Universidade de Brasília (UnB)
Departamento de Ciências da Computação (CIC)

RELATÓRIO

Trabalho Final de TR2

Matrícula	**Nome Completo**
17/0056465	Ayllah Ahmad
17/0162290	Guilherme Braga

Prof. André Drummond

Brasília, 11 de Maio de 2021

Índice

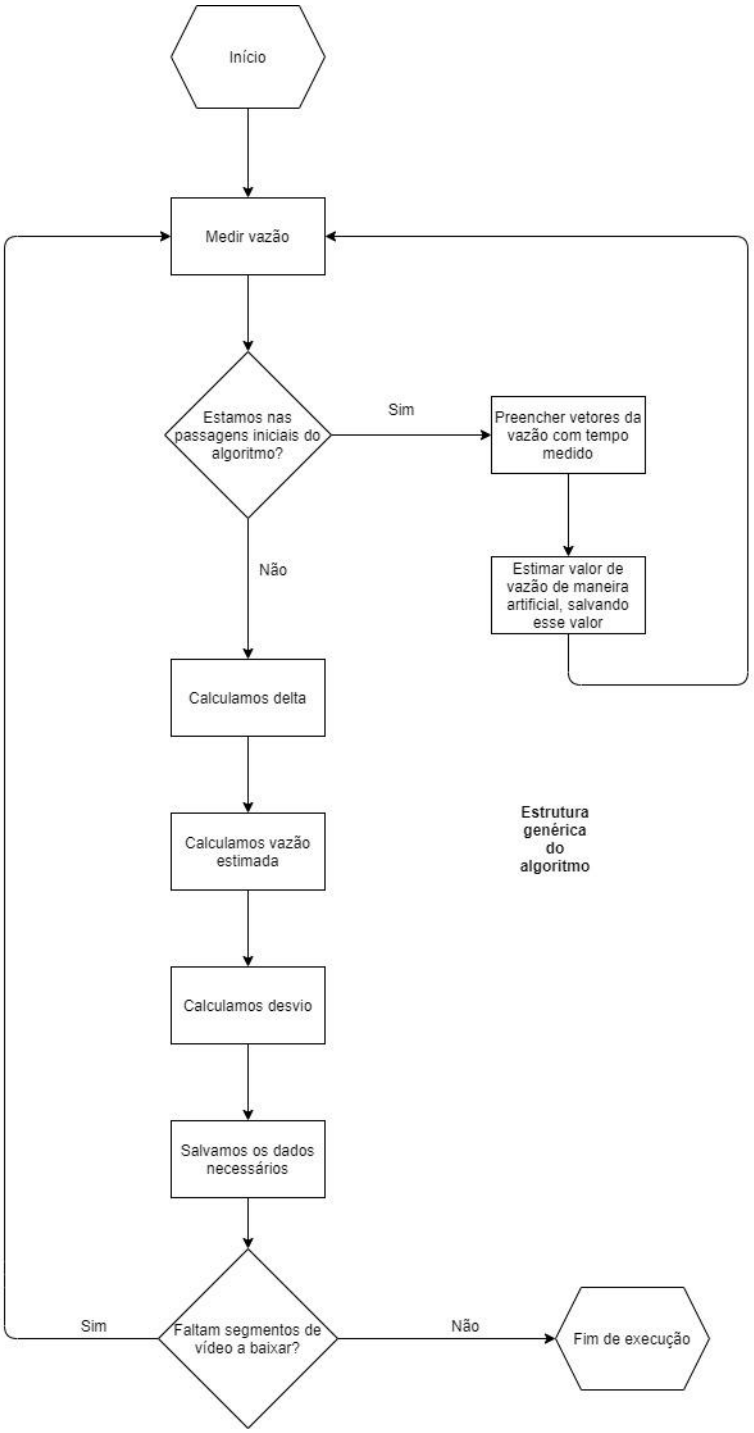
1- Introdução

Decidiu-se ter como base o artigo “Adaptive Streaming of Audiovisual Content Using MPEG DASH”, de Truong Cong Thang e Anh T. Pham, para que então os conceitos abordados neste artigo possam primeiramente ser implementados, para posteriormente serem testados e potencialmente melhorados. Tanto o código mostrado aqui como imagens de testes podem ser visualizados no Github.

A motivação inicial por trás da adoção deste algoritmo foi sua simplicidade e maleabilidade, pois como será discutido, ele é altamente customizável. Diferente de uma abordagem no qual se pega unicamente médias de vazões medidas, esta abordagem envolve calcular novamente esta média dando diferentes pesos para diferentes vazões dependendo de como a aplicação se comporta, suavizando mudanças e descartando valores muito antigos, que podem não ser tão relevantes na hora de se tomar uma decisão recente.

2- Algoritmo ABR

A estrutura do algoritmo implementado segue, genericamente, a seguinte implementação:



Se é medida a vazão da requisição, para que seja calculado um determinado delta e o desvio entre uma medição na vazão atual e uma medição mais antiga. Nas primeiras

passagens do algoritmo original, utiliza-se a noção de que é necessário preencher uma lista de vazões estimadas com simples vazões que foram medidas, para que esses valores sejam utilizados em passagens futuras, e com o tempo os valores estimados vão ser normalizados com valores realistas. Ao final, quando não há mais segmentos de vídeo para serem baixados, se executa o que está no buffer e o algoritmo é encerrado. A vantagem maior, como será mostrado mais adiante, é que este algoritmo é teoricamente capaz de se moldar na medida que o algoritmo tem cada vez mais iterações, sempre podendo se adaptar, priorizando valores mais recentes no lugar de valores antigos, que não necessariamente refletem o estado de uma conexão naquele momento no qual uma decisão deverá ser tomada.

2.1 - Funcionamento do algoritmo de referência

Inicialmente uma vazão estimada é definida como a vazão medida anteriormente, para que então possa ser atualizada com a seguinte equação:

$$T_e(i) = (1 - \delta)\, T_e(i - 2) + \delta T_s(i - 1)\,,\ i > 0$$

Onde $T_e(i)$ é a vazão estimada em uma passagem "i" e $T_s(i)$é a vazão efetivamente medida naquela passagem. Logo, a vazão estimada é obtida ponderando-se a vazão estimada em 2 passagens anteriores, somando com a vazão medida da última passagem do algoritmo. A ideia por trás de se ponderar essas duas medições é a de ter a capacidade de suavizar trocas quando necessário, e torná-las mais bruscas dependendo da variação pela qual as medições passaram. Isso é obtido por meio do cálculo de δ, que vai levar em consideração o desvio entre a vazão que foi estimada para uma passagem "i" e a vazão efetivamente medida naquela passagem do algoritmo. O cálculo, como descrito na referência do trabalho, ocorre da seguinte maneira:

$$\delta = \frac{1}{1+e^{-K(p-P_0)}}$$

O valor de δvai ser o determinante para o cálculo da estimativa, e vai de uma valor teórico de 0 para 1. As constantes KeP_0são valores obtidos a partir de testes relacionados ao comportamento da rede que o algoritmo está utilizando. Elas podem ser usadas para incentivar ou atenuar mudanças do resultado de δ.

$$p = \frac{|T_s(i) - T_e(i)|}{T_e(i)}$$

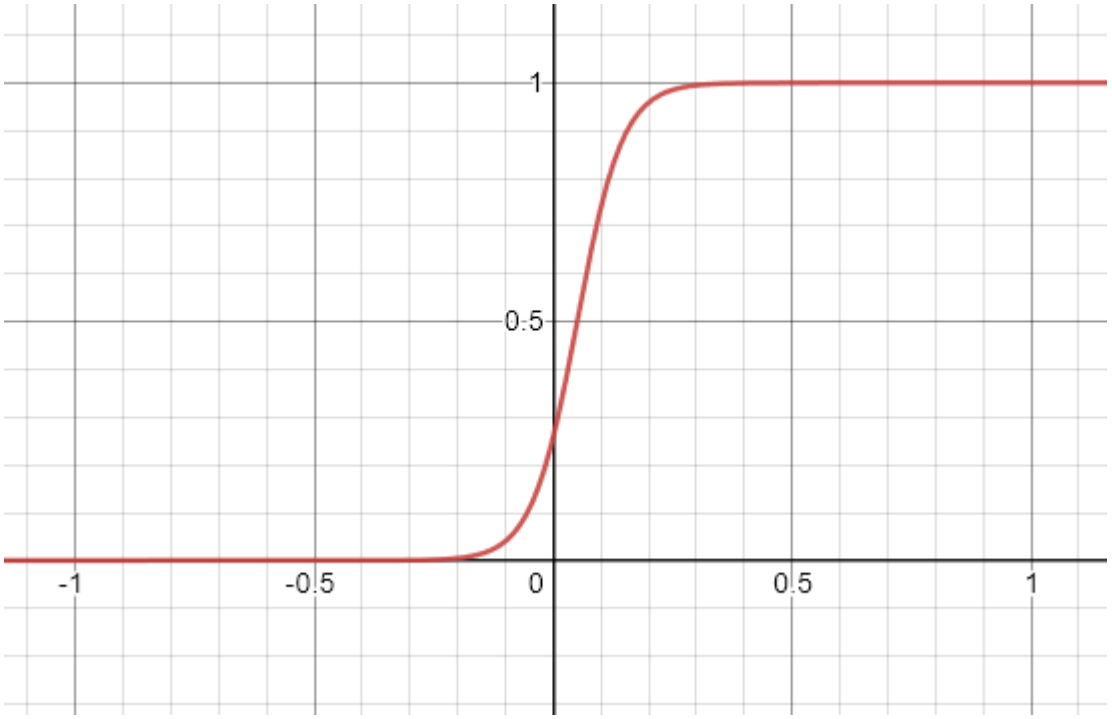
O valor de "p" representa o desvio normalizado entre uma medição e uma estimativa de throughput. Quando é muito elevado, representa uma brusca mudança entre um valor estimado e um valor medido, e quando o valor é muito baixo, a situação contrária é representada.

2.2- Testes no cálculo de delta

O artigo original do trabalho menciona valores pré-fixados para as constantes do cálculo de δ, como sendo Kigual a 21 e P_0igual a 0.05, caracterizando uma rede cabeada. Para a equação com os valores já aplicados, obtém-se:

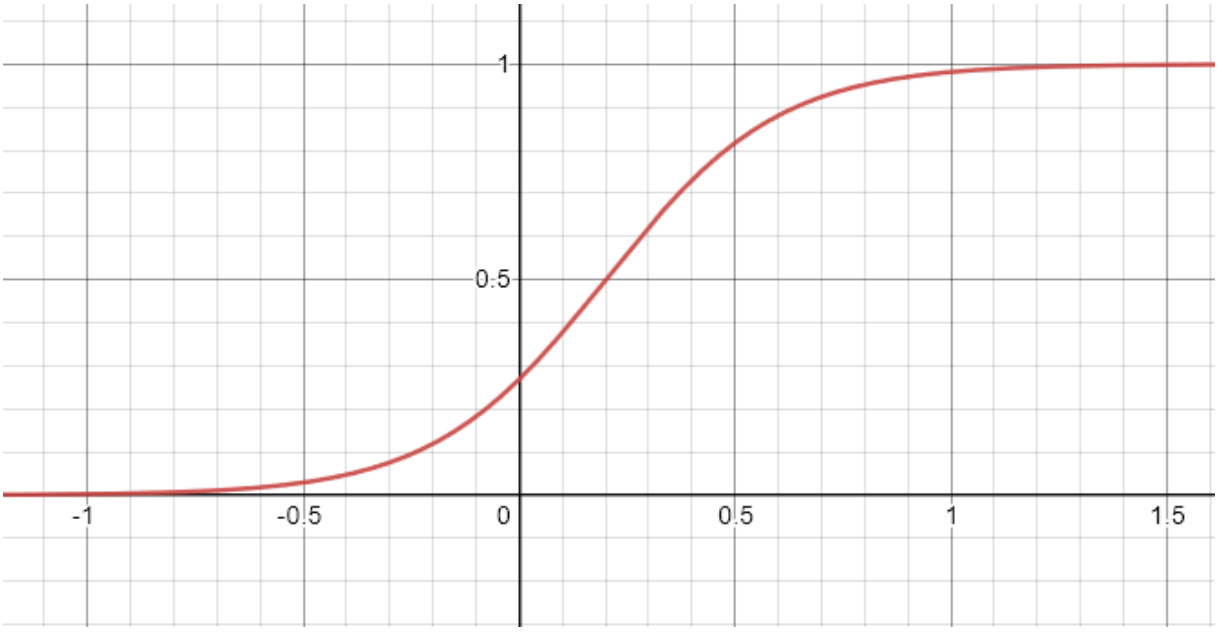
$$\frac{1}{1+e^{-21(p-0.05)}}$$

No eixo y, se encontram os valores de δ calculados, para o valor do desvio respectivamente no eixo x.



Percebe-se que δ é negativo para valores do desvio mais ou menos abaixo de 0.3, e positivo para valores acima de 0.3, o que significa que o delta é positivo para a maioria dos casos, o que representa uma prioridade ao peso do termo que leva a vazão medida, e não a estimada. O artigo menciona que uma rede Wi-Fi utilizaria um valor de P_0de 0.2, que para cortar o eixo “y” mais ou menos na mesma região da função anterior, apresentaria um Kigual a 5. Como pode então ser demonstrado respectivamente na função com os valores atualizados e seu gráfico:

$$\frac{1}{1+e^{-5(p-0.2)}}$$



Significa que, para parâmetros considerados de Wi-Fi para o artigo de referência, com P_0com o valor de 0.2 (como especificado), um K com o valor de 5 (calculado posteriormente) vai ser capaz de interceptar o eixo “y” mais ou menos no mesmo ponto que os parâmetros de rede cabeada interceptavam. Uma interpretação deste resultado é o de que o valor de delta demora muito mais para variar com o parâmetro P_0 da rede Wi-Fi do que o parâmetro da rede cabeada. Existirão casos onde o termo que representa a influência da vazão estimada vai apresentar um peso muito maior do que comparado com o termo que representa a vazão medida. Ou seja, para uma instabilidade maior na rede Wi-Fi (como é de se esperar por natureza ao se comparar com a estabilidade de uma rede por cabo) a mudança ocorre de maneira mais dinâmica, aumentando o peso da vazão estimada em seu cálculo do que comparada com o peso da vazão medida. Isso pode ser benéfico, se a rede Wi-Fi do usuário que está baixando o vídeo for boa porém momentaneamente instável, a mudança na qualidade do vídeo baixado pode ser mais precisa por se dividir em intervalos maiores.

2.3- Comportamento Teórico desta implementação

Definido o funcionamento, na teoria, vão existir algumas diferentes possibilidades para o comportamento da aplicação. Fazendo um exemplo, para uma passagem na qual foi medido 808057 bps seguido de 620705 bps. Se estávamos em uma tendência de manter os mesmos resultados diante de uma margem de erro, todos na casa dos 800 kbps, logo:

$$p_{antigo} = 0.1$$

$$p_{novo} = 0.3$$

$$\delta = \frac{1}{1+e^{-21(0.1-0.05)}} = 0.74$$

$$T_e(i) = (1 - 0.74)\,T_e(i - 2) + 0.74T_s(i - 1),$$

Supondo que as estimativas e a medição usadas não apresentam grandes variações entre si ($T_e(i - 2) \approx T_s(i - 1) \approx 800\text{kbps}$), a medição para esta passagem não vai ser afetada de maneira direta, tendendo ao resultado anterior do que a medição atual. Perceba que este valor de delta é recorrente para passagens que não variam muito (para estes valores de ke de P_0, propostos no artigo), logo a mudança nunca será brusca.

Porém, na próxima passagem, $T_s(i - 1)$ vai passar a ser na casa dos 600kbps, com um desvio menor, por já ter sido calculado p_{novo}na última passagem, supondo que a medição voltou a ser 600kbps:

$$\delta = \frac{1}{1+e^{-21(0.3-0.05)}} = 0.99$$

$$T_e(i) = (1 - 0.99)\,T_e(i - 2) + 0.99T_s(i - 1),$$

Assim fazendo o resultado fazer quase que completamente ir para a medição dos 600kbps. Um problema a ser testado, de fato, é como que o buffer da aplicação vai se comportar para estas mudanças. Se supõe então que há a necessidade de se adicionar uma restrição que satisfaça a existência de um buffer avantajado, que seja capaz de manter alguns segmentos de vídeo em memória para que essas mudanças passem de maneira menos perceptível. Uma contrapartida desta idéia seria que um usuário com a conexão estável e perfeita para o download dos segmentos de vídeo teria uma implementação que abaixaria um pouco a qualidade de seu arquivo para fazer um buffer que teoricamente não seria usado por conta de sua conexão não ter essa necessidade. Na prática esse caso não é muito prático, logo a necessidade da construção de um buffer é real.

2.4- Modelando uma Restrição

Uma restrição básica para melhorar o algoritmo original seria fazer um gargalo na hora de decidir qual qualidade deverá ser efetivamente escolhida. Ao se decidir pegar apenas uma porcentagem do que foi medido como estimativa, podemos abaixar um pouco os valores usados quando comparado com valores medidos, para que então o programa tenha uma boa margem para apresentar um buffer que aguente mais instabilidades na rede sem demonstrar que houve um travamento no vídeo. Mas então, diferentes questionamentos surgem:

- É vantajoso diminuir a qualidade da imagem para todos os que vão utilizar aquele algoritmo, mesmo que o usuário tenha uma conexão impecável?
- A restrição deve ser aplicada sempre ou apenas quando existe grande desvio na medição?
- A restrição deve ser aplicada quando o buffer está vazio? Ou quando o buffer está cheio? Existe um valor intermediário para que todos os casos de teste tenham uma restrição a ser aplicada em valor intermediário?

O que ocorre na prática é que existiu a necessidade de se testar coisas como essa, para se chegar na melhor restrição possível, existindo uma troca entre um buffer mais cheio e uma qualidade mais alta que vai resultar em mais travamentos, não é possível apresentar um algoritmo que faça as duas coisas nos cenários testados. Assim, decidiu-se adotar a metodologia mais otimizada, como pode ser visto a seguir.

```
selected_qi = self.qi[3]

if self.p > 0.4:
    for i in self.qi:
        if (estimativa_atual*0.4) > i:
            selected_qi = i
elif self.p > 0.1 and self.p < 0.4:
    for i in self.qi:
        if restricao > i:
            selected_qi = i
else:
    for i in self.qi:
        if estimativa_atual > i:
            selected_qi = i
```

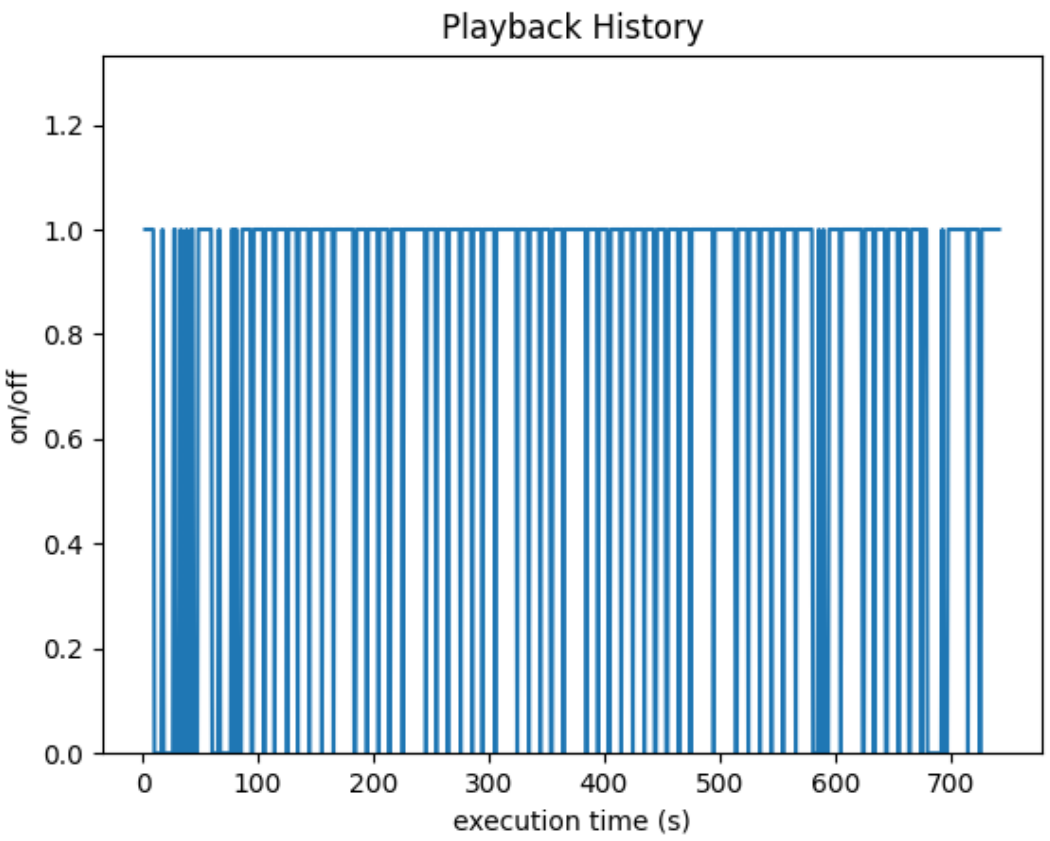
Nessa metodologia, a variância da taxa de transferência é usada para calcular uma margem de segurança para a taxa de transferência estimada. Sendo assim, para valores onde a taxa de transferência (p) é maior que 0.4, δ tenderá a 1 e representará uma brusca mudança entre um valor estimado e um valor medido. Tendo isso em mente, condições foram impostas de modo que não ocorram mudanças com variâncias tão altas. Caso p > 0.4, a estimativa atual calculada da vazão será utilizada em torno de 40%, por outro lado, se 0.1 < p < 0.4, será utilizada uma qualidade em torno da restrição estipulada. O cálculo desta restrição é dado por:$restrição = 1 - mi \cdot estimativa\_atual$, cabe destacar que o mi utilizado para o cálculo da restrição foi colocado em seu máximo, ou seja, 0.5. Por fim, caso p < 0.1, a qualidade selecionada estará em torno da própria estimativa de vazão calculada.

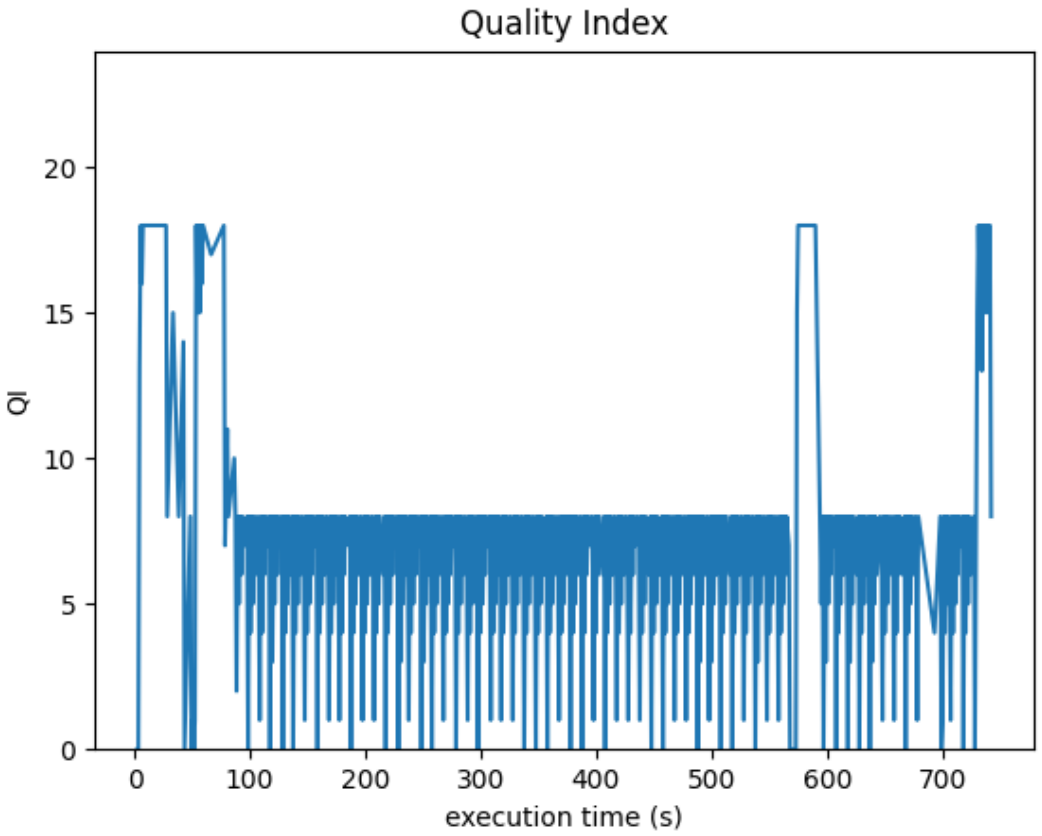
3- Comportamento da Implementação

Todas as imagens resultantes dos testes executados para este relatório podem ser encontradas no Github onde o trabalho foi armazenado ao longo de seu desenvolvimento.

3.1- Algoritmo básico funcionando

A priori, testou-se o algoritmo em sua forma pura, sem tentar nenhuma otimização para ver o que acontecia. O resultado foi uma grande quantidade de travamentos, por conta de que o buffer passava uma boa parte de sua execução vazio. Assim, por exemplo, para K igual a 21 e P_0 igual a 0.05:



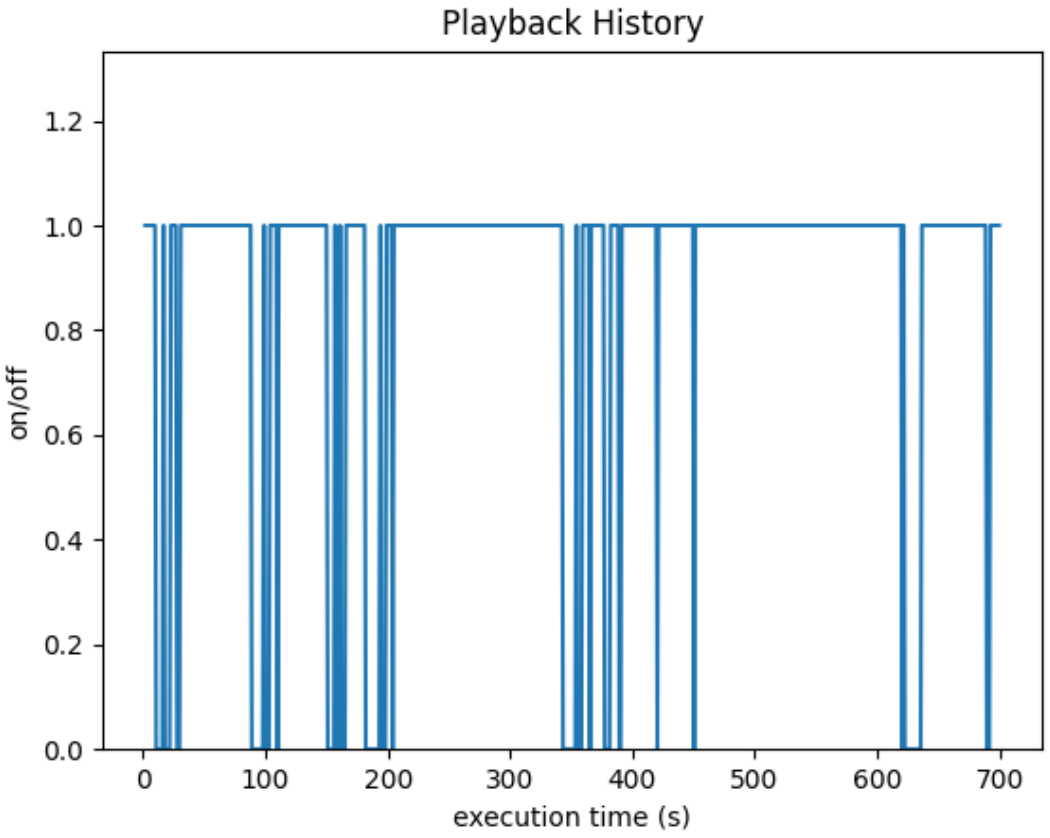


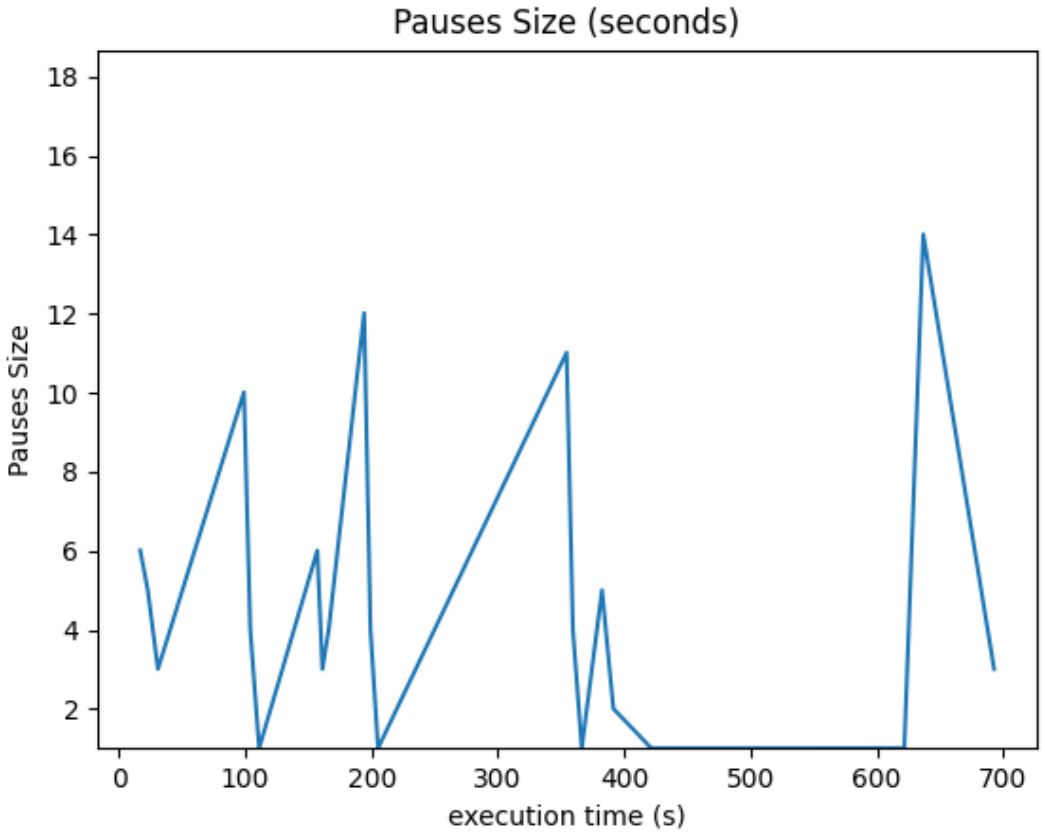
E também, testando-se para P_0igual a 0.2 e k igual a 5 como havia sido previamente calculado, fazendo o teste no mesmo ambiente, o resultado também obtido foi análogo. Na prática as pausas não foram muito prolongadas, porém, eram recorrentes mesmo em um ambiente estável. Logo, a intenção de uma modificação neste algoritmo é o de manter uma qualidade perto da abordagem original, porém, com menos travamentos, se possível nenhum.

