Universidade de Brasília
Departamento de Ciência da Computação
CIC0103 - Transmissão de Dados

Algoritmo Adaptativo para Transmissão de Vídeo Baseado na Teoria de Controle

Grupo 1 - Turma A:

Davi Salomão Soares Corrêa, 18/0118820

Francisco Henrique da Silva Costa, 18/0120174

Matheus Teixeira de Sousa, 18/0107101

Brasília, DF
2022

Conteúdo

1. Introdução

A revolução na comunicação proporcionada pela *internet* é um marco no desenvolvimento da civilização, ela é onipresente para maioria das pessoas nas grandes cidades ao redor do planeta. O amplo compartilhamento de informação e a maior integração entre as diferentes culturas são algumas das características do paradigma criado por essa revolução. De acordo com a projeção do relatório anual da Cisco, o tráfego de vídeo vai representar 82% do tráfego total consumido globalmente até 2022 [Cisco, 2018].

Essa tendência de crescimento, aliada ao crescimento do número de pessoas conectadas à *internet* ao longo dos anos, faz com que a busca pelo melhor desempenho na experiência do usuário seja um dos desafios na transmissão de vídeo. Como o estado da rede pode variar para cada cliente e região, diversos modelos foram propostos para tentar melhorar a forma como o vídeo é transmito para o usuário de forma adaptativa. Bentaleb et al. [2019] classifica esses modelos em quatro grupos: os baseados no cliente, os baseados no servidor, os assistidos por rede e os híbridos, que utilizam partes das combinações do outros três.

Os modelos baseados no lado do cliente, escopo desse trabalho, buscam se adaptar ao estado da rede por meio da análise de variáveis como a largura de banda e o espaço disponível no *buffer* [Bentaleb et al., 2019]. Dentre aqueles que avaliam a primeira variável, podemos citar Li et al. [2014], que estima a largura da banda disponível e utiliza esse valor para tentar reduzir as oscilações na qualidade do vídeo. Enquanto isso, dentre aqueles que avaliam a segunda variável, podemos citar Mueller et al. [2015], que utiliza, essencialmente, o tamanho do *buffer* para, assim como no caso anterior, tentar suavizar as trocas de qualidade no vídeo.

Alternativamente, Ito et al. [2015] propõe uma arquitetura que se baseia, principalmente, no tamanho do *buffer* do cliente e na taxa média de *download* na rede. Nesse modelo, um sistema de controle foi projetado para calcular a qualidade do vídeo a ser requisitada a partir das duas variáveis descritas com o objetivo de manter o tamanho do *buffer* próximo de um valor fixo para evitar estouro ou esvaziamento do *buffer*. Então, a partir desse valor e o do estado da rede, uma máquina de estados seleciona a qualidade do vídeo indicada.

É com base nas definições apresentadas até então que este trabalho tem por objetivo descrever e explorar a implementação do algoritmo proposto por Ito et al. [2015] em *A Fine-Tuned Control-Theoretic Approach for Dynamic Adaptive Streaming Over HTTP*. Como será discutido em mais detalhes na seção 2, a lógica de desenvolvimento do algoritmo pode ser dividida em três etapas bem definidas para calcular a qualidade do próximo pedaço de vídeo, avaliar o estado da execução e ajustar os parâmetros selecionados.

Após delinear os detalhes da implementação realizada, a seção 3 apresentará os resultados obtidos durante as simulações com a plataforma *pyDash* desenvolvida por Marotta et al. [2021], destacando a influência de cada cenário para o desempenho do algoritmo. Por fim, o trabalho é concluído na seção 4, dedicada a observações finais sobre o projeto, os resultados obtidos e trabalhos futuros.

2. Metodologia

Como mencionado na seção anterior, o desenvolvimento do algoritmo implementado pode ser dividido em três etapas: o controlador, a máquina de estados e a seleção dos parâmetros. Na primeira etapa, o controlador calcula a qualidade do próximo pedaço de vídeo a partir das variáveis de entrada e a envia para a máquina de estados. Então, na segunda etapa, a máquina de estados avalia o cálculo do controlador e o estado do *buffer* para decidir a qualidade do próximo pedaço de vídeo.

Por fim, na terceira etapa, é realizado o ajuste fino (*fine tune*) dos parâmetros do controlador, mas, uma vez que os parâmetros são ajustados, a lógica do algoritmo se resume às dois primeiros procedimentos, ou seja, cálculo da qualidade e decisão com base no estado. Para cada uma das etapas, será apresentado o funcionamento e, depois, como nós a implementamos. A explicação geral sobre o artigo que propõe o algoritmo implementado pode ser encontrada em vídeo por meio deste link

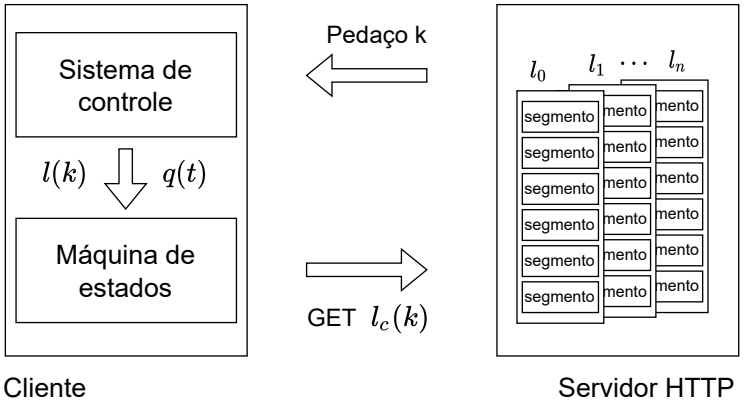

Figura 1. Esquemático do funcionamento do algoritmo

A figura 1 apresenta o esquemático do funcionamento geral do algoritmo. Depois das etapas já descritas, de cálculo da qualidade e da seleção pela máquina de estados, a qualidade é adicionada à mensagem de requisição. Então, o próximo pedaço de vídeo é requisitado ao servidor HTTP (*Hypertext Transfer Protocol*). Ao final do *download* desse pedaço, o processo é repetido para o próximo pedaço de vídeo até que todos os pedaços sejam solicitados.

