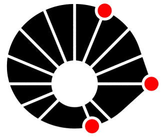

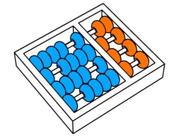

Avaliação de desempenho do HTTP/3 para transmissão de vídeo 360º com taxa de bits adaptável

G. A. Tabchoury *C. Melo* *D. M. Casas-Velasco*
N. L. S. da Fonseca

Relatório Técnico - IC-PFG-21-41
Projeto Final de Graduação
2021 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

The contents of this report are the sole responsibility of the authors.
O conteúdo deste relatório é de única responsabilidade dos autores.

Avaliação de desempenho do HTTP/3 para transmissão de vídeo 360º com taxa de bits adaptável

Gabriel Alves Tabchoury[*] Cesar Melo[†] Daniela M. Casas-Velasco[†]
Nelson L. S. da Fonseca[†]

Resumo

1 Introdução

Realidade virtual é um assunto que se popularizou ao longo dos últimos anos. Em 2021 a empresa matriz do Facebook alterou o nome para Meta, criando assim uma nova identidade para a empresa que se propõe a explorar a realidade virtual no âmbito das conexões sociais.

Os vídeos 360º, que já estão disponíveis no YouTube desde 2015, tornaram a realidade virtual acessível para muitas pessoas. Apenas com um celular e uma conexão a Internet é possível assistir um vídeo em realidade virtual gratuitamente.

No entanto, ainda existe muito a se explorar com os vídeos 360º, principalmente no que tange a séries e filmes nos principais serviços de *streaming* atuais, como Netflix e Amazon Prime.

O principal desafio para a transmissão de um vídeo 360º é, sem dúvida, o volume de dados que necessita ser trafegado pela Internet, uma vez que um vídeo 360º possui muito mais *pixels* comparado a um vídeo tradicional. Por conta disso, para que o usuário tenha uma boa experiência, ou seja, um vídeo com uma qualidade alta e sem travamentos, é imprescindível uma boa conexão com a Internet.

Pensando nisso, este trabalho tem como objetivo avaliar o desempenho da terceira e próxima versão do principal protocolo de comunicação web, o HTTP/3, para a transmissão de vídeos 360º. Será utilizada a técnica DASH para transmitir o vídeo com uma taxa de bits adaptável, ou seja, uma mudança automática na qualidade do vídeo de acordo com a conexão do usuário.

Com a ajuda do Mininet[9], serão realizados experimentos simulando diversos cenários de rede a fim de entender os requisitos mínimos para que o usuário possa ter uma ótima experiência ao assistir um vídeo 360º.

[*]Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 13083-852 Campinas, SP.
[†]Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

2 Vídeo 360º

Os vídeos 360º têm se tornado cada vez mais populares, principalmente após o surgimento dos óculos de realidade virtual. Com o objetivo de entregar uma melhor experiência para o usuãrio, os vídeos 360º são geralmente gravados com câmeras especiais que registram as imagens em todas as direções. Desta forma, durante a reprodução, o usuário possui controle total do ângulo de visão, fazendo com que o mesmo se sinta no local em que o vídeo foi gravado.

Embora a imersão total seja possível apenas com um óculos de realidade virtual (também conhecidos como óculos 3D), também é possível a repodução de vídeos 360º com *players* que possuem suporte, ou seja, que possibilitam o controle da direção da visualização.

No entanto, o grande problema dos vídeos 360º é o tamanho dos arquivos gerados. Como esperado, os vídeos 360º são muito mais pesados do que os os vídeos convencionais, o que dificulta a transmissão pela Internet.

Desta forma, para não ter que enviar todo o vídeo em uma única requisição, o vídeo será segmentado em trechos menores, e o cliente irá requisitar conforme necessário, ou seja o vídeo será transmitido sob demanda do cliente. No experimento, um vídeo 360º de 6 segundos será dividido em 6 segmentos de 1 segundo.

Além disso, será utilizada a técnica de *video tiling*, que consiste na divisão do vídeo em diversos quadrantes (chamados de *tiles*) que posteriormente serão agrupados no momento de reprodução. Para o experimento cada segmento será dividido em 200 *tiles*.

Por fim, como uma pessoa não possui uma visão 360º, parte dos *tiles* não estarão no campo de visão do usuário em um determinado momento de reprodução, ou seja, como o usuário possui controle da direção de visualização do vídeo, a cada momento pode mudar os *tiles* no campo de visão do usuário. Sendo assim, os *tiles* no campo de visão do usuário serão requisitados com uma maior prioridade e o servidor irá administrar as requisições de modo a dar uma maior vazão a esses *tiles* de maior prioridade.

3 Taxa de bits adaptável

Durante uma transmissão de vídeo, a conexão com a Internet do usuário sofre oscilações que podem prejudicar a reprodução do vídeo, gerando inclusive travamentos. Algumas plataformas de vídeos disponibilizam mais de uma resolução para o mesmo vídeo (como por exemplo 360p, 720p, 1080p e 2160p) e o usuário tem a possibilidade de escolher a resolução que deseja assistir. No entanto, quanto maior a resolução, maior o volume de dados que deve ser transmitido e portanto melhor deve ser a conexão com a Internet do usuário.

A DASH (*Dynamic Adaptive Streaming over HTTP*), também conhecida como MPEG-DASH é uma técnica de *streaming* que permite uma adaptação do *bitrate* (taxa de bits), ou seja, uma mudança automática na qualidade do vídeo em tempo de transmissão.

Sendo assim, serão utilizados algoritmos DASH que irão determinar, com base nas métricas de transmissão dos últimos segmentos, qual será o *bitrate* do próximo segmento de vídeo. Desta forma, é possível diminuir a qualidade do vídeo automaticamente quando a conexão estiver baixa (evitando travamentos) e, da mesma forma, aumentar a qualidade

caso a conexão melhore. Com isso, é esperada uma melhor experiência para o usuário, garantindo a melhor qualidade possível de acordo com a sua conexão.

Para a simulação, serão utilizados os algoritmos *basic_dash* e *basic_dash2* do repositírio AStream[1] e os seguintes *bitrates*: 4.156Mbps, 7.033Mbps e 9.721Mbps que estarão disponíveis para todos os *tiles* de todos os segmentos.

