



Projeto de Programação Algoritmos Adaptativos de Streaming de Video MPEG-DASH

1 Introdução

Dynamic Adaptive Streaming over HTTP (DASH), também conhecido como **MPEG-DASH**, é uma técnica adaptativa de streaming de taxa de bits de video, que permite o streaming de alta qualidade de conteúdo de mídia pela Internet, entregues a partir de servidores HTTP web convencionais. O **MPEG-DASH** funciona dividindo o conteúdo em uma sequência de pequenos segmentos, que são servidos por HTTP. Cada segmento contém um curto intervalo de tempo de reprodução de conteúdo. A união de todos esses segmentos pode formar o conteúdo de um filme ou a transmissão ao vivo de um evento esportivo. De acordo com protocolo **MPEG-DASH**, um conteúdo é disponibilizado em uma variedade de taxas de bits diferentes. Enquanto o conteúdo está sendo reproduzido por um cliente **MPEG-DASH**, ele utiliza um algoritmo de adaptação de taxa de bits (ABR)[3] para selecionar automaticamente o segmento com a maior taxa de bits possível que pode ser baixado, sem causar travamentos ou re-buffering na reprodução. Portanto, um cliente **MPEG-DASH** adapta-se às mudanças nas condições da rede e fornece a reprodução de alta qualidade com poucos eventos de bloqueio ou re-buffering [6].

A Figura 1 ilustra um cenário simples de streaming entre um servidor HTTP e um cliente DASH. Os formatos e funcionalidades dos blocos em vermelho são definidos pelo padrão **MPEG-DASH**. Os blocos em verde são objeto deste projeto. Observe que o servidor HTTP não possui nenhuma funcionalidade especial neste contexto, ele apenas atua como repositório e meio para a obtenção dos arquivos MPD (XML contendo a descrição do conteúdo multimídia armazenado) e dos arquivos contendo os segmentos do conteúdo multimídia. No lado do cliente DASH, basicamente existem dois blocos para realizar o parsing dos ar-



quivos baixados, outro para a comunicação com o servidor HTTP e, finalmente, os dois blocos principais que são o Tocador de Mídia e o módulo de Controle que inclui o algoritmo de adaptação de taxa de bits (ABR).

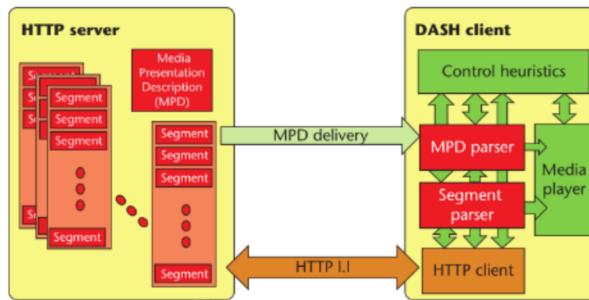


Figura 1: Padrão MPEG-DASH [5].

Para ter uma visão geral sobre o padrão **MPEG-DASH**, acesse o site [1] e leia o artigo [5]. Na aula disponível em [2] também são abordados os conceitos de DASH.

2 Objetivo

O objetivo do trabalho de programação é o desenvolvimento do Algoritmo ABR em um cliente DASH.

3 Especificação

O padrão **MPEG-DASH** define uma infinidade de formatos e funcionalidades necessários para abranger todos os cenários de streaming de conteúdo multimídia [4]. Casos de uso mais avançados podem incluir e alternar entre várias visualizações de câmera, streaming de conteúdo multimídia 3D, streams de vídeo com legendas, inserção dinâmica de anúncios, streaming ao vivo de baixa latência, streaming misto e reprodução de conteúdo pré-armazenado entre outros [5].

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



//45.171.101.167/DASHDataset/) segmentado de 6 maneiras diferentes (vide Fig. 2) e codificado em 20 formatos distintos, variando de uma taxa de bits de 46980bps até 4726737bps (vide Fig. 3 e Fig. 4).

Index of /datasets/DASHDataset2014/BigBuckBunny

Name	Last modified	Size	Description
Parent Directory	-	-	
1sec/	2014-10-16 14:08	-	
2sec/	2014-10-16 14:09	-	
3sec/	2014-10-16 14:09	-	
4sec/	2014-10-16 14:10	-	
5sec/	2014-10-16 14:10	-	
6sec/	2014-10-16 14:10	-	
10sec/	2014-10-16 14:10	-	
15sec/	2014-10-16 14:10	-	

Figura 2: Possibilidades de duração de segmentos.