2.1. Padrão MPEG-DASH

Antes de tratar, efetivamente, do desenvolvimento do algoritmo implementado, é relevante apresentar alguns conceitos sobre o padrão MPEG-DASH. De acordo com Caetano [2022], o MPEG-DASH (ou DASH, *Dynamica Adaptive Streaming over HTTP*, é uma técnica de transmissão de bits de vídeo que permite a transmissão em alta de qualidade de conteúdo de mídia. O padrão funciona dividindo o vídeo em uma sequência de segmentos com um pequeno intervalo de tempo que são servidos por HTTP e codificados em diferentes qualidades. Então, enquanto o conteúdo é reproduzido no cliente, um algoritmo de adaptação de transmissão de bits seleciona automaticamente o segmento com a maior qualidade possível sem que sejam causados travamentos ou *rebuffering* na reprodução.

2.2. Controlador

Para o controlador, o projeto se baseia em um problema de regulação, ou seja, definir o sinal necessário para levar a saída do sistema para zero dentro de determinadas condições. No caso do algoritmo implementado pelos autores, o objetivo é levar o tempo de *buffer* $q(t)$ para o referencial q_0 por meio da seleção do valor apropriado de $l(k)$. Para tanto, o processo foi modelado e linearizado, e sua representação em diagrama de blocos no domínio de Laplace pode ser vista na figura 2.

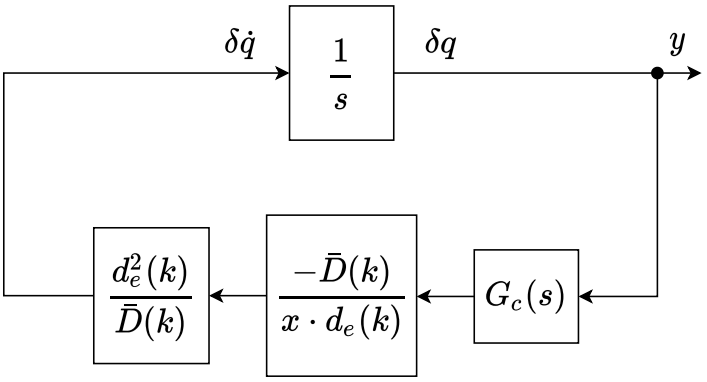

Figura 2. Diagrama de blocos no domínio de Laplace

Na figura, as variáveis $\bar{D}(k)$, x, $d_e(k)$ e $G_c(s)$ são, respectivamente, a taxa média de *download* k-ésimo pedaço de vídeo, a duração do pedaço de vídeo, a taxa de extração do k-ésimo pedaço de vídeo e o controlador a ser projetado. Da teoria de controle de moderno [Ogata, 2010], temos que é possível escrever o sistema do diagrama de blocos por meio da sua representação no espaço de estados. Assim, considerando δq como o estado e a multiplicação dos blocos na realimentação como a matriz de estado, temos

$$\begin{aligned} \delta\dot{q}(t) &= G(t)\delta q(t), \\ y(t) &= \delta q(t), \\ G(t) &= \frac{-G_c(t) \cdot d_e(k)}{x}. \end{aligned} \tag{1}$$

Em que $\delta q(t) = q(t) - q_0$, é a saída do sistema. Dessa forma, supondo $G_c(t)$ um ganho proporcional k_p e resolvendo equação diferencial envolvendo o estado, temos que δq pode ser calculado como

$$\delta q(t) = e^{-G \cdot t + \delta q(0)}. \tag{2}$$

Em que $\delta q(0)$ é a condição inicial. Como $d_e(k)$, x e k_p são definidos maiores que zero, o ganho G é positivo e, por tanto, $\delta q(t)$ é uma exponencial decrescente que vai para zero à medida que o tempo passa, como desejado pelo problema de regulação. Na prática, $\delta q(t)$ ir para zero significa que o tempo de *buffer* atual é igual ao tempo desejado pela referência. No processo de linearização do sistema, foram definidas as seguintes equações:

$$\delta l = l(k) - l_0, \tag{3}$$

$$l_0 = \frac{\bar{D}(k)}{d_e(k)}, \tag{4}$$

$$\delta l(k) = \frac{\bar{D}(k)}{x \cdot d_e(k)} \cdot \delta q(t), \quad \text{e} \tag{5}$$

$$\delta \dot{q}(t) = -\frac{d_e^2(k)}{\bar{D}(k)} \cdot \delta l(k). \tag{6}$$

Em que l_0 é a qualidade do vídeo ao redor do ponto de operação da linearização e δl é a diferença na qualidade do vídeo. Dessa forma, se substituirmos a equação 2 na equação 5, temos

$$\delta l(k) = \frac{\bar{D}(k)}{x \cdot d_e(k)} \cdot e^{-G \cdot t + \delta q(0)} \rightarrow l(k) = \left(\frac{e^{-G \cdot t + \delta q(0)}}{x} + 1 \right) \cdot l_0. \tag{7}$$