O algoritmo *basic_dash* compara o tempo de *download* do último segmento com o tempo médio de *download* para determinar o próximo *bitrate*. Caso o tempo de *download* do segmento atual seja maior do que a média, o algoritmo reduz um *bitrate*. No entanto, se o tempo de *download* for menor e a razão entre o tempo médio de *download* e o tempo de *download* do último segmento for maior do que a razão entre o *bitrate* atual e o próximo *bitrate*, o algoritmo aumenta o *bitrate*.

O algoritmo *basic_dash2* utiliza como base a taxa de *download* (*download rate*) que é dada pela razão entre o tamanho dos arquivos baixados e o tempo gasto para baixar os arquivos. O algoritmo compara a taxa de *download* dos últimos segmentos e aumenta ou diminui o *bitrate* em caso de aumento ou diminuição da taxa, respeitando uma margem de 20%.

4 HTTP/3 e QUIC

O HTTP/3 é a terceira versão do HTTP que, diferentemente das versões anteriores, utiliza o QUIC (*Quick UDPInternet Connections*) como transporte.

O QUIC, que foi introduzido em 2012 pela Google, é um protocolo de transporte que tem como principal objetivo a diminuição da latência HTTP. Atualmente, o QUIC já é utilizado na maior parte das conexões do navegador Chrome com os servidores da Google e a expectativa é que o mesmo se torne o novo protocolo padrão para navegaçao WEB.

Para diminuir a latência e melhorar o desempenho das aplicaçoes, o QUIC utiliza *multiplexing*, que faz com que várias conexões multiplexadas sejam estabelecidas entre o cliente e servidor, permitindo o envio de dados paralelamente. Embora já seja possível enviar dados de forma paralela com o HTTP/2, como o QUIC utiliza o UDP (*User Datagram Protocol*), o fenômeno de *head of line blocking*, que ocorre quando pacotes TCP são perdidos ou atrasados, não acontece.

Desta forma, o QUIC possibilita o envio de um grande volume de dados com uma baixa latência, o que indica que o mesmo terá um bom desempenho na transmissão de vídeos 360º, principalmente com as técnicas de segmentação e vídeo tiling que iremos utilizar.

Para realizar os experimentos, serão utilizadas como base as implementações de cliente e servidor do Gustavo Fernandez[2] que utilizam a biblioca Aioquic da linguagem Python[8], que implementa o QUIC.

5 Implementação

Para a realização dos experimentos foi utilizado o vídeo “Chariot Race”[5] que originalmente é um arquivo MP4 na resolução 3840x2160.

Primeiramente, foi necessário converter o vídeo para o formato YUV com a ferramenta *ffmpeg* para utilizar o codificador *Kvazaar*.

Posteriormente, utilizamos o codificador *Kvazaar* para dividir o vídeo em *tiles* e também gerar os arquivos para os diferentes *bitrates*.

Por fim, utilizou-se o empacotador *MP4Box* para segmentar o vídeo em segmentos de 1 segundo e gerar os arquivos para os correspondentes *bitrates*, segmentos e *tiles*.

Sendo assim, como serão utilizados 3 *bitrates*, 50 segmentos e 200 *tiles*, teremos um total de 30000 arquivos de vídeo que serão transmitidos entre o servidor e o cliente.

5.1 Servidor

Como citado anteriormente, o servidor foi implementado em Python utilizando a biblioteca Aioquic. Como o servidor é capaz de atender múltiplos clientes sob demanda, uma vez executado, o mesmo entra em um *loop* aguardando as requisições.

A principal responsabilidade do servidor é fornecer o arquivo solicitado por um determinado cliente. Para isso, cada requisição deve conter os seguintes parâmetros:

- **id do cliente**: como o servidor é capaz de atender múltiplos clientes, é necessário um identificador único para cada cliente
- ***bitrate***: através dos algoritmos DASH o cliente irá solicitar o arquivo em um *bitrate* específico
- **segmento**: número do segmento solicitado pelo cliente
- ***tile***: número do *tile* solicitado pelo cliente
- **prioridade**: 1 para *tiles* no campo de visão do usuário e 2 para *tiles* fora do campo de visão

Para administrar a prioridade dos *tiles*, o servidor utiliza uma política de enfileiramento. No caso do presente trabalho, será utilizada a WFQ (*Weighted Fair Queuing*), que irá atribuir um peso 2 vezes maior para os *tiles* no campo de visão.

Além disso, para aproveitar o *multiplexing* do QUIC, o servidor terá duas *streams* de conexão com o cliente, uma para os *tiles* no campo de visão e outra para os *tiles* fora do campo de visão. Desta forma, o impacto dos *tiles* de menor prioridade para os *tiles* de maior prioridade é reduzido. Como o número de *tiles* fora do campo de visão é, em média, 6 vezes maior que o número de *tiles* no campo de visão, vamos ter uma maior vazão para os *tiles* de alta prioridade, o que pode proporcinar uma melhor experiência para o usuário.

Com o intuito de melhorar ainda mais o desempenho, o servidor utiliza uma técnica chamada *server push*, que consiste em uma resposta do servidor para uma requisição que ainda não foi feita pelo cliente. Ou seja, o *server push* é uma técnica que tenta prever uma próxima requisição do cliente.

Neste caso, o *server push* foi implementado de modo a enviar os arquivos do próximo segmento antes do cliente solicitar. Considerando um contexto de visualização de vídeo 360º, a tendência é que não haja uma alta variação nos *tiles* do campo de visão de um

segmento para o outro e portanto o *server push* pode auxiliar na transmissão. No entanto, vale ressaltar que essa política se torna ineficiente nos momentos de variações altas nos *tiles* do campo de visão, uma vez que essas variações irão acarretar em previsões incorretas do servidor e consequentemente tráfego desnecessário.

5.2 Cliente

Assim como o servidor, o cliente também foi desenvolvido em Pyhton com a biblioteca Aioquic. Diferentemente do servidor que roda em *loop*, o cliente é encerrado após o término da transmissão do vídeo.

Para executar o cliente, é necessário informar o endereço do servidor e o algoritmo dash a ser utilizado. Uma vez executado, o cliente irá disparar duas *threads* que serão executadas de forma paralela.

A primeira *thread* é a responsável por requisitar os arquivos ao servidor, enquanto que a segunda é responsável por receber os arquivos do servidor, inclusive os arquivos recebidos via *server push*.