Index of /datasets/DASHDataset2014/BigBuckBunny/1sec

Name	Last modified	Size	Description
Parent Directory	-	-	
BigBuckBunny_1s_onDemand_2014_05_09.mpd	2014-10-16 14:04	8.7K	
BigBuckBunny_1s_simple_2014_05_09.mpd	2014-10-16 14:04	4.3K	
bunny_46980bps/	2014-09-15 11:41	-	
bunny_91912bps/	2014-09-15 11:43	-	
bunny_135410bps/	2014-09-15 11:37	-	
bunny_187365bps/	2014-09-15 11:37	-	
bunny_226106bps/	2014-09-15 11:38	-	
bunny_270316bps/	2014-09-15 11:38	-	
bunny_357546bps/	2014-09-15 11:39	-	
bunny_424520bps/	2014-09-15 11:41	-	
bunny_537825bps/	2014-10-15 13:22	-	
bunny_620705bps/	2014-10-15 13:22	-	
bunny_808057bps/	2014-09-15 11:43	-	
bunny_1071529bps/	2014-09-15 11:36	-	
bunny_1312787bps/	2014-09-15 11:36	-	
bunny_1662809bps/	2014-09-15 11:37	-	
bunny_2234145bps/	2014-09-15 11:38	-	
bunny_2617284bps/	2014-09-15 11:38	-	
bunny_3305118bps/	2014-09-15 11:39	-	
bunny_3841983bps/	2014-09-15 11:40	-	
bunny_4242973bps/	2014-09-15 11:41	-	
bunny_4726737bps/	2014-09-15 11:42	-	

Figura 3: Arquivo MPD e possibilidades de codificação.

Em linhas gerais, o algoritmo ABR a ser desenvolvido receberá demandas do Tocador de Mídia para (i) baixar o arquivo MPD referente ao filme de interesse



Index of /datasets/DASHDataset2014/BigBuckBunny/1sec/bunny_46980bps

Name	Last modified	Size	Description
Parent Directory	-	-	
?BigBuckBunny_1s1.m4s	2014-09-10 13:14	6.1K	
?BigBuckBunny_1s2.m4s	2014-09-10 13:14	6.9K	
?BigBuckBunny_1s3.m4s	2014-09-10 13:14	3.8K	
?BigBuckBunny_1s4.m4s	2014-09-10 13:14	5.3K	
?BigBuckBunny_1s5.m4s	2014-09-10 13:14	6.1K	
?BigBuckBunny_1s6.m4s	2014-09-10 13:14	6.1K	
?BigBuckBunny_1s7.m4s	2014-09-10 13:14	6.6K	
?BigBuckBunny_1s8.m4s	2014-09-10 13:14	5.6K	
?BigBuckBunny_1s9.m4s	2014-09-10 13:14	6.4K	
?BigBuckBunny_1s10.m4s	2014-09-10 13:14	6.2K	
?BigBuckBunny_1s11.m4s	2014-09-10 13:14	6.3K	
?BigBuckBunny_1s12.m4s	2014-09-10 13:14	6.5K	

Figura 4: Segmentos para a codificação em 46980bps (lista truncada).

e (ii) baixar cada um dos segmentos do filme.

Cabe ao algoritmo ABR decidir qual segmento deverá ser baixado, considerando as alternativas de codificação disponíveis, de forma que a execução do vídeo ocorra com a melhor qualidade possível e com o mínimo de interrupções. Para tal, o ABR deve monitorar continuamente o desempenho da rede e/ou o buffer de reprodução do Tocador de Mídia para tomar a melhor decisão.

3.1 Arquivo MPD

Está disponível no servidor HTTP um arquivo MPD (*Media Presentation Description*) para cada padrão de segmentação (1s, 2s, 4s, 6s, 10s, e15s). Cada arquivo MPD é um arquivo XML que possui um formato padrão, conforme o exemplo mostrado na Fig. 5. Observe que neste MPD há apenas um *Period* e dentro deste apenas um *AdaptationSet*. Dentro deste último está definido o padrão dos segmentos e suas representações (codificações), conforme detalhes abaixo:

```
<SegmentTemplate
    timescale="96"
    media="bunny_${Bandwidth}bps/BigBuckBunny_1s$Number$.m4s"
    startNumber="1"
    duration="96"
    initialization="bunny_${Bandwidth}bps/BigBuckBunny_1s_init.mp4"
/>
```



```
<!-- MPD file Generated with GPAC version 0.5.1-DEV-rev5379 on 2014-09-10T13:14:57Z -->
<!MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500000s" type="static" mediaPresentationDuration="PT0H9M55.465" profiles="urn:mpeg:dash:profile:isoff-live:2011">
  <ProgramInformation moreInformationURI="http://gpac.sourceforge.net">
    <Title>dashed/BigBuckBunny_ls_simple_2014_05_09.mpd generated by GPAC</Title>
  </ProgramInformation>
  <Period duration="PT0H9M55.465">
    <AdaptationSet segmentAlignment="true" group="1" maxWidth="480" maxHeight="360" maxFrameRate="24" par="4:3">
      <SegmentTimeline timescale="96" media="group1.mpd" bandwidth="480" startNumber="1" endNumber="96" startNumber="1" duration="96"
        initialization="group1.mpd" bandwidth="480" startNumber="1" endNumber="96" startNumber="1" duration="96"
        initialisation="group1.mpd" bandwidth="480" startNumber="1" endNumber="96" startNumber="1" duration="96">
        <Representation id="320x240 47.0kbps" mimeType="video/mp4" codecs="avc1.42c00d" width="320" height="240" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="46980"/>
        <Representation id="320x240 92.0kbps" mimeType="video/mp4" codecs="avc1.42c00d" width="320" height="240" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="9191"/>
        <Representation id="320x240 135.0kbps" mimeType="video/mp4" codecs="avc1.42c00d" width="320" height="240" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="135419"/>
        <Representation id="480x360 182.0kbps" mimeType="video/mp4" codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="182366"/>
        <Representation id="480x360 226.0kbps" mimeType="video/mp4" codecs="avc1.42c00d" width="480" height="360" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="226106"/>
        <Representation id="480x360 270.0kbps" mimeType="video/mp4" codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="270218"/>
        <Representation id="480x360 314.0kbps" mimeType="video/mp4" codecs="avc1.42c00d" width="480" height="360" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="314329"/>
        <Representation id="480x360 425.0kbps" mimeType="video/mp4" codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="424529"/>
        <Representation id="654x488 538.0kbps" mimeType="video/mp4" codecs="avc1.42c01e" width="654" height="488" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="537825"/>
        <Representation id="654x488 621.0kbps" mimeType="video/mp4" codecs="avc1.42c01e" width="654" height="488" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="620785"/>
        <Representation id="1280x720 808.0kbps" mimeType="video/mp4" codecs="avc1.42c01f" width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="888857"/>
        <Representation id="1280x720 1.1Mbps" mimeType="video/mp4" codecs="avc1.42c01f" width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="1071529"/>
        <Representation id="1280x720 1.3Mbps" mimeType="video/mp4" codecs="avc1.42c01f" width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="1312788"/>
        <Representation id="1280x720 1.5Mbps" mimeType="video/mp4" codecs="avc1.42c01f" width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="1563999"/>
        <Representation id="1920x1080 2.0Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="2234145"/>
        <Representation id="1920x1080 2.6Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="2617284"/>
        <Representation id="1920x1080 3.3Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="3305118"/>
        <Representation id="1920x1080 3.6Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="3841983"/>
        <Representation id="1920x1080 4.2Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="4242923"/>
        <Representation id="1920x1080 4.7Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="4726737"/>
    </AdaptationSet>
  </Period>
</MPD>
```