Como o termo dentro do parenteses é sempre maior ou igual a um e o valor de l_0 é sempre maior ou igual a zero, o valor de $l(k)$ é sempre maior ou igual a zero. Esse resultado faz sentido, uma vez que $l(k)$ define a qualidade do vídeo e não pode existir uma qualidade negativa. Na equação 7, o valor de k_p, x e $d_e(k)$ são escolhidos, enquanto os valores de l_0 e $\bar{D}(k)$ são calculados. A equação 4 calcula o valor de l_0 e o valor de $\bar{D}(k)$ pode ser calculado pela expressão

$$\bar{D}(k) = \frac{S(k)}{t_e(k) - t_s(k)}. \tag{8}$$

Em que $t_s(k)$, $t_e(k)$ e $S(k)$ são, respectivamente, o tempo de início do *download*, o tempo de final do *download* e o tamanho em bits do pedaço de vídeo que foi baixado do servidor. Um vez escolhidos os valores de k_p, x e $d_e(k)$, basta calcular o $l(k)$ pela equação 7 para ter a qualidade escolhida para o vídeo.

2.3. Implementação do controlador

A implementação foi realizada por uma classe chamada **R2A_FineTunedControl** através de quatro métodos: dois proveniente da camada superior, *player*, para tratar a requisição xml (**handle_xml_request** e **handle_segment_size_request**) e dois da camada inferior, *ConnectionHandler*, para tratar os segmentos de vídeo (**handle_xml_response** e **handle_segment_size_response**). No método **handle_xml_request** (figura 3), o tempo de início do *download*, $t_s(k)$, é obtido por meio da função **perf_counter()** da biblioteca *time*.

Enquanto isso, no método **handle_xml_response** (figura 4), o **parser_mpd** é utilizado para obter a lista de qualidades qi. Depois, a máquina de estados é inicializada com os parâmetros $qi[0]$, q_{max}, q_{min}, m e n. Então, a função **perf_counter()** é utilizada novamente para obter o tempo de final de *download*. O tamanho em bits do pedaço de vídeo $S(k)$ é calculado com o método **msg.get_bit_length**, que retorna o tamanho do segmento

que acabou de ser recebido. Com esses valores, $\bar{D}(k)$ e l_0 são calculados, conforme as equações 8 e 4, respectivamente. Por fim, G e $l(k)$ são calculados por meio das respectivas equações, 1 e 7.

```
1
2     def handle_xml_request(self, msg):
3         self.ts = time.perf_counter()
4         self.send_down(msg)
```

Figura 3. Método handle_xml_request

```
1
2     def handle_xml_response(self, msg):
3         # Captura a lista de qualidades disponíveis
4         self.parsed_mpd = parse_mpd(msg.get_payload())
5         self.qi = self.parsed_mpd.get_qi()
6         self.statemachine = FSM(self.qi[0], self.q_max,
              self.q_min, self.m, self.n)
7
8         # Calcula o valor do lk
9         self.te = time.perf_counter() - self.ts
10        self.S = msg.get_bit_length()
11        self.D = self.S/(self.te)
12        self.l0 = self.D/self.de
13        G = -self.de*self.kp/self.x
14        self.lk = (np.exp(G*(time.perf_counter()-self.t0))/self.x
              + 1)*self.l0
15
16        self.send_up(msg)
```

Figura 4. Método handle_xml_response

Para a primeira execução do método **handle_segment_size_request** (figura 5), os parâmetros da máquina de estados são atualizados com os valores calculados em **handle_xml_response**. Para todas as outras, os valores vem do próximo método a ser descrito. Depois, pelo método **set_state** da máquina de estados, o valor da qualidade l é recebido. Então, é verificado se o estado da máquina é IDLE e, se for o caso, executa a ação correspondente. Por fim, o valor de qualidade é definido como o maior valor existente no servidor que seja menor que o l vindo da máquina de estados. Isso ocorre para que não seja solicitado uma qualidade que não existe no servidor.

O método **handle_segment_size_response** (figura 6) calcula o valor de $l(k)$ de maneira semelhante à realizada no método **handle_xml_response**, mas o primeiro método repete esse processo a cada novo pedaço de vídeo, enquanto o segundo só realiza uma vez. O valor tempo de *buffer* para referência do estado IDLE é atualizado por meio do método **get_amount_video_to_play** do objeto **whiteboard**. Então, o método **finalization** (figura 7) salva os valores relevantes para a construção dos gráficos para análise.

```
1
2     def handle_segment_size_request(self, msg):
3         # Passa os valores de lk e q
4         self.statemachine.set_params(self.lk, self.qt)
5         l = self.statemachine.set_state()
6
7         # Verifica se está no estado IDLE e dá uma pausa
8         if (self.statemachine.current_state == 'IDLE'):
9             interval = self.statemachine.get_IDLE_time(self.lk,
                  self.D, self.x)
10            time.sleep(interval)
11
12        # Seleciona o maior valor que satisfaz o l definido pelo
              FSM
13        selected_qi = self.qi[0]
14        for i in self.qi:
15            if l > i:
16                selected_qi = i
17
18        # Incrementa o contador do estado
19        self.states[self.statemachine.current_state] += 1
20
21        msg.add_quality_id(selected_qi)
22        self.ts = time.perf_counter()
23        self.send_down(msg)
```

Figura 5. Método handle_segment_size_request

```
1
2     def handle_segment_size_response(self, msg):
3         # Calcula a qualidade l(k) e o tempo de buffer
4         self.te = time.perf_counter() - self.ts
5         self.S = msg.get_bit_length()
6         self.D = self.S/(self.te)
7         self.qt = self.whiteboard.get_amount_video_to_play()
8         self.l0 = self.D/self.de
9         G = -self.de*self.kp/self.x
10        self.lk = (np.exp(G*(time.perf_counter()-self.t0))/self.x
              + 1)*self.l0
11
12        self.statemachine.prev_q = self.qt
13
14        self.send_up(msg)
```