Para simular a entrada do usuário, ou seja, as mudanças nas direções de visualização, utilizamos um arquivo CSV gerado a partir do estudo "360° Video Viewing Dataset in HeadMounted Virtual Reality"[7] que identificou padrões de movimentos dos usuários em vídeos 360º do YouTube.

O arquivo contém os *tiles* no campo de visão para cada *frame* de um vídeo, e é por meio dele que o cliente irá identificar os *tiles* de maior e menor prioridade.

Além disso, antes de requisitar o servidor, para cada segmento o cliente irá executar o algortimo DASH escolhido que tem como saída o *bitrate* a ser utilizado no segmento. Para que os algoritmos decidam qual o próximo *bitrate*, o tempo de *download* dos arquivos bem como o tamanho dos arquivos são armazenados pela *thread* de recebimento de arquivos.

Após a definição do *bitrate*, o cliente requisita os 200 *tiles* com prioridade 1 para os *tiles* no campo de visão e prioridade 2 para os *tiles* fora do campo de visão.

Para avaliarmos o desempenho da transmissão é importante coletarmos a métrica de *missing ratio*, que é dada pela razão da quantidade de *tiles* perdidos pela quantidade de *tiles* total. No nosso experimento, o vídeo possui 30 *frames* por segundo, de modo que cada *frame* tem uma duração de aproximadamente 33ms. Embora as requisições sejam feitas por segmentos, precisamos calcular o *missing ratio* em cada *frame* para posteriormente calcularmos o *missing ratio* total da transmissão. Para isso, o cliente simula a reprodução do *frame* aguardando o tempo de duração de cada *frame* antes de verificar o *missing ratio*, ou seja, antes de verificar se o arquivo correspondente ao *tile* e ao *frame* já foi baixado. Desse modo, conseguimos calcular o *missing ratio* de forma justa, considerando o tempo de reprodução do vídeo.

Por fim, após a requisição de todos os segmentos e as esperas para simulação de reprodução, o cliente é encerrado e as métricas finais são calculadas.

6 Experimentos

Como dito anteriormente, iremos utilizar os primeiros 50 segundos do vídeo "Chariot Race" [5], divididos em 6 segmentos de 1 segundo e 200 *tiles*, com os seguintes *bitrates*: 4.156Mbps, 7.033Mbps e 9.721Mbps. Além disso, iremos avaliar os algoritmos *basic_dash* e *basic_dash2* do repositírio AStream[1]. Para isso, iremos utilizar o Mininet[9] para emular uma rede e avaliar o desempenho em diferentes combinações de carga, banda e atraso:

- **Carga**: 10% e 30%
- **Banda**: 10Mpbs e 8Mbps
- **Atraso**: 5ms, 50ms, 75ms, 100ms, 125ms e 150ms

Além disso, iremos considerar uma taxa de perda da rede de 2% em todos os cenários. A Figura 1 representa a tepologia de rede utilizada no Mininet.

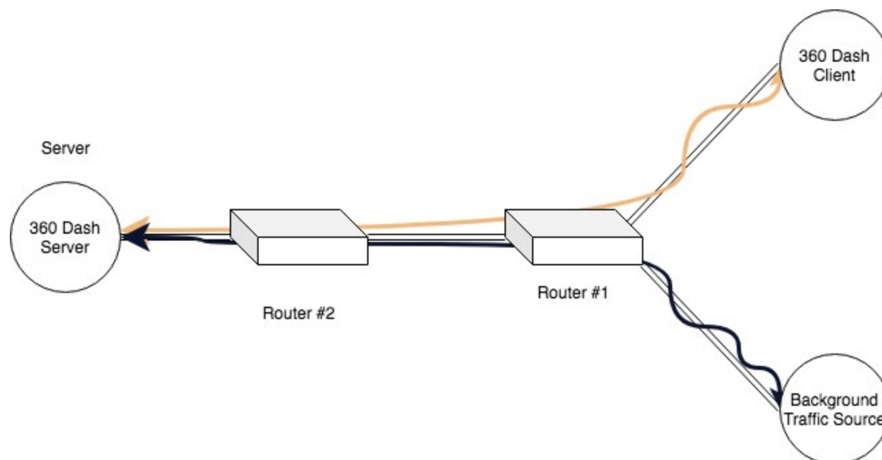

Figura 1: Topologia da rede Mininet.

Para cada combinação de carga, banda e atraso, iremos coletar as seguintes métricas:

- ***Bitrate* médio**: Média dos *bitrates* utilizados na transmissão, considerando todos os segmentos.
- ***Missing ratio* total**: Razão do número de *tiles* perdidos pelo número total de *tiles*.
- ***Missing ratio* no campo de visão**: *Missing ratio* apenas para os segmentos no campo de visão do usuário.
- **Contagem de *rebuffering***: Quantidade de vezes que foi necessário *rebuffering*.
- **Tempo de *rebuffering***: Tempo total gasto em *rebuffering*.
- **Uso do Canal**: Porcentagem de utilização do canal de transmissão.

7 Resultados

Avaliamos o desempenho dos algoritmos *basic_dash* e *basic_dash2* para todas as combinações possíveis de carga, banda e atraso. Para uma melhor análise dos resultados, foram gerados gráfricos comparativos para as métricas definidas anteriormente.

7.1 *Bitrate* médio

O *Bitrate* médio (de todos os experimentos) do algortimo *basic_dash* foi de 8.16Mbps enquanto que o do algortimo *basic_dash2* foi de 9.5Mpbs.