Figura 5: Exemplo de um arquivo MPD.

O mais importante neste trecho é identificar o padrão em *media* que informa o formato do caminho para acessar cada segmento, sendo que *Bandwidth* deve ser substituído pela taxa de interesse e *Number* deve ser substituído pelo número do segmento, iniciando em *startNumber*. Finalmente, *initialization* indica o segmento de inicialização do filme.

Em seguida, o arquivo MPD lista as diversas representações disponíveis para o filme (20 neste caso), indicando em *bandwidth* a taxa de bits desta codificação (necessária para compor o caminho para acessar os segmentos).

```
<Representation
  id="320x240 47.0kbps"
  mimeType="video/mp4"
  codecs="avc1.42c00d"
  width="320"
  height="240"
  frameRate="24"
  sar="1:1"
  startWithSAP="1"
  bandwidth="46980"
/>
```



4 Arquitetura

Para implementação e avaliação dos algoritmos ABR, foi proposta uma plataforma básica funcional chamada *pyDash*. O código para esta plataforma encontra-se versionado no repositório Git disponível em:

- <https://github.com/mfcaetano/pydash>

Nas seções seguintes abordaremos a sua arquitetura básica sob a perspectiva da implementação de um novo protocolo ABR usando esta ferramenta. Contudo, não iremos entrar nos detalhes de toda a modelagem feita para a implementação do *pyDash*. Pelo contrário, uma premissa do trabalho inclui a navegação pelo código disponibilizado e compreensão por conta própria das técnicas e padrões de projeto implementadas.

Antes de abordarmos como essa implementação é feita, vamos primeiramente analisar os principais componentes que fazem parte desta arquitetura.

4.1 Cliente DASH

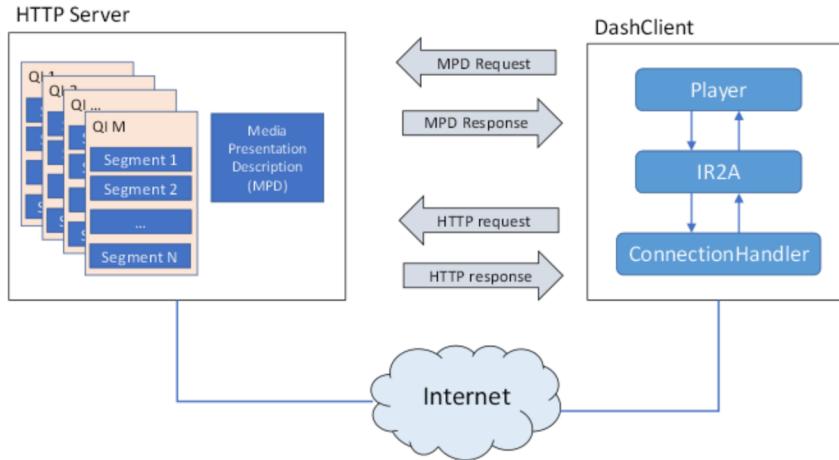


Figura 6: Arquitetura da Solução DashClient.