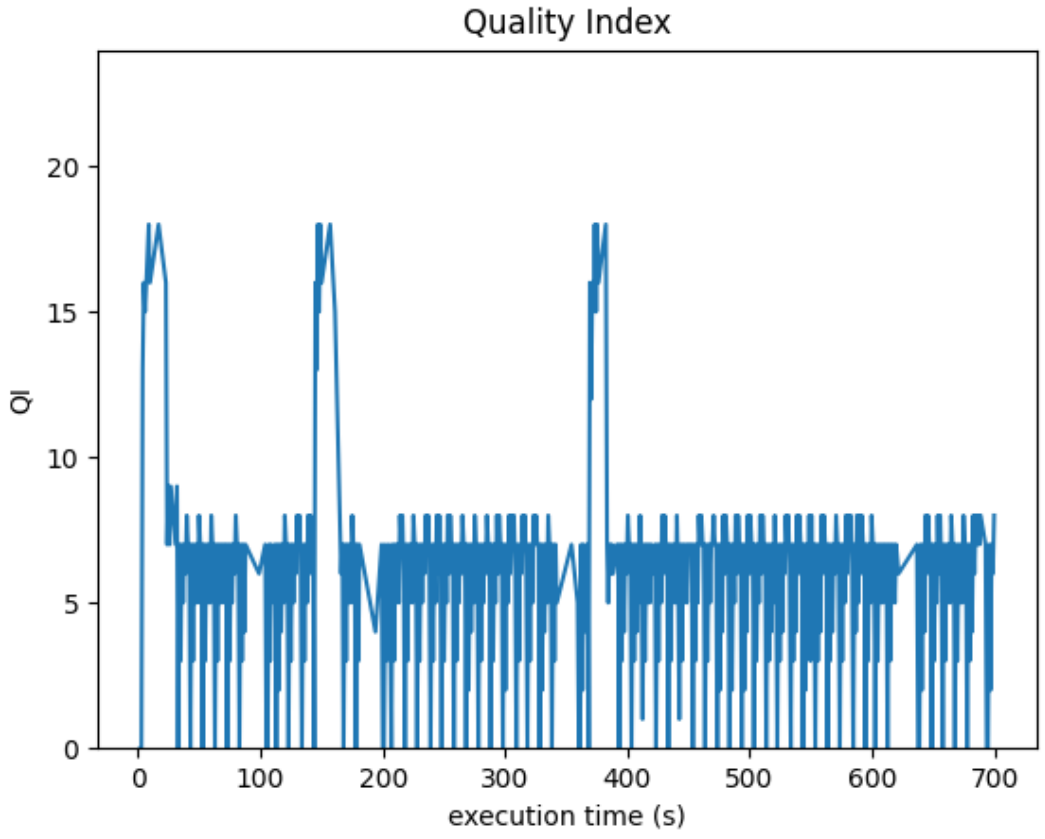
3.2- Primeiras modificações

De início, com uma imposição de determinada restrição quando o buffer chegava a 5, obtivemos os seguintes resultados:

```
if self.whiteboard.get_amount_video_to_play() >= 5:
    for i in self.qi:
        if estimativa_atual > i:
            selected_qi = i
else:
    for i in self.qi:
        if restricao > i:
            selected_qi = i
```



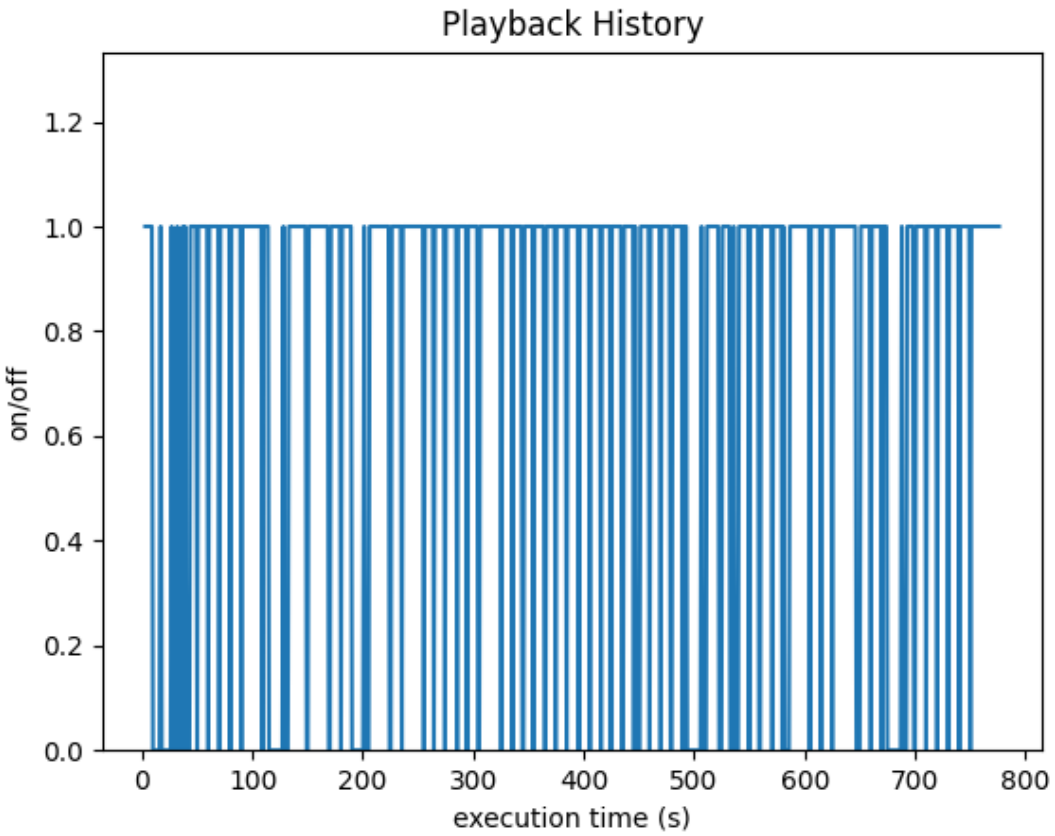

Pauses Size (seconds)
Pauses Size
execution time (s)
18
16
14
12
10
8
6
4
2
0
100
200
300
400
500
600
700

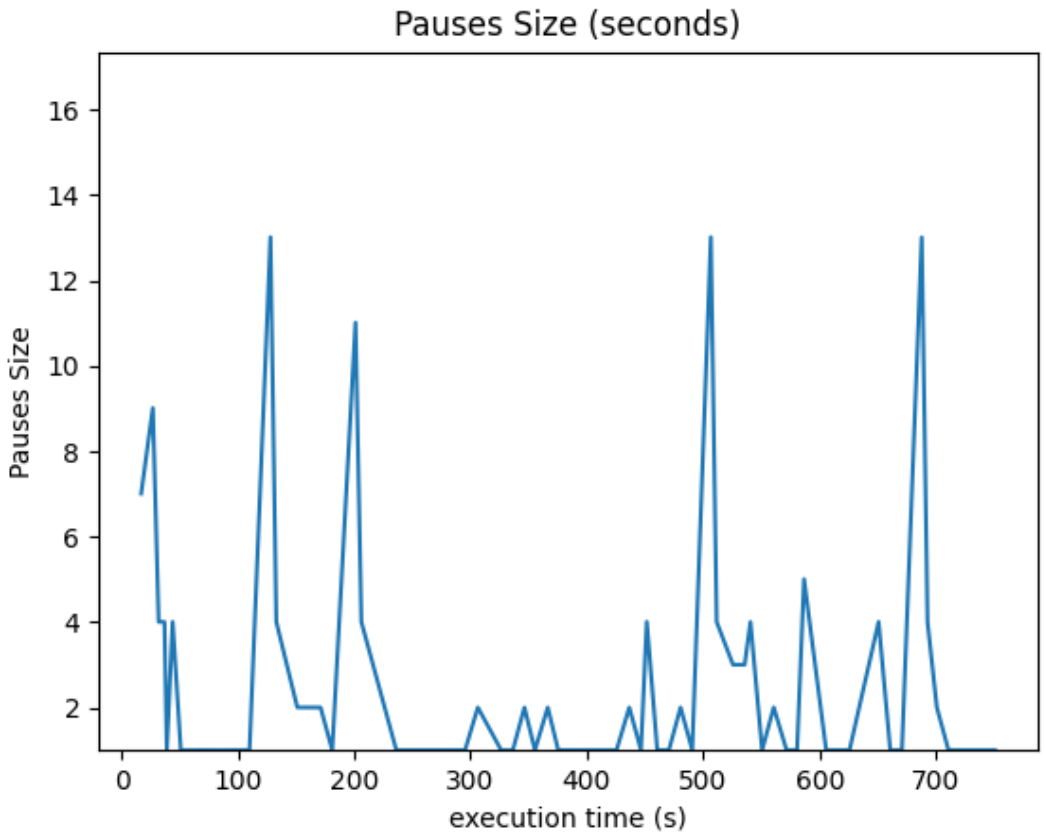

Quality Index
QI
execution time (s)
20
15
10
5
0
0
100
200
300
400
500
600
700

O buffer nestes testes permaneceu relativamente cheio, porém geralmente não passava dos 8 segmentos guardados. Quase sempre que a aplicação era interrompida, o buffer era imediatamente gasto.

Com uma imposição de determinada restrição apenas quando o buffer estava vazio, obtivemos os seguintes resultados:

```
if self.whiteboard.get_amount_video_to_play() > 0:
    for i in self.qi:
        if estimativa_atual > i:
            selected_qi = i
else:
    for i in self.qi:
        if restricao > i:
            selected_qi = i
```

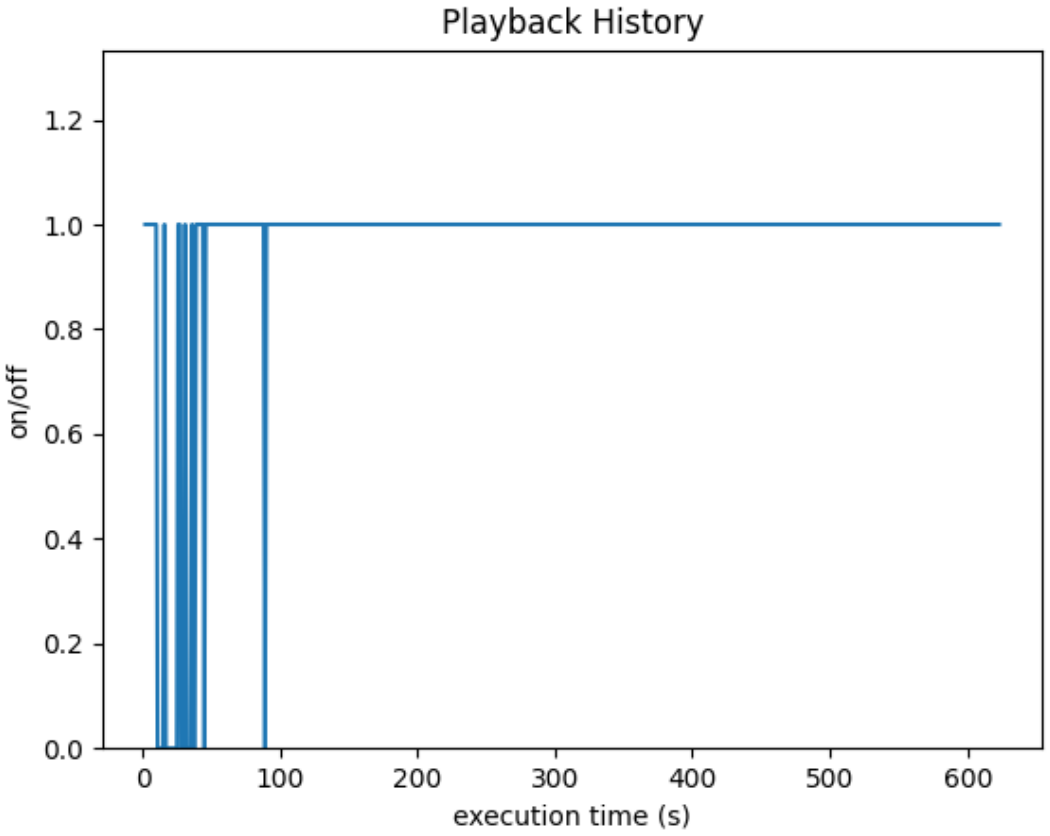


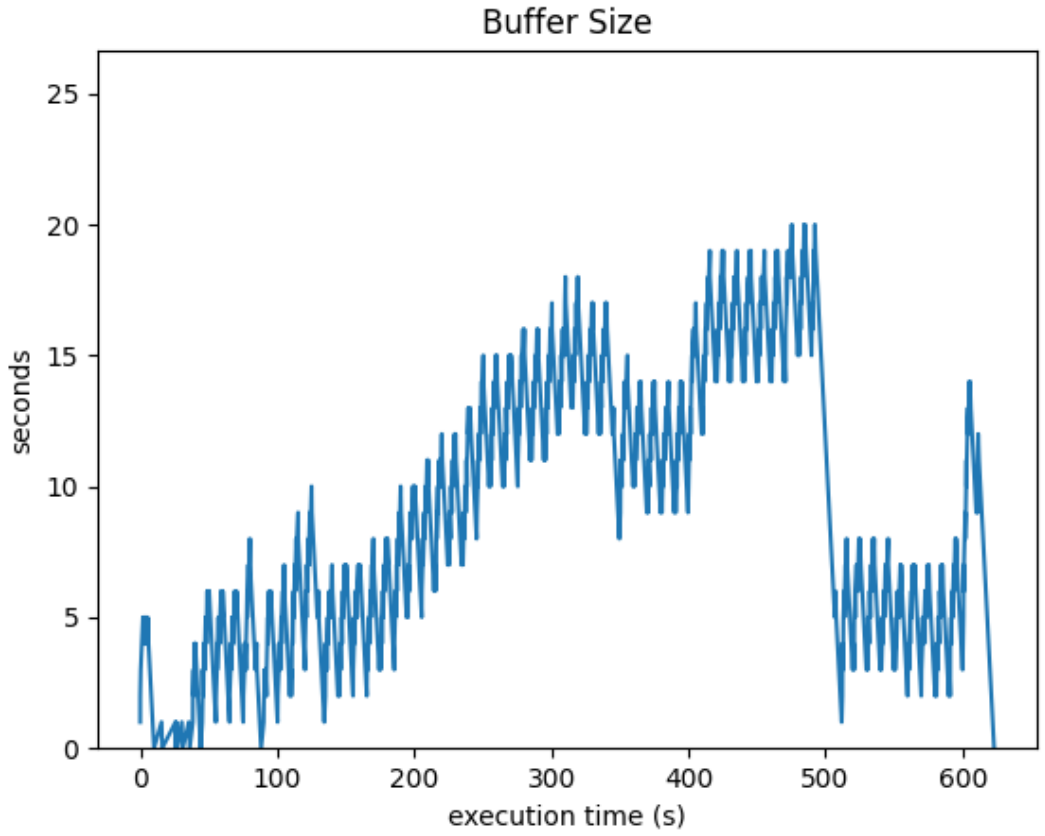


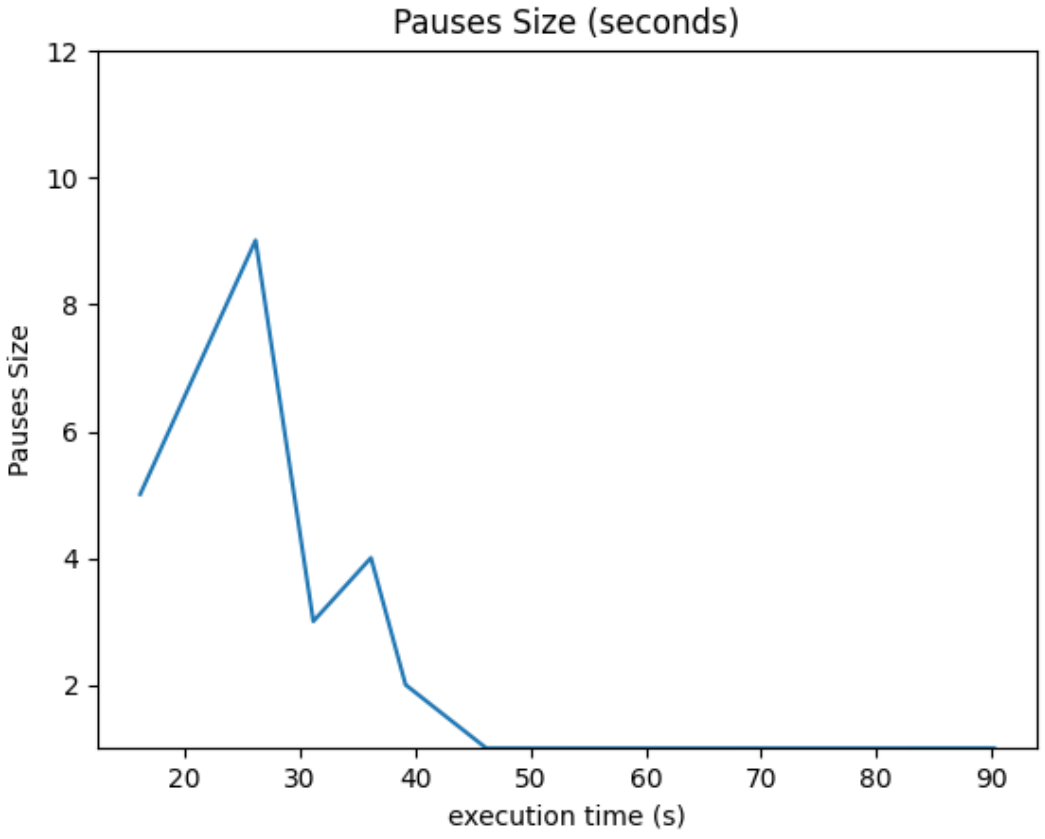
Percebeu-se uma quantidade muito grande de pausas no meio do vídeo quando a restrição era aplicada unicamente quando o buffer não estava vazio, assim a folga de se aplicar a restrição quando o buffer estava com um valor abaixo de 5 fez uma boa diferença. O buffer neste caso também diminui ainda mais.

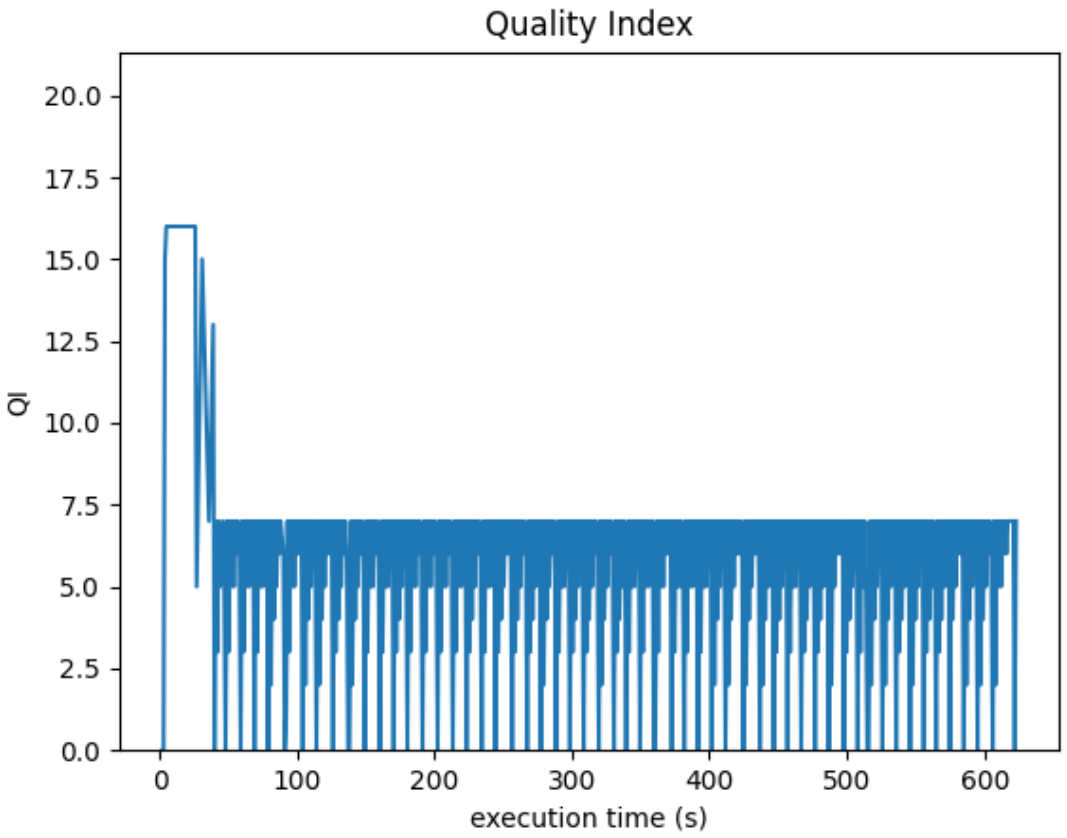
Também testou-se a escolha sempre levando em conta a restrição. Percebe-se um buffer muito cheio e quase nenhuma pausa depois do começo (que começa na primeira vazão medida), o que é positivo, porém a qualidade está um pouco mais baixa do que em casos anteriores. A restrição nesse caso e em todos os anteriores, é a de passar adiante sempre 80% da estimativa calculada previamente.

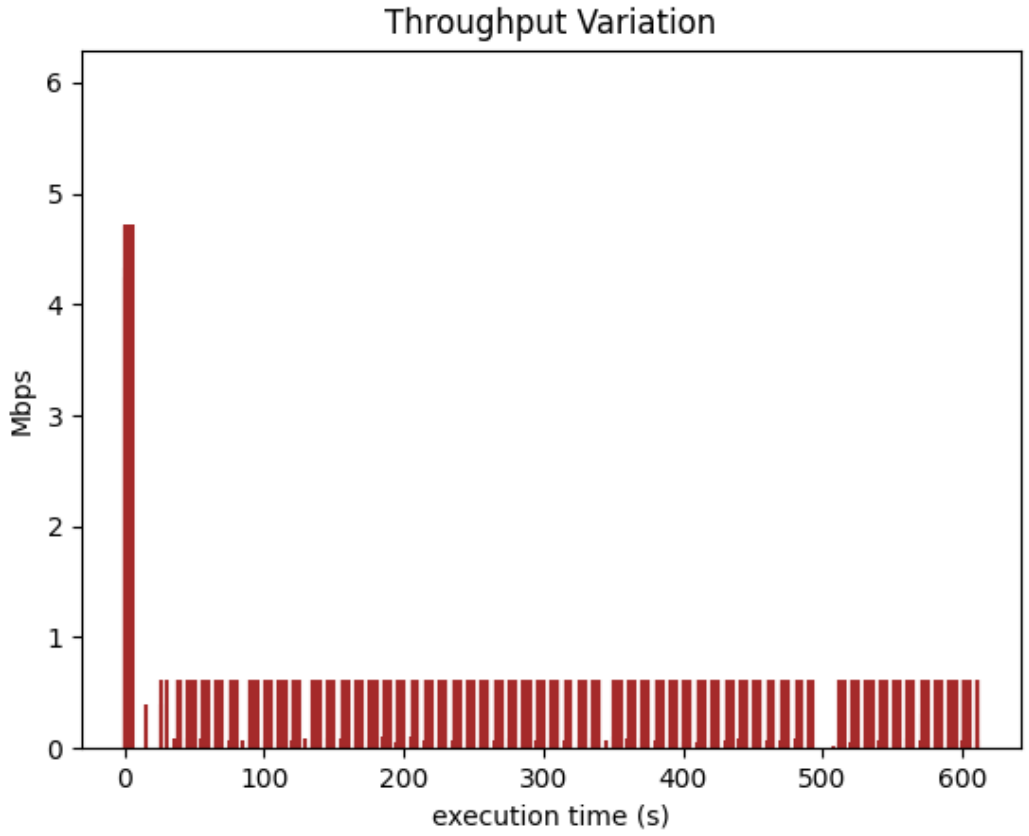
```
for i in self.qi:
    if restricao > i:
        selected_qi = i
```




Buffer Size
seconds
execution time (s)
0
5
10
15
20
25
0
100
200
300
400
500
600


Pauses Size (seconds)
Pauses Size
execution time (s)
2
4
6
8
10
12
20
30
40
50
60
70
80
90


Quality Index
QI
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
100
200
300
400
500
600
execution time (s)

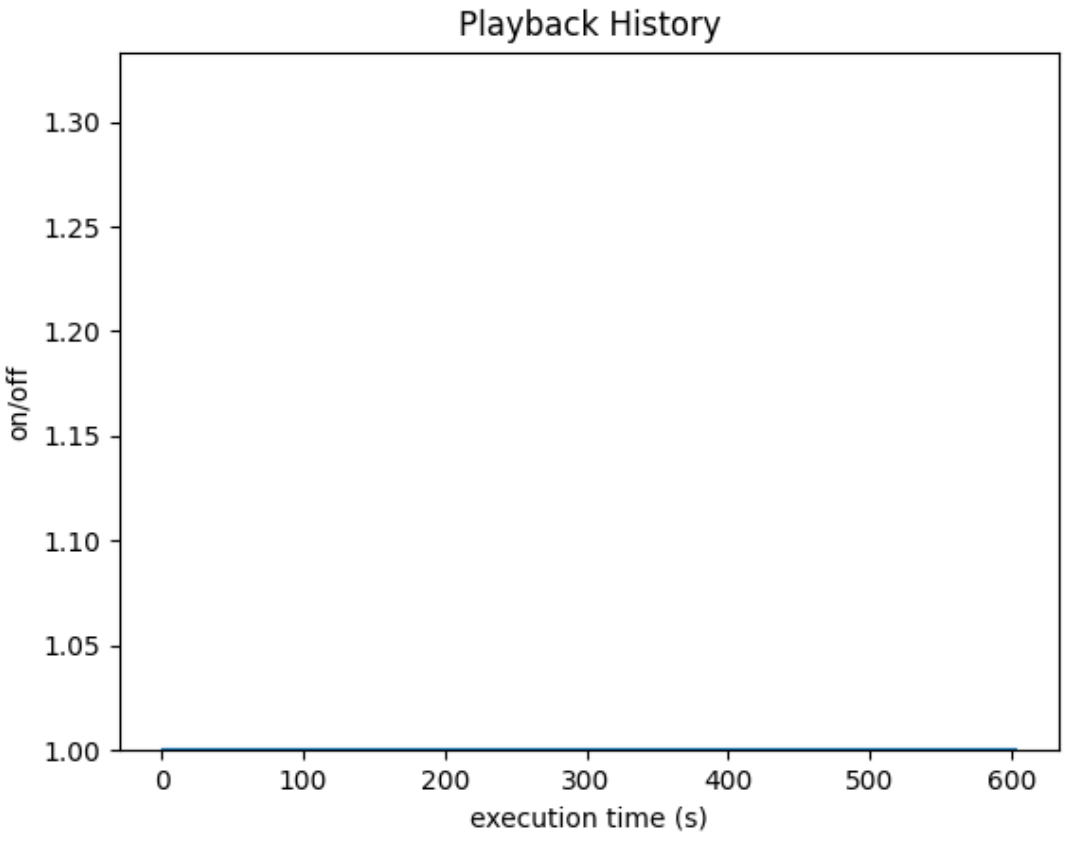

Throughput Variation
Mbps
6
5
4
3
2
1
0
0
100
200
300
400
500
600
execution time (s)

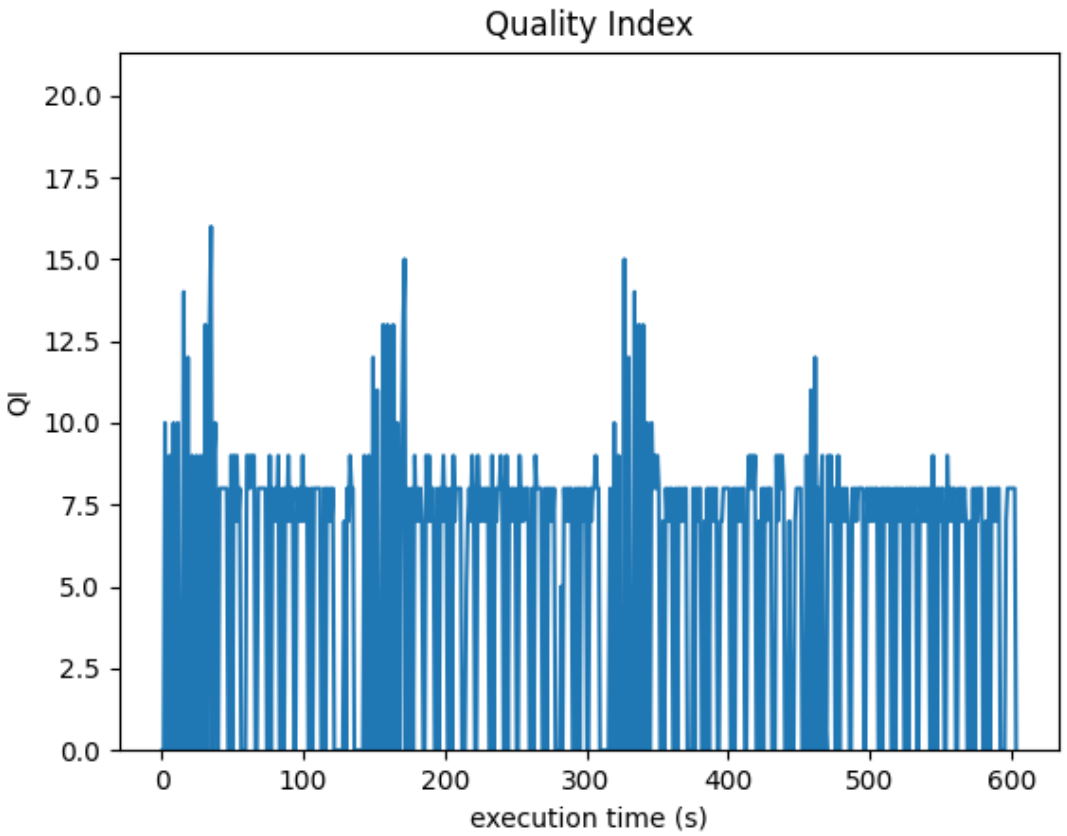
O problema dessa abordagem justamente está na qualidade obtida, que diminuiu um pouco, em nome de ter um buffer que pode ficar muito grande. Assim, manipular quando aplicar uma restrição de excesso no buffer acabou sendo o foco de novos testes.