7.2 *Missing ratio* total (%) x Atraso (ms)

Banda 10Mbps / Carga 10%

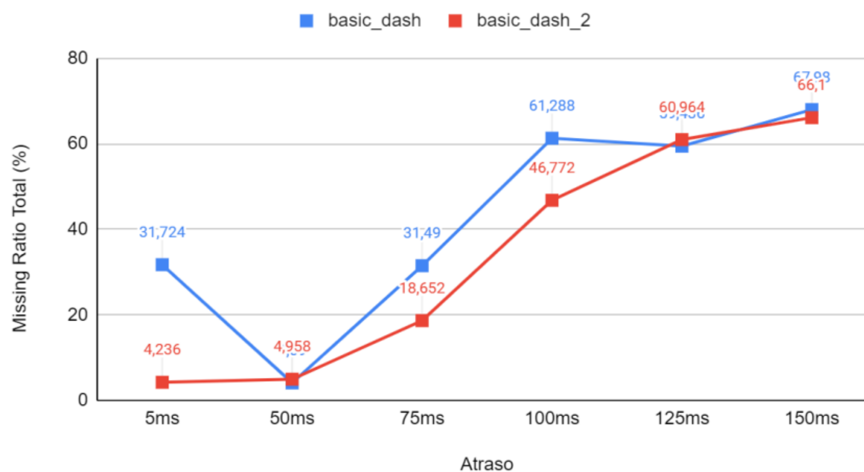

Figura 2: *Missing ratio* total (%) x Atraso (ms) - Banda 10Mbps / Carga 10%

Banda 8Mbps / Carga 10%

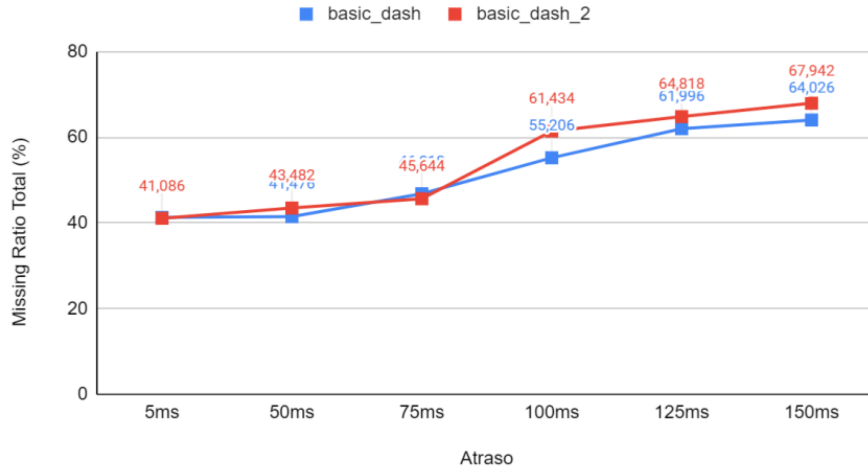

Figura 3: *Missing ratio* total (%) x Atraso (ms) - Banda 8Mbps / Carga 10%

Banda 10Mbps / Carga 30%

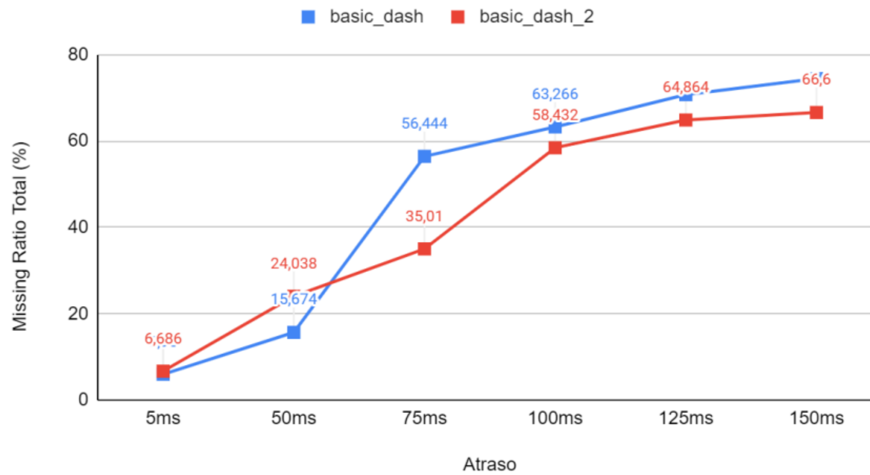

Figura 4: *Missing ratio* total (%) x Atraso (ms) - Banda 10Mbps / Carga 30%

Banda 8Mbps / Carga 30%

Figura 5: *Missing ratio* total (%) x Atraso (ms) - Banda 8Mbps / Carga 30%

7.3 *Missing ratio* no campo de visão (%) x Atraso (ms)

Banda 10Mbps / Carga 10%

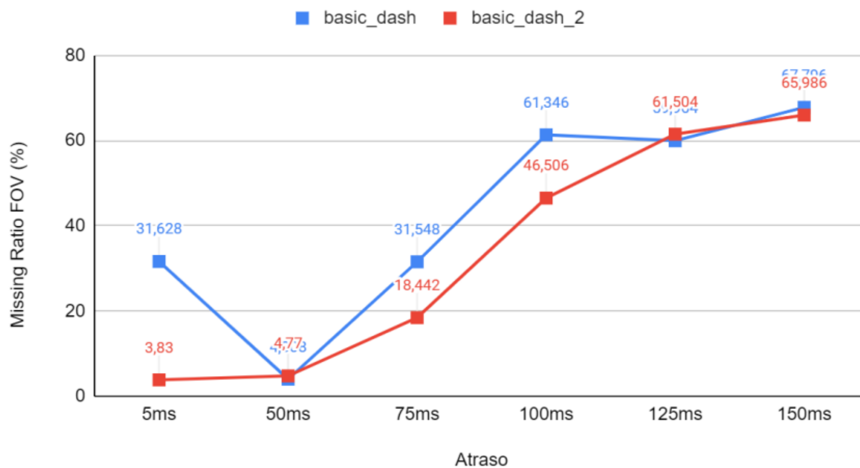

Figura 6: *Missing ratio* no campo de visão (%) x Atraso (ms) - Banda 10Mbps / Carga 10%

Banda 8Mbps / Carga 10%

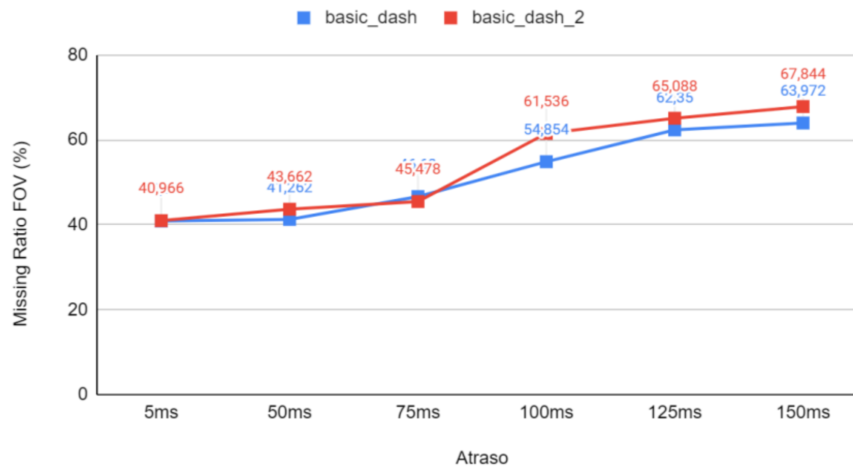

Figura 7: *Missing ratio* no campo de visão (%) x Atraso (ms) - Banda 8Mbps / Carga 10%

