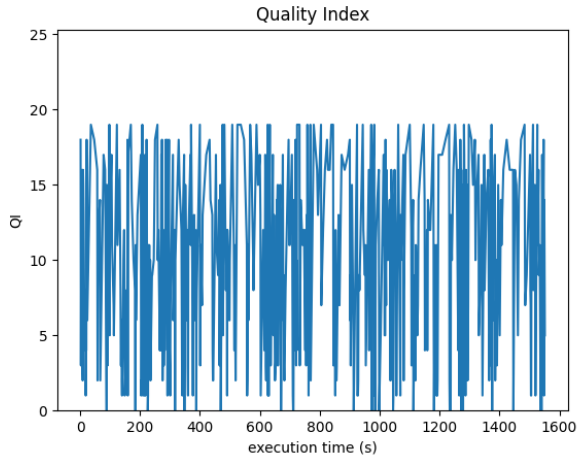
(a) ABR Dinâmico



(b) ABR Fixed



(c) ABR Average



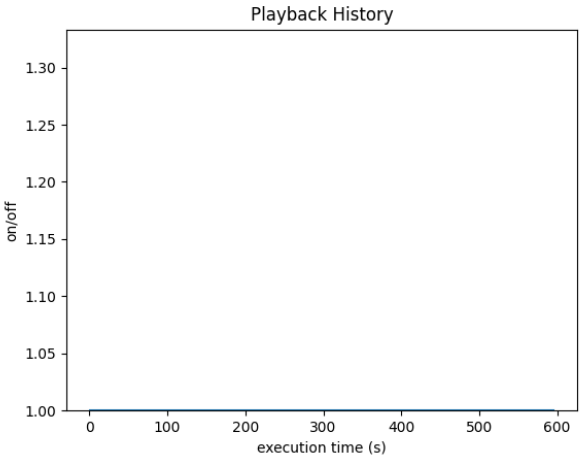
(d) ABR Random

Fig. 5: Comparação da variação da qualidade em diferentes algoritmos

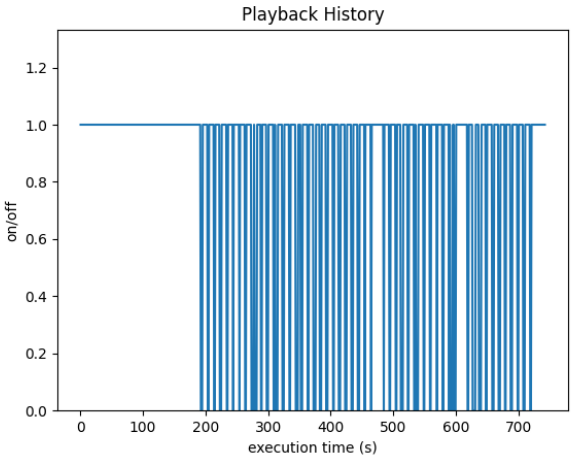
Analisando os gráficos é possível perceber que:

- (figura 5a) O algoritmo dinâmico teve algumas variações bruscas mas na maioria do tempo se manteve no nível 6 de qualidade, com média geral de qualidade 4.56, tendo alguns picos na qualidade máxima (19) e em alguns momentos se manteve em zero.
- (figura 5b) O algoritmo *Fixed* se mantém fixo - como o nome já sugere - na qualidade 7 que foi escolhida arbitrariamente pelo grupo por ser próxima da qualidade média porém, menor que ela, afim de ocorrer menos pausas.
- (figura 5c) O algoritmo *Average* tem poucas variações de qualidade, com média de 6.96, estas variações não são tão bruscas quanto as do dinâmico e seus degraus são bem definidos, isto é, se mantêm nas mesmas qualidades por tempo razoável antes de mudar seu valor.
- (figura 5d) O algoritmo *Random* tem a maior variação de todos os algoritmos, como esperado, valor de qualidade 9.41 em média, variando quase a cada segundo para valores "aleatórios".