A Figura 6 apresenta a relação da classe `DashClient` e seus principais componentes, bem como sua interação com um servidor HTTP, que armazena e



publica vídeos seguindo o formato MPEG-DASH. A classe `DashClient` é formada por três atributos principais, que estão organizados em uma estrutura de pilha, são eles:

- `Player` – A classe `Player` implementa o comportamento do player de vídeo. Ou seja, implementa a estrutura do tipo `buffer`, que utiliza para fazer o controle sobre quais segmentos precisam ser recuperados do servidor HTTP e quais já foram armazenados. Dos segmentos armazenados, o `Player` rotula quais foram reproduzidos pelo usuário e quais ainda serão consumidos. Além disso, o `Player` monitora diversas métricas de desempenho, como exemplo: o número de pausas, o tamanho das pausas, as qualidades selecionadas, etc. O `Player` gera, também, as estatísticas das métricas monitoradas, as quais podem ser utilizadas para acompanhar o comportamento do `buffer` durante a execução do vídeo. O `Player` é responsável por definir qual será o próximo segmento a ser recuperado do servidor HTTP. Ele monta esta requisição, em uma mensagem do tipo `base.SSMessage`. Após montada, a mensagem do tipo `base.SSMessage` é encaminhada para a camada abaixo (`IR2A`).
- `IR2A` (*Interface of Rate Adaptation Algorithms*) – Classe Abstrata que define uma interface comum a ser implementada pelos novos algoritmos ABR. As interfaces definem métodos abstratos, os quais precisam ser obrigatoriamente implementados pelo algoritmo ABR para que o mesmo possa ser validado. Um algoritmo ABR válido pode ser carregado de forma automática na plataforma de avaliação `pyDash`. Após ser carregado, o algoritmo ABR se localizará entre as classes `Player` e `ConnectionHandler`, fazendo a intermediação entre as mensagens que descem (`request`) e sobem (`response`) na pilha. Da classe `Player`, o algoritmo ABR recebe a mensagem do tipo `base.SSMessage` e, baseado em sua lógica e políticas de seleção, define qual qualidade deverá ser utilizada na requisição de um segmento criando uma mensagem. O algoritmo encaminha essa mensagem para a camada de baixo (`base.SSMessage`). O trabalho a ser desenvolvido concentrar-se-á exclusivamente no algoritmo ABR que deverá implementar a `IR2A`.
- `ConnectionHandler` – classe responsável pela comunicação entre a plataforma `pyDash` e o servidor HTTP definido no arquivo `dash-client.json`. Converte uma requisição definida por uma mensagem do tipo `base.SSMessage` em uma conexão funcional HTTP (vice-versa). Ao receber uma mensagem do tipo `base.SSMessage`, vinda da camada superior, realiza uma conexão



no servidor HTTP e encaminha para camada superior a resposta obtida, utilizando para isso o formato `base.SSMessage`. Esta classe também implementa o algoritmo de *traffic shaping* destinado ao controle da banda e consequente teste do protocolo ABR implementado.

É importante destacar, que os códigos das classes `Player` e `ConnectionHandler` não devem ser alterados durante a implementação do trabalho. A sua implementação de um novo algoritmo ABR deve estar limitada somente a sua classe que herdará a `IR2A`, a qual deverá ser colocada na pasta `r2a`. Solicitações de correção ou adição de novas funcionalidades na plataforma pyDash devem ser encaminhadas ao professor, eventuais atualizações do código serão disponibilizadas no repositório Git. Os testes e a avaliação dos trabalhos serão feitos com o código disponível no repositório. Ou seja, **quaisquer alterações feitas nas demais classes da biblioteca `pyDash` não serão consideradas**.

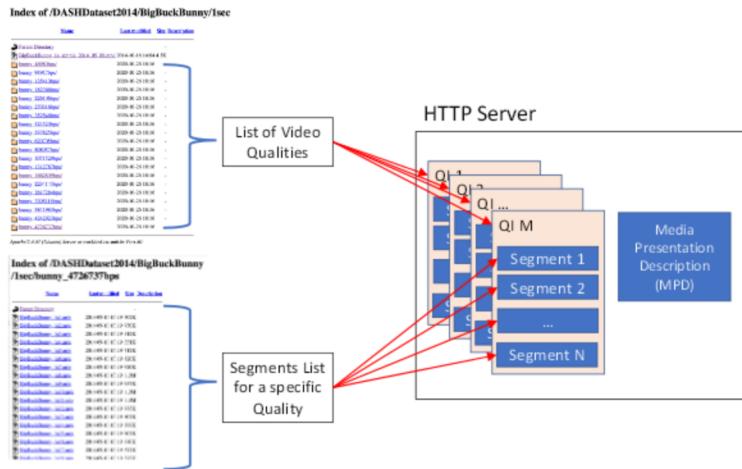


Figura 7: Detalhes da Arquitetura `DashClient` lado Servidor

Conforme discutido em sala, do lado do Servidor¹, os segmentos de vídeo estão organizados conforme apresentado pela Figura 7. Em nosso exemplo, estamos trabalhando com o vídeo *BigBuckBunny*, publicado em um servidor HTTP no seguinte endereço:

¹vídeo aula: DASH e CDN - <https://youtu.be/Tzm9uDi8ZG0>



- <http://45.171.101.167/DASHDataset/BigBuckBunny>

Como pode ser observado na Figura 8, o mesmo vídeo foi segmentado em tamanhos fixos diferentes. Considerando o tamanho de segmentos de 1 segundo, verifica-se 20 qualidades diferentes em que este vídeo foi codificado, sendo a menor qualidade de 46980 bps e a maior 4726737 bps. No escopo da plataforma *pyDash*, é disponibilizado um parser, para conteúdo **mpd**, implementado pela classe `pydash.player.mpd_node`. Entraremos a frente em maiores detalhes sobre o uso desta funcionalidade.