Banda 10Mbps / Carga 30%

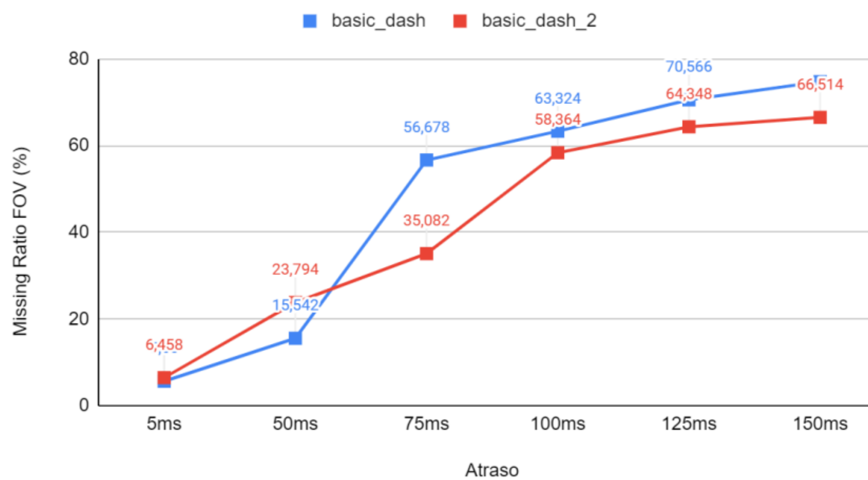

Figura 8: *Missing ratio* no campo de visão (%) x Atraso (ms) - Banda 10Mbps / Carga 30%

Banda 8Mbps / Carga 30%

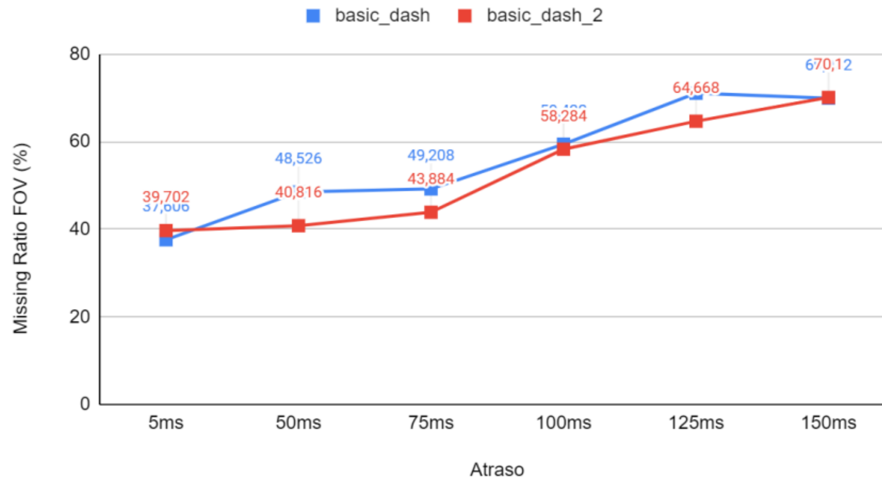

Figura 9: *Missing ratio* no campo de visão (%) x Atraso (ms) - Banda 8Mbps / Carga 30%