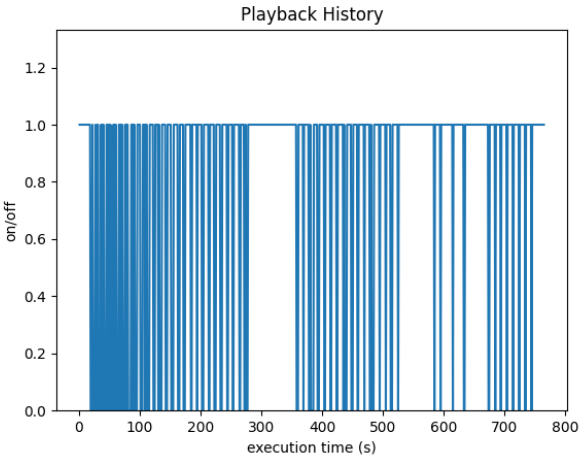
Mas a informação da qualidade no tempo sozinha não é suficiente para esclarecer o impacto do algoritmo na experiência do usuário. Para isso, podemos observar também como fica o histórico de reprodução do vídeo.



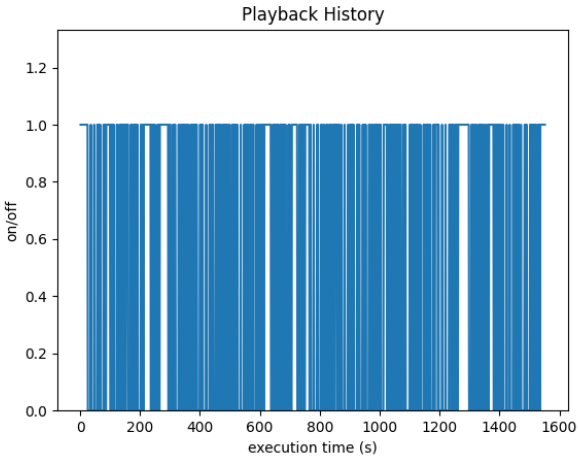
(a) ABR Dinâmico



(b) ABR Fixed



(c) ABR Average



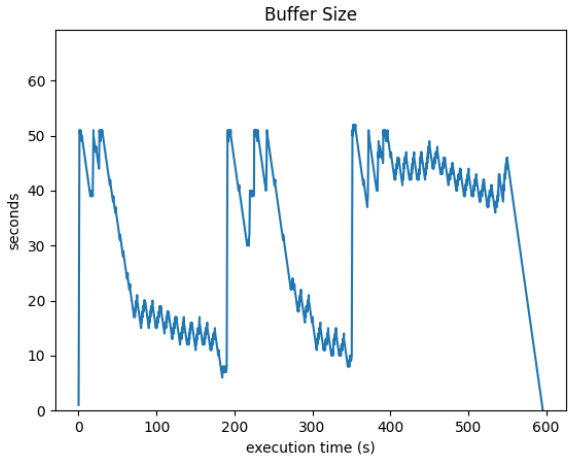
(d) ABR Random

Fig. 6: Comparação das pausas em diferentes algoritmos

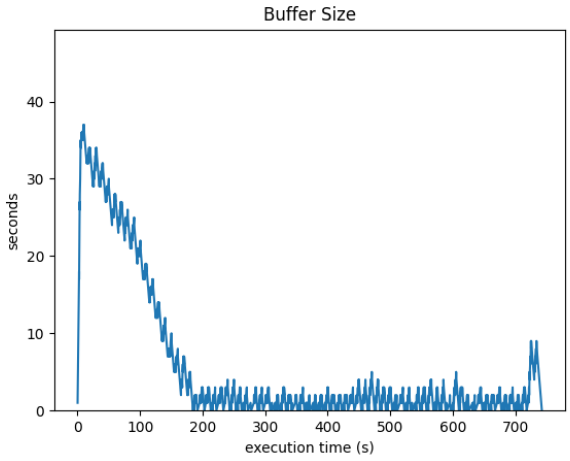
Analisando os gráficos é possível perceber que:

- (figura 6a) O algoritmo dinâmico teve sua reprodução completa, sem pausas, no tempo de vídeo.
- (figura 6b) O algoritmo *Fixed* começa reproduzindo por quase 200 segundos e então começa a sofrer pausas. De maneira geral, as pausas são curtas (média de 2.58 segundos), que totalizam 54 pausas e a execução é finalizada em aproximadamente 700 segundos.
- (figura 6c) O algoritmo *Average* tem pausas durante toda a reprodução, 72 no total, em alguns poucos momentos a reprodução sem pausas possuí durações mais longas chegando próximas a 100 segundos, as pausas no geral possuem média de 2.25 segundos e finaliza sua reprodução em aproximadamente 760 segundos de execução.
- (figura 6d) O algoritmo *Random* tem o maior número de pausas, 208 no total, ocorrendo durante toda a reprodução que durou aproximadamente 1500 segundos, sendo a reprodução mais demorada entre as analisadas.

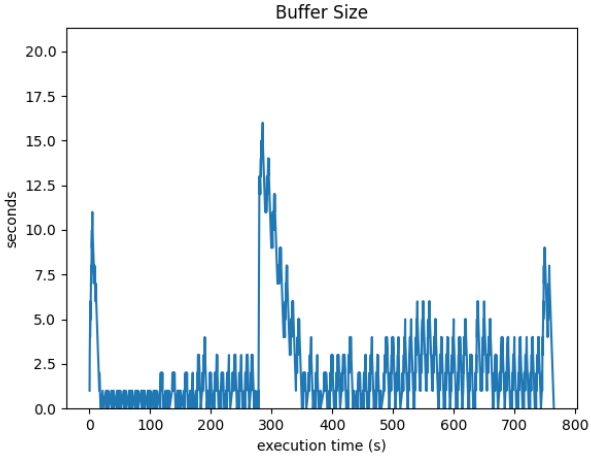
Complementando as informações sobre eficiência obtidas pelos gráficos anteriores, podemos também observar o comportamento do buffer no tempo:



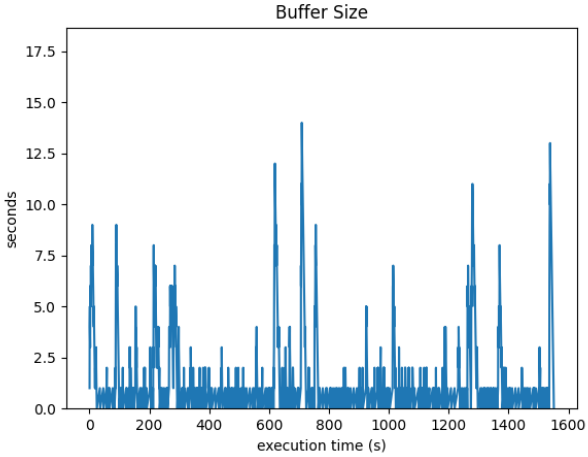
(a) ABR Dinâmico



(b) ABR Fixed



(c) ABR Average



(d) ABR Random

Fig. 7: Comparação do comportamento do buffer em diferentes algoritmos

Aqui podemos realmente identificar o motivo das pausas ocorrerem, isto é, quando o buffer chega a zero. Fica fácil perceber que não aconteceram pausas no algoritmo dinâmico pois o seu buffer foi bem administrado, em nenhum momento o buffer chegou a zero, sempre que seu valor baixava, o algoritmo tratava a qualidade de forma a recuperar o buffer, preocupação inexistente ou mal executada nos outros algoritmos. Podemos assumir então, que para este tráfego o algoritmo dinâmico é muito mais eficiente que os outros mostrados.