Index of /DASHDataset2014/BigBuckBunny

Name	Last modified	Size	Description
Parent Directory	-	-	-
1sec/	2020-10-26 18:16	-	
2sec/	2020-10-26 18:16	-	
4sec/	2020-10-26 18:16	-	
6sec/	2020-10-26 18:16	-	
10sec/	2020-10-26 18:16	-	
15sec/	2020-10-26 18:16	-	

Apache/2.4.41 (Ubuntu) Server at workbird.cic.unb.br Port 80

Figura 8: vídeo *BigBuckBunny* dividido em segmentos de tamanhos fixos.

É importante destacar que ao realizar o parser do conteúdo **mpd**, é possível extraír um vetor **QI** (*quality index*) contendo a lista de todas as qualidades em que o vídeo foi codificado. Onde, `qi[0]` indica a menor qualidade e `qi[19]` a maior. Por fim, para cada qualidade é possível encontrar os segmentos que representam o vídeo original (cópias do mesmo vídeo, codificado em qualidades diferentes). Para o vídeo *BigBuckBunny*, com tamanhos de segmento de 1 segundo, por exemplo, temos 596 arquivos que compõem o vídeo original.

A Figura 9 apresenta em maiores detalhes os atributos que formam a classe `DashClient`, bem como a sua interação. Ao iniciar a execução da plataforma *pyDash*, a classe `Player` iniciará o processo montando uma requisição para recuperar o arquivo descriptor `mpd`. Para isso, obterá a url para o `mpd` através do arquivo `dash_client.json` (`url_mpd`). Esta requisição será montada utilizando uma mensagem do tipo `base.Message`. Esta mensagem será encaminhada para camada de baixo, conforme indicado pela figura. O algoritmo ABR, carregado no local do `IR2A`, receberá essa requisição através do método sobrescrito `handle_xml_request(self, msg)`. Neste ponto será a oportunidade do novo algoritmo ABR executar alguma operação relacionada com a requisição deste arquivo descriptor `mpd`, para em seguida encaminhar a mensagem (alterada ou não) para camada de baixo, utilizando para isso a função `self.send_down(msg)`. A men-

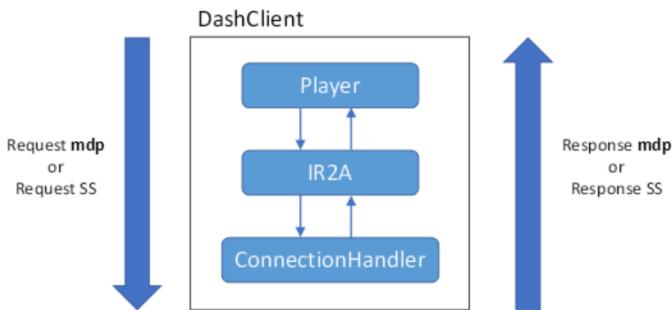


Figura 9: A arquitetura `DashClient` e sua relação entre as entidades

sagem será recebida pela classe `ConnectionHandler`, onde será lida e convertida em uma requisição HTTP, onde o arquivo *mpd* será efetivamente recuperado. Sim, você pode acompanhar esta requisição utilizando para isso o *Wireshark*.

Ao recuperar o conteúdo do arquivo *mpd*, a classe `ConnectionHandler` encaminhará à camada superior o seu conteúdo, devidamente encapsulado em uma mensagem do tipo `base.Message`. Este conteúdo será recebido pelo algoritmo ABR, através do método sobrescrito `handle_xml_response(self, msg)`. Novamente, o algoritmo ABR terá a oportunidade de executar alguma instrução, como por exemplo, extrair a lista de qualidades possíveis do arquivo *mpd*. Em seguida, a mensagem deverá ser enviada a camada superior através da função `self.send_up(msg)`.

A classe `Player` de posse do conteúdo do arquivo *mpd*, iniciará o processo de recuperação dos segmentos de vídeos (ss). A partir deste ponto, montará mensagens do tipo `base.SSMMessage`, informando qual o próximo segmento deverá ser recuperado. Contudo, sem informar a qualidade deste segmento, uma vez que esta é uma atribuição do algoritmo ABR. A mensagem será encaminhada à camada de baixo, e será recebida pelo algoritmo ABR através do método sobrescrito `handle_segment_size_request(self, msg)`. Na implementação deste método, o algoritmo definirá qual qualidade será utilizada na requisição HTTP. Esta escolha é feita através do método `msg.add_quality_id(x)`, onde x é um valor inteiro positivo que representa a qualidade em *bps*. Por fim, após a execução das instruções necessárias para o funcionamento correto da abordagem implementada no algoritmo ABR, a mensagem será encaminhada à camada de baixo através da execução da função `self.send_down(msg)`. Da mesma forma como aconteceu com a requisição ao arquivo *mpd*, a classe `ConnectionHandler`, converterá a mensagem



msg em uma requisição HTTP, onde o segmento de vídeo *ss* será efetivamente recuperado. Lembre-se, novamente, pode acompanhar esta requisição utilizando para isso o *Wireshark*.