3.3- Execução com o menor número de travamentos possível

E assim, foi possível efetuar uma passagem do algoritmo sem travamentos, com a desvantagem da qualidade ter caído:

```
if self.p < 0.4:
    if self.whiteboard.get_amount_video_to_play() > 0:
        for i in self.qi:
            if estimativa_atual > i:
                selected_qi = i
    else:
        for i in self.qi:
            if restricao > i:
                selected_qi = i
else:
    selected_qi = self.qi[0]
```

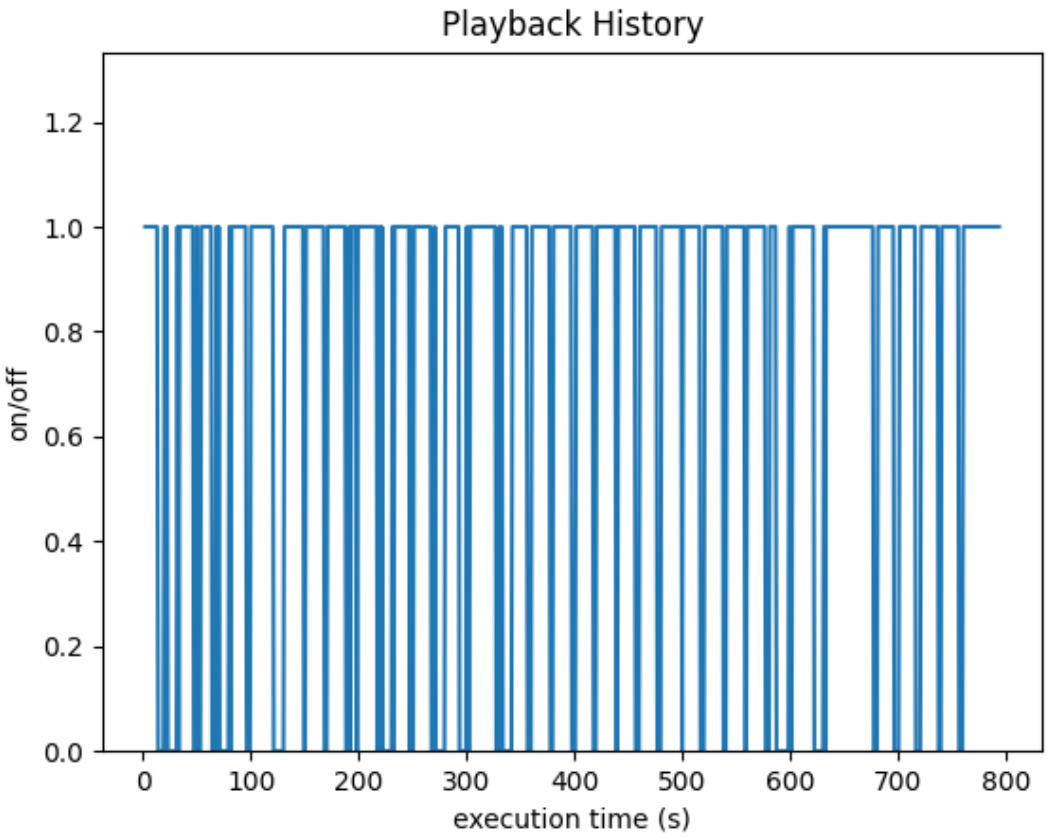


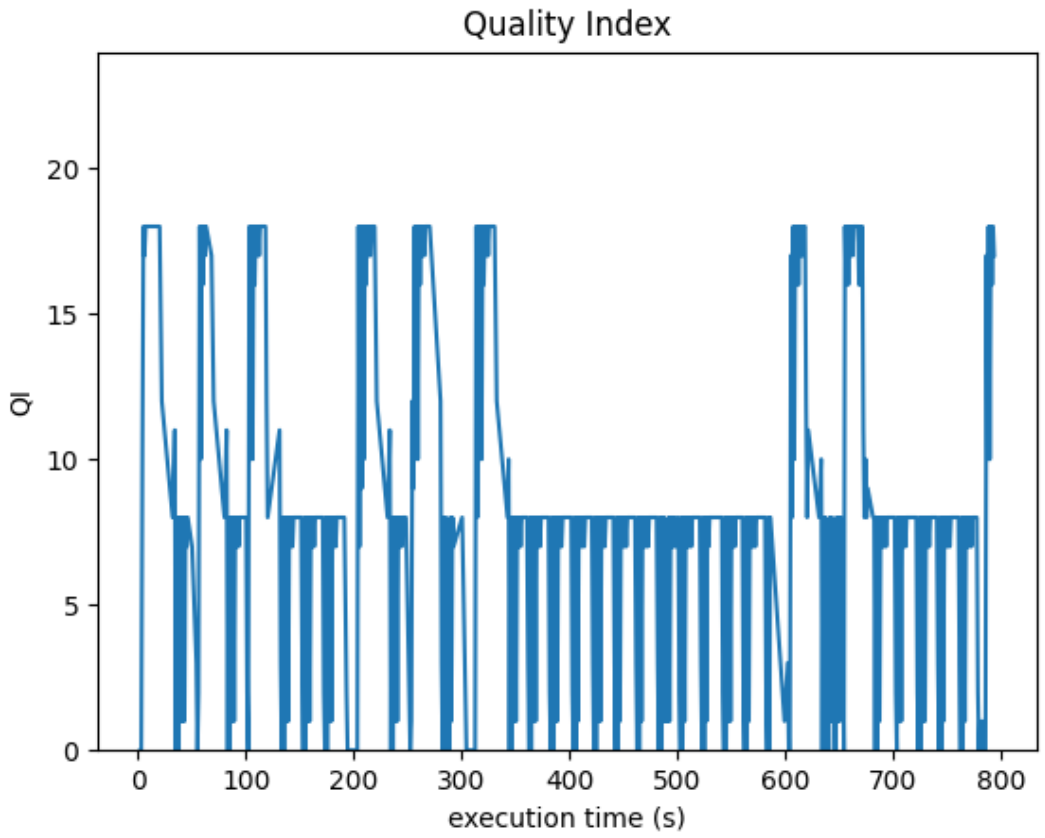


Foi possível observar uma qualidade média decente, porém com muitas quedas para o valor mais baixo possível ser escolhido, o que resulta no buffer encher mais rápido.

Tentando aumentar os valores mínimos que o algoritmo chega, para que não seja requisitado o menor possível quando há um grande desvio, testou-se um algoritmo com uma restrição um pouco maior, que repassa 30% do valor que havia sido estimado.

```
if self.p < 0.4:
    if self.whiteboard.get_amount_video_to_play() > 0:
        for i in self.qi:
            if estimativa_atual > i:
                selected_qi = i
    else:
        for i in self.qi:
            if restricao > i:
                selected_qi = i
else:
    for i in self.qi:
            if restrição_recuperação > i:
                selected_qi = i
```





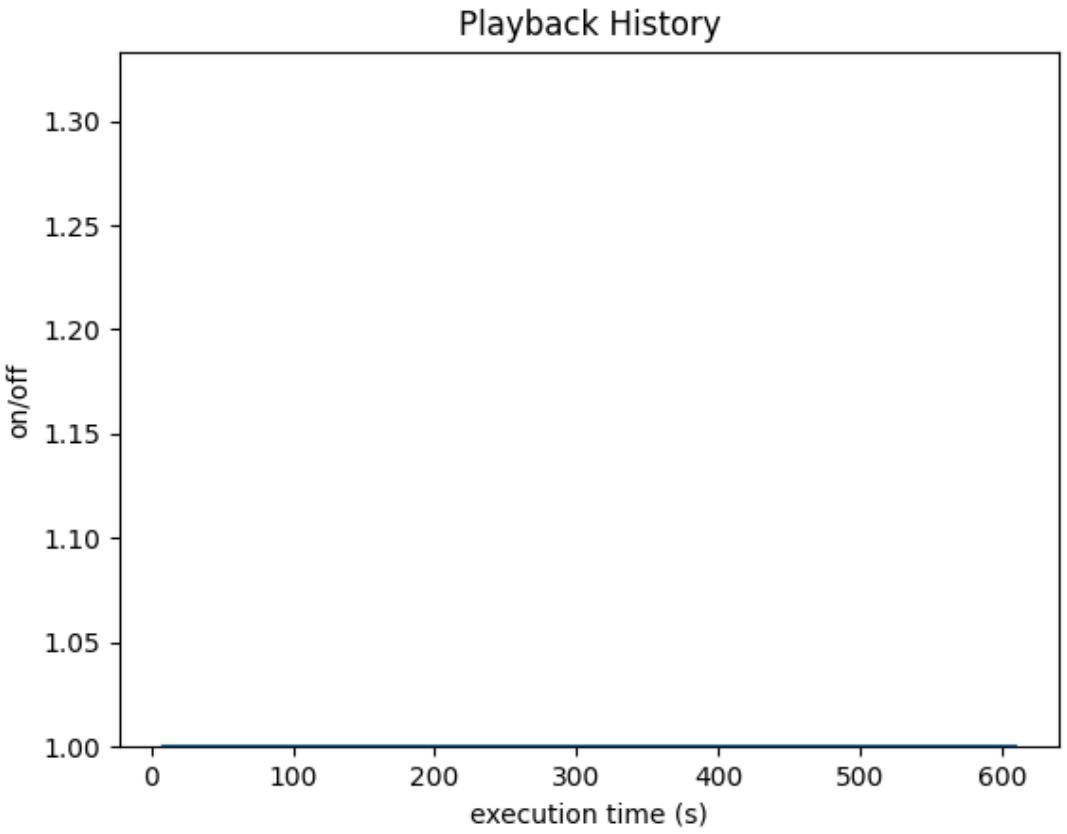
Apesar dos travamentos recorrentes, eles foram um pouco mais espaçados, e foi mais recorrente o aumento da qualidade momentânea, juntamente de atenuamento de quedas.

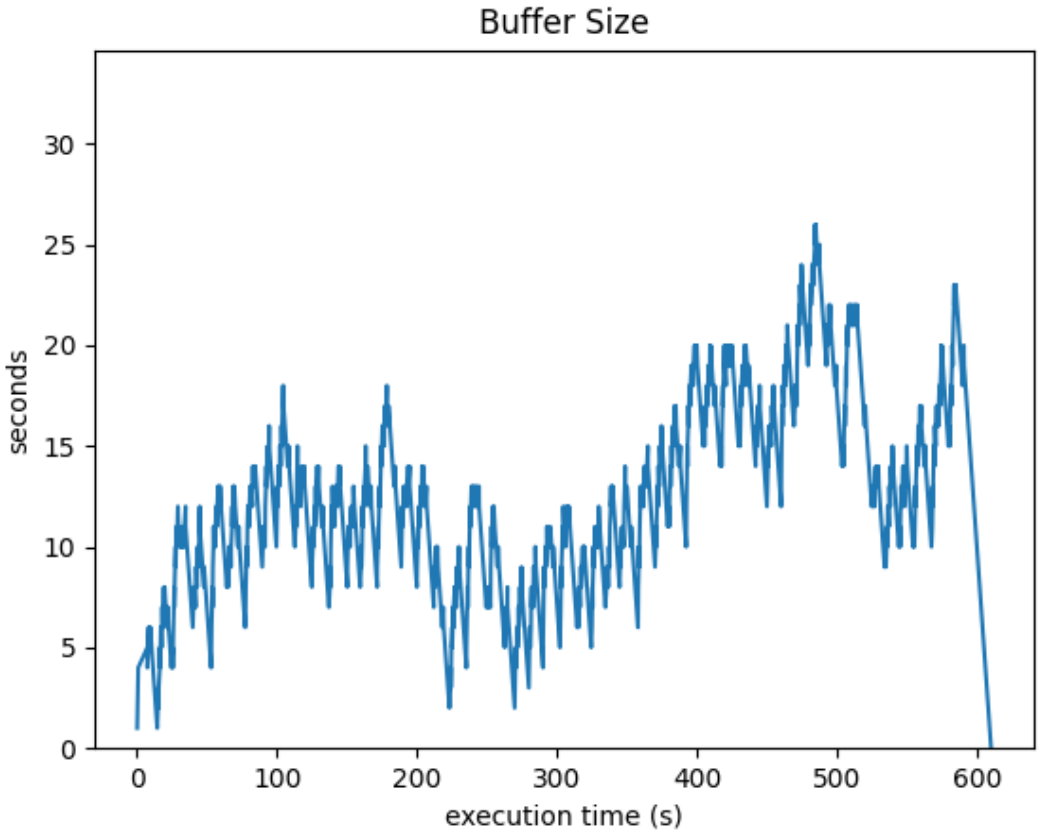
3.4- Modelo final selecionado

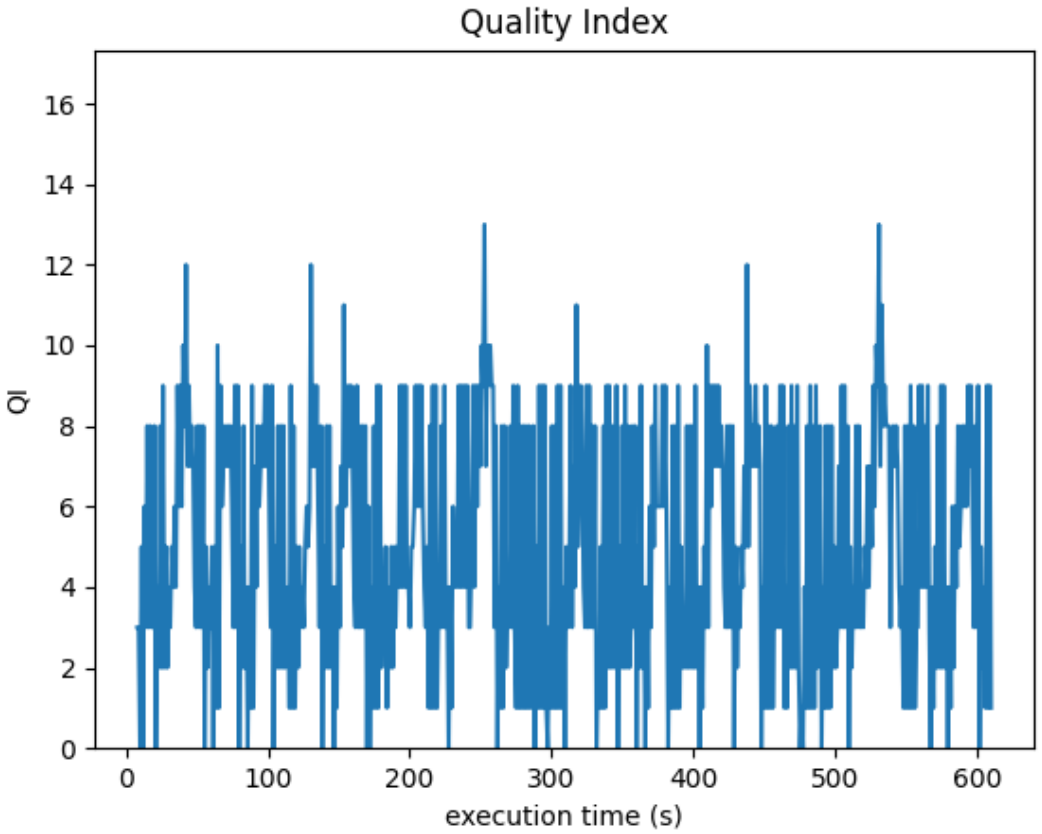
Por fim, o modelo a seguir foi onde se obteve melhores resultados com poucos travamentos e uma qualidade mediana. Além do que é mostrado na figura, cabe destacar que os valores usados para p0 e k foram 0.2 e 21, respectivamente.

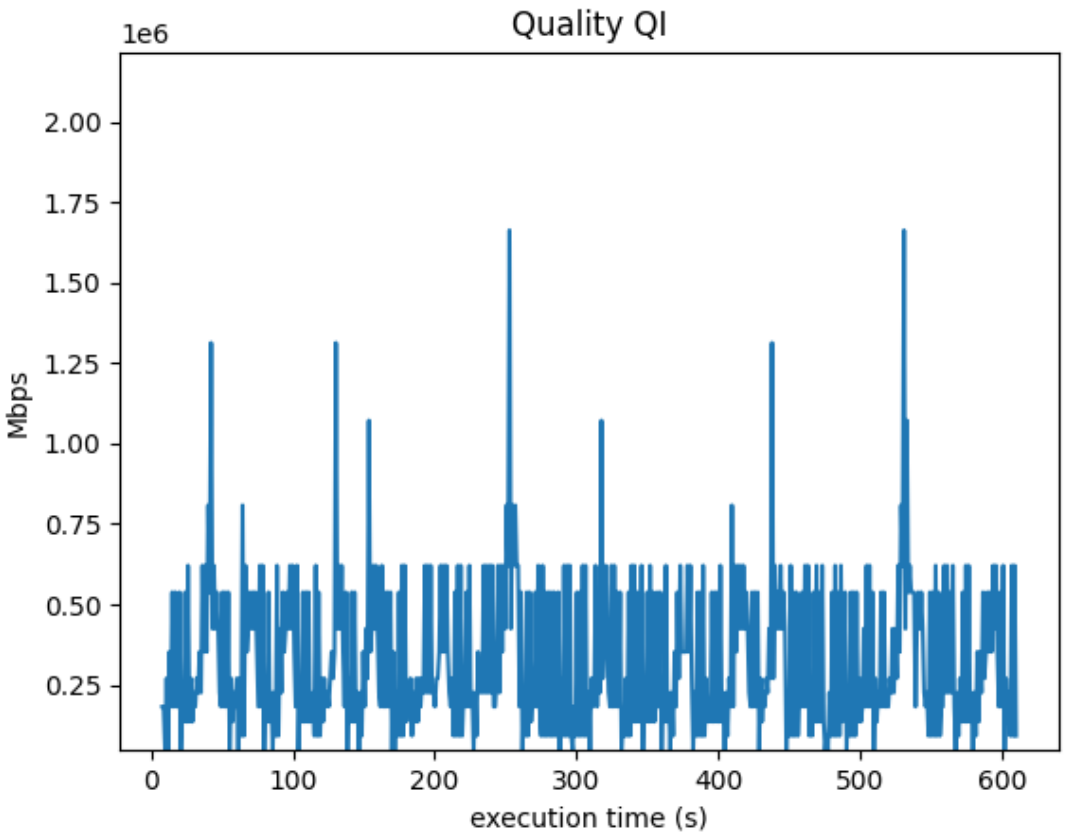
```
selected_qi = self.qi[3]

if self.p > 0.4:
    for i in self.qi:
        if (estimativa_atual*0.4) > i:
            selected_qi = i
elif self.p > 0.1 and self.p < 0.4:
    for i in self.qi:
        if restricao > i:
            selected_qi = i
else:
    for i in self.qi:
        if estimativa_atual > i:
            selected_qi = i
```

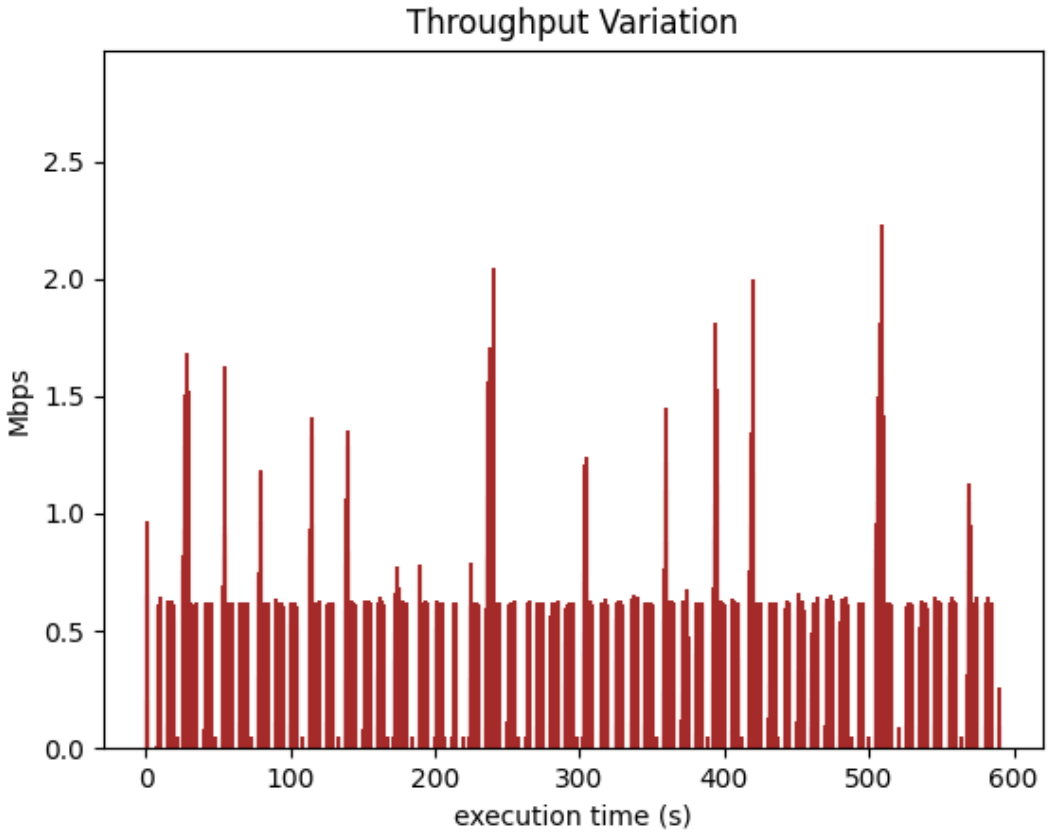
Com esse método foi possível obter uma quantidade de pausas igual a 0 para o cenário “LMH” já definido inicialmente. Além disso, as qualidades obtidas ficaram em torno de uma média consideravelmente boa.


Playback History
1.30
1.25
1.20
1.15
1.10
1.05
1.00
on/off
0
100
200
300
400
500
600
execution time (s)


Buffer Size
30
25
20
15
10
5
0
seconds
0
100
200
300
400
500
600
execution time (s)


Quality Index
QI
execution time (s)
0
2
4
6
8
10
12
14
16
0
100
200
300
400
500
600

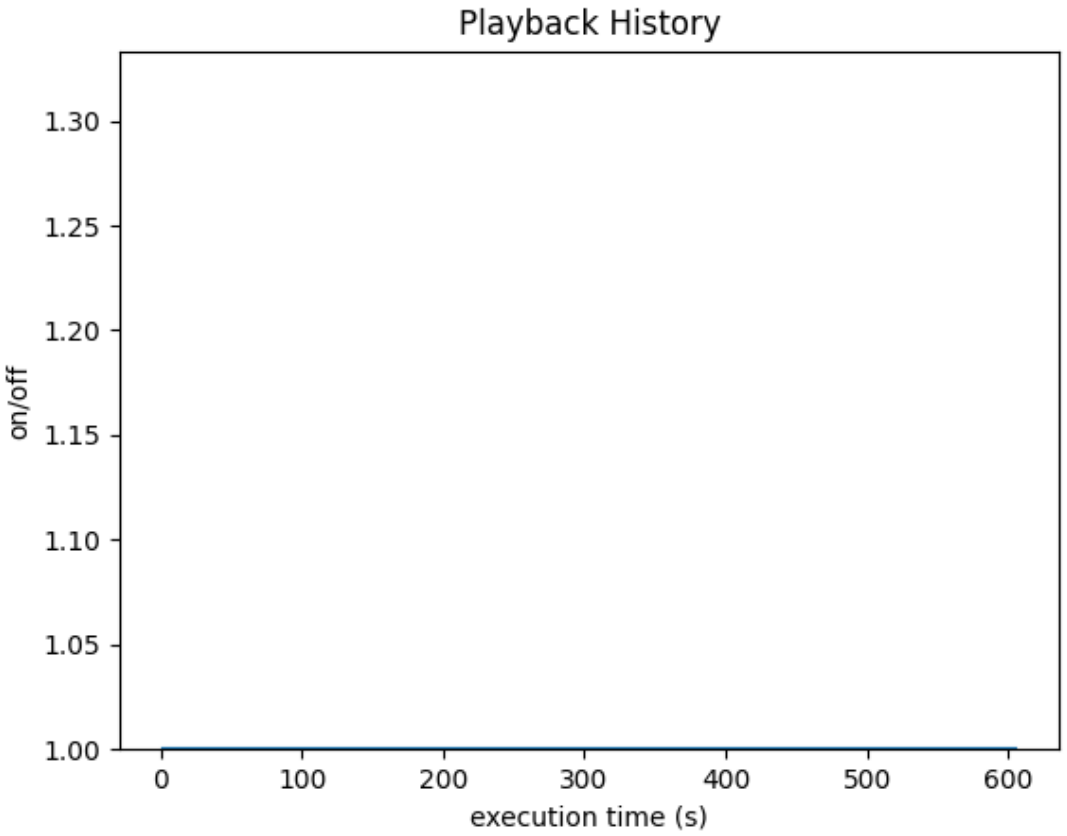

Quality QI
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Mbps
0
100
200
300
400
500
600
execution time (s)

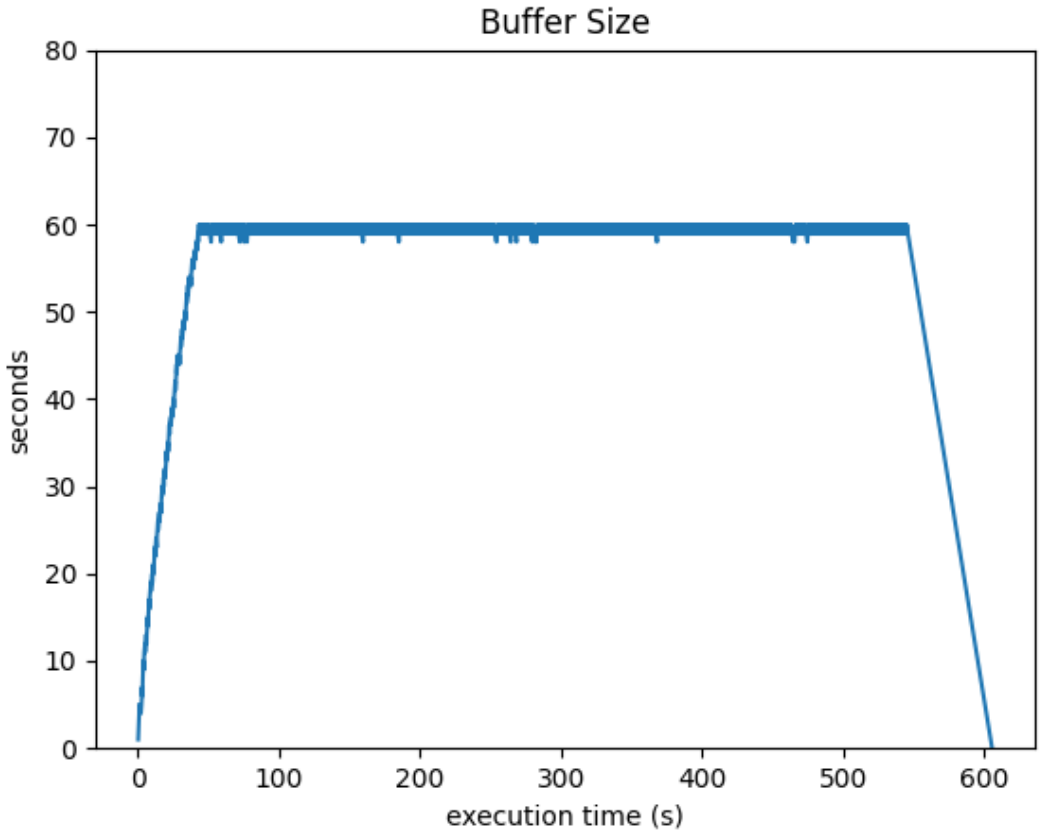

Throughput Variation
2.5
2.0
1.5
1.0
0.5
0.0
Mbps
0
100
200
300
400
500
600
execution time (s)

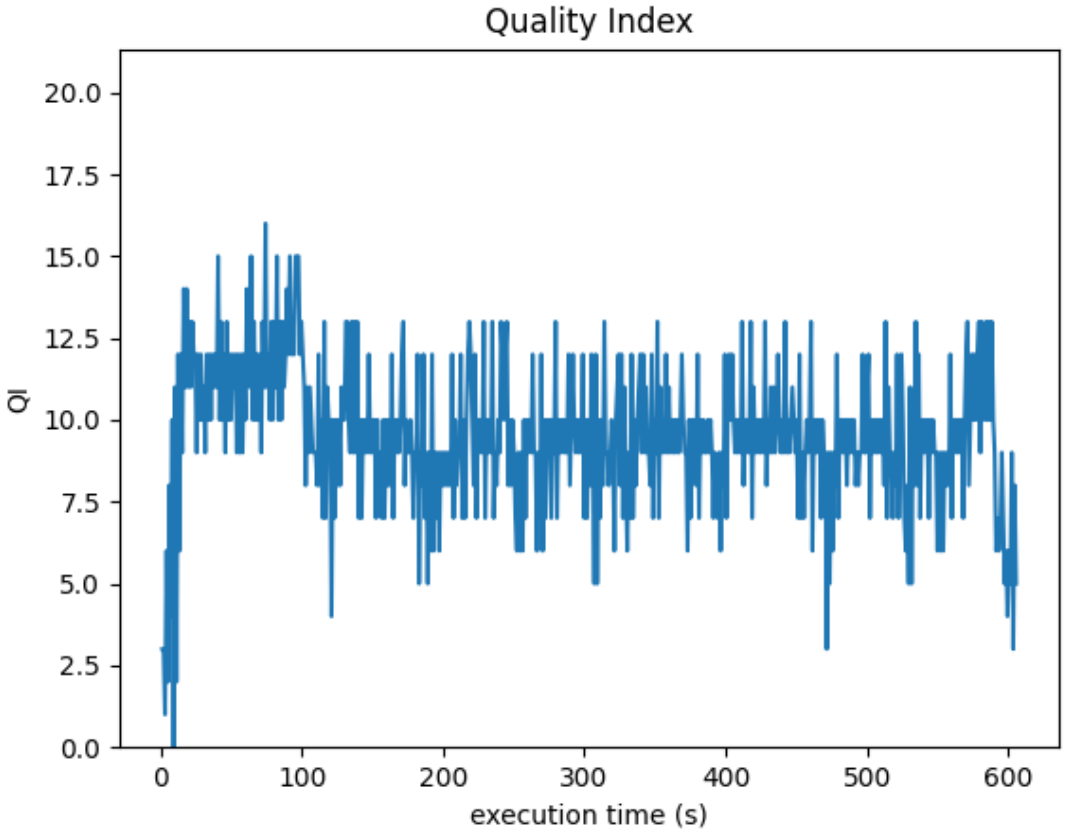
4- Cenários de teste

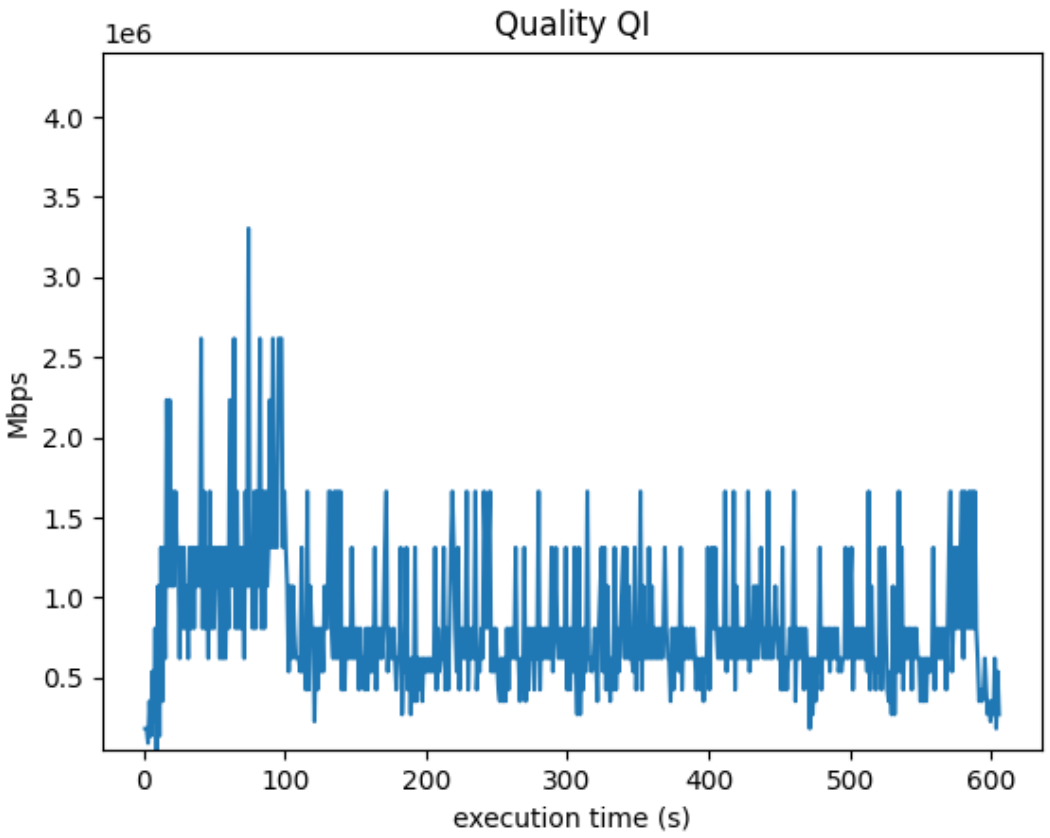
Alguns cenários foram testados a fim de se obter mais informações a respeito do desempenho do código desenvolvido.