IV. Conclusão

A implementação do algoritmo foi bem sucedida com a ressalva de que a adaptação direta para qualidade de vídeo não é a melhor abordagem. Foi necessário criar um controle de tamanho de buffer para que o comportamento do algoritmo adaptado fosse mais aceitável. A abordagem que se mostrou interessante foi fazer o controle da qualidade selecionada de acordo com o tamanho do buffer disponível, fixando a qualidade do vídeo na mais baixa para um buffer de até 10 segundos e fixando a qualidade no valor mais alto disponível para um buffer superior a 50 segundos, sendo o buffer máximo de 60 segundos.

O motivo principal, por ter sido adicionado ao algoritmo esse controle do tamanho do buffer, é que o algoritmo original aumenta a qualidade de vídeo quando a vazão de rede fica estável, para amostras de vazão muito próximas. Em alguns casos isso é interessante, mas fazendo testes com amostras de vazões baixas e com valores próximos, o algoritmo se mostrou menos eficiente, sem o controle de tamanho do buffer baixado. Para este caso, a rede está estável, porém com uma taxa de transmissão muito baixa, então o comportamento desejado seria o de utilizar uma qualidade menor para evitar pausas durante a reprodução do vídeo, o que não estava acontecendo, como a rede estava estável. A qualidade de reprodução do vídeo estava aumentando e gerando pausas na reprodução do vídeo cada vez maiores, porque para baixar um segmento com uma qualidade superior com uma baixa taxa de transmissão demora mais tempo que fazer o mesmo processo para um segmento de qualidade inferior.

Para evitar que a qualidade do vídeo aumentasse para vazões baixas e "estáveis", optou-se por fixar a qualidade de vídeo selecionada para o valor mais baixo disponível até que o tamanho do buffer armazenado fosse maior que 10 segundos. Esta abordagem é interessante para a redução do número de pausas durante a reprodução e para evitar que seja selecionado qualidades de vídeo muito superiores à taxa de transmissão disponível da rede.

Já o valor fixo da qualidade de vídeo para o caso do buffer maior que 50 segundos foi implementado, por motivo de se o buffer chegou a esse nível significa que pode indicar que a rede está apta a operar com a qualidade máxima de reprodução. Assim é possível proporcionar uma experiência melhor ao usuário que estiver assistindo ao vídeo, supondo que a rede seja capaz de proporcionar isso.

E, quando o tamanho do buffer estiver entre as faixas superior a 10 segundos e inferior a 51 segundos, o algoritmo irá escolher a melhor qualidade suportada pela rede.

Por fim, chegamos a esse tipo de abordagem levando em consideração os testes feitos, em que a opção com o controle do tamanho do buffer se mostrou a mais eficiente. Com a finalidade de melhorar a reprodução do vídeo com o menor número possível de pausas e com a qualidade mais adequada para cada caso.

V. Referências

CAETANO, Marcos. Projeto de Programação Algoritmos Adaptativos de Streaming de Vídeo MPEG-DASH, 2021.Disponível em: https://aprender3.unb.br/pluginfile.php/701421/mod_resource/content/3/Especificacao_pydash.pdf.

CAETANO, Marcos; MAROTTA, Marcelo. pyDash: A Framework Based Educational Tool for Adaptive Streaming Video Algorithms Study, 2021. Disponível em: https://github.com/mfcaetano/pydash.

RUETHER, Traci. MPEG-DASH: Dynamic Adaptive Streaming Over HTTP Explained (Update), 2021. Disponível em: https://www.wowza.com/blog/mpeg-dash-dynamic-adaptive-streaming-over-http.

KREGLICKI,Piotr. HLS, MPEG-DASH – What is ABR?, 2021. Disponível em: https://sfiblog.telestream.net/2017/05/what-is-abr/.

L. Bedogni, M. Di Felice and L. Bononi, "Dynamic segment size selection in HTTP based adaptive video streaming," 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2017, pp. 665-670, doi: 10.1109/INFCOMW.2017.8116456.