7.4 Contagem de Rebuffering x Atraso (ms)

Banda 10Mbps / Carga 10%

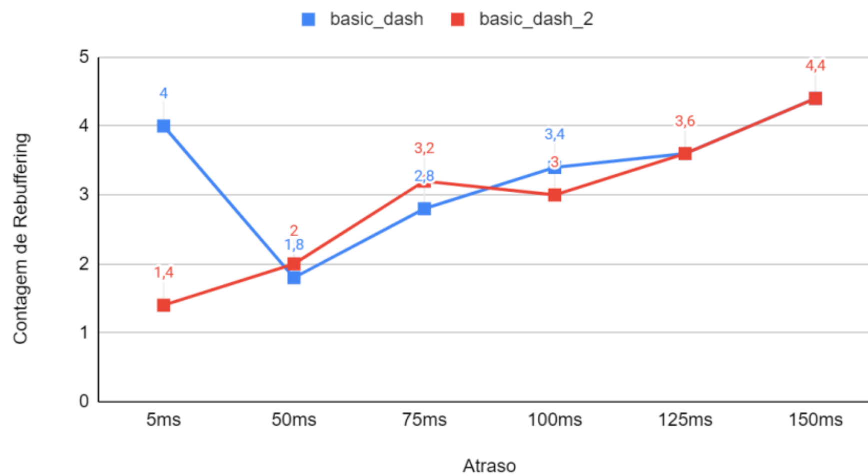

Figura 10: Contagem de Rebuffering x Atraso (ms) - Banda 10Mbps / Carga 10%

Banda 8Mbps / Carga 10%

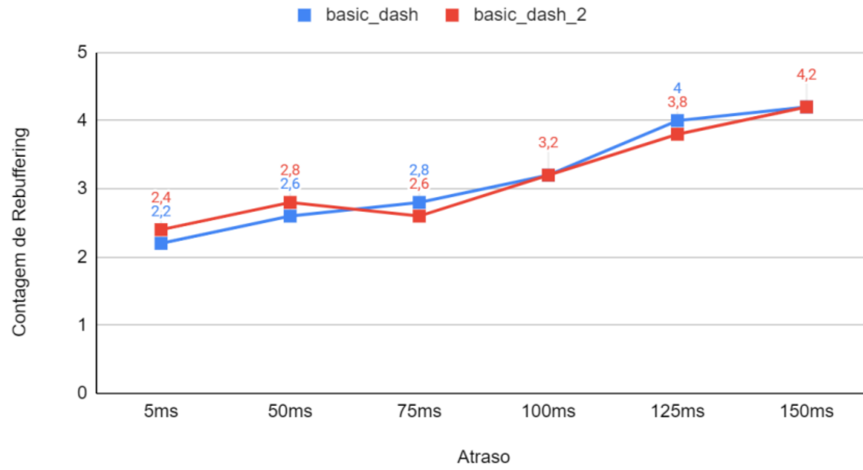

Figura 11: Contagem de Rebuffering x Atraso (ms) - Banda 8Mbps / Carga 10%

Banda 10Mbps / Carga 30%

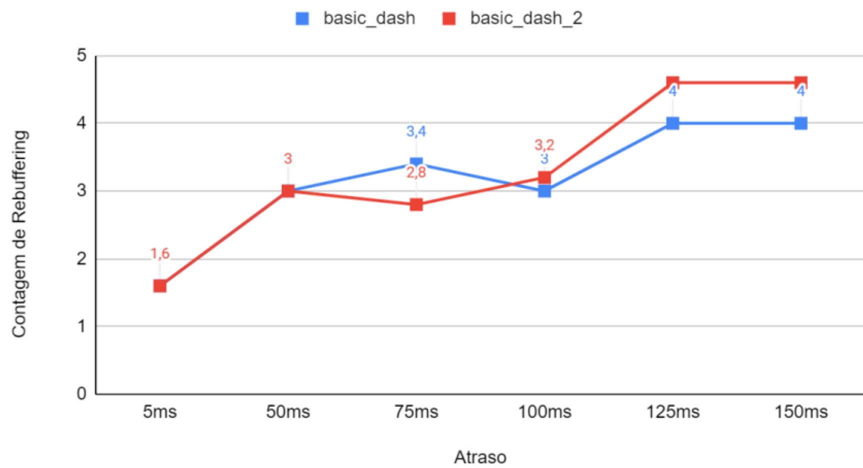

Figura 12: Contagem de Rebuffering x Atraso (ms) - Banda 10Mbps / Carga 30%

Banda 8Mbps / Carga 30%

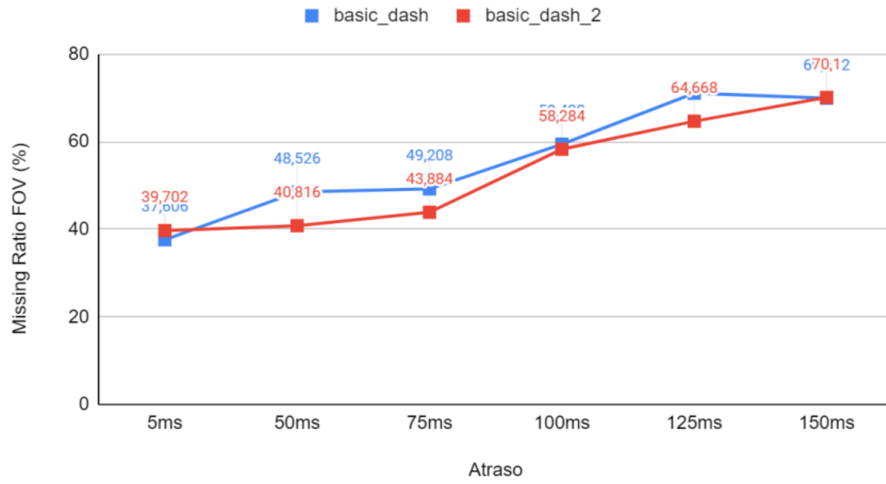

Figura 13: Contagem de Rebuffering x Atraso (ms) - Banda 8Mbps / Carga 30%

7.5 Tempo de Rebuffering (s) x Atraso (ms)

Banda 10Mbps / Carga 10%

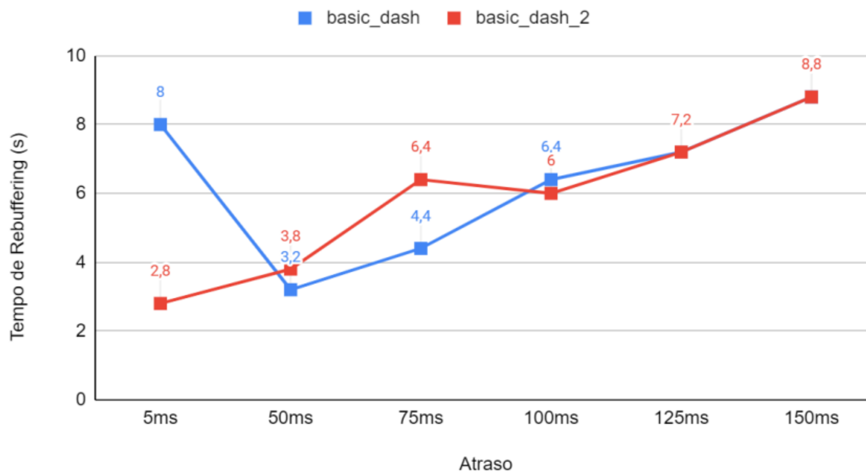

Figura 14: Tempo de Rebuffering (s) x Atraso (ms) - Banda 10Mbps / Carga 10%

Banda 8Mbps / Carga 10%

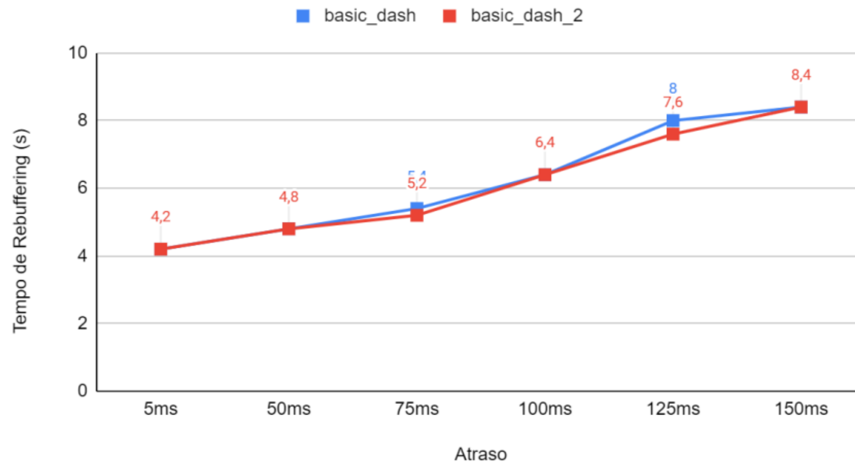

Figura 15: Tempo de Rebuffering (s) x Atraso (ms) - Banda 8Mbps / Carga 10%

Banda 10Mbps / Carga 30%

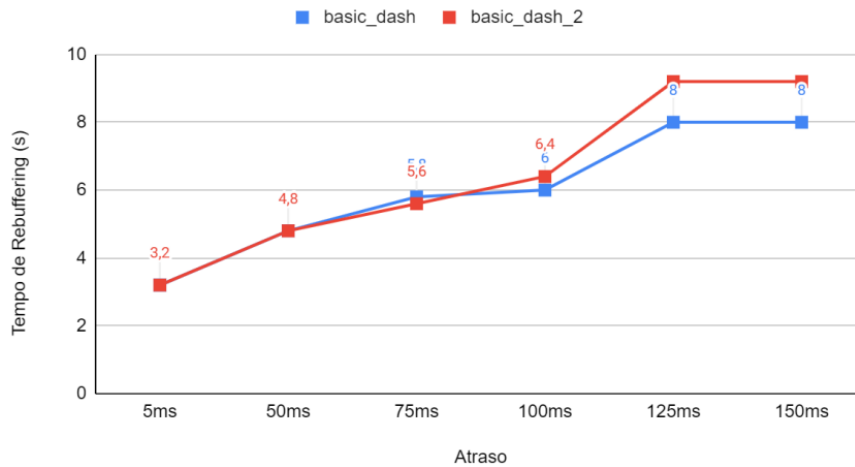

Figura 16: Tempo de Rebuffering (s) x Atraso (ms) - Banda 10Mbps / Carga 30%

Banda 8Mbps / Carga 30%

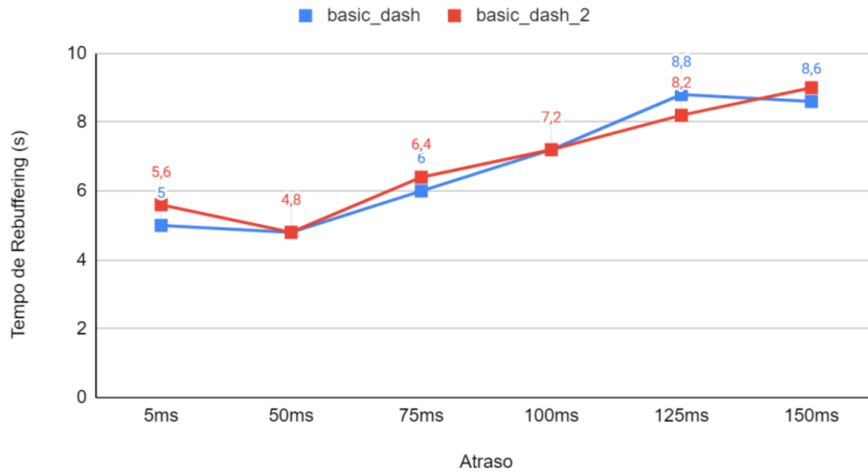

Figura 17: Tempo de Rebuffering (s) x Atraso (ms) - Banda 8Mbps / Carga 30%

7.6 Uso do Canal (%) x Atraso (ms)

Banda 10Mbps / Carga 10%

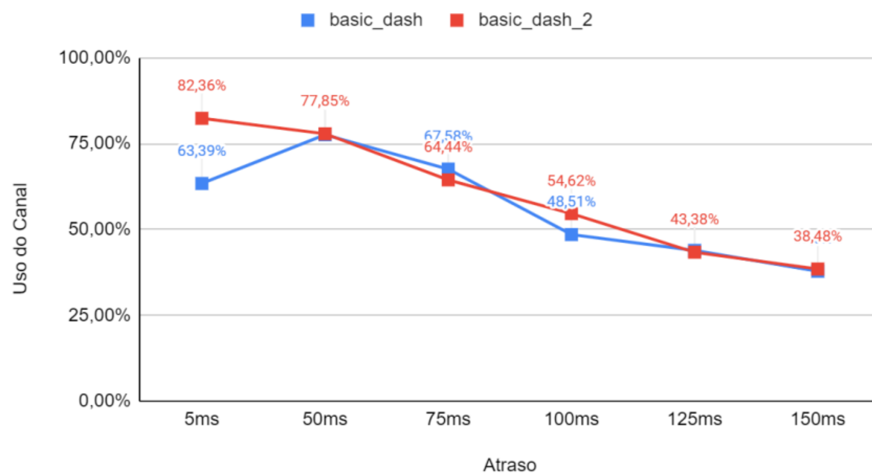

Figura 18: Uso do Canal (%) x Atraso (ms) - Banda 10Mbps / Carga 10%

Banda 8Mbps / Carga 10%

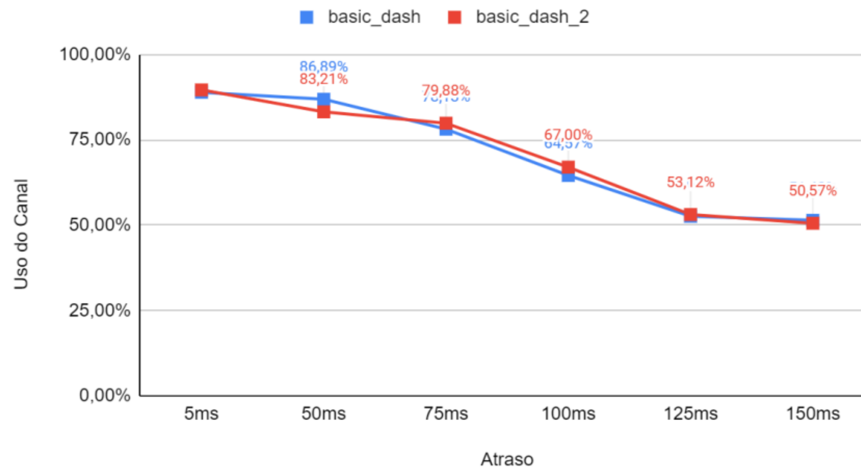

Figura 19: Uso do Canal (%) x Atraso (ms) - Banda 8Mbps / Carga 10%

Banda 10Mbps / Carga 30%

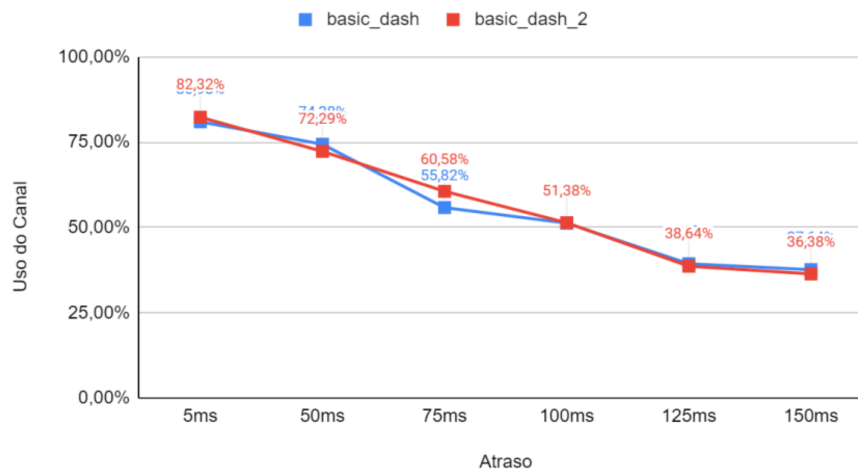