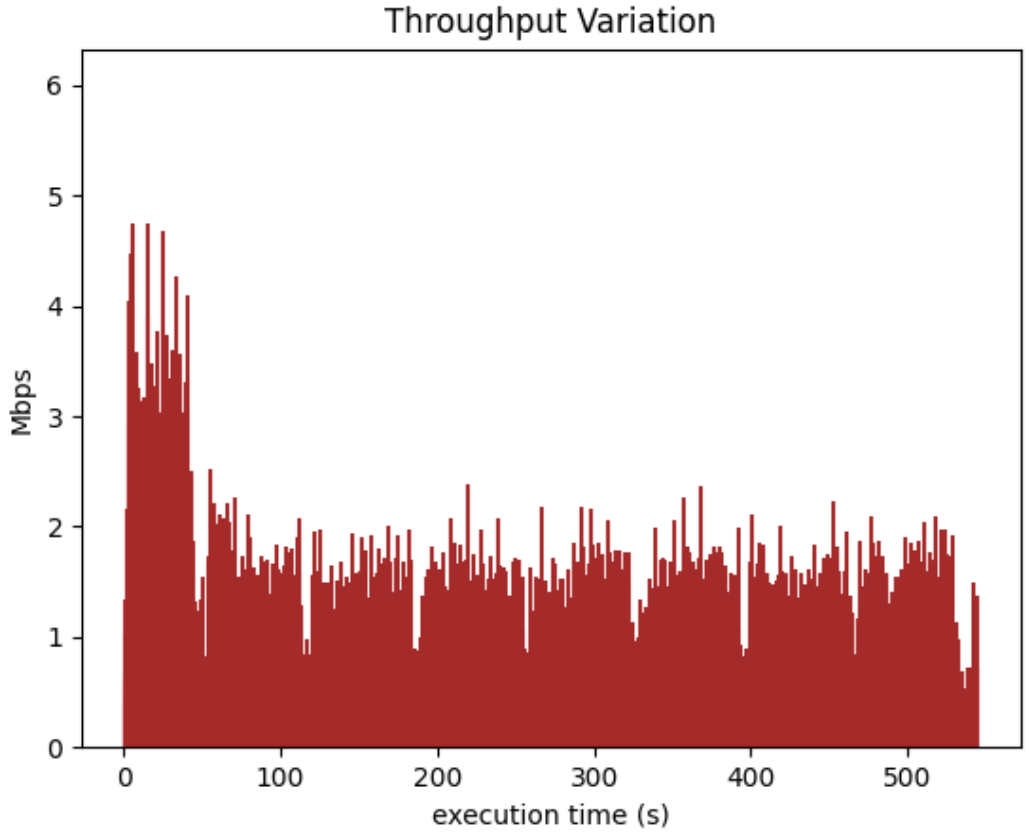
4.1- Cenário “LLL”



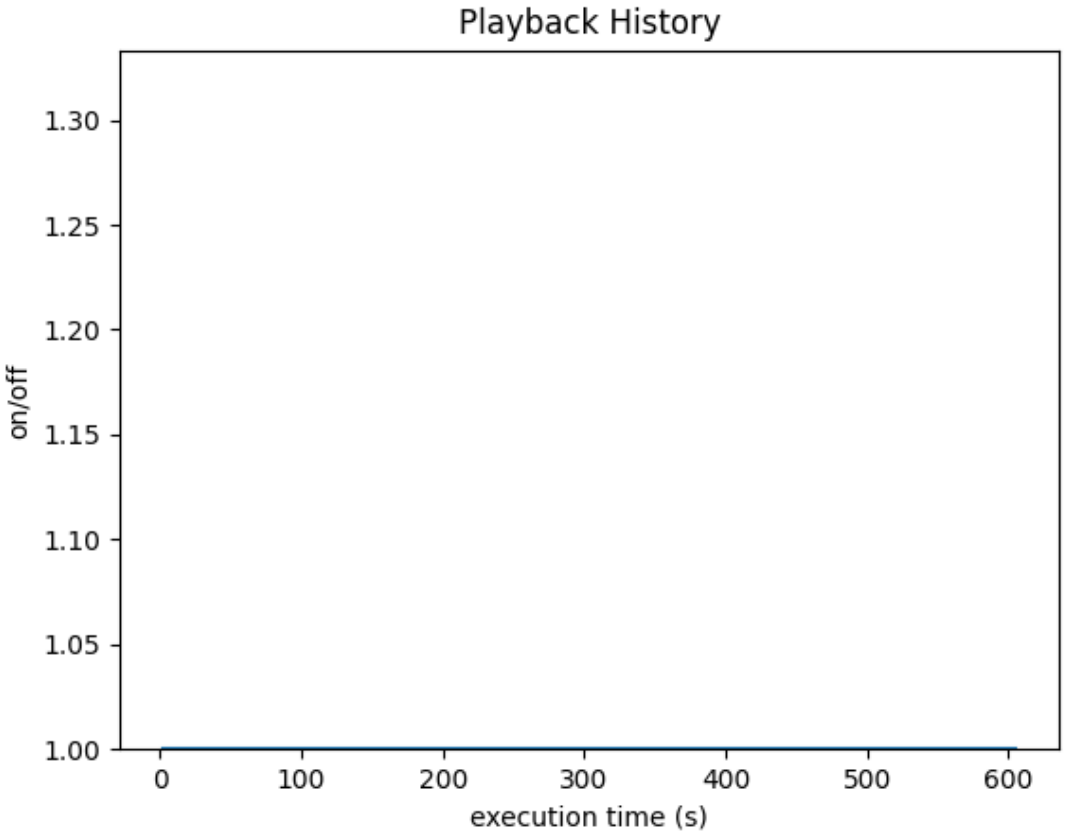

Buffer Size
seconds
execution time (s)
0
10
20
30
40
50
60
70
80
0
100
200
300
400
500
600

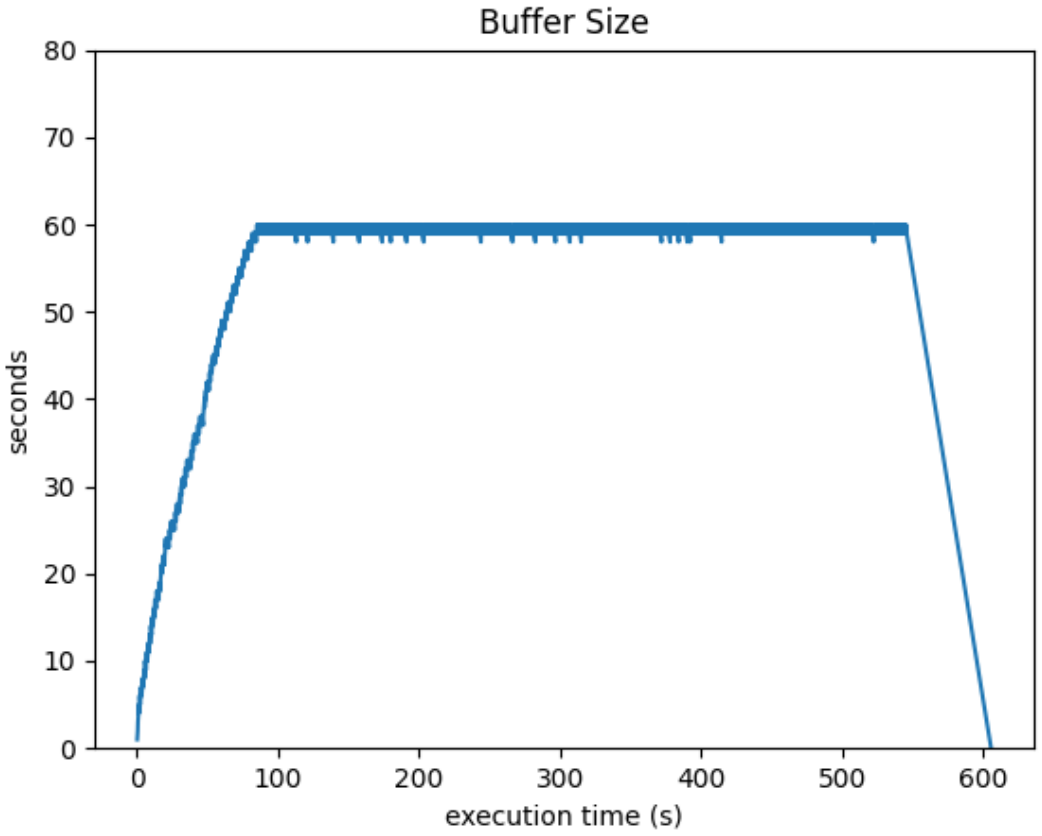

Quality Index
QI
execution time (s)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
100
200
300
400
500
600

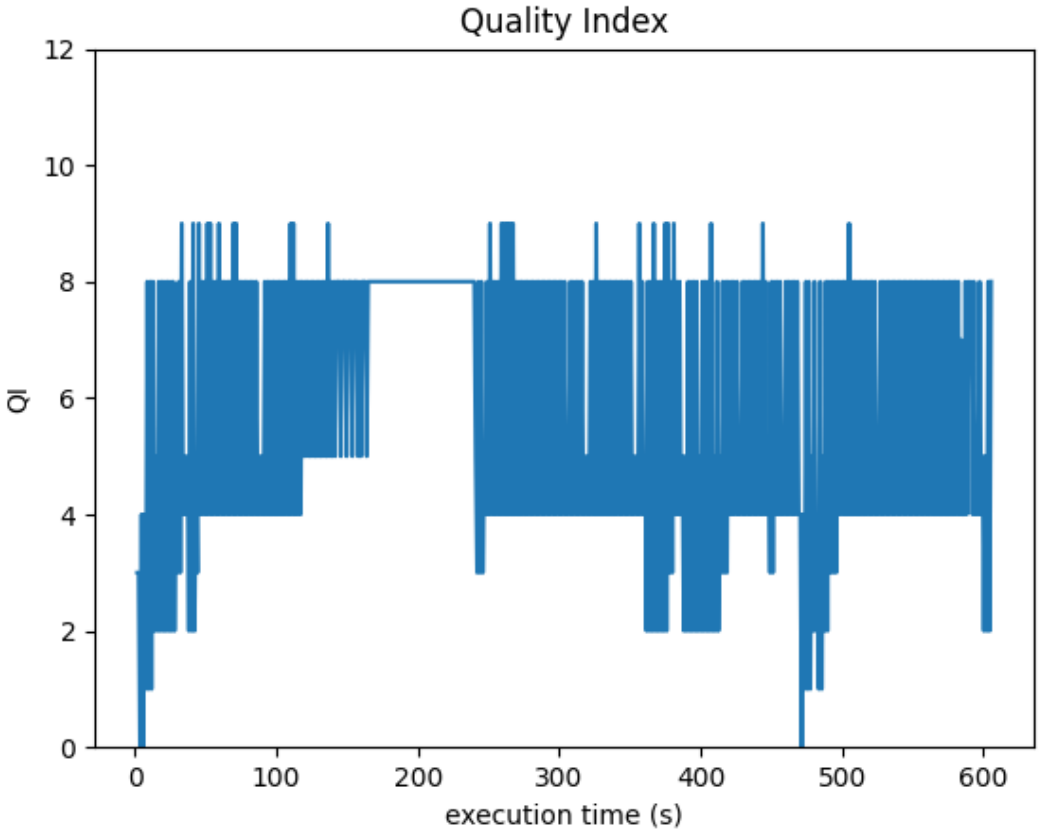

Quality QI
1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
Mbps
0
100
200
300
400
500
600
execution time (s)

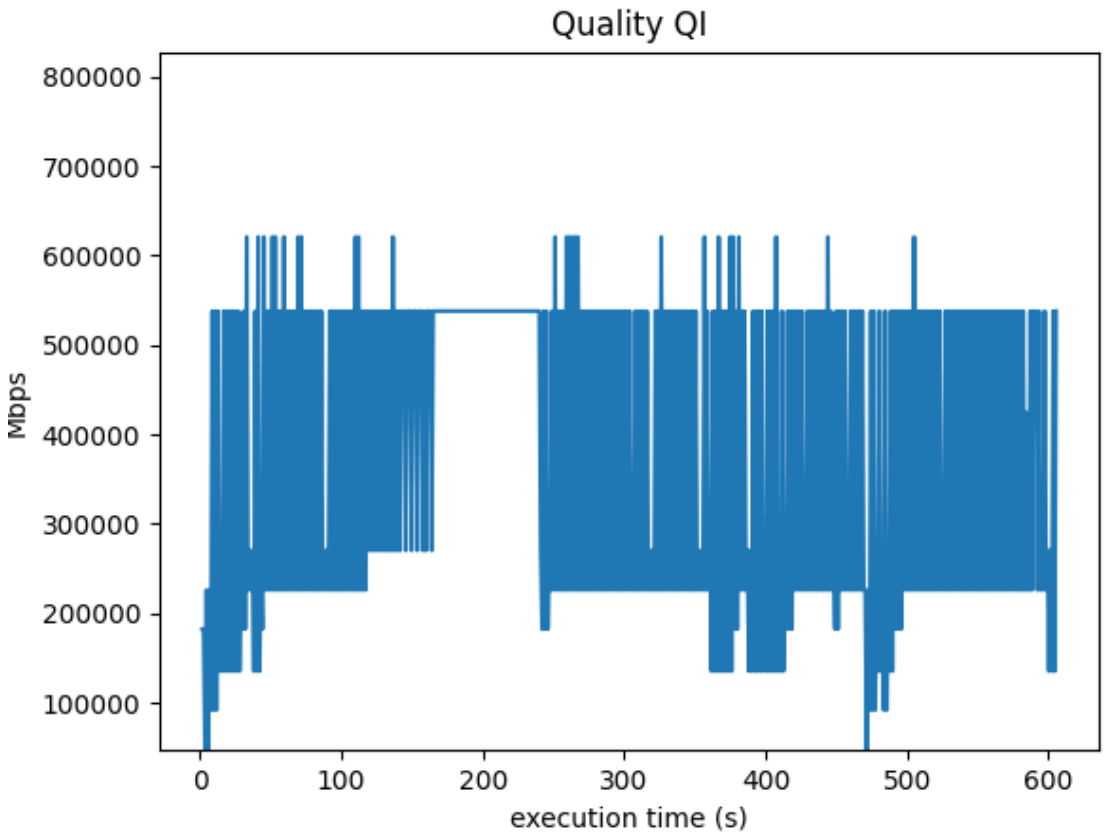

Throughput Variation
6
5
4
3
2
1
0
Mbps
0
100
200
300
400
500
execution time (s)

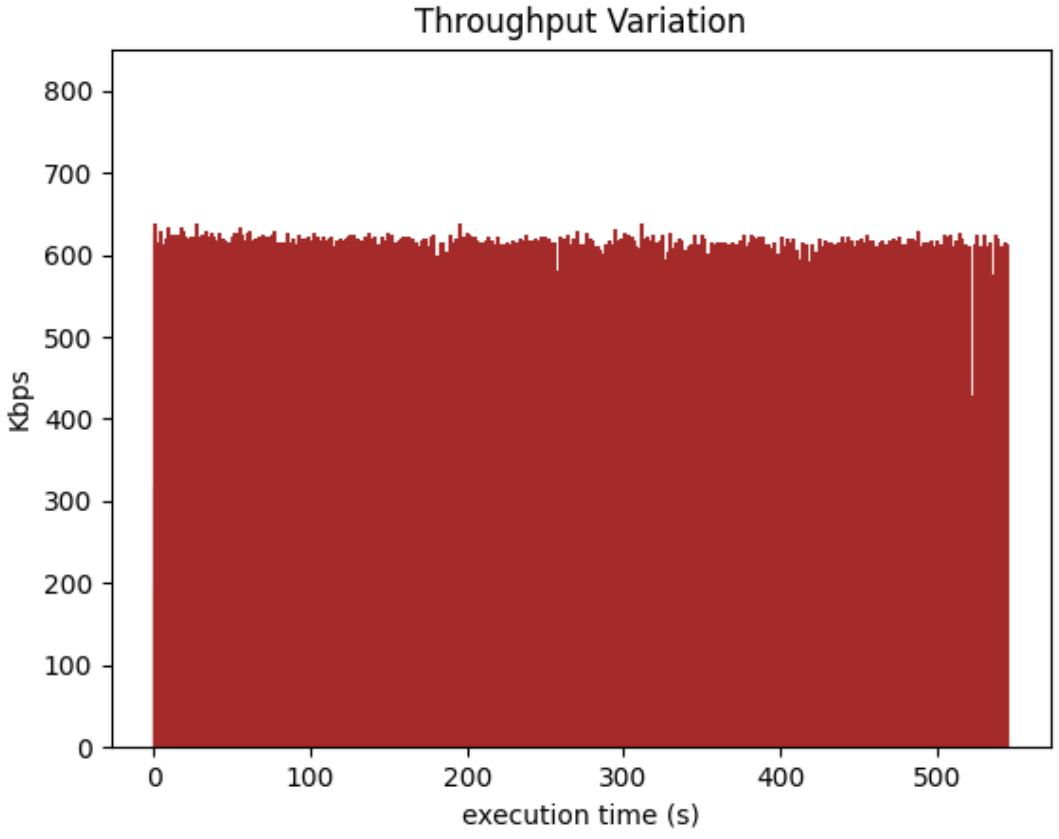
4.2- Cenário “MMM”



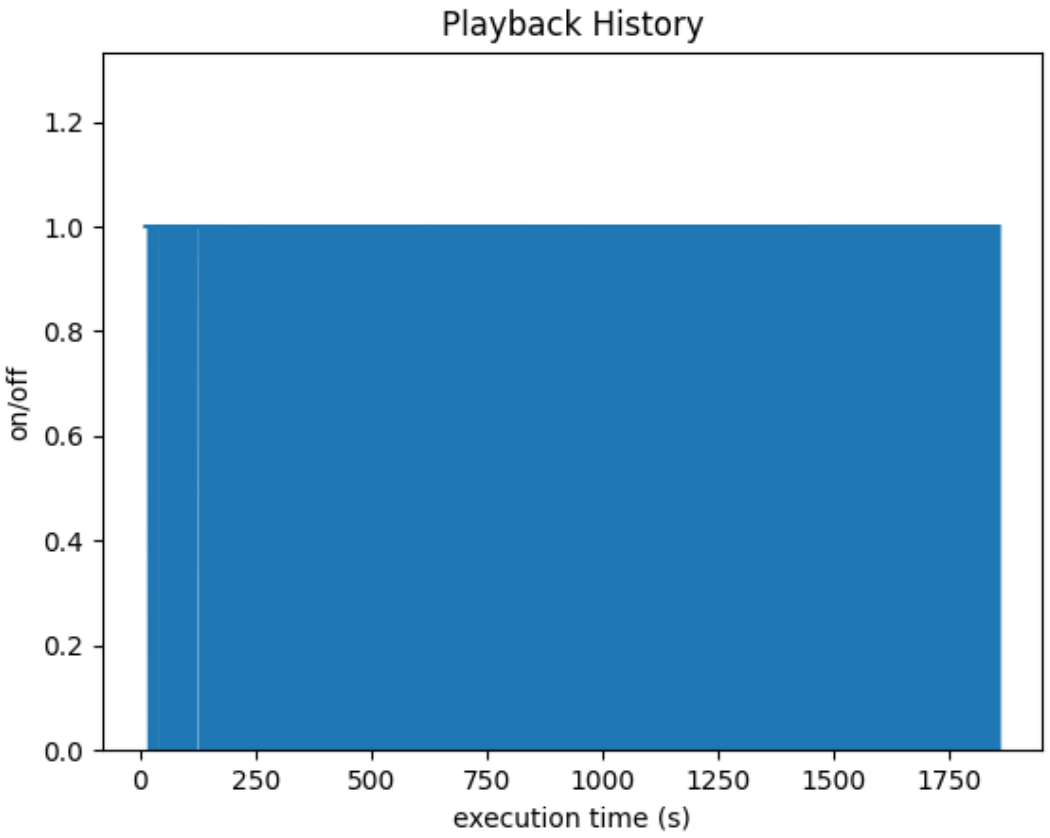

Buffer Size
seconds
execution time (s)
0
10
20
30
40
50
60
70
80
0
100
200
300
400
500
600

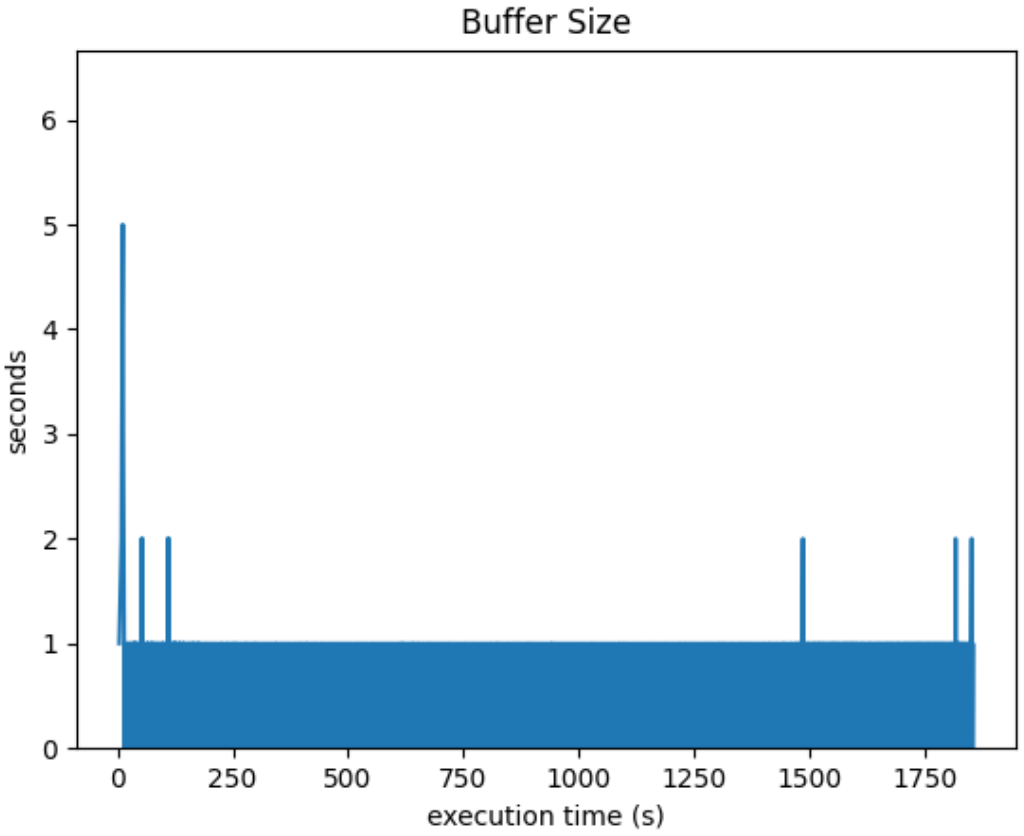

Quality Index
QI
execution time (s)
0
2
4
6
8
10
12
0
100
200
300
400
500
600

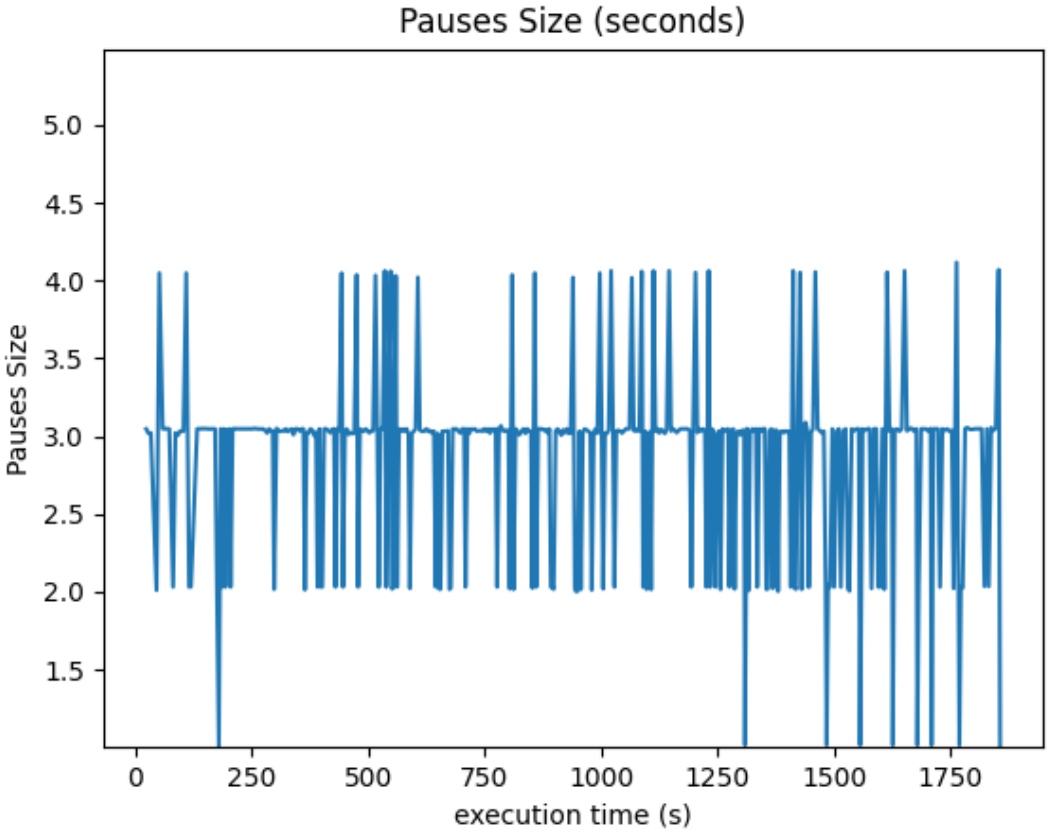

Quality QI
800000
700000
600000
500000
400000
300000
200000
100000
Mbps
0
100
200
300
400
500
600
execution time (s)

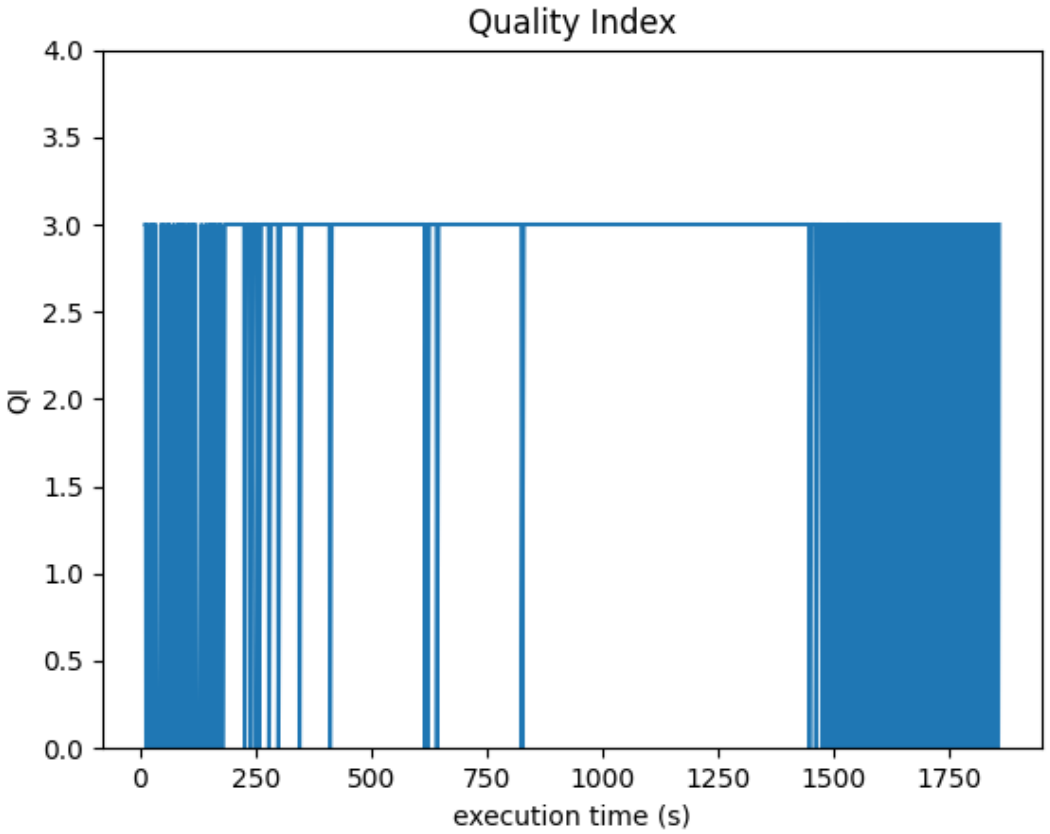

Throughput Variation
800
700
600
500
400
300
200
100
0
Kbps
0
100
200
300
400
500
execution time (s)

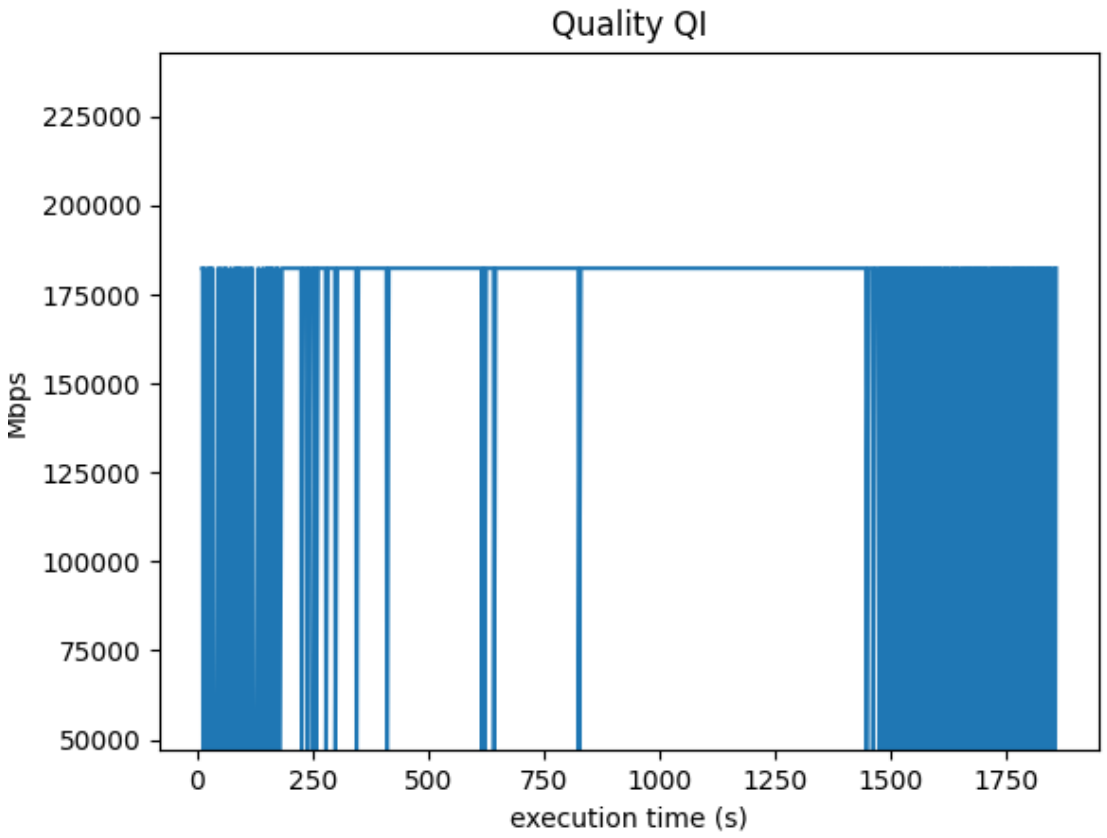
4.3- Cenário “HHH”