Figura 6. Método handle_segment_size_response

```
1
2     def finalization(self):
3         states_vector = [self.states['lc'], self.states['l0'],
              self.states['l+'], self.states['l-'],
              self.states['IDLE']]
4         qi = self.whiteboard.get_playback_qi()
5         buffer = self.whiteboard.get_playback_buffer_size()
6         pauses = self.whiteboard.get_playback_pauses()
7         history = self.whiteboard.get_playback_history()
8
9         # Salva os dados
10        np.save('results/qi.npy', qi)
11        np.save('results/buffer.npy', buffer)
12        np.save('results/pauses.npy', pauses)
13        np.save('results/history.npy', history)
14        np.save('results/states.npy', states_vector)
```

Figura 7. Método finalization

2.4. Máquina de estados

Para a máquina de estados, os parâmetros $l(k)$ e $q(t)$ são avaliados para que seja definido qual a qualidade do próximo pedaço de ser vídeo a ser solicitado. Para isso, a máquina de estados (figura 8) conta com cinco estados: l_c, l_+, l_-, l_0 e IDLE. O estado l_c representa o estado atual da máquina, por isso ele é apresentado centralizado. Os estados l_+ e l_- representam, respectivamente, o acréscimo e o decréscimo na qualidade do vídeo. O estado l_0 representa uma condição crítica na qual o *buffer* está abaixo do mínimo. Por fim, o estado IDLE é dual ao estado l_0 e representa a condição na qual o *buffer* está acima do limite.

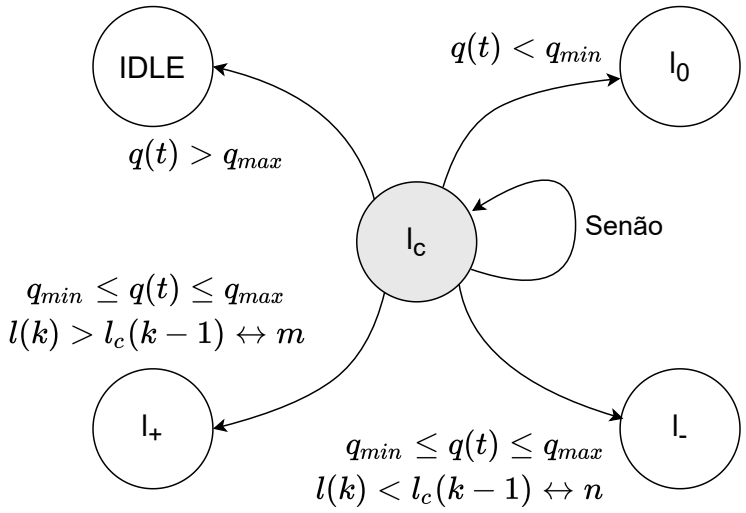

Figura 8. Diagrama da máquina de estados

Considerando os valores q_{max} e q_{min} como os limites superior e inferior de controle do tempo de *buffer*, a máquina de estados faz as seguintes avaliações:

1. Se $q(t) < q_{min}$, então a qualidade do vídeo será a menor disponível para evitar o esvaziamento do *buffer*. Logo após executar essa ação, a máquina retorna para o estado l_c;
2. Senão, se $q(t) > q_{max}$, então o *download* será pausado por um tempo ΔI para o nível do *buffer* possa ser esvaziado. O valor do intervalo ΔI é calculado por:

$$\Delta_I(k) = q(t_e(k-1)) + x - \frac{l_n \cdot x}{\bar{D}(k)} - q_{max}. \tag{9}$$

   Em que $q(t_e(k-1))$ é o tempo de *buffer* ao final do *download* do último pedaço de vídeo. Depois do tempo de pausa, a máquina retorna para o estado l_c e seleciona a qualidade definida antes de entrar no estado IDLE;
3. Senão, se $q_{min} \leq q(t) \leq q_{max}$ e $l(k) > l_c(k-1)$ para m pedaços de vídeo, então a qualidade de vídeo requisitada aumentará para a sugerida. Depois de executar essa ação, a máquina retorna para o estado l_c;
4. Senão, se $q_{min} \leq q(t) \leq q_{max}$ e $l(k) < l_c(k-1)$ para n pedaços de vídeo, então a qualidade de vídeo requisitada diminuirá para a sugerida. Depois de executar essa ação, a máquina retorna para o estado l_c;
5. Senão, a qualidade do vídeo se mantém igual a solicitada da última vez.

2.5. Implementação da máquina de estados

A máquina de estados foi implementada por meio de uma classe chamada **FSM**, que possui, além da inicialização, quatro métodos: **set_params**, **get_conditions**, **set_state** e **get_IDLE_time**. A ordem na qual esses métodos são chamados dentro do código principal do algoritmo já foi apresentada na explicação sobre os métodos **handle_xml_response** e **handle_segment_size_request**. Nessa subseção, será detalhado o funcionamento de cada método da máquina de estados.

É no método de inicialização, o primeiro método, que variáveis como m, n, q_{max} e q_{min} são definas, assim como quase todas as outras variáveis utilizadas na classe. Depois, no método **set_params** (figura 9), um dicionário é definido para relacionar o nome do estado com a qualidade de vídeo correspondente caso ele seja selecionado. Além disso, os valores atuais da qualidade $l(k)$ e do tempo de *buffer* $q(t)$ são salvos.