Ao receber o segmento de vídeo *ss*, a classe `ConnectionHandler` iniciará o processo de subida na pilha desta informação, de forma que o segmento *ss* chegue até a classe `Player`. A representação do segmento *ss*, no formato da classe `base.SSMessage`, será recebido pelo algoritmo ABR, através do método sobrescrito `handle_segment_size_response(self, msg)`. Novamente, o algoritmo ABR poderá executar as instruções necessárias para o seu correto funcionamento, finalizando esta execução, ao enviar a mensagem *msg* para a camada superior, através da função `self.send_up(msg)`. Ao receber o segmento, a classe `Player` armazenará o seu conteúdo no *buffer* e computará as estatísticas necessárias, dando continuidade a execução da plataforma *pyDash*, através da requisição do próximo segmento de vídeo.

4.2 Implementando Um Novo Algoritmo ABR

Conforme descrito na seção anterior, a arquitetura da plataforma *pyDash* está organizada em três camadas. Na camada do meio, encontra-se a classe abstrata `IR2A`. Esta classe desempenha um papel de interface para os novos algoritmos ABR a serem incorporados à plataforma. Para isso, é necessária a implementação desta interface, conforme será descrito nesta seção.

A Figura 10 apresenta a relação de como um novo algoritmo ABR pode ser acoplado dinamicamente à plataforma *pyDash*. Implementados todos os métodos abstratos definidos pela classe `IR2A`, a plataforma carrega de forma dinâmica o objeto da nova classe (`R2ANewAlgorithm1`), obtendo o seu nome pelo campo `r2a_algorithm` presente no arquivo *dash-client.json*. Note que para este procedimento ser bem sucedido, é mandatório que a classe que implementa o novo algoritmo ABR, esteja armazenada na pasta `r2a` e tanto o nome da classe quanto o nome do arquivo sejam os mesmos (classe `R2ANewAlgorithm1` contida no arquivo `r2a/r2anewalgorithm1.py`).

A Figura 11 apresenta os métodos abstratos da classe `IR2A`, a serem implementados por um novo algoritmo ABR. Esta nova política deverá ser implementada em uma classe que herdará os atributos de `IR2A`. Como exemplo de algoritmos, na pasta `r2a` encontram-se duas políticas: `R2AFixed` e `R2ARandom`. Na primeira, o algoritmo ABR sempre seleciona a mesma qualidade, independentemente da realidade da capacidade da rede. Na segunda abordagem, a escolha da qualidade é feita de forma aleatória. Não é preciso mencionar que estas abordagens são triviais, ou até mesmo desastrosas, como políticas ABR.

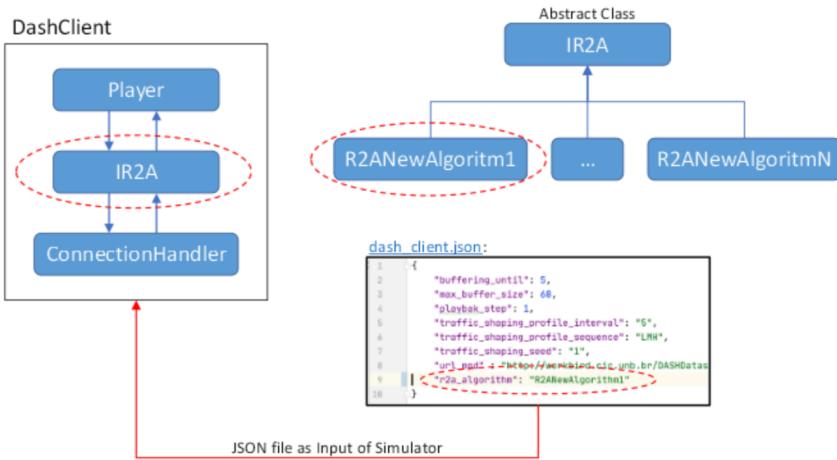


Figura 10: Detalhes da Arquitetura `DashClient` lado Cliente

Ao implementar os métodos abstratos definidos na Figura 11, é necessário considerar os atributos e o comportamento descritos na seção anterior e representados na Figura 9. Neste contexto, os métodos para tratamento de requisições (`handle_segment_size_request(self, msg)` e `handle_xml_request(self, msg)`) são oriundos da camada superior (`Player`) e suas implementações devem ser finalizadas encaminhando a mensagem `msg` para camada inferior (`ConnectionHandler`), utilizando para isso o método `self.send_down(msg)`. Por outro lado, os métodos para tratamento das respostas (`handle_segment_size_response(self, msg)` e `handle_xml_response(self, msg)`) são oriundos da camada inferior (`ConnectionHandler`) e suas implementações devem ser finalizadas encaminhando a mensagem `msg` para camada superior (`Player`), utilizando para isso o método `self.send_up(msg)`.