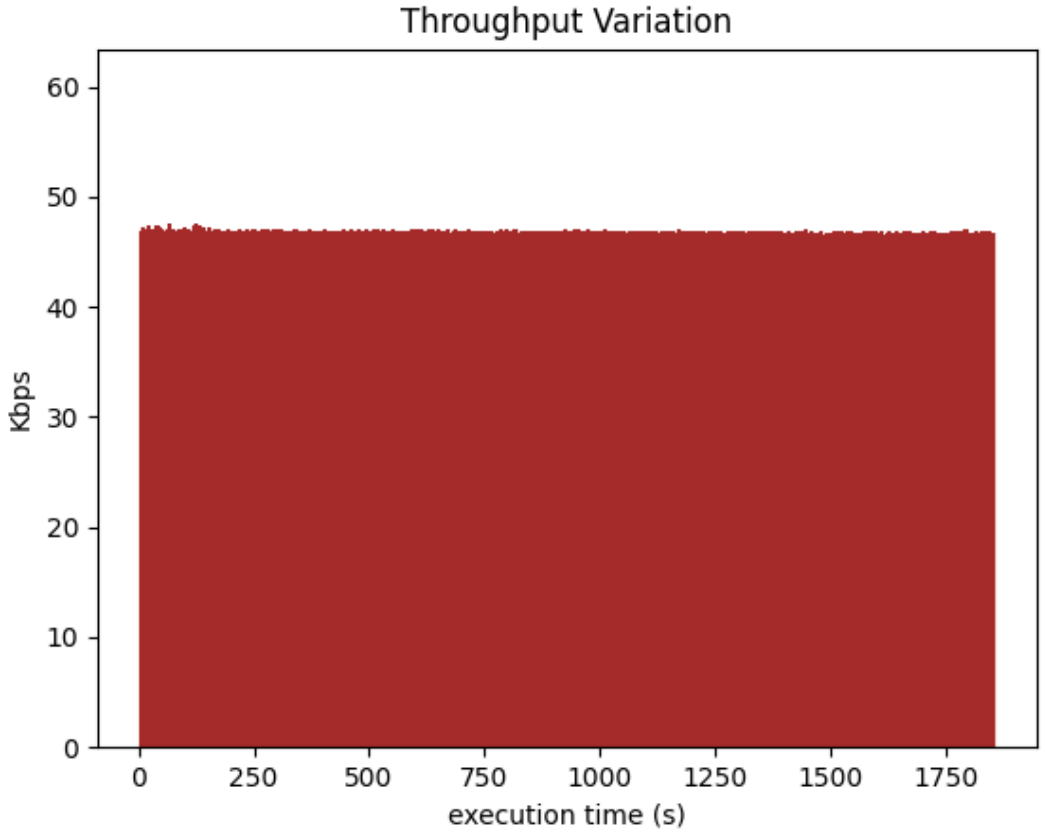



Buffer Size
seconds
0
1
2
3
4
5
6
0
250
500
750
1000
1250
1500
1750
execution time (s)

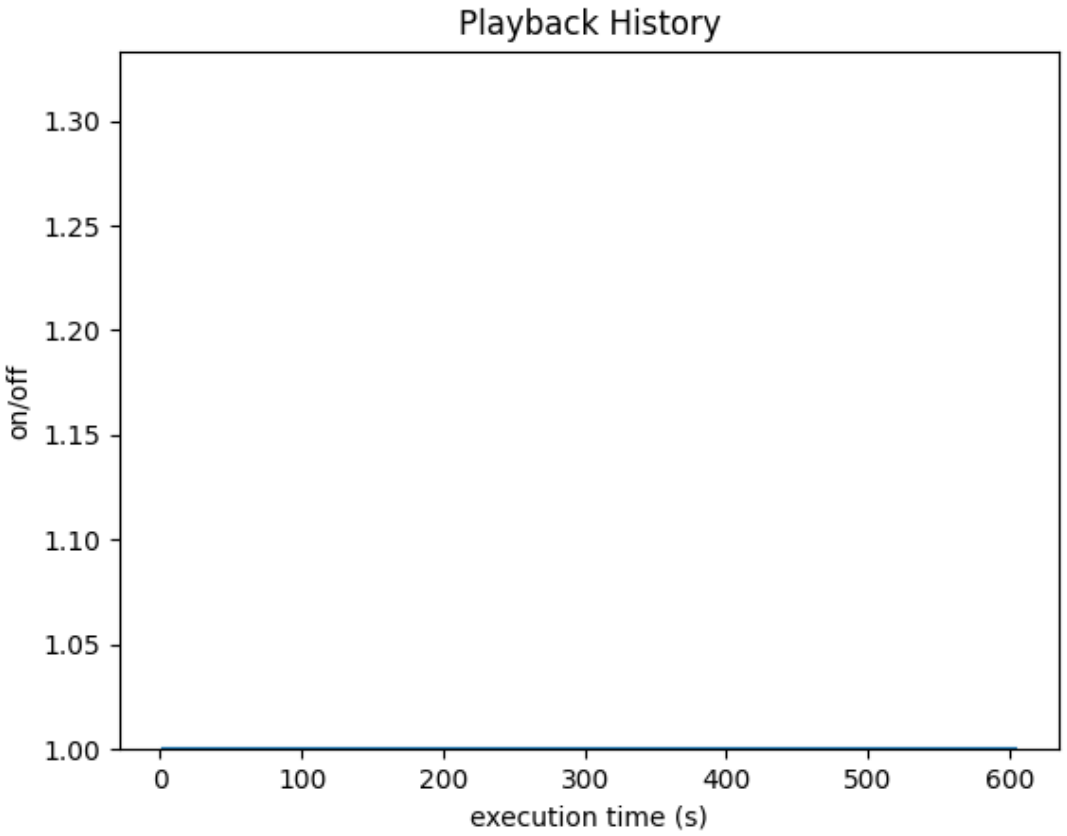

Pauses Size (seconds)
Pauses Size
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
0
250
500
750
1000
1250
1500
1750
execution time (s)

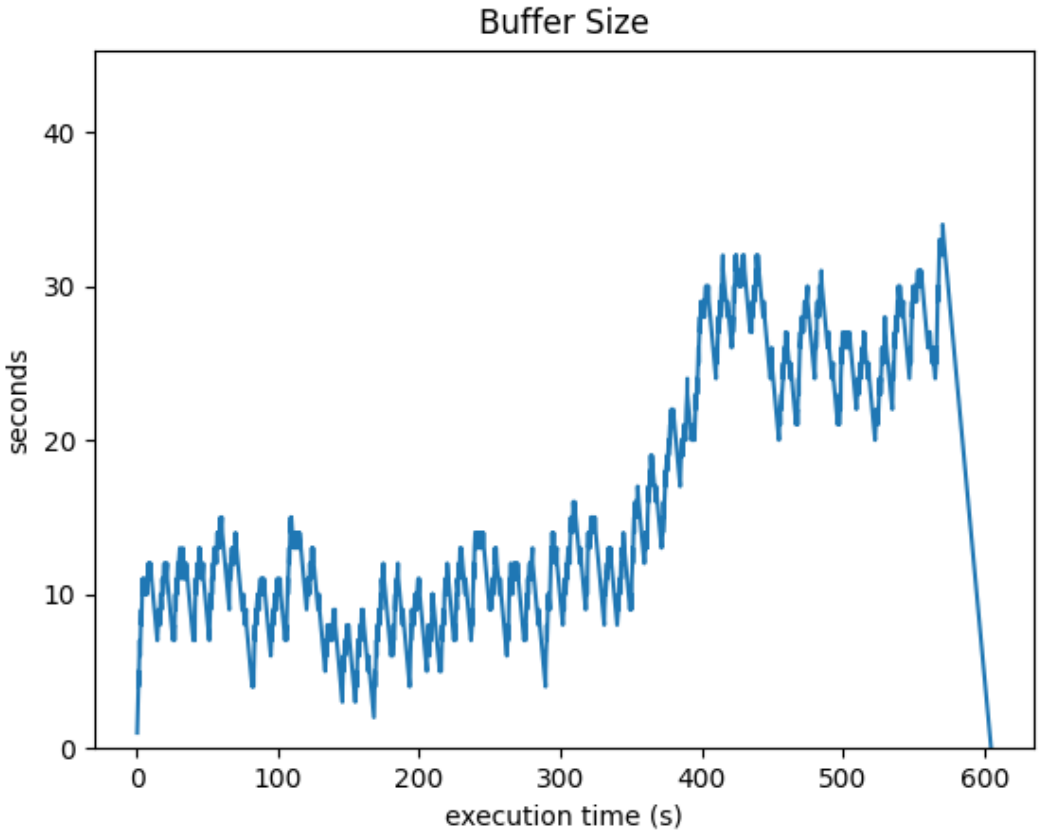

Quality Index
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
QI
0
250
500
750
1000
1250
1500
1750
execution time (s)

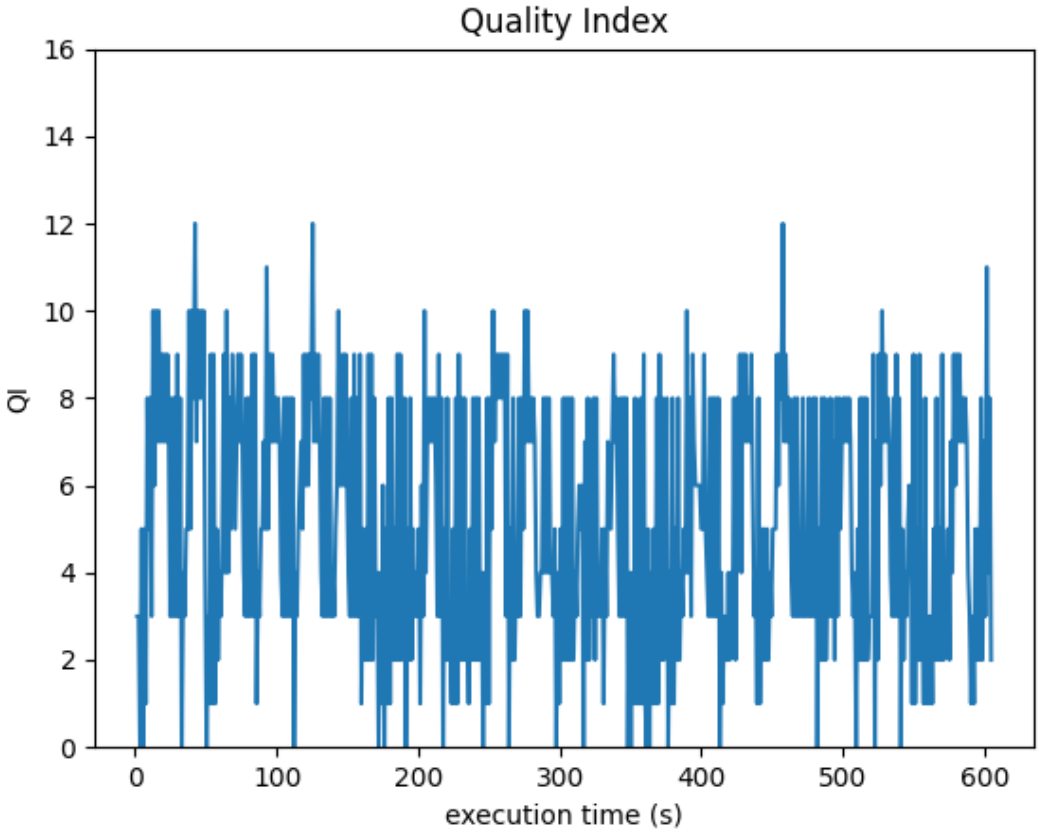

Quality QI
225000
200000
175000
150000
125000
100000
75000
50000
Mbps
0
250
500
750
1000
1250
1500
1750
execution time (s)

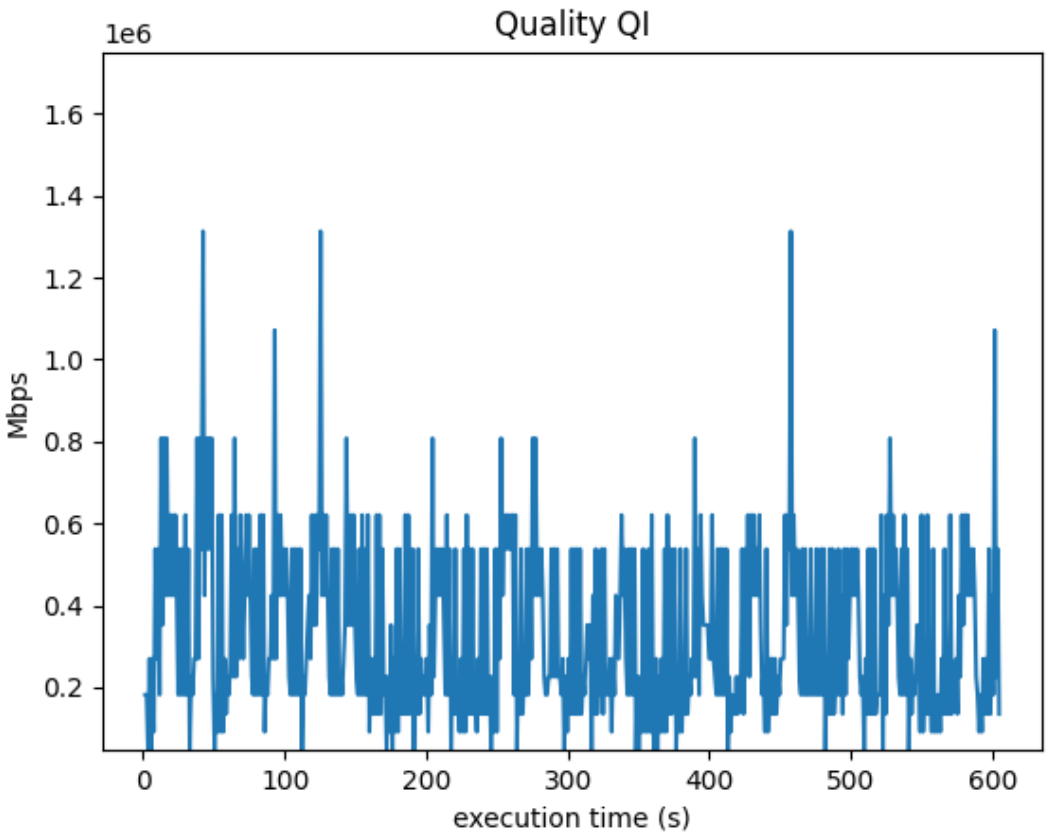

Throughput Variation
60
50
40
30
20
10
0
Kbps
0
250
500
750
1000
1250
1500
1750
execution time (s)

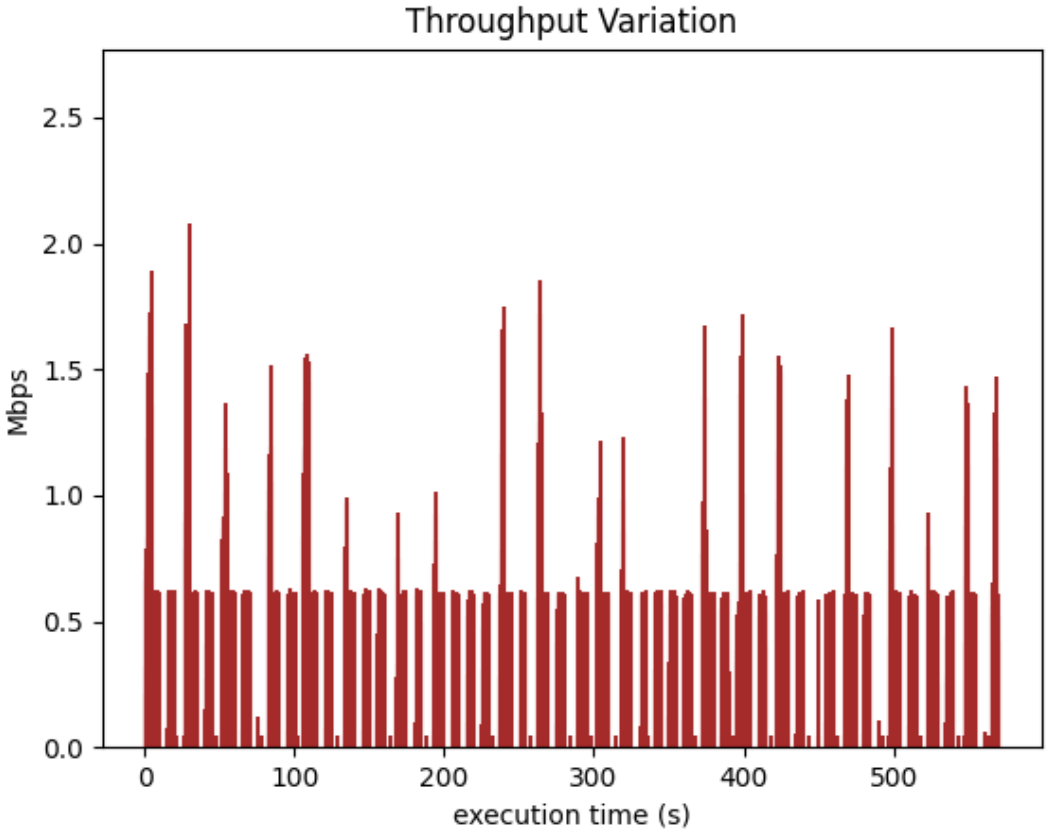
4.4- Cenário “LMH”



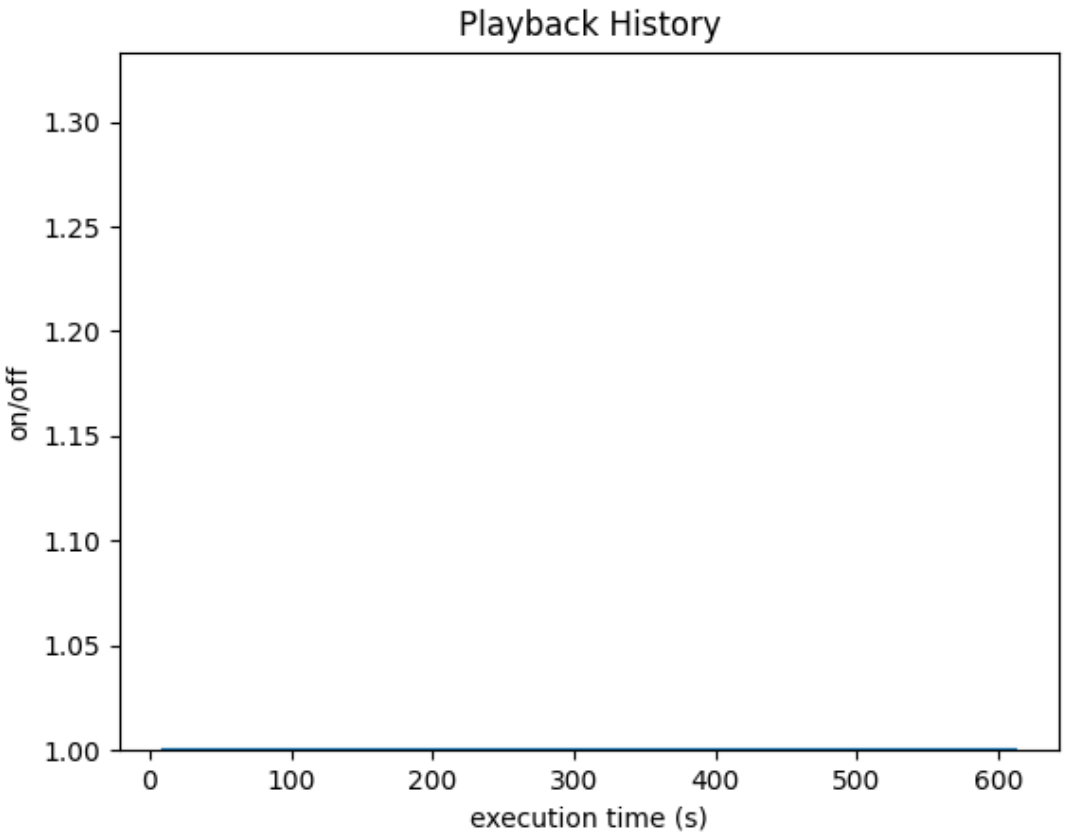

Buffer Size
seconds
0
10
20
30
40
0
100
200
300
400
500
600
execution time (s)

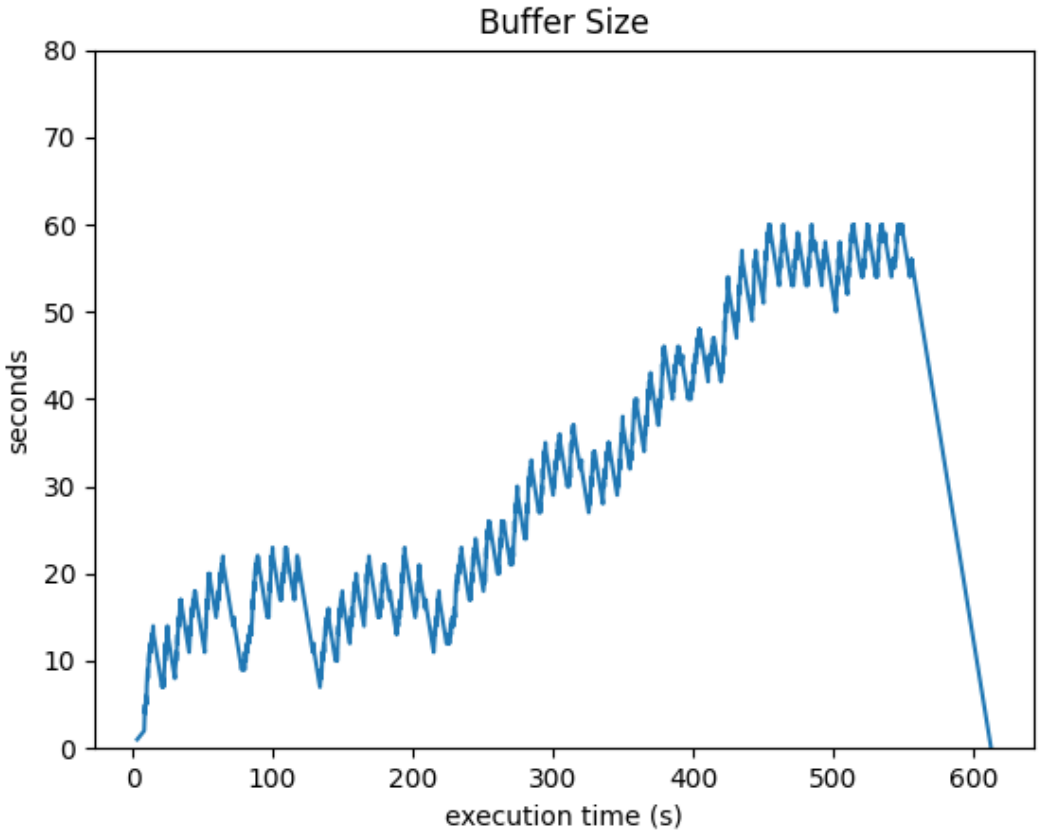

Quality Index
QI
0
2
4
6
8
10
12
14
16
0
100
200
300
400
500
600
execution time (s)

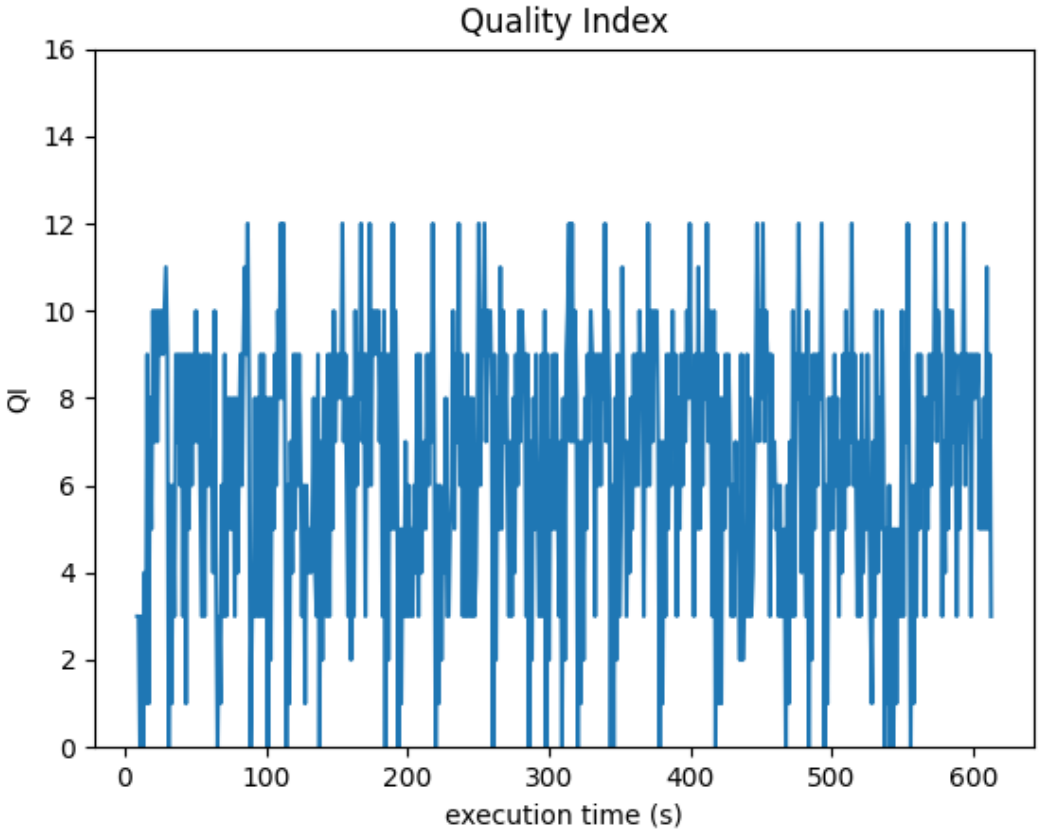

Quality QI
1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
Mbps
0
100
200
300
400
500
600
execution time (s)

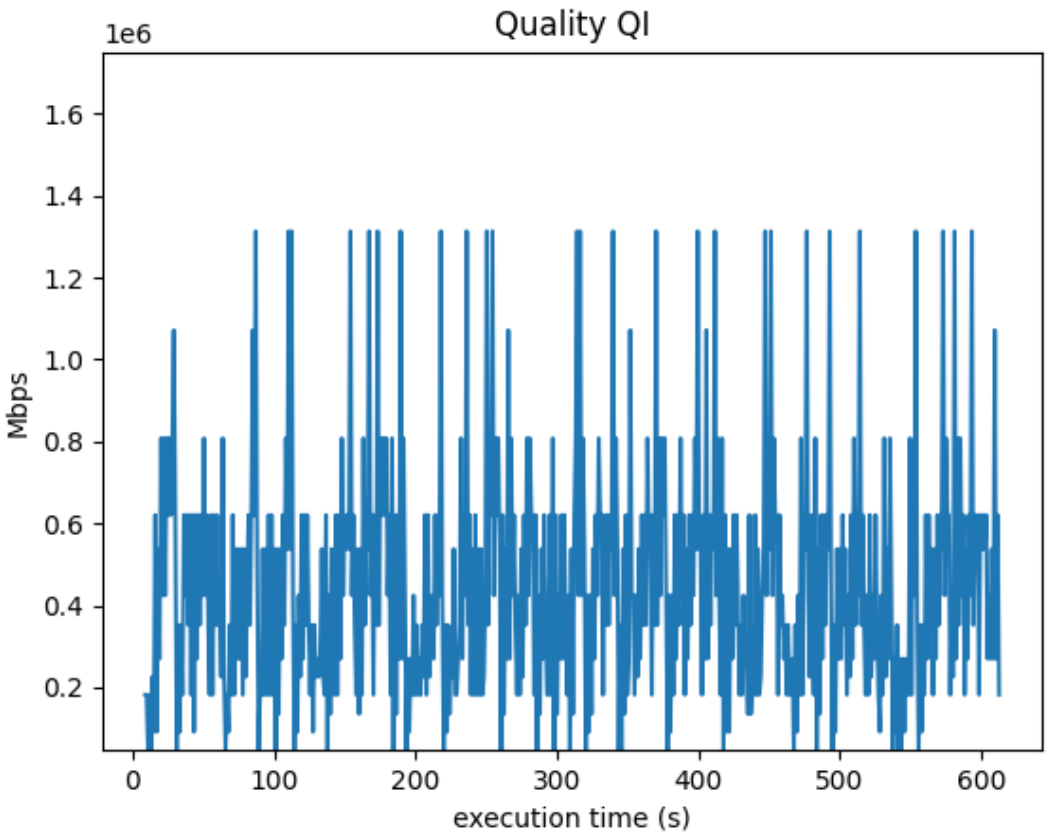

Throughput Variation
2.5
2.0
1.5
1.0
0.5
0.0
Mbps
0
100
200
300
400
500
execution time (s)

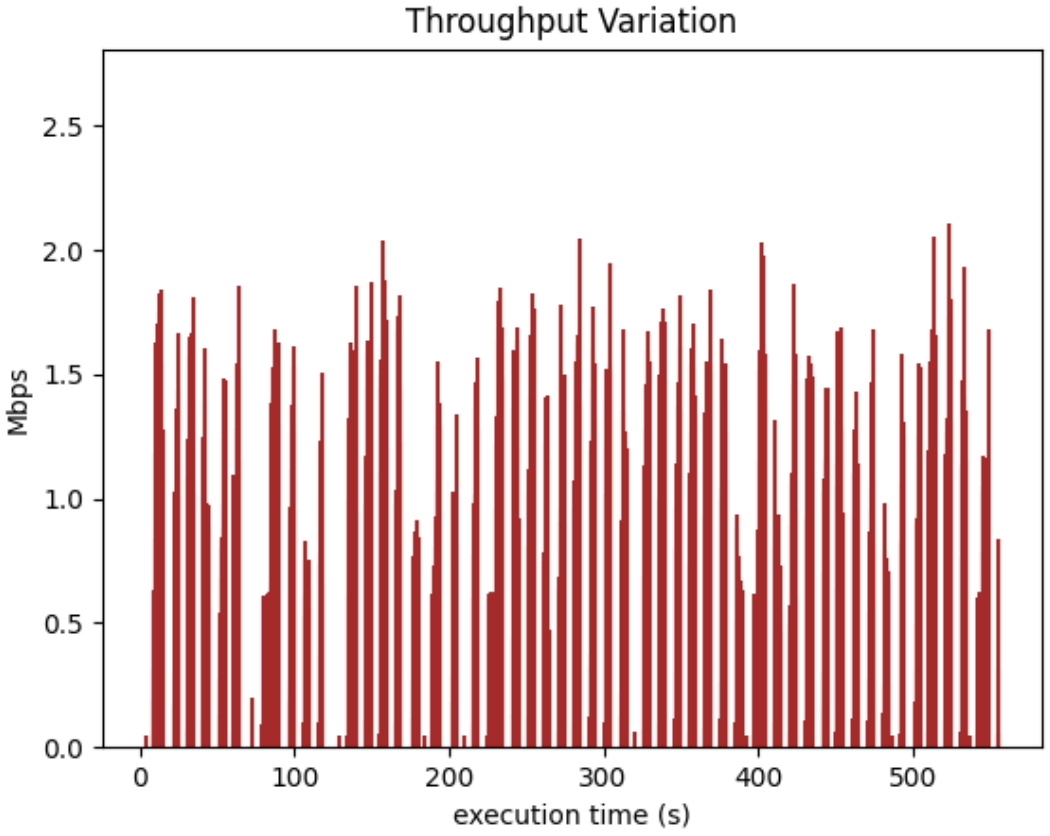
4.5- Cenário “HML”