Então, o método **get_conditions** (figura 10) realiza a avaliação dos valores atuais da qualidade $l(k)$ e do tempo de *buffer* $q(t)$ conforme descrito na subseção anterior, e retorna o estado correspondente. É nesse método que a quantidade de solicitações de acréscimo ou decréscimo na qualidade é contabilizada, pelas variáveis $self.m\_chks$ e $self.n\_chks$. Esse são os valores comparados com os limites de m e n estabelecidos para que haja a troca na qualidade do vídeo.

```
1
2     # Define o dicionário com os estados
3     def set_params(self, l, q):
4         self.current_l = l
5         self.current_q = q
6         self.states = {'lc': self.prev_lc, 'l0': self.min_rate,
              'l+': self.current_l, 'l-': self.current_l, 'IDLE':
              self.current_l}
```

Figura 9. Método set_params

```
1
2     # Verifica as codições atuais
3     def get_conditions(self):
4         # Se q(t) < q_min --> estado l0
5         if (self.current_q < self.q_min):
6             return 'l0'
7
8         # Se não, se q(t) > q_max --> estado IDLE
9         elif (self.current_q > self.q_max):
10            return 'IDLE'
11
12        # Se não, se q_min <= q(t) <= q_max --> estado l+, l- ou lc
13        elif (self.current_q >= self.q_min and self.current_q <=
              self.q_max):
14            if (self.current_l > self.prev_lc and self.m_chks >=
                  self.m_max):
15                self.m_chks = 0
16                self.n_chks = 0
17                return 'l+'
18
19            elif (self.current_l > self.prev_lc and self.m_chks <
                  self.m_max):
20                self.m_chks += 1
21                return 'lc'
22
23            elif (self.current_l < self.prev_lc and self.n_chks >=
                  self.n_max):
24                self.n_chks = 0
25                self.m_chks = 0
26                return 'l-'
27
28            elif (self.current_l < self.prev_lc and self.n_chks <
                  self.n_max):
29                self.n_chks += 1
30                return 'lc'
31
32            else:
33                return 'lc'
```

Figura 10. Método get_conditions

```
1
2       # Atualiza o estado e retorna o próximo lc
3       def set_state(self):
4           self.prev_state = self.current_state
5           self.current_state = self.get_conditions()
6           self.prev_lc = self.states[self.current_state]
7
8           return self.states[self.current_state]
```

Figura 11. Método set_state

```
1
2       # Calcula o tempo de pausa no estado IDLE
3       def get_IDLE_time(self, l, D_x, x):
4           delta_I = self.prev_q + x - (l*x)/D_x - self.q_max
5           return delta_I
```

Figura 12. Método get_IDLE_time

No terceiro método, **set_state** (figura 11), o estado da máquina e o valor da qualidade são atualizados. Então, com o retorno do método **get_conditions** e com o dicionário dos estados já definido, esse método retorna a qualidade selecionada para o vídeo. Por fim, o método **get_IDLE_time** (figura 12) calcula o tempo de pausa do estado IDLE conforme especificado pela equação 9 e retorna esse valor.

2.6. Seleção dos parâmetros

Para a seleção dos parâmetros, os autores realizaram diversos testes com os ganhos dos controladores, o valor de q_0, de m e de n. Os melhores resultados foram obtidos com os conjuntos 1, 5 e 9, que são apresentados na tabela 1. Como, no desenvolvimento do nosso controlador somente o ganho proporcional foi considerado, por questão de simplificação, é natural que a escolha de parâmetros seja pelo conjunto 5.

Conjunto	1	5	9
q_0	20	20	40
k_p	0,01	0,01	0,01
k_i	0	0	0
k_d	0,01	0	0,01
m	10	10	10
n	3	3	3

Tabela 1. Configurações da seleção de parâmetros

De acordo com Ito et al. [2015], o aumento no ganho proporcional pode fazer com que a variável de controle $q(t)$ apresente altas oscilações e, inclusive, baixo nível do tempo de *buffer*. Dessa forma, o valor de k_p foi mantido como 0,01. No entanto, o valor

que de q_0 foi adaptado para 30 a fim de deixar um espaço de 10 segundos como limites do *buffer* de modo que q_{min} é igual a 10 segundos e q_{max} é igual a 50 segundos. Assim como utilizado no artigo base, o valor de $d_e(k)$ foi escolhido como 1 e os valores de m e n foram mantidos como definido no conjunto 5 da tabela 1.

2.7. Definição dos cenários de teste

Para avaliar o algoritmo implementado, utilizamos a plataforma *pyDash* desenvolvida por Marotta et al. [2021]. A configuração da simulação é definida pelo arquivo **dash_client.json**, a figura 13 apresenta a estrutura desse arquivo. Cada um dos campos define um parâmetro da simulação. Por exemplo, **traffic_shaping_profile_sequence** define a sequência do perfil de transferência, em que L é a menor restrição na banda e H a maior, e **traffic_shaping_profile_interval** define por quanto tempo um determinado perfil será aplicado antes de ser trocado pelo próximo na sequência.

```
{
    "buffering_until": 5,
    "max_buffer_size": 60,
    "playbak_step": 1,
    "traffic_shaping_profile_interval": "5",
    "traffic_shaping_profile_sequence": "LMH",
    "traffic_shaping_seed": "1",
    "url_mpd" : "http://45.171.101.167/DASHDataset/BigBuckBunny/
    1sec/BigBuckBunny_1s_simple_2014_05_09.mpd",
    "r2a_algorithm": "R2A_FineTunedControl"
}
```

Figura 13. Estrutura do arquivo dash_client

Somente os valores do intervalo do perfil e a sequência do perfil de restrição na banda foram alterados durantes os testes a fim de verificar o desempenho do algoritmo para as diferentes sequências e intervalos de duração. Dessa forma, foram realizadas três baterias de teste: na primeira, mantivemos o intervalo de cada perfil, mas utilizamos uma sequência com somente um perfil; na segunda, o intervalo de cada perfil subiu para 30 segundos e utilizamos, novamente, somente um perfil na sequência; Por fim, na terceira, foi definido um intervalo de tempo intermediário e uma sequência com diferentes perfis de restrição. A tabela 2 apresenta um resumo dos cenários utilizados.