Figura 20: Uso do Canal (%) x Atraso (ms) - Banda 10Mbps / Carga 30%

Banda 8Mbps / Carga 30%

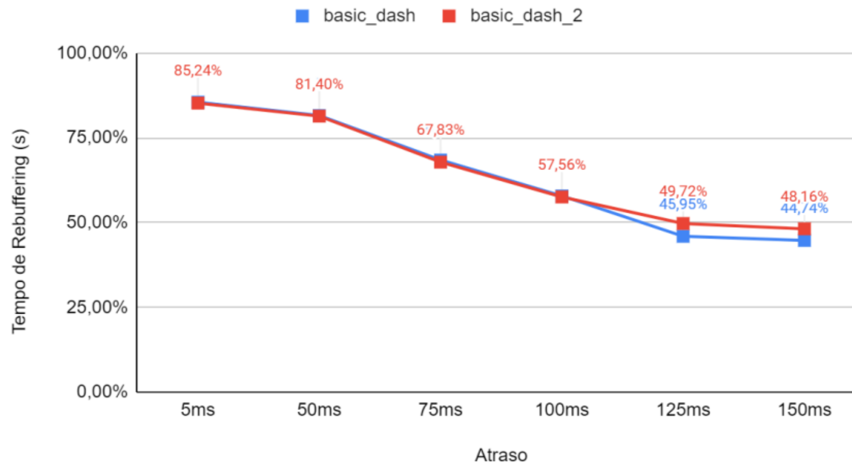

Figura 21: Uso do Canal (%) x Atraso (ms) - Banda 8Mbps / Carga 30%

8 Conclusões

A primeira conclusão obtida com os resultados é que o algortimo *basic_dash2* obteve um *bitrate* médio 16% acima do algoritmo *basic_dash*, o que indica uma melhor qualidade visual do vídeo a ser reproduzido.

Analisando a métrica de *Missing ratio* total, podemos perceber que o algortimo *basic_dash2* obteve melhores resultados em praticamente todos os cenários, exceto no cenário de banda de 8Mbps e carga de 10% em que o algortimo *basic_dash* obteve resultados ligeiramente melhores.

Em relação a métrica de *Missing ratio* no campo de visão, os resultados foram muito parecidos, com diferenças de no máximo 1%. Isso indica que a política de priorização utilizada (WFQ - *Weighted Fair Queuing*) não se mostrou satisfatória, uma vez que era esperado uma diminuição significativa para o *missing ratio* no campo de visão.

Em relação ao *rebuffering*, os resultados obtidos não indicam superioridade de nenhum algoritmo, podemos perceber que tanto na quantidade quanto no tempo total de *rebuffering* as linhas são similares, as vezes até sobrepostas. Sendo assim, podemos concluir que a escolha do algoritmo não possui grande impacto no *rebuffering*.

Assim como o *rebuffering*, o uso do canal se mostrou muito parecido para ambos os algoritmos, independente da combinação de carga, banda e atraso.

Sendo assim, podemos concluir que mesmo com um *bitrate* médio superior, o algoritmo *basic_dash2* obteve um *missing ratio* menor do que o algoritmo *basic_dash*, demonstrando ser um algoritmo mais eficiente para a transmissão de vídeos 360º com taxa de bits adaptável.