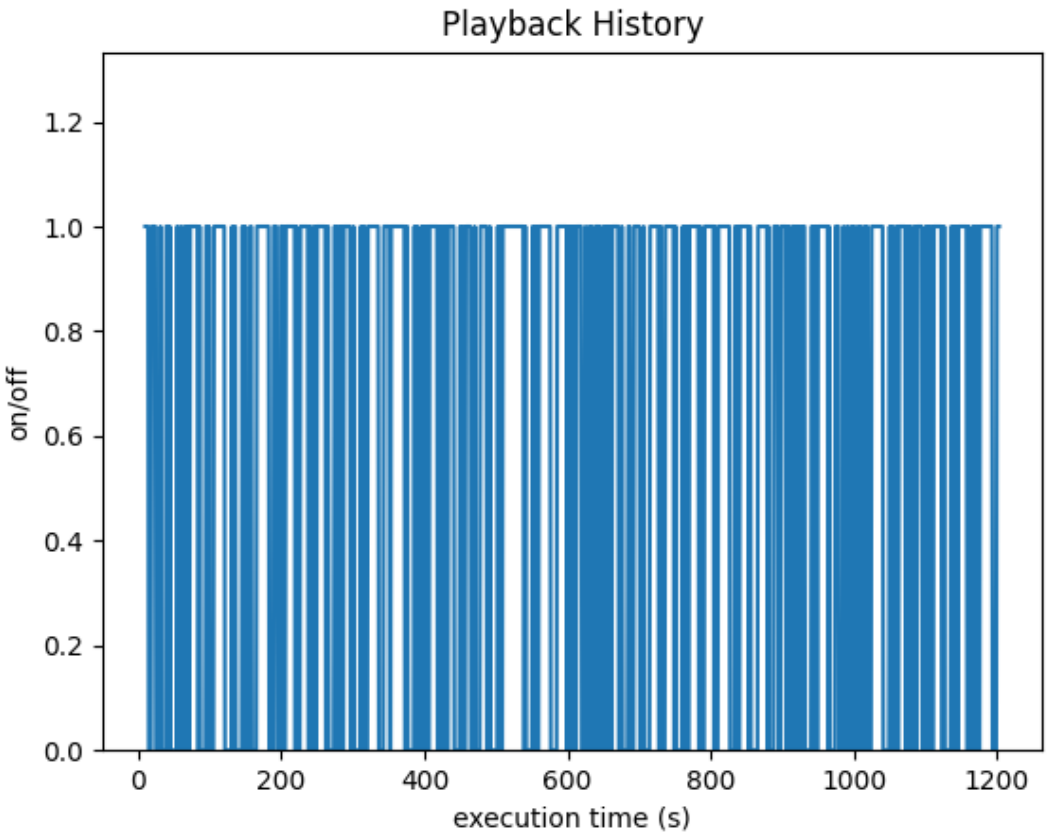

Buffer Size
seconds
execution time (s)
0
10
20
30
40
50
60
70
80
0
100
200
300
400
500
600

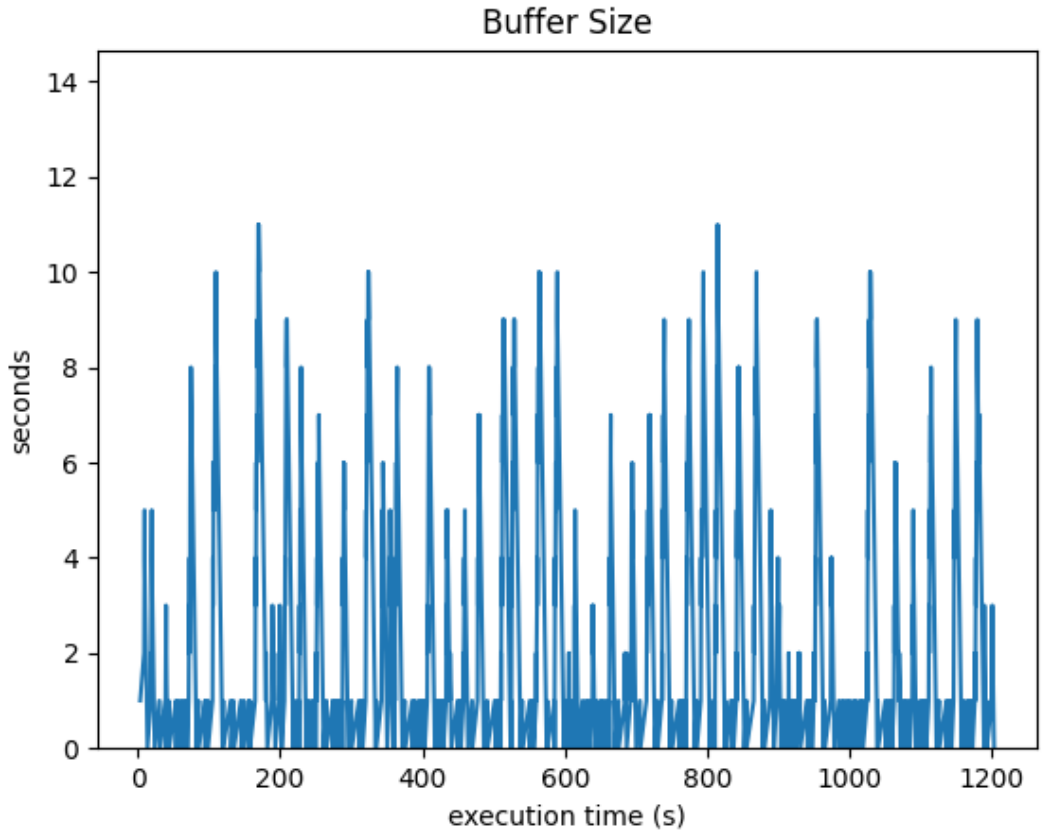

Quality Index
QI
execution time (s)
0
2
4
6
8
10
12
14
16
0
100
200
300
400
500
600

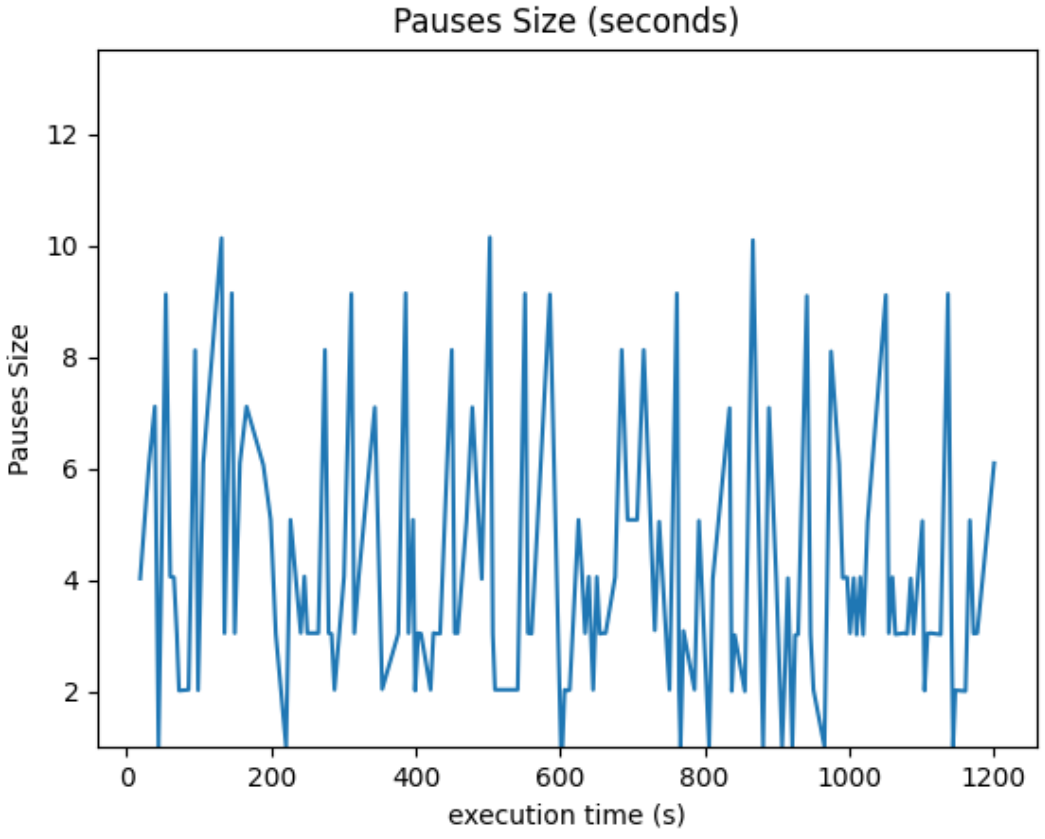

Quality QI
1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
Mbps
0
100
200
300
400
500
600
execution time (s)

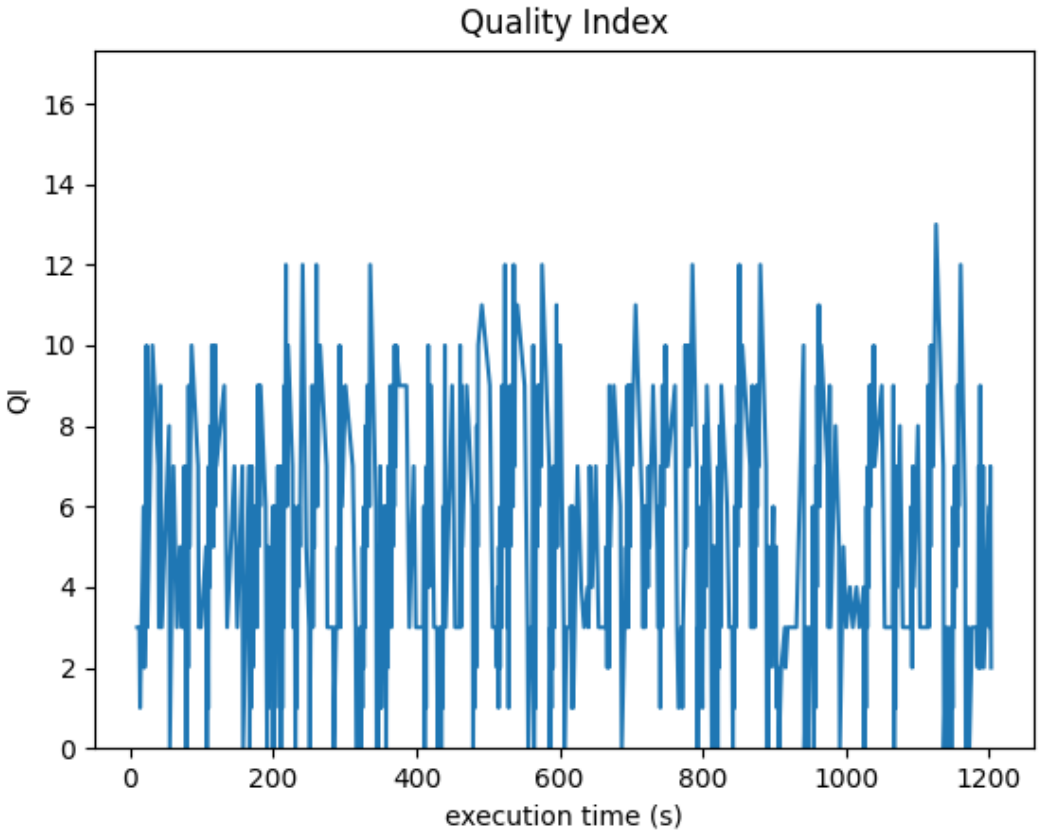

Throughput Variation
2.5
2.0
1.5
1.0
0.5
0.0
Mbps
0
100
200
300
400
500
execution time (s)

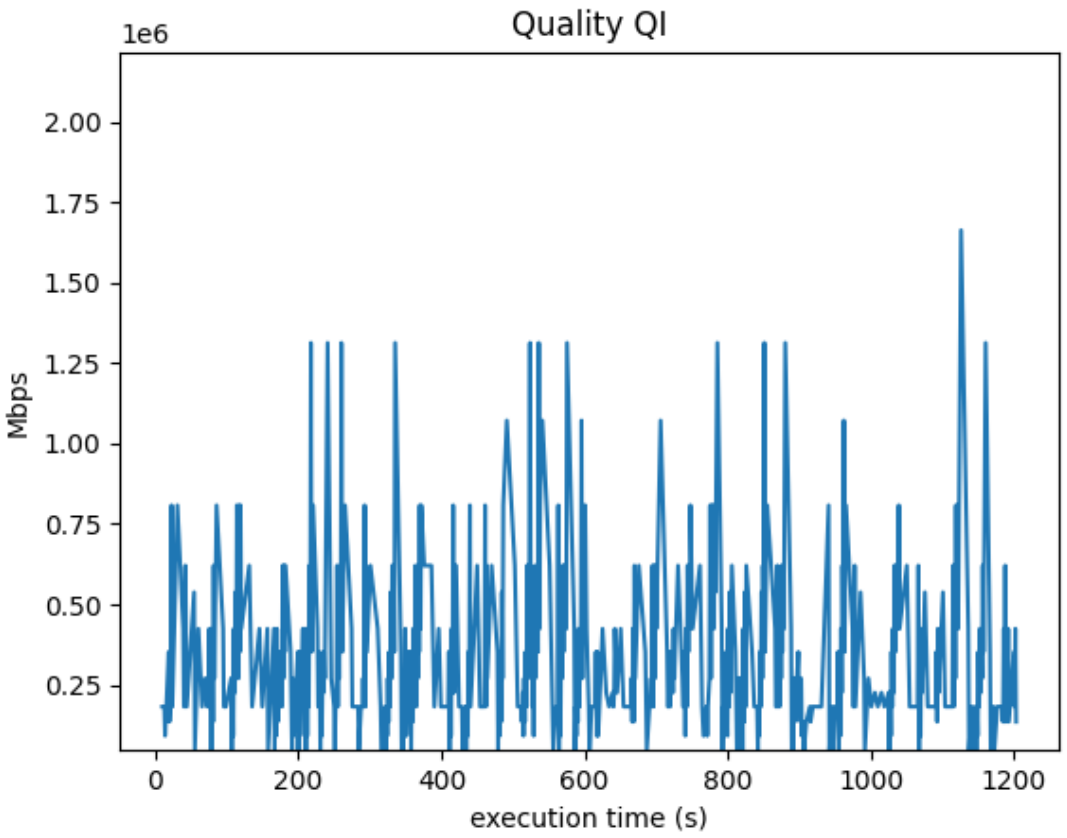
4.6- Cenário “HLH”

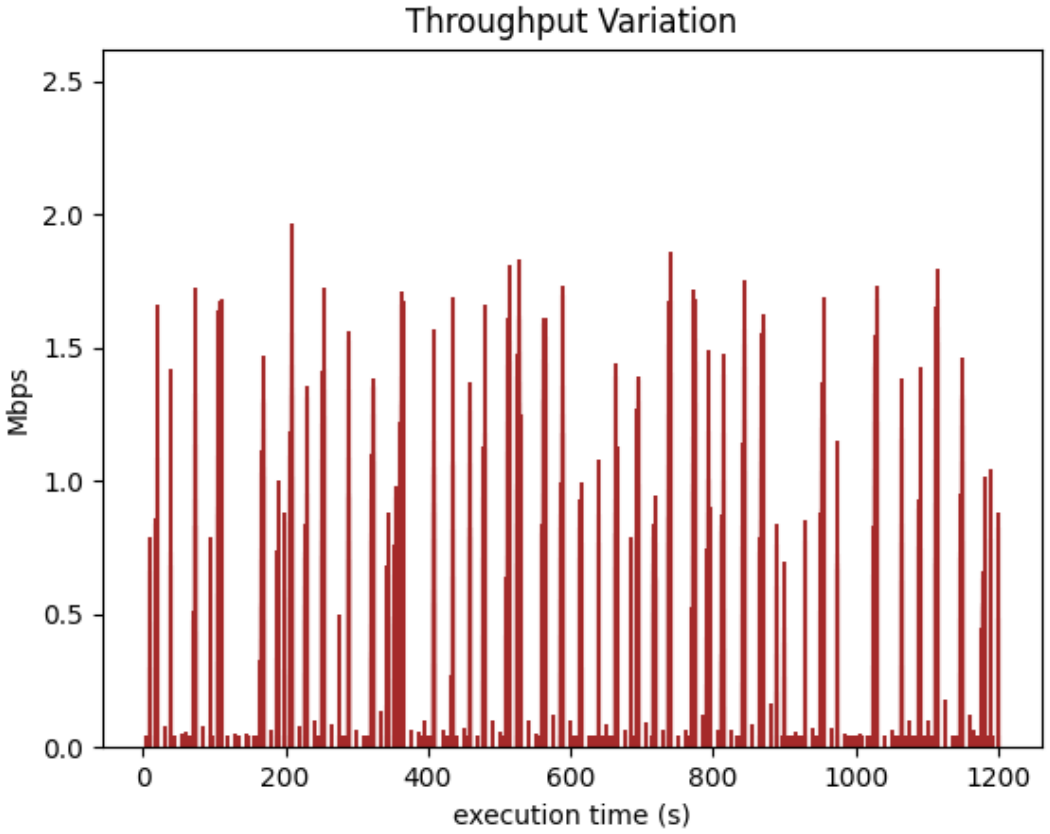



Buffer Size
seconds
execution time (s)
0
2
4
6
8
10
12
14
0
200
400
600
800
1000
1200

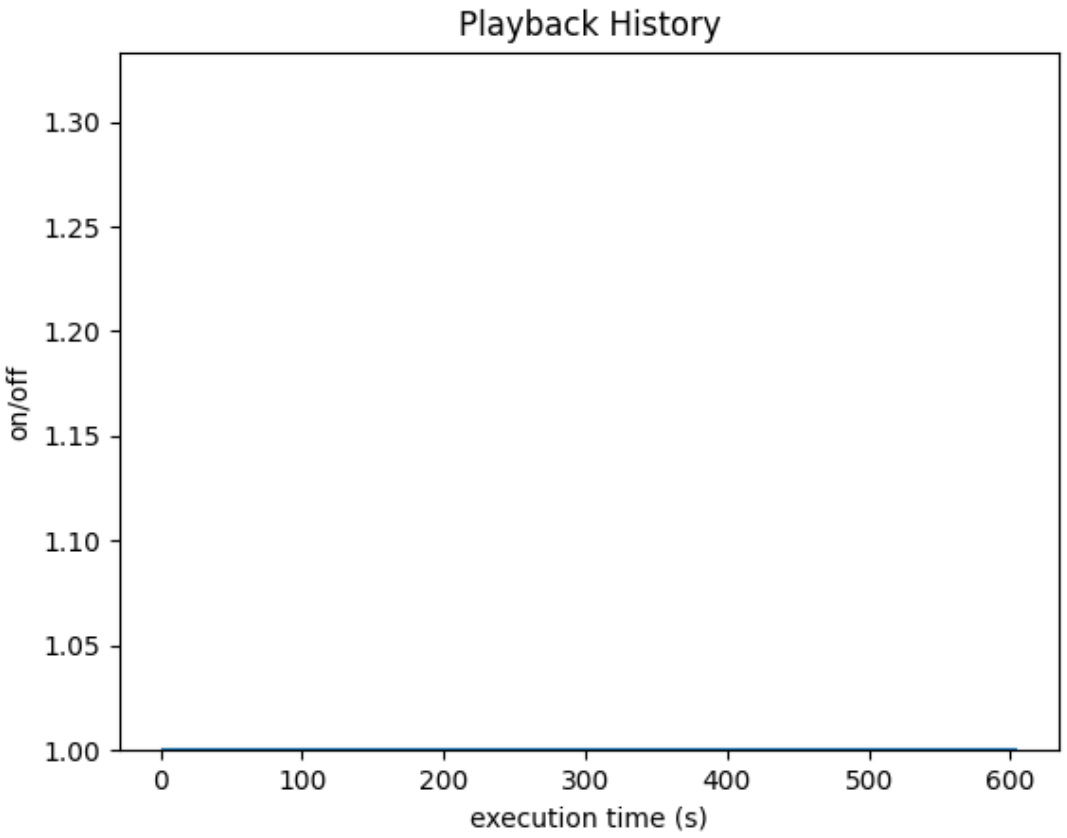

Pauses Size (seconds)
Pauses Size
execution time (s)
2
4
6
8
10
12
0
200
400
600
800
1000
1200

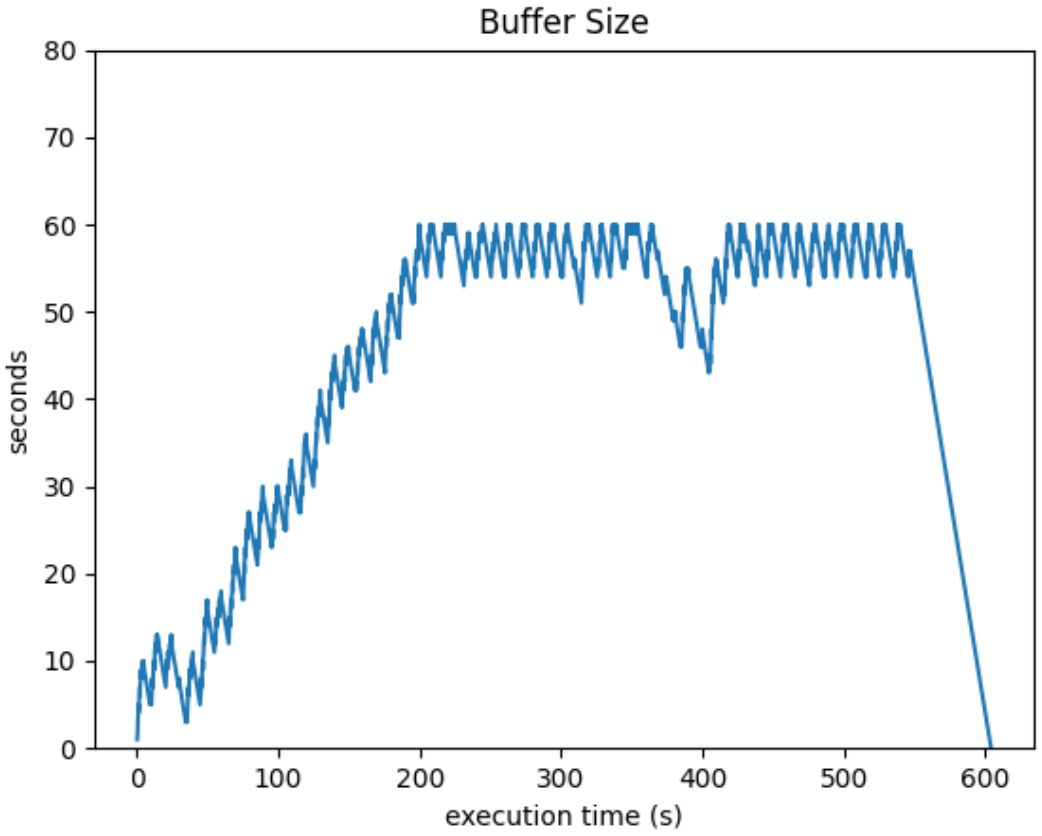

Quality Index
QI
execution time (s)
0
2
4
6
8
10
12
14
16
0
200
400
600
800
1000
1200

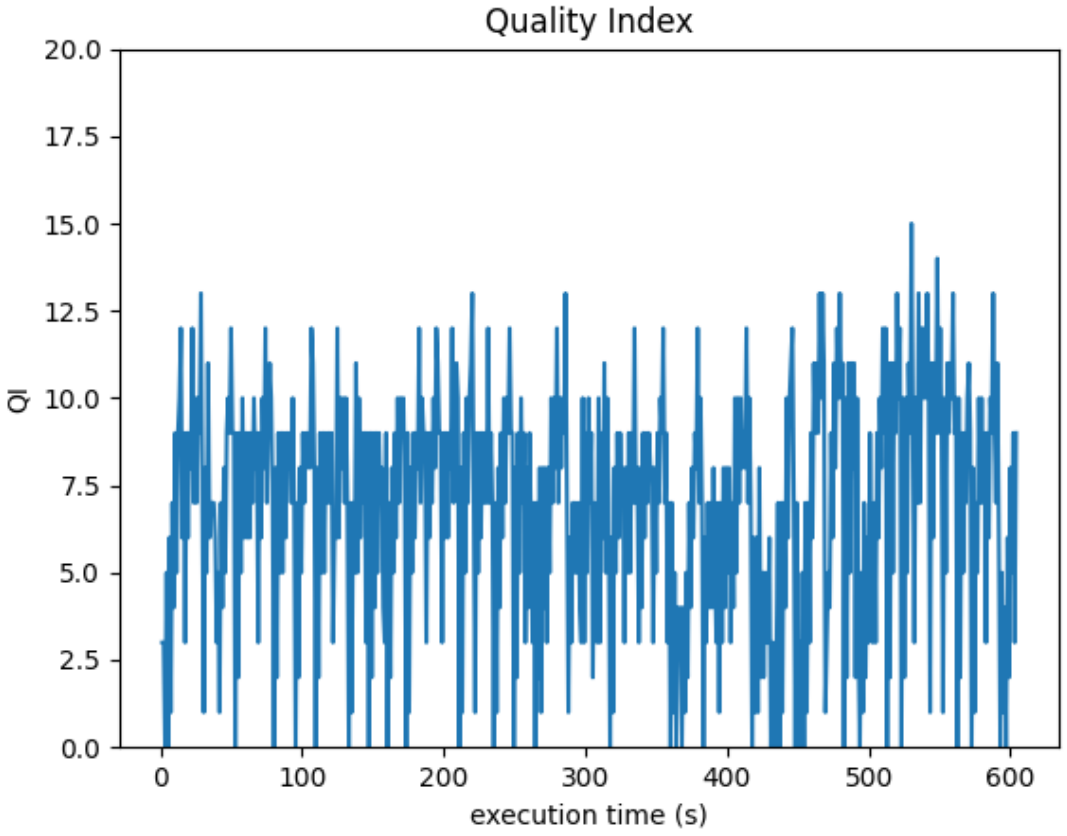

Quality QI
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Mbps
0
200
400
600
800
1000
1200
execution time (s)

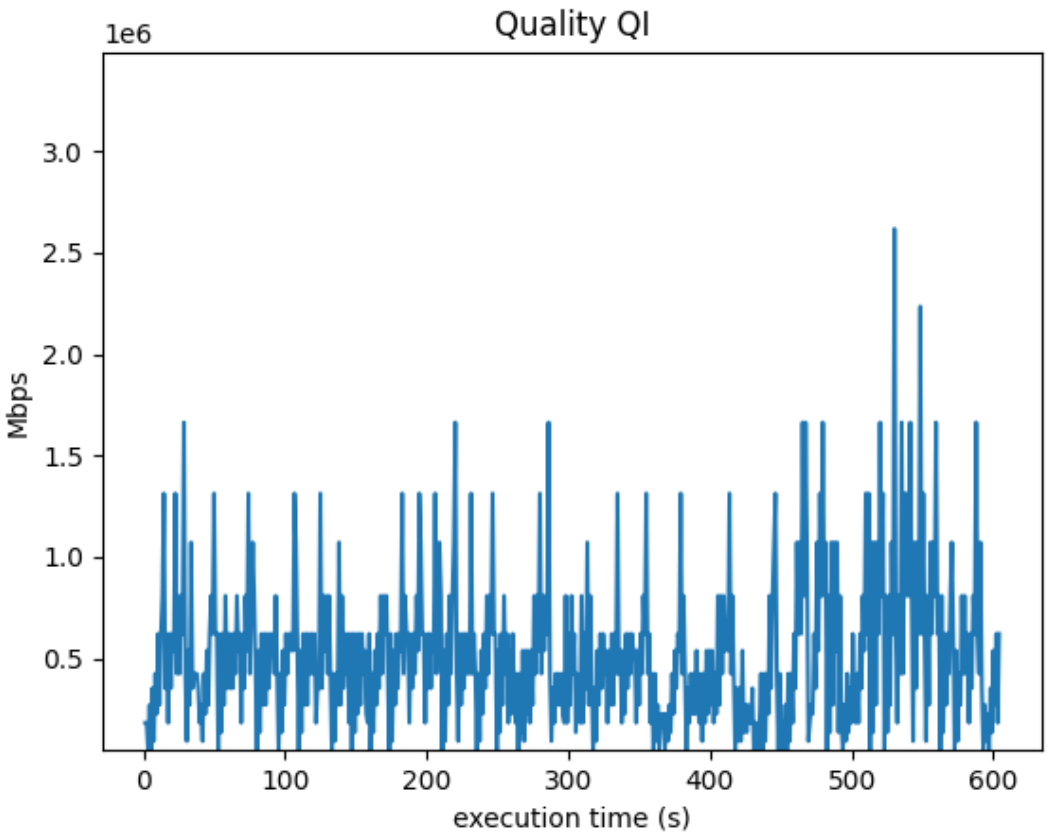

Throughput Variation
2.5
2.0
1.5
1.0
0.5
0.0
Mbps
0
200
400
600
800
1000
1200
execution time (s)