Nº do teste	Intervalo do perfil	Perfil de restrição
1	5	LLL
2	5	MMM
3	5	HHH
4	30	LLL
5	30	MMM
6	30	HHH
7	20	HHHMMMMMLLLLLLL
8	20	MHMMLHHHHMLMMHLL
9	20	MMLMLHHLLLMLHM

Tabela 2. Cenários de intervalo do perfil e perfil de restrição

Como, em todas as simulações, somente o vídeo com segmentos de tamanho igual a 1 segundo foi utilizado, a variável x da equação 7 foi definido como 1.

3. Resultados

Os gráficos das figuras 14 e 15 apresentam três curvas cada, correspondentes aos testes 1 (curva azul), 2 (curva laranja) e 3 (curva verde) apresentados na tabela 2. Pelo gráfico 14, é possível observar que, quando a condição na banda é a menos restritiva, o algoritmo consegue solicitar o conteúdo do vídeo em alta qualidade muito rapidamente e manter isso de forma quase constante durante toda a execução com uma média de qualidade igual a 17,98 (equivalente a uma qualidade de 3,31Mbps). Pelo gráfico 15, é possível observar que não ocorreram eventos de estouro ou esvaziamento do *buffer* e o seu tempo médio foi de 36,04s, um pouco maior que o valor de referência q_0.

Enquanto isso, para a condição na qual a restrição na banda é média, é possível observar que a qualidade alcançada é bem inferior, com média de 8,14 (equivalente a uma qualidade de 537,82kbps). Inclusive, nesse cenário, há momentos em que a qualidade selecionada é a mínima, de 46,98kbps, quando o tempo de *buffer* fica inferior ao valor mínimo de 10s. Apesar disso, o algoritmo conseguiu evitar o estouro ou esvaziamento do *buffer* e o seu tempo médio foi de 18,68s, menor que o valor de referência q_0.

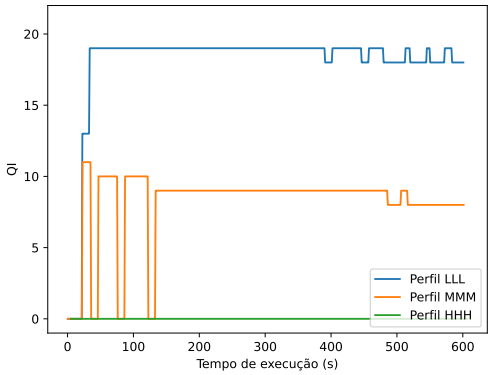

Figura 14. Evolução da qualidade nos teste 1, 2 e 3

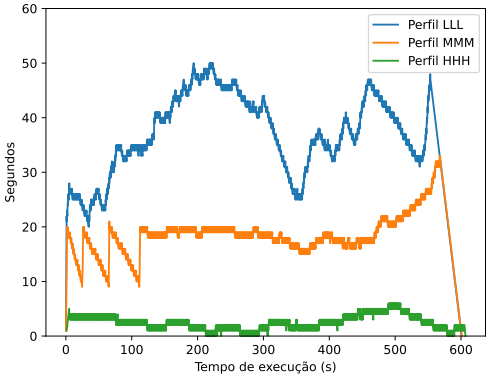

Figura 15. Evolução do tempo de buffer nos testes 1, 2 e 3

Depois, para a condição com a maior restrição na banda, o gráfico 14 apresenta sempre a qualidade mínima. De fato, o tempo de *buffer* ficou abaixo do mínimo durante toda a execução (média de 2,47s). Então, pela regra definida na máquina de estados, a qualidade selecionada nesse caso deve ser a mínima para tentar evitar o esvaziamento. Apesar do *buffer* diminuir várias vezes, somente em duas oportunidades houve uma pausa na execução do vídeo com média de 1,01 segundo por pausa. Esse comportamento evidência um controle do algoritmo uma vez que, mesmo na condição com maior restrição de transferência, só pausou o vídeo em dois momentos durante uma execução de 600 segundos.

No gráfico da figura 16 estão os resultados da evolução da qualidade nos testes 4, 5 e 6. Se compararmos a figura 16 com a figura 16, é possível observar que as curvas obtidas são bastantes semelhantes. É razoável supor que esse seja o resultado esperado, uma vez que o perfil de restrição não troca ao final do intervalo. Enquanto isso, para perfis

diferentes, o aumento no tempo de cada perfil pode resultar em qualidades maiores por mais tempo ou em esvaziamento do *buffer*, dependendo da duração e da ordem escolhidas para os perfis.