Referências

[1] Parikshit Juluri. AStream: A rate adaptation model for DASH. Disponível em: `https://github.com/pari685/AStream`. Acesso em: 12 de dez. de 2021.

[2] Gustavo Fernandez. 360º Video Streaming over QUIC. Disponível em: `https://github.com/gufernandez/aioquic-360-video-streaming`. Acesso em: 12 de dez. de 2021.

[3] WEI, Wenjia, et al. MP-VR: An MPTCP-Based Adaptive Streaming Framework for 360-degree Virtual Reality Videos. In: ICC 2021-IEEE International Conference on Communications. IEEE, 2021. p. 1-6.

[4] SAIF, Darius; LUNG, Chung-Horng; MATRAWY, Ashraf. An early benchmark of quality of experience between http/2 and http/3 using lighthouse. In: ICC 2021-IEEE International Conference on Communications. IEEE, 2021. p. 1-6.

[5] Paramout Pictures at Youtube. BEN-HUR (2016) - Chariot Race 360° Video - Paramount Pictures. Disponível em: `https://www.youtube.com/watch?v=jMyDqZe0z7M`. Acesso em: 12 de dez. de 2021.

[6] Daniel Stenberg. HTTP/3 explained. Ementa (descrição). Disponível em: `https://http3-explained.haxx.se/`. Acesso em: 12 de dez. de 2021.

[7] W. Lo, C. Fan, J. Lee, C. Huang, K. Chen, C. Hsu. 360° Video Viewing Dataset in Head-Mounted Virtual Reality. (2017)

[8] Python Software Foundation. "Python Language Reference, version 3.9.7". Disponível em: `http://www.python.org/`. Acesso em: 12 de dez. de 2021.

[9] Mininet Project Contributors. An Instant Virtual Network on your Laptop (or other PC). Disponível em: `http://mininet.org/`. Acesso em: 12 de dez. de 2021.