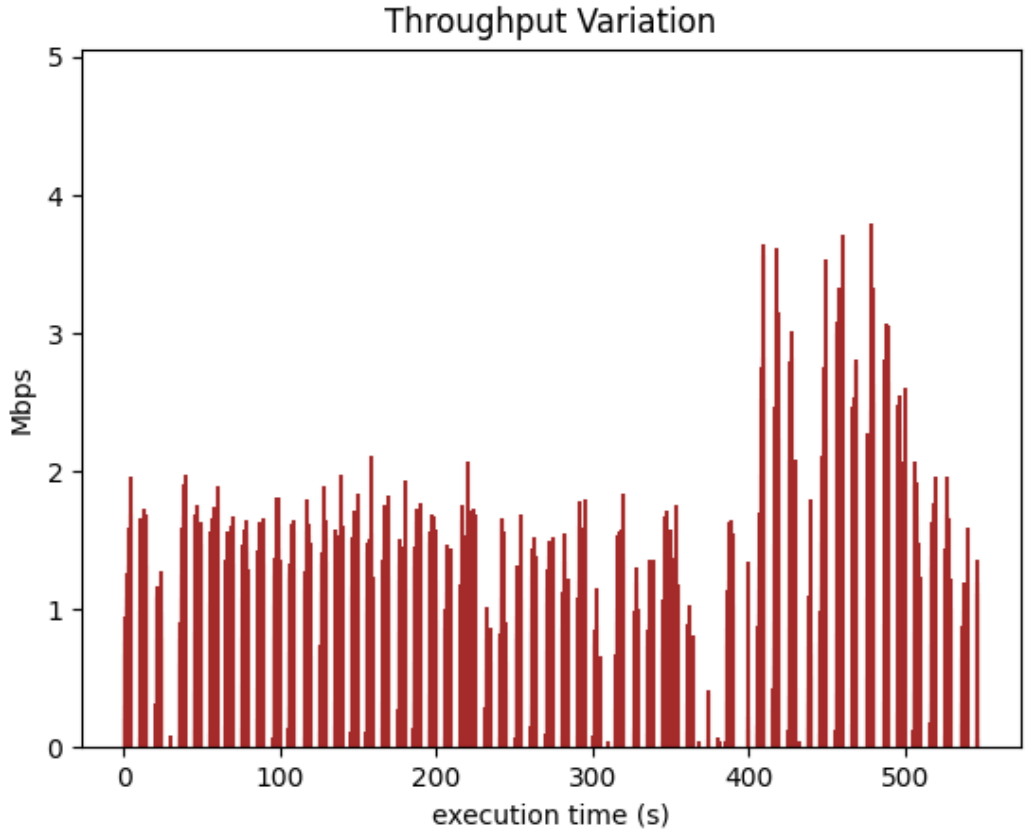
4.7- Cenário “LHL”



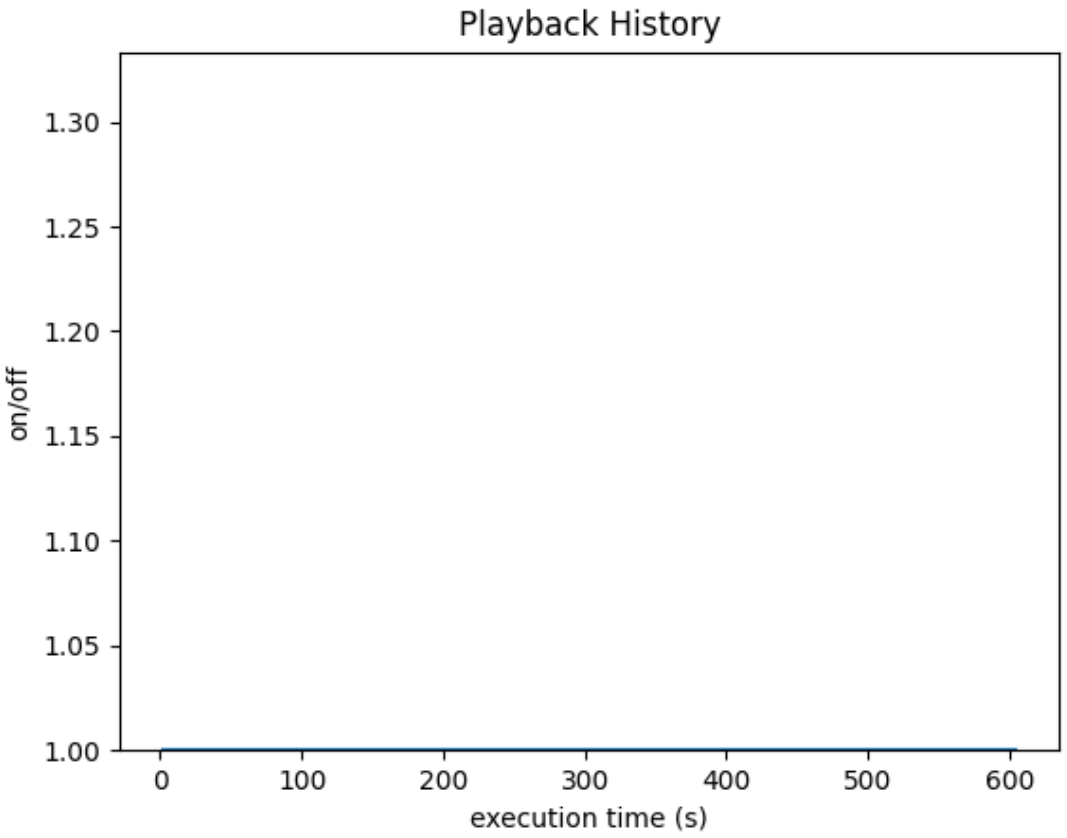

Buffer Size
seconds
execution time (s)
0
10
20
30
40
50
60
70
80
0
100
200
300
400
500
600

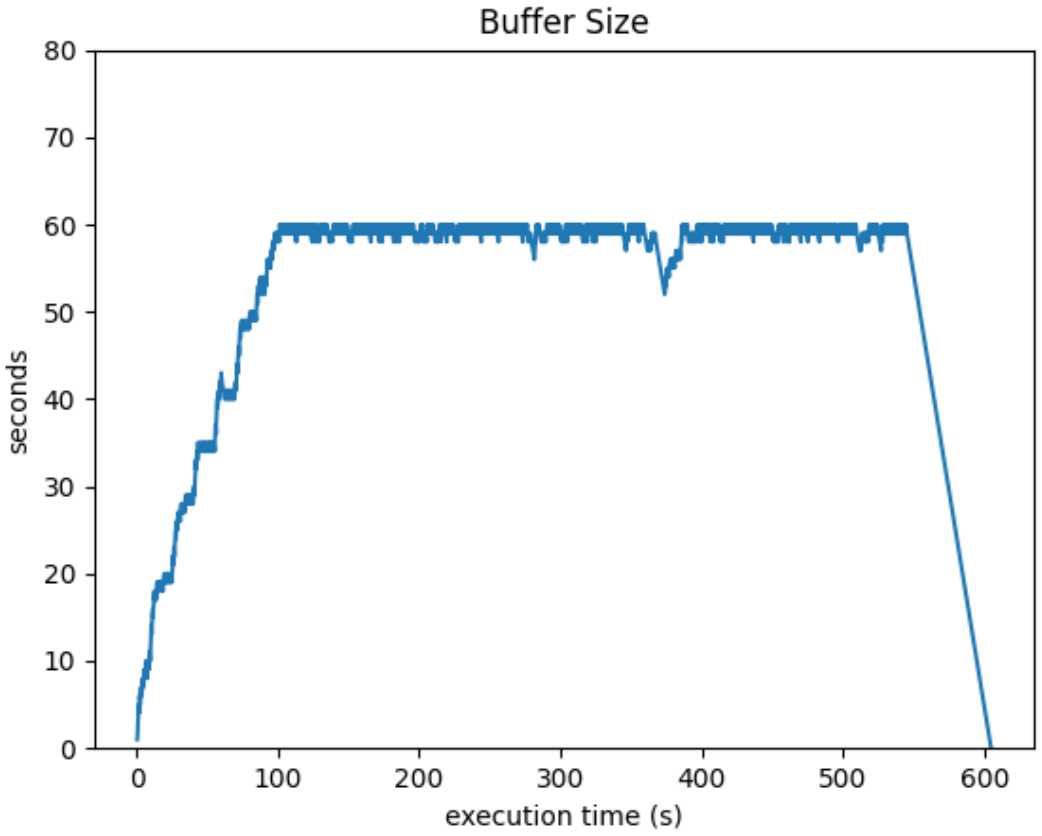

Quality Index
QI
execution time (s)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
100
200
300
400
500
600

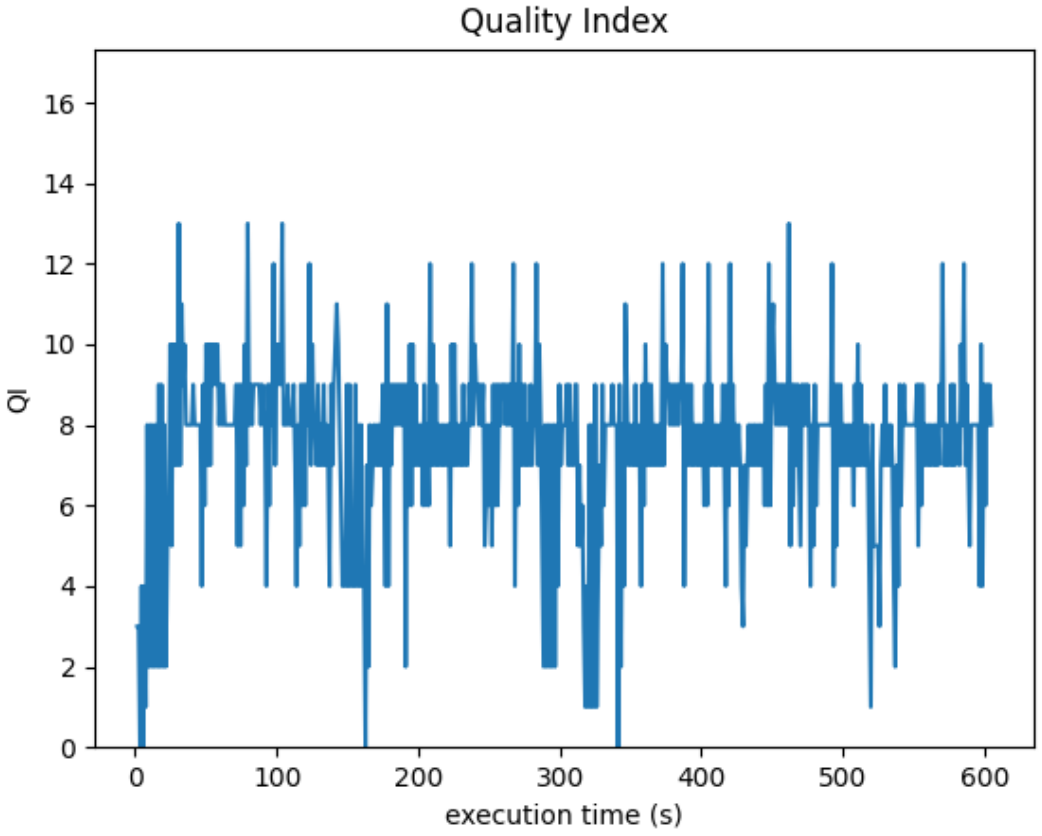

Quality QI
1e6
3.0
2.5
2.0
1.5
1.0
0.5
Mbps
0
100
200
300
400
500
600
execution time (s)

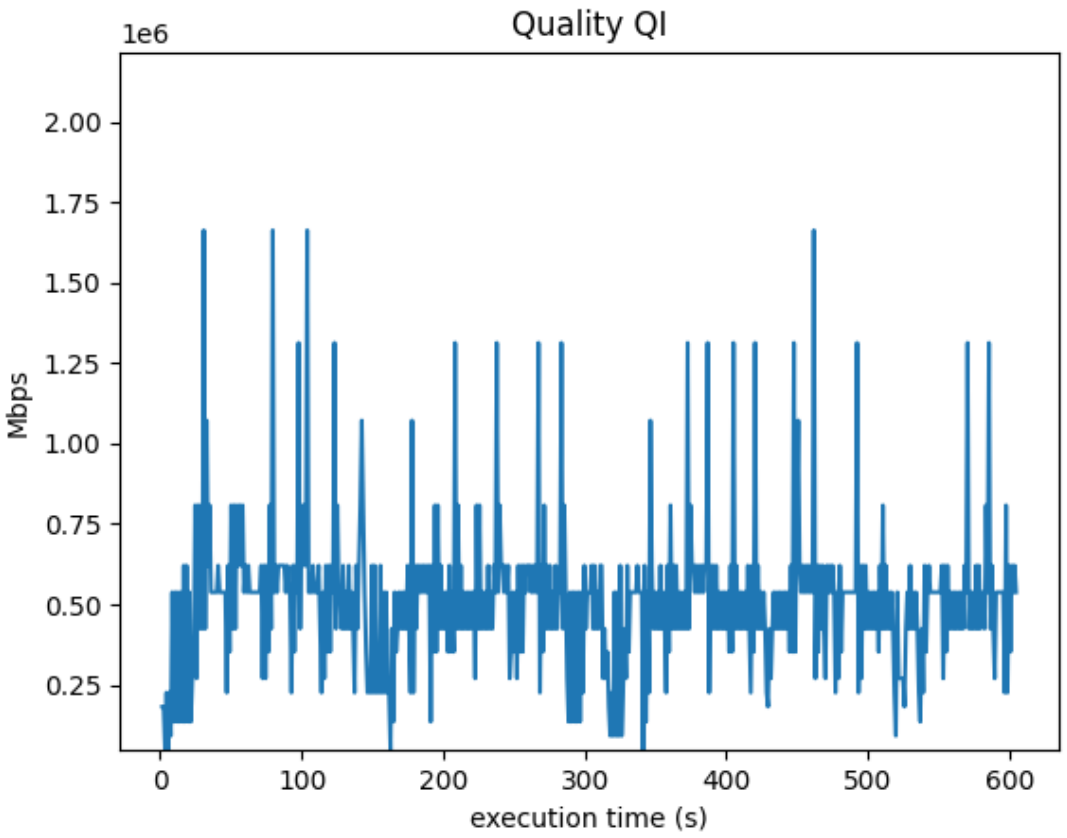

Throughput Variation
5
4
3
2
1
0
Mbps
0
100
200
300
400
500
execution time (s)

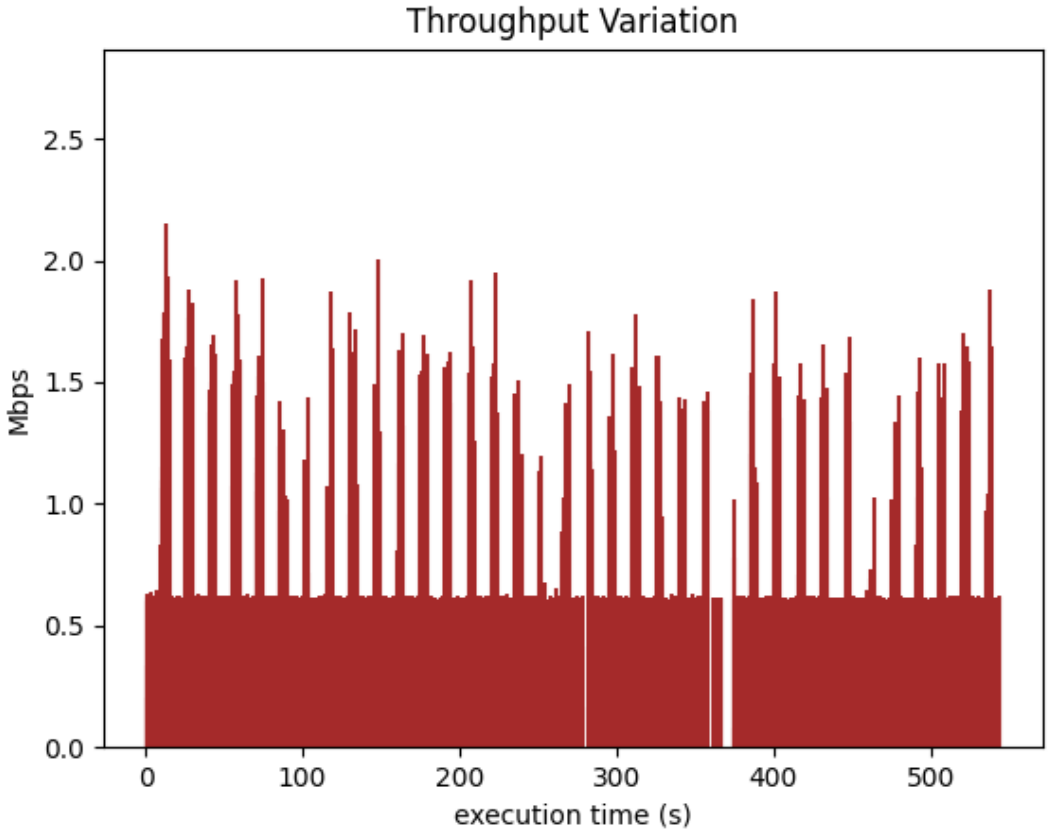
4.8- Cenário “MML”



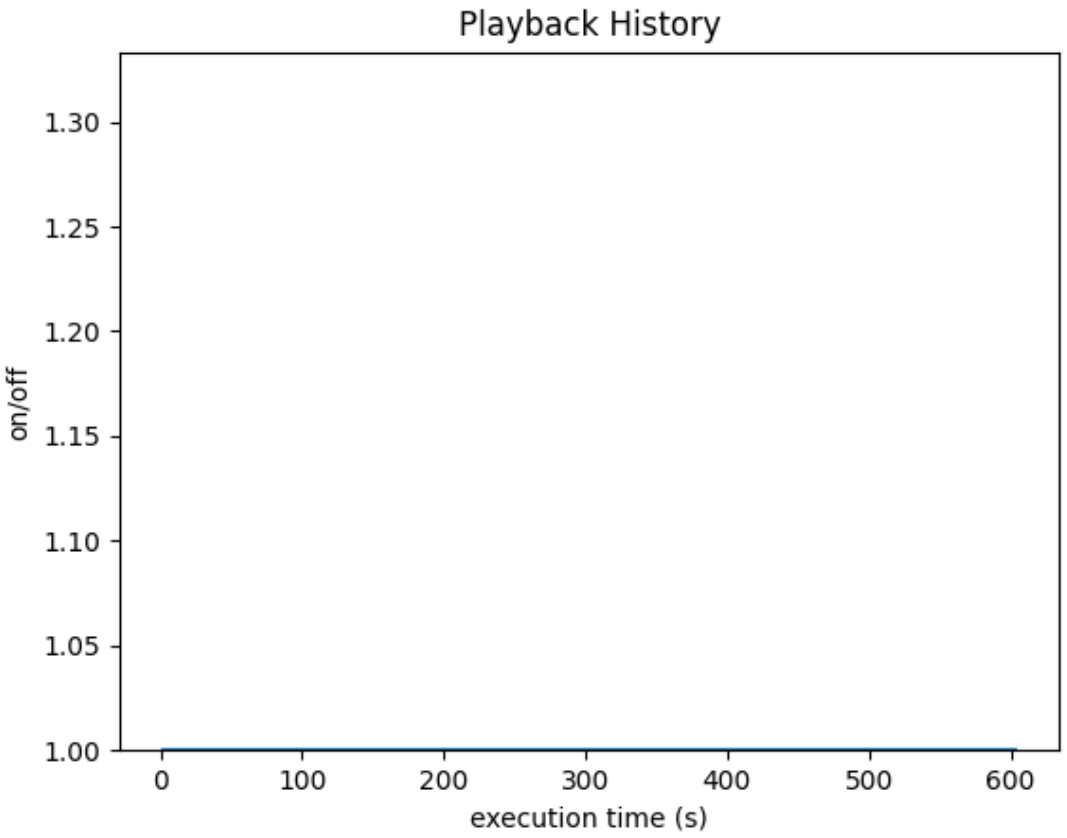

Buffer Size
seconds
execution time (s)
0
10
20
30
40
50
60
70
80
0
100
200
300
400
500
600

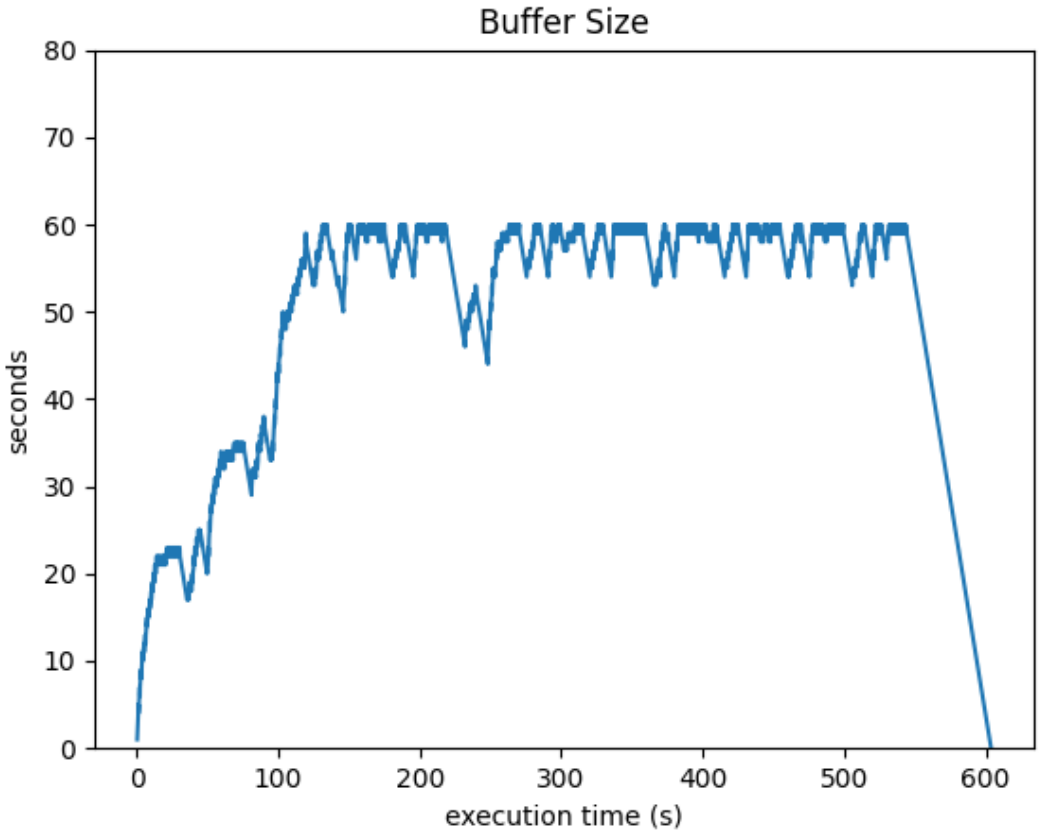

Quality Index
QI
execution time (s)
0
2
4
6
8
10
12
14
16
0
100
200
300
400
500
600

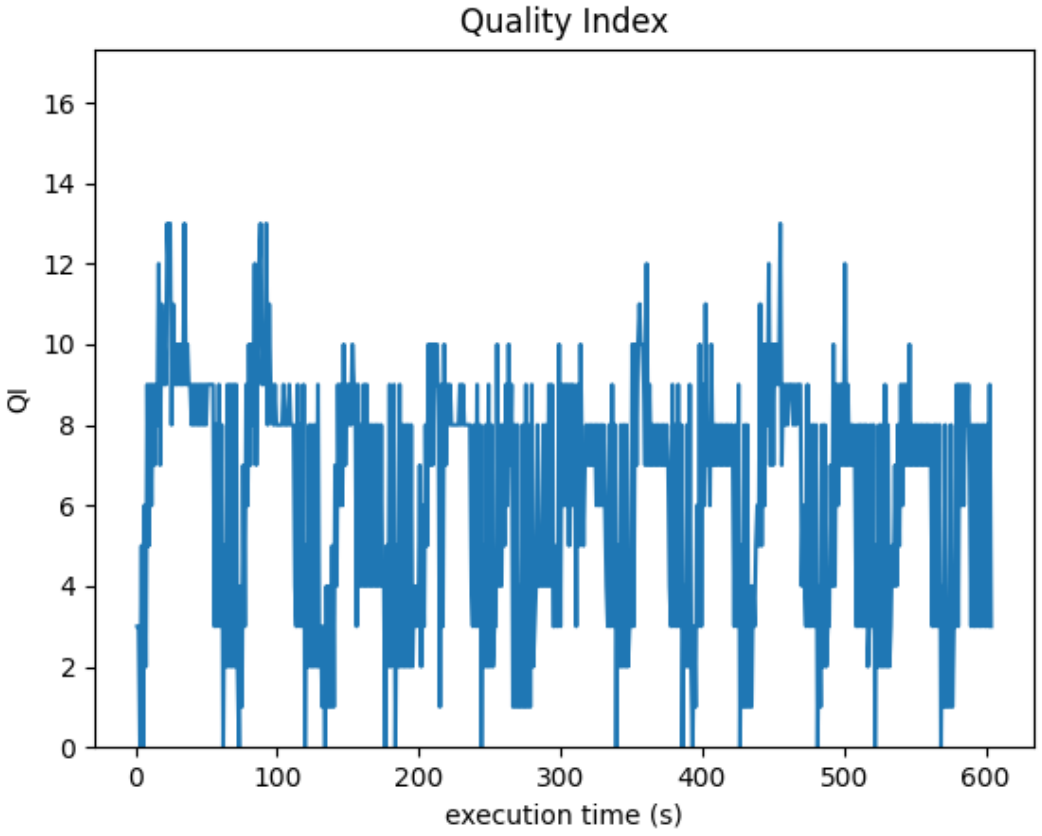

Quality QI
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Mbps
0
100
200
300
400
500
600
execution time (s)

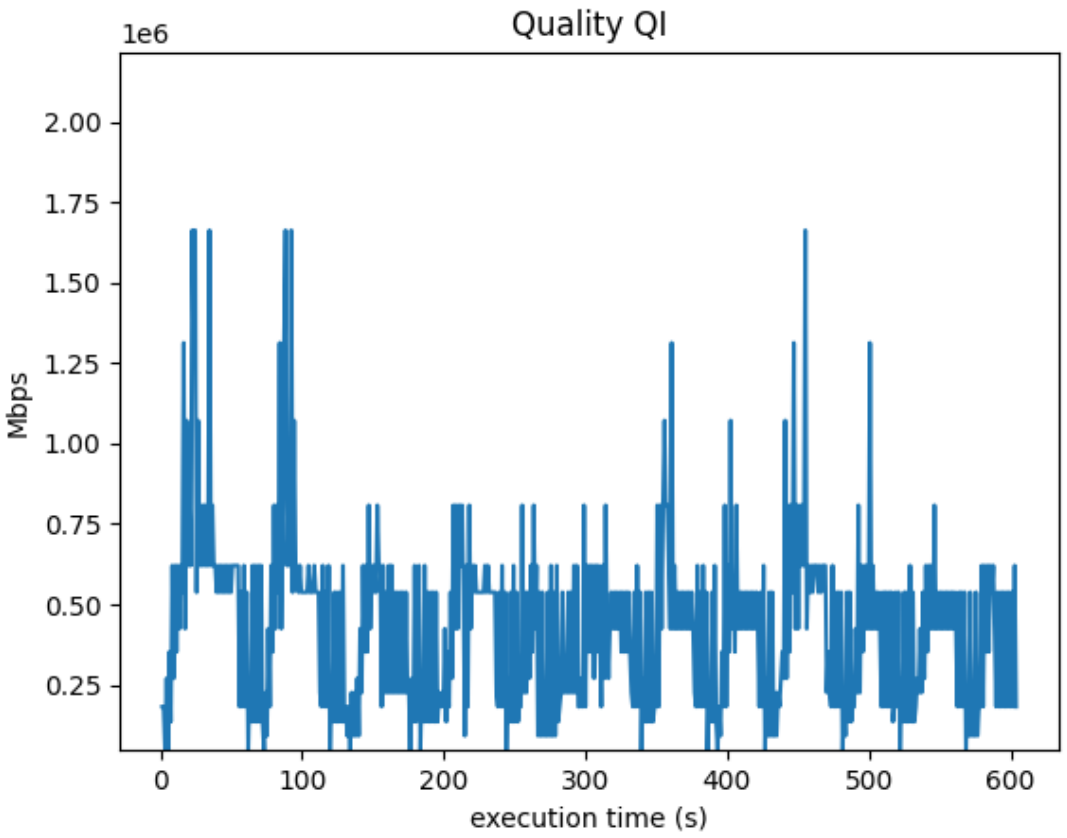

Throughput Variation
2.5
2.0
1.5
1.0
0.5
0.0
Mbps
0
100
200
300
400
500
execution time (s)

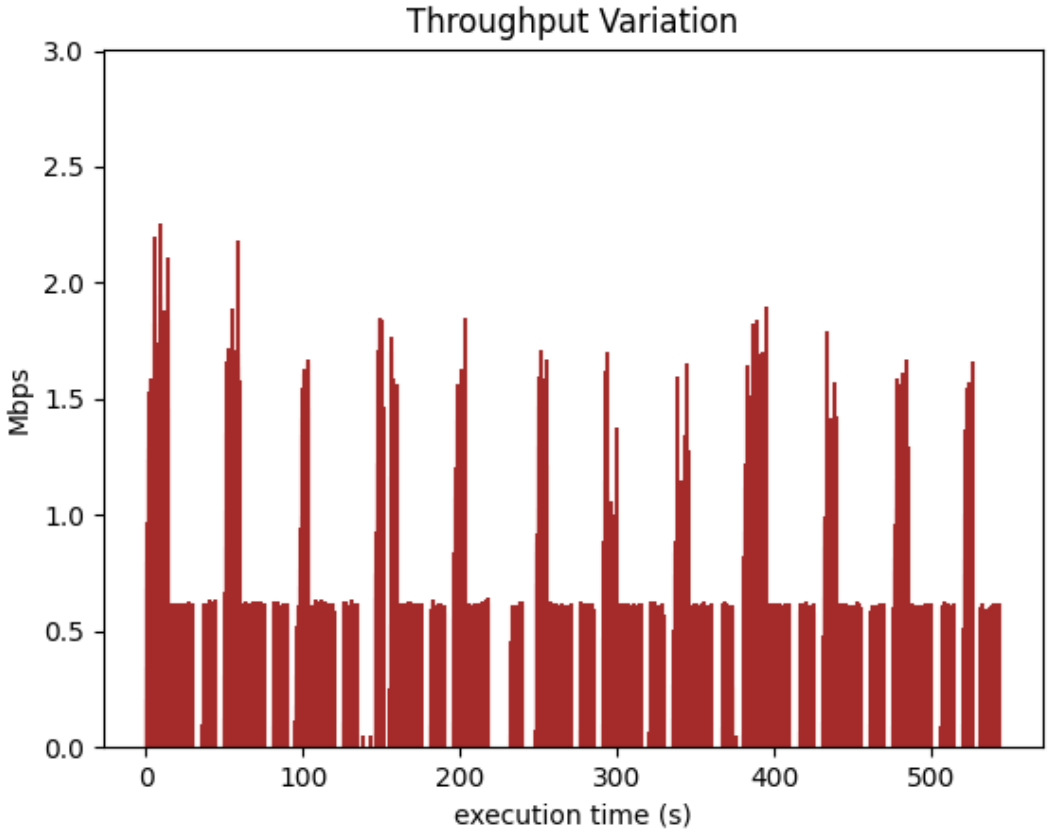
4.9- Cenário “LLLMMMHMMMH”