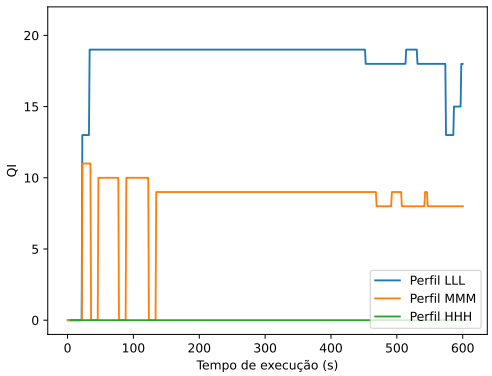

Figura 16. Evolução da qualidade nos testes 4, 5 e 6

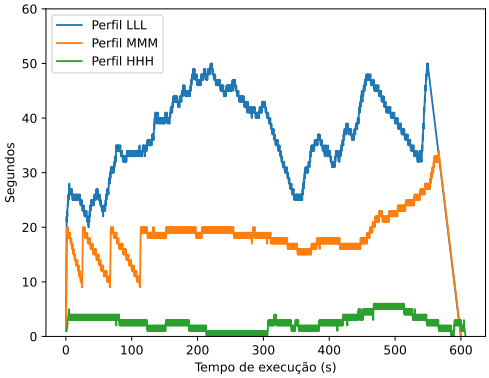

Figura 17. Evolução do tempo de buffer nos testes 4, 5 e 6

Com relação ao teste 7 apresentado na tabela 2, observa-se pela figura 18 dois estágios com qualidade mínima. O primeiro permanece em zero devido ao perfil de alta restrição de banda H, permanece até a transição para média restrição de banda M e, por sensibilidade, zera novamente. Em seguida, o perfil tende a ser predominantemente M, a qualidade aumenta e atinge estabilidade até a transição do M para o perfil de baixa restrição de banda L, sendo possível observar pela figura 19, que houve zeramento do *buffer* no período de intervalo entre 300 e 400 segundos.

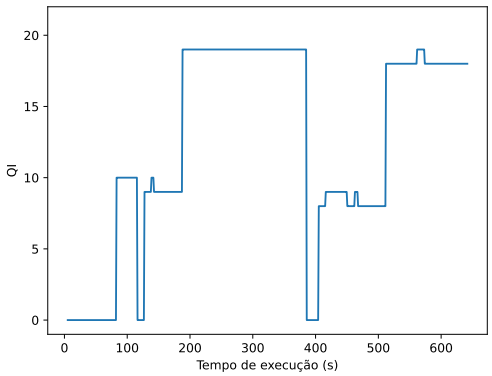

Figura 18. Evolução da qualidade no teste 7

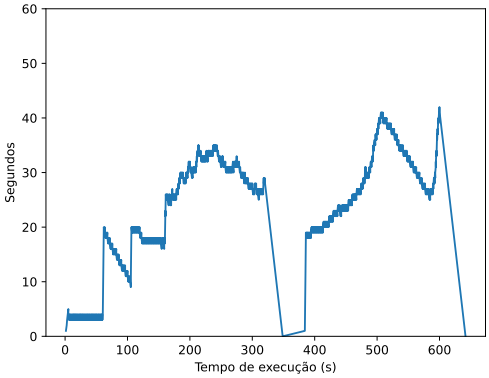

Figura 19. Evolução do tempo de buffer no teste 7

Isso se relaciona ao estado ocioso do IDLE em que ocorre uma pausa no sistema para que não ocorra estouro de buffer. No que se segue, o perfil permanece em L, tende a aumentar a qualidade e estabilizar. Ainda vale notar, pela figura 19, a estabilidade relativa do controle do *buffer*, que apresentou apenas eventuais transições de estados com um tempo médio de 23,25s no *buffer*, melhor que o obtido no caso com restrição média na banda. Além disso, a qualidade média obtida foi de 12 (equivalente a uma qualidade de

1,31Mbps), também melhor que o caso de restrição média.

Com relação ao teste 8 apresentado na tabela 2, foram obtidos alguns resultados bem variados se comparado com o teste anterior devido a alta variabilidade das entradas escolhidas na analise. Essa escolha acaba por fazer com que o *buffer* não acumule valores muito maiores que o valor alvo q_0. Esse efeito, por sua vez, acaba gerando uma quantidade maior de *buffer* durante os instantes em que a qualidade sobe e quantidade menores nos pontos em que a qualidade desce.

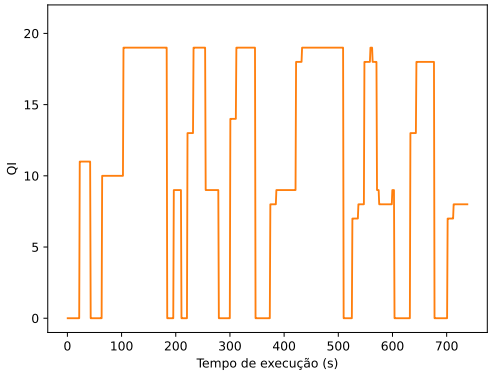

Figura 20. Evolução da qualidade no teste 8

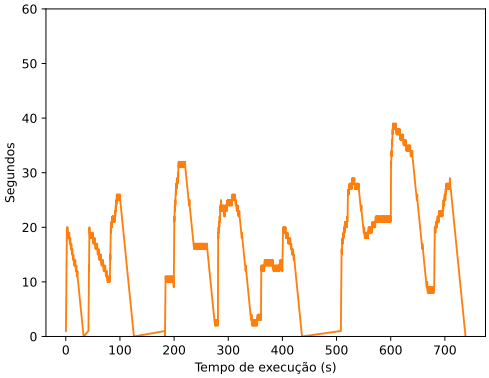

Figura 21. Evolução do tempo de buffer no teste 8

Durante o teste, ocorreram 3 instantes em que a execução é pausada (com um tempo médio de 45,39s por pausa e desvio padrão de 33,21s), sendo uma durante a primeira restrição alta, representada por H, no inicio do teste quando ainda não havia tempo de *buffer* suficiente para manter o vídeo sem travar. Os outros dois mostram instantes em que há uma queda de qualidade da forma mais extrema possível, indo da maior qualidade disponível para a pior e permanecendo nela por um período de tempo. Apesar do valor médio da qualidade ser maior que o caso médio dos primeiros testes (8,85), o tempo de *buffer* foi levemente menor e as pausas alongadas prejudicaram a execução do vídeo.