Com relação aos métodos abstratos apresentados na Figura 11, seguem abaixo as suas respectivas descrições:

- `handle_xml_request(self, msg)` – método recebe como parâmetro uma mensagem `msg` do tipo `base.Message`, que representa a requisição ao arquivo `mpd`. Mensagem gerada na camada superior (`Player`), deve ser encaminhada a camada inferior (`ConnectionHandler`) ao final da execução deste método (`self.send_down(msg)`).



```
class IR2A(SimpleModule):
    ...

    @abstractmethod
    def handle_xml_request(self, msg):
        pass

    @abstractmethod
    def handle_xml_response(self, msg):
        pass

    @abstractmethod
    def handle_segment_size_request(self, msg):
        pass

    @abstractmethod
    def handle_segment_size_response(self, msg):
        pass

    @abstractmethod
    def initialize(self):
        ...
        pass

    @abstractmethod
    def finalization(self):
        ...
        pass
```

Figura 11: Métodos abstratos da classe `IR2A`.

- `handle_xml_response(self, msg)` – método recebe como parâmetro uma mensagem `msg` do tipo `base.Message`, que representa a resposta à requisição ao arquivo `mpd`. Ou seja, o *payload* desta mensagem é o conteúdo xml do arquivo `mpd` recuperado do servidor HTTP. Seu conteúdo pode ser acessado pelo método `msg.get_payload()`. Para obter-se a lista de qualidades disponíveis, é possível utilizar as funcionalidades do parser `mpd` disponibilizado pela plataforma `pyDash`. Abaixo é apresentado um exemplo de



código em que a lista de qualidades (`self.qi`) é criada através do parser feito no conteúdo do arquivo *mpd*.

```
from player.parser import *
...
self.parsed_mpd = parse_mpd(msg.get_payload())
self.qi = self.parsed_mpd.get_qi()
```

Ao final da execução do método `handle_xml_response(...)`, a mensagem *msg* deve ser encaminhada à camada superior (`Player`) através do uso da função (`self.send_up(msg)`).

- `handle_segment_size_request(self, msg)` – método recebe como parâmetro uma mensagem *msg* do tipo `base.SSMessage`, que representa a requisição a um segmento de vídeo (ss). Esta requisição contém o número do segmento de vídeo solicitado pela classe `Player`, contudo não é definida a qualidade que este segmento deve ser requisitado. Na implementação desta função, a qualidade deve ser definida. A definição de qualidade é feita através do método `msg.add_quality_id(...)`, seguindo a política ABR implementada. O código abaixo apresenta um exemplo de implementação do método `handle_segment_size_request(self, msg)`, seguindo uma política fixa para definição de qualidade. Neste exemplo, o algoritmo ABR sempre define a maior qualidade possível das 20 disponíveis.

```
def handle_segment_size_request(self, msg):
    msg.add_quality_id(self.qi[19])
    self.send_down(msg)
```

Por fim, a mensagem gerada na camada superior (`Player`), deve ser encaminhada à camada inferior (`ConnectionHandler`) ao final da execução deste método (`self.send_down(msg)`).

- `handle_segment_size_response(self, msg)` – método recebe como parâmetro uma mensagem *msg* do tipo `base.SSMessage`, que representa a resposta para a requisição de um segmento de vídeo específico. Ou seja, nesta mensagem encontra-se todas as informações necessárias para a representação do segmento de vídeo requisitado. Dentre as inúmeras medições passíveis de serem computadas em um algoritmo ABR, é possível obter o tamanho do segmento recebido através do método `msg.get_bit_length()` e calcular a diferença de tempo transcorrido entre o momento da requisição realizada (`handle_segment_size_request()`) e a resposta recebida



(`handle_segment_size_response()`). Com isso, o algoritmo ABR é capaz de calcular a taxa de transferência obtida para a mensagem `msg` recebida. Por fim, ao final da execução do método, a mensagem `msg` deve ser encaminhada à camada superior (`Player`) através do uso da função `self.send_up(msg)`.

- `initialize(self)` – é o primeiro método a ser executado pela plataforma `pyDash`. Este método é executado durante a fase de inicialização da plataforma. Assim como no construtor da classe, é possível utilizar esta função para inicializar os atributos que serão utilizados pelo algoritmo ABR. Contudo, é importante destacar que os atributos da classe devem ser declarados no construtor da classe a ser implementada.
- `finalization(self)` – é o último método a ser executado pela plataforma `pyDash`. Este método é executado durante a fase de finalização da plataforma. Pode ser utilizado para gerar estatísticas finais computadas pelo protocolo ABR implementado. A plataforma `pyDash` gera por padrão uma série de estatísticas e gráficos ao final da sua execução. Contudo, o desenvolvedor pode utilizar este método para adicionar informações estatísticas complementares.