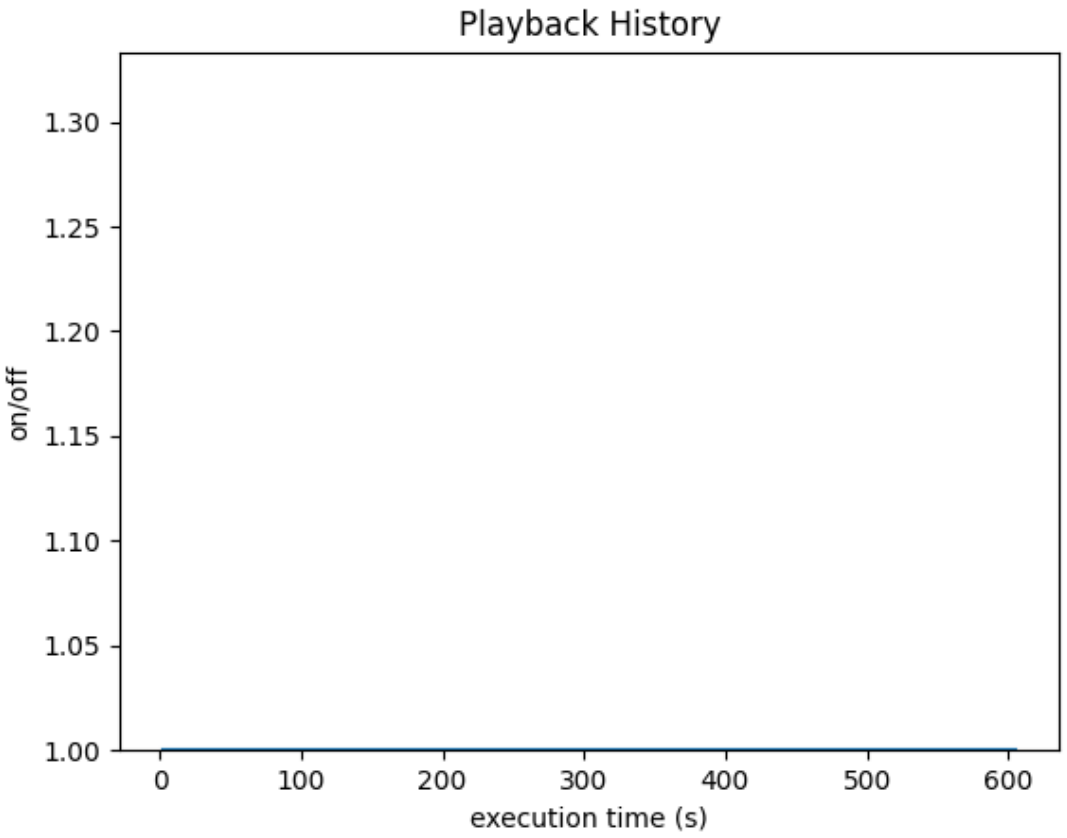

Buffer Size
seconds
execution time (s)
0
10
20
30
40
50
60
70
80
0
100
200
300
400
500
600

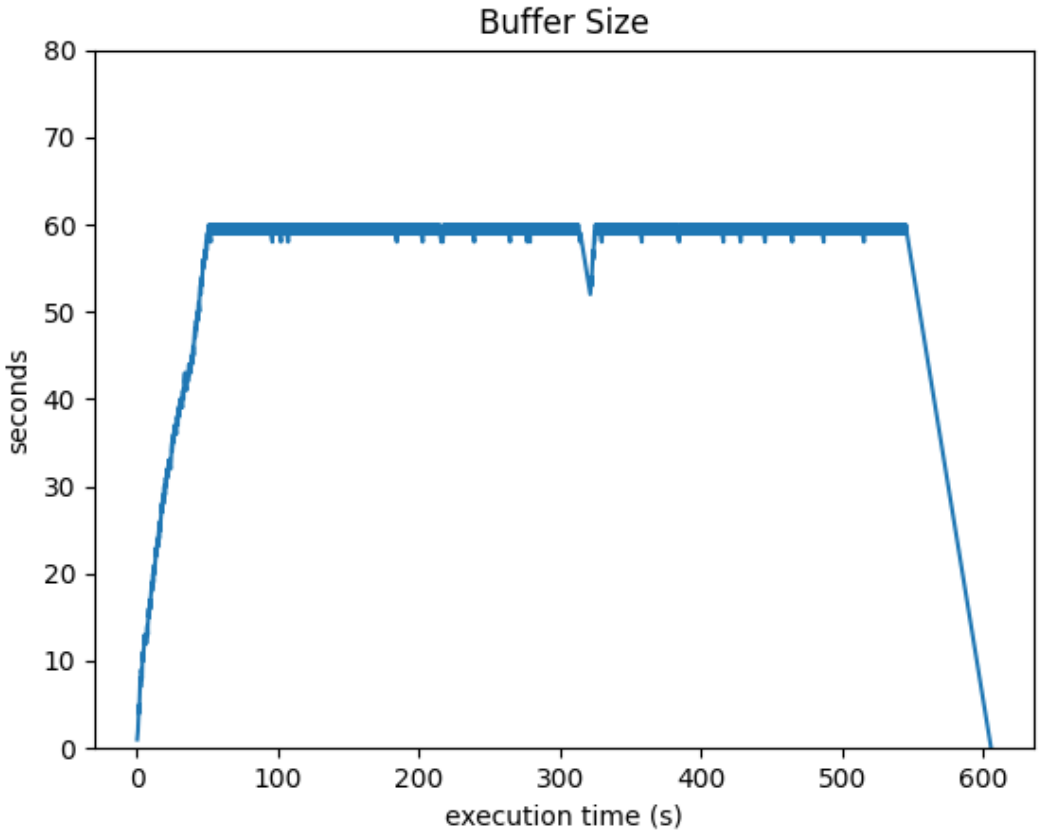

Quality Index
QI
execution time (s)
0
2
4
6
8
10
12
14
16
0
100
200
300
400
500
600

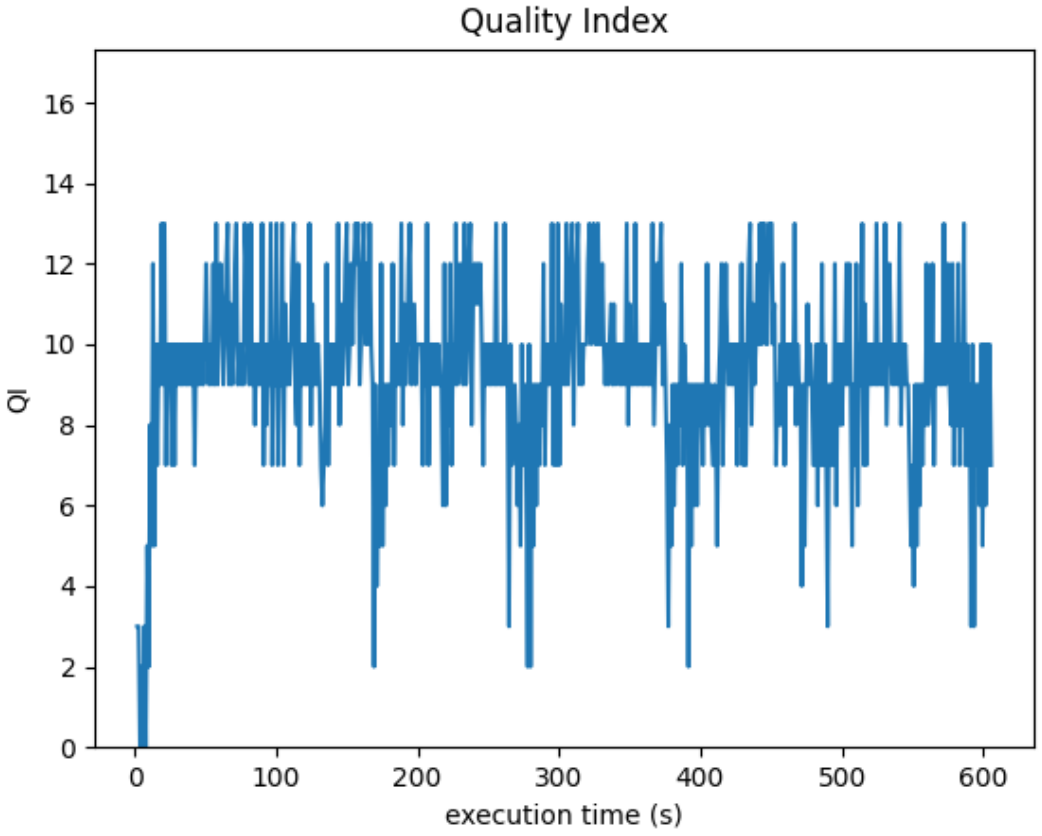

Quality QI
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Mbps
0
100
200
300
400
500
600
execution time (s)

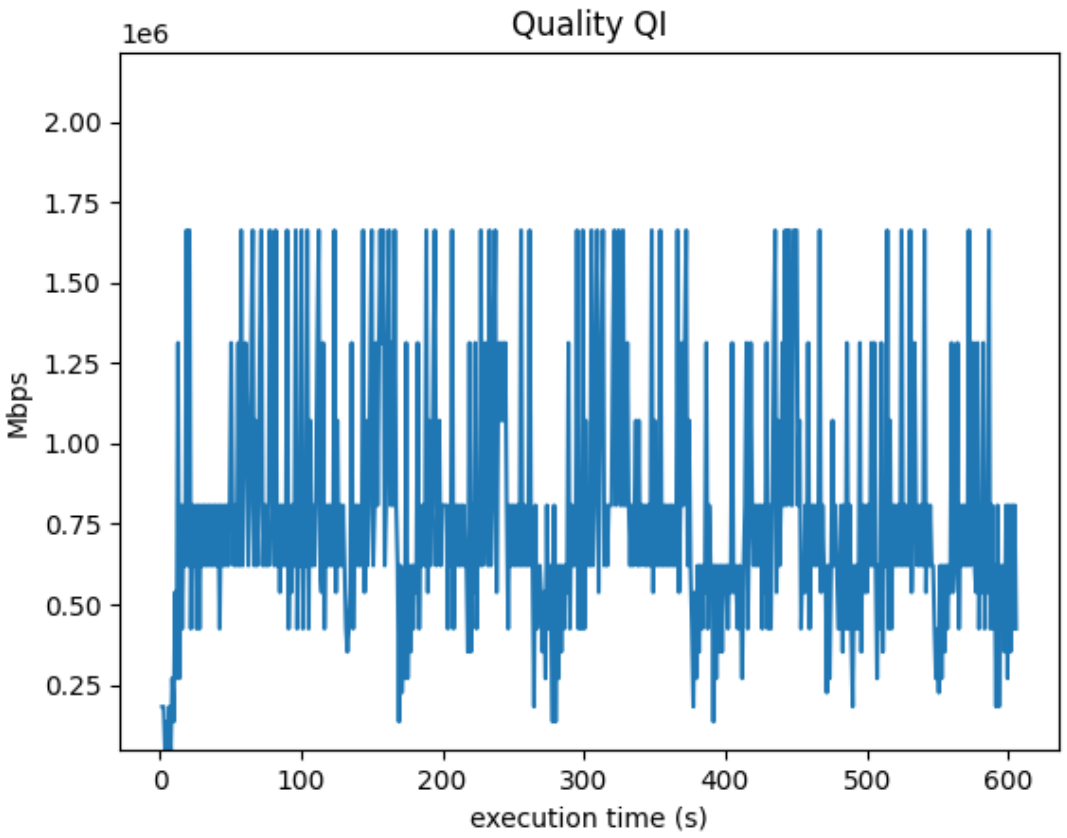

Throughput Variation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Mbps
0
100
200
300
400
500
execution time (s)

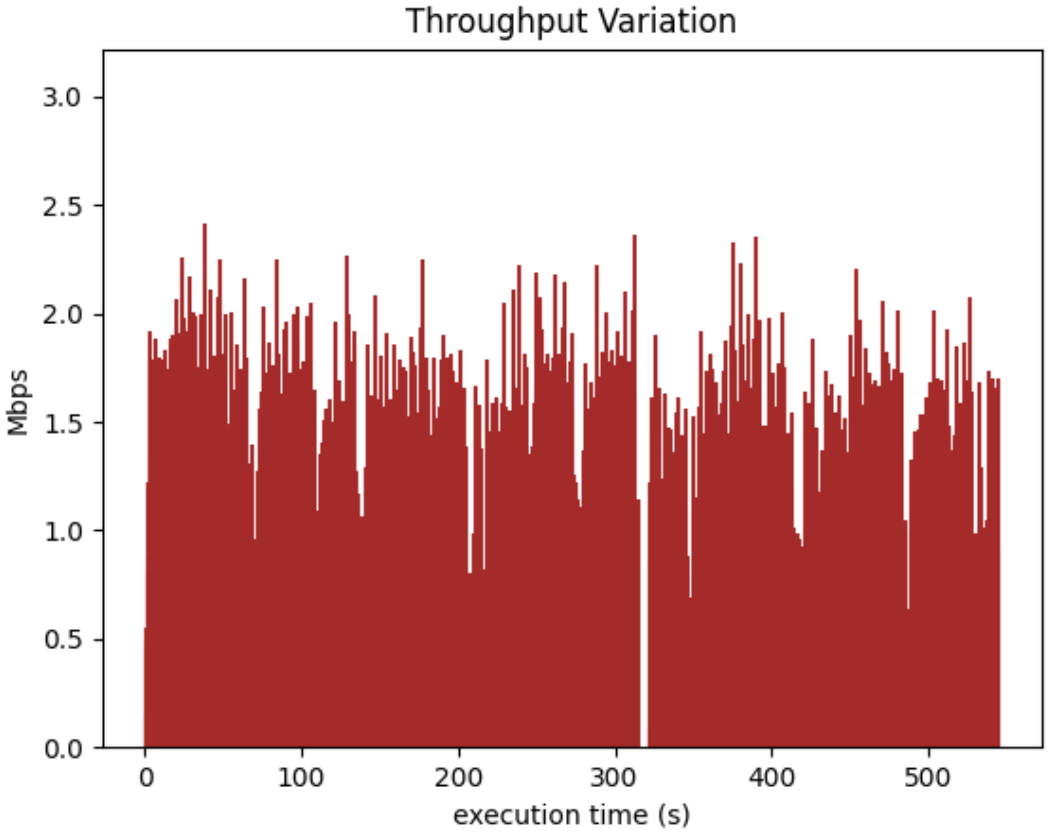
4.10- Cenário “LL”



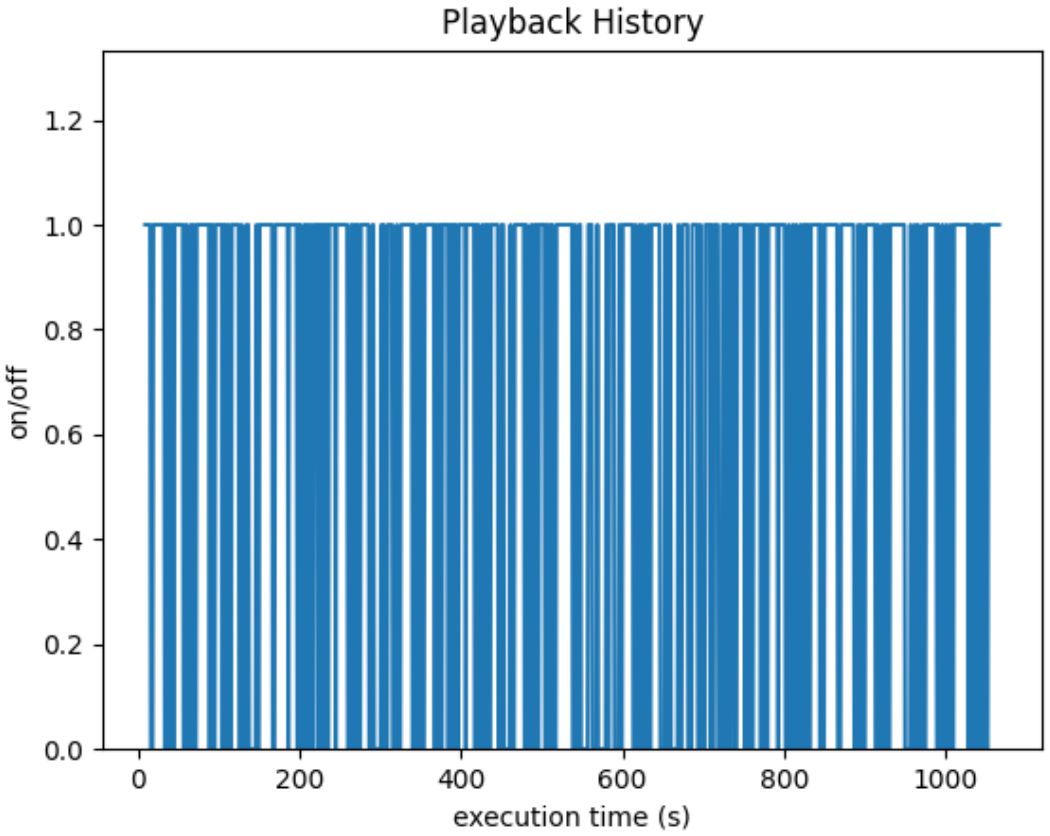

Buffer Size
seconds
execution time (s)
0
10
20
30
40
50
60
70
80
0
100
200
300
400
500
600

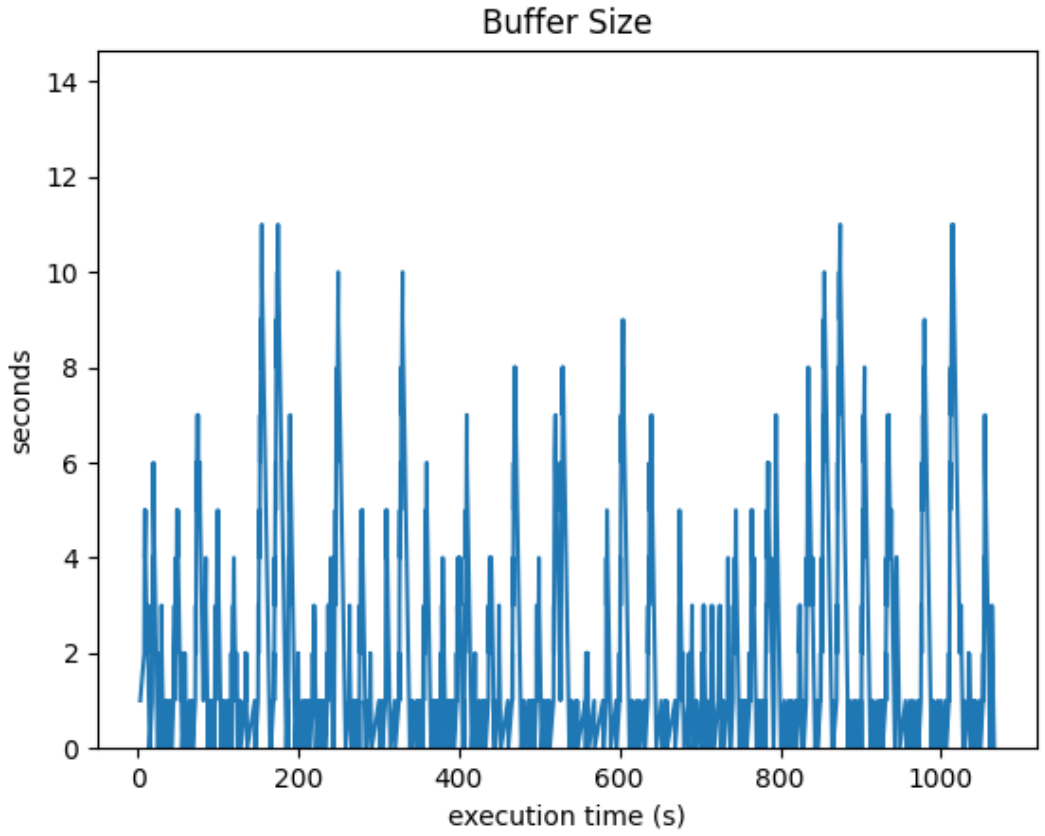

Quality Index
QI
execution time (s)
0
2
4
6
8
10
12
14
16
0
100
200
300
400
500
600

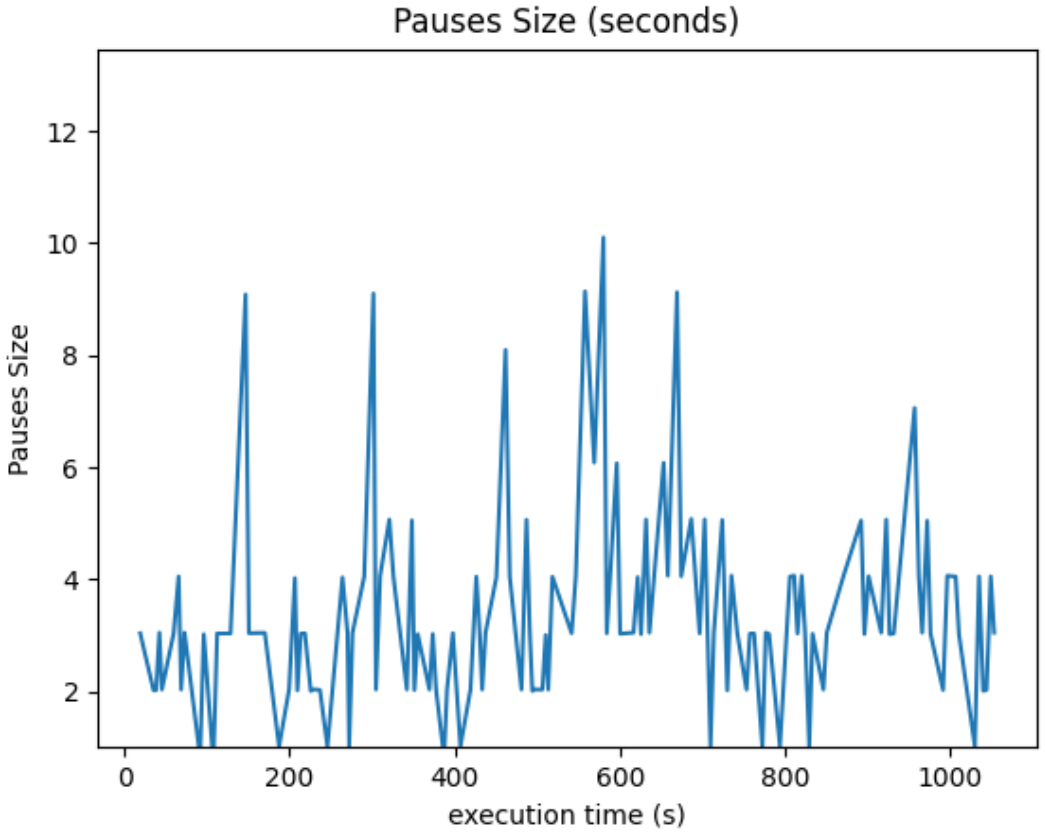

Quality QI
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Mbps
0
100
200
300
400
500
600
execution time (s)

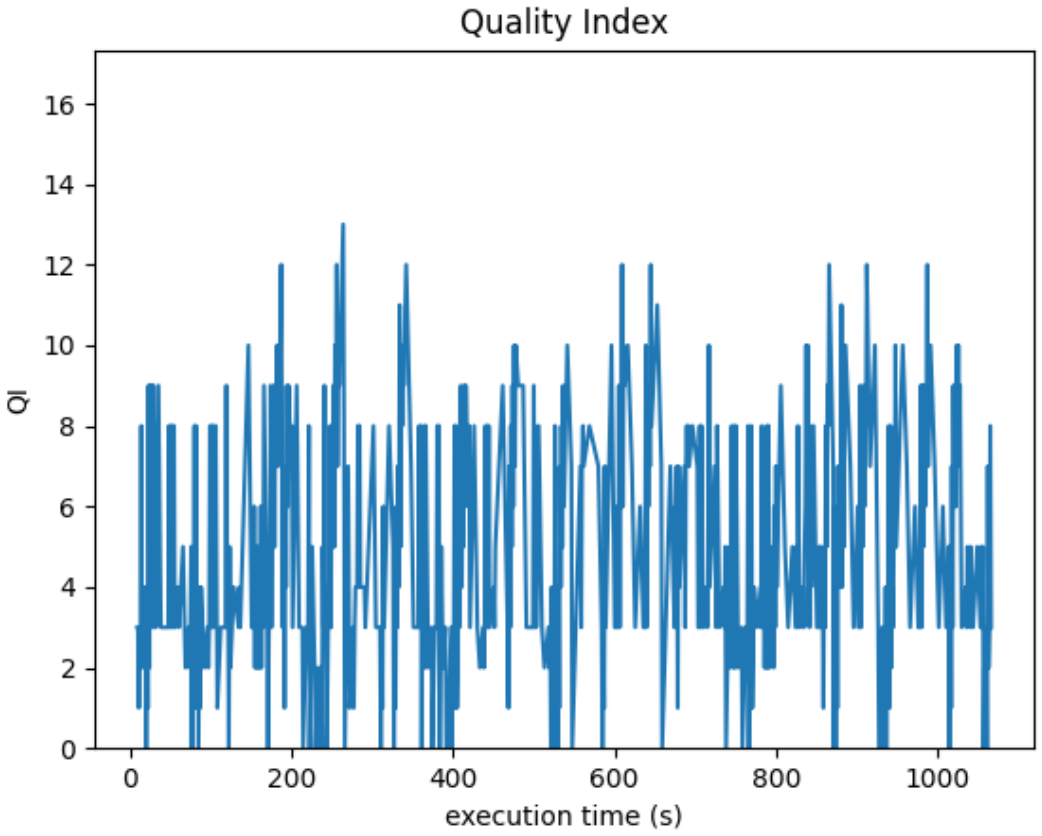

Throughput Variation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Mbps
0
100
200
300
400
500
execution time (s)

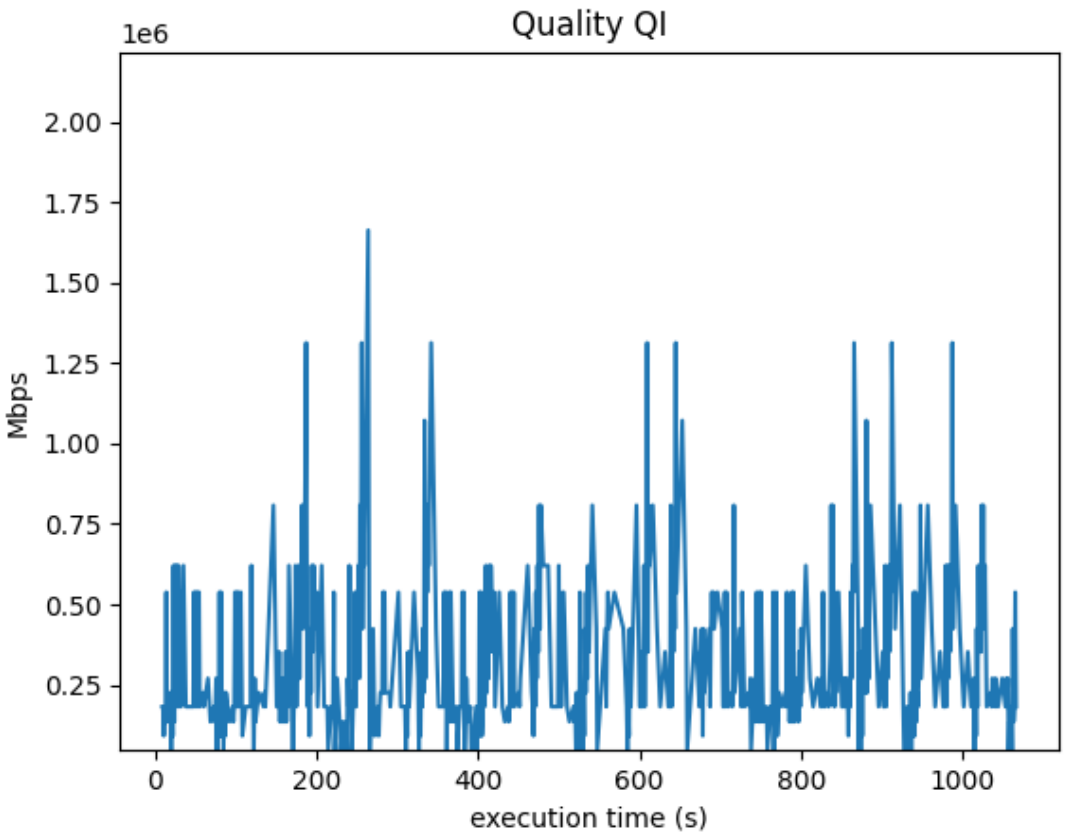
4.11- Cenário “HMHLH”

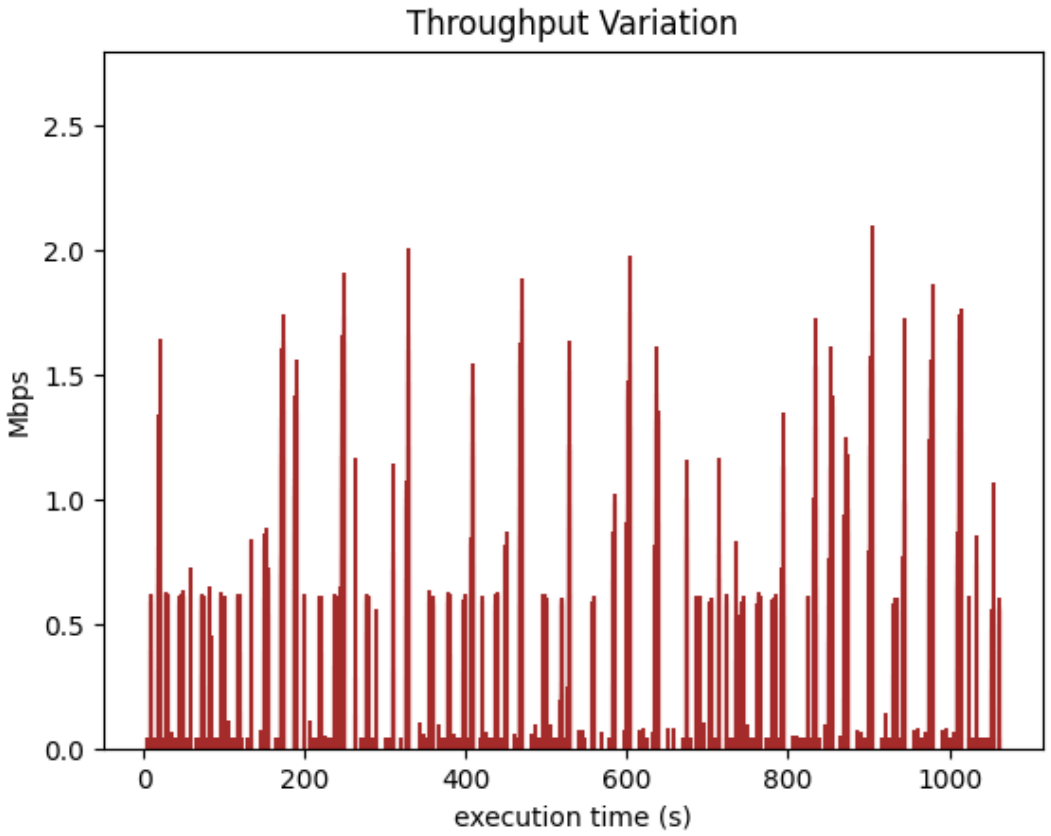



Buffer Size
seconds
execution time (s)
0
2
4
6
8
10
12
14
0
200
400
600
800
1000


Pauses Size (seconds)
Pauses Size
execution time (s)
2
4
6
8
10
12
0
200
400
600
800
1000


Quality Index
QI
execution time (s)
0
2
4
6
8
10
12
14
16
0
200
400
600
800
1000


Quality QI
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Mbps
0
200
400
600
800
1000
execution time (s)


Throughput Variation
2.5
2.0
1.5
1.0
0.5
0.0
Mbps
0
200
400
600
800
1000
execution time (s)

5- Conclusão

Assim foi possível fazer um algoritmo eficiente com um número de pausas e qualidade dentro de uma margem aceitável, dada a existência de um *tradeoff* entre ter uma qualidade melhor com um vídeo que pausa mais ou uma quantidade de pausas menor com um vídeo de menor qualidade.

6- Referências

- Adaptive Streaming of Audiovisual Content Using MPEG DASH, de Truong Cong Thang e Anh T. Pham
- Material mostrado em sala de aula