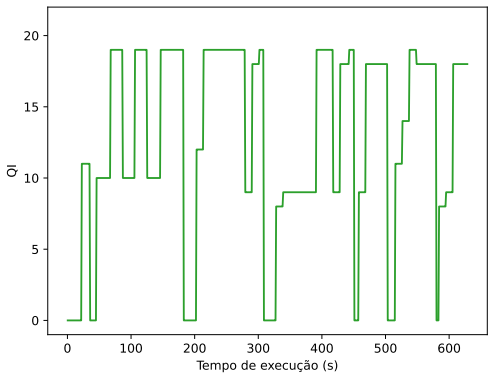

Figura 22. Evolução da qualidade no teste 9

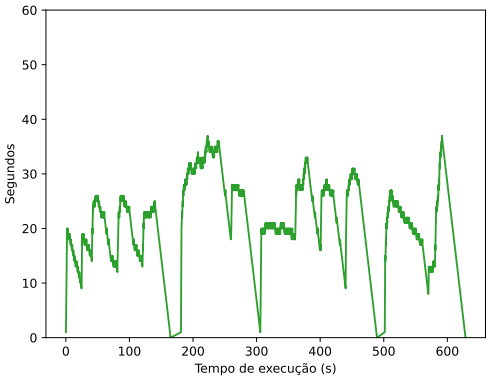

Figura 23. Evolução do tempo de buffer no teste 9

Por fim, com relação ao teste 9 apresentado na tabela 2, é perceptível pela figura 22 que aconteceram 2 pausas (com um tempo médio de 14,12s por pausa) durante a execução do código. Essa pausas são observadas de forma parecida ao teste anterior, em que ocorre a pausa durante a queda de uma qualidade mais alta possível para a menor possível por uma quantidade estendida de tempo. Esse esvaziamento do *buffer* pode ser explicado pela sequência de restrição escolhida.

Como o teste 9 apresenta menos estados de alta restrição na banda que o teste 8, é natural esperar que o algoritmo tenha um resultado melhor nesse caso. De fato, o algoritmo apresenta qualidade média e tempo médio de *buffer* maiores, 12,34 e 21,34s respectivamente. Os resultados obtidos sugerem que o algoritmo implementado apresenta um bom desempenho quando não existem trocas bruscas no perfil de restrição da banda ou períodos prolongados de alta restrição na rede.

4. Conclusão

Ao longo deste trabalho, realizamos a implementação de um algoritmo adaptativo baseado em um sistema com um controlador e uma máquina de estados. A proposta de baseia na teoria de controle e utiliza desta para garantir parâmetros como a estabilidade do sistema. Para verificar o desempenho da proposta, realizamos nove testes por meio da plataforma *pyDash* com variações em relação ao intervalo do perfil de restrição da banda e da sequência desses perfis. Os resultados obtidos segurem que o algoritmo é capaz de apresentar um bom desempenho na transmissão dos segmentos de vídeo quando não há mudanças bruscas no perfil de restrição ou períodos prolongados de alta restrição da rede.

Durante o processo de elaboração do trabalho, foi possível desenvolver habilidades de programação em Python, leitura e compreensão de textos em inglês, e trabalho em grupo. Aprendemos a ideia básica por trás do padrão MPEG-DASH para transmissão de conteúdo de mídia e lidamos com as dificuldades inerentes do processo de implementação de um algoritmo adaptativo, como a busca pelo ajuste correto das variáveis do sistema. Como trabalho futuro, uma proposta é a revisão dos parâmetros que regulam a sensibilidade do sistema para acréscimo ou decréscimo da qualidade. Alternativamente, outra proposta é investigar outras formas de projeto do sistema de controle utilizando controle ótimo e controle robusto.

5. Referências

Bentaleb, A., Taani, B., Begen, A. C., Timmerer, C., e Zimmermann, R., 2019. A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP. *IEEE Communications Surveys Tutorials*, 21(1):562–585.

Caetano, M. F. *Projeto de Programação - Algoritmos Adaptativos de Streaming de Video MPEG-DASH*, 2022. Disciplina: Transmissão de Dados.

Cisco. Visual Networking Index - Global Device Growth Traffic Profiles. `https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_Device_Growth_Traffic_Profiles.pdf`, 2018. [Online; Acesso em: 21 de abr. de 2022].

Ito, M. S., Bezerra, D., Fernandes, S., Sadok, D., e Szabo, G., 2015. A fine-tuned control-theoretic approach for dynamic adaptive streaming over HTTP. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, páginas 301–308.

Li, Z., Zhu, X., Gahm, J., Pan, R., Hu, H., Begen, A. C., e Oran, D., 2014. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733.

Marotta, M. A., Souza, G. C., Holanda, M., e Caetano, M. F., 2021. PyDash - A Framework Based Educational Tool for Adaptive Streaming Video Algorithms Study. In *2021 IEEE Frontiers in Education Conference (FIE)*, páginas 1–8.

Mueller, C., Lederer, S., Grandl, R., e Timmerer, C., 2015. Oscillation compensating dynamic adaptive streaming over http. In *2015 IEEE International Conference on Multimedia and Expo (ICME)*, páginas 1–6.

Ogata, K. *Engenharia de Controle Moderno*. Pearson, London, 5ª ed edition, 2010.