4.3 Estrutura de *Whiteboard*

Durante a execução da plataforma `pyDash`, diversas estatísticas são coletadas. Ao final, a plataforma gera resultados na pasta `results`. Durante a execução do `pyDash` é possível ao algoritmo ABR ter acesso às estatísticas que estão sendo geradas. A Figura 12 apresenta a relação de troca de informação entre a classe `Player` e o algoritmo ABR implementado. Como o novo algoritmo ABR herda os atributos da classe abstrata `IR2A`, ele tem acesso ao objeto `self.whiteboard`, que permite acesso em tempo real às estatísticas geradas pela plataforma.

Através do objeto `self.whiteboard`, as seguintes estatísticas podem ser obtidas pelo novo algoritmo ABR implementado.

- `get_playback_qi()` – método retorna uma lista de tuplas, onde o primeiro elemento é o momento em que o dado foi coletado (segundos) e o segundo elemento é a qualidade QI observada durante a reprodução do trecho do vídeo. A lista está ordenada de forma crescente pelo campo tempo.
- `get_playback_pauses()` – método retorna uma lista de tuplas, onde o primeiro elemento é o momento em que o dado foi coletado (segundos) e o

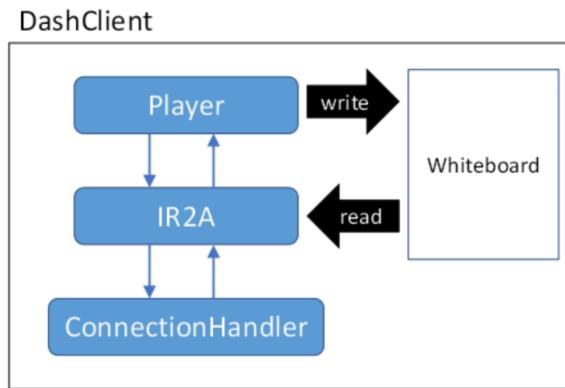


Figura 12: Área de transferência de estatística *Whiteboard*.

segundo elemento é o tamanho da pausa de vídeo (segundos) ocorrida. A lista está ordenada de forma crescente pelo primeiro campo de tempo.

- `get_playback_buffer_size()` – método retorna uma lista de tuplas, onde o primeiro elemento é o momento em que o dado foi coletado (segundos) e o segundo elemento é o tamanho do *buffer* observado durante a reprodução do vídeo. A lista está ordenada de forma crescente pelo campo tempo.
- `get_playback_history()` – método retorna uma lista de tuplas, onde o primeiro elemento é o momento em que o dado foi coletado (segundos) e o segundo elemento representa o status de reprodução do vídeo. Ou seja, terá o valor um (1) quando foi possível reproduzir o vídeo e zero (0) quando não foi possível. Esta lista representa o histórico de reprodução do vídeo. A lista está ordenada de forma crescente pelo campo tempo.

4.4 Estatísticas e gráficos

Ao final da execução da plataforma *pyDash*, alguns gráficos são gerados na pasta `results`. A plataforma também apresenta no terminal algumas estatísticas durante a sua execução. Por fim, é importante lembrar que os valores utilizados para gerar os gráficos podem ser acessados através do objeto `self.whiteboard`, conforme descrito na seção anterior.



4.5 Arquivo de entrada *json*

A plataforma *pyDash* aceita a passagem de parâmetros de configuração através do arquivo `dash_client.json`. A Figura 13 apresenta um exemplo do arquivo json de configuração da plataforma². Com relação ao arquivo json, os seguintes parâmetros podem ser fornecidos à plataforma *pyDash*:

- `buffering.until` – define o tamanho do *buffer* inicial antes do vídeo começar a ser reproduzido. Este deve ser um valor inteiro positivo e representa o tempo na escala de segundos.
- `max_buffer_size` – tamanho máximo do *buffer* que armazena segmentos de vídeo. Ao atingir este tamanho, o `Player` passa a limitar as próximas requisições. Este deve ser um valor inteiro positivo e representa o tempo na escala de segundos.
- `playbak_step` – passo na reprodução do vídeo. Por padrão, este valor é definido como 1 (um). Ou seja, a “reprodução” do vídeo será feita a cada segundo. **Este valor padrão não deve ser alterado.**
- `traffic_shaping.profile_interval` – Este parâmetro faz parte da política de *traffic shaping* (TF) implementada pela plataforma *pyDash*. Define por quanto tempo o perfil corrente de TF será aplicado antes de ser modificado. Este deve ser um valor inteiro positivo e representa o tempo na escala de segundos.
- `traffic_shaping.profile_sequence` – Este parâmetro faz parte da política de *traffic shaping* (TF) implementada pela plataforma *pyDash*. Foram definidos três perfis para TF:
 - **Low (L)** – perfil com a menor restrição de banda, fornecendo taxa de transferência suficiente para que a maior qualidade de vídeo seja recuperada³ sem pausas no `Player`.
 - **Medium (M)** – perfil com restrição média de banda, fornecendo taxa de transferência suficiente para que a qualidade média de vídeo seja recuperada sem pausas no `Player`.

²verifique o repositório git do projeto para ter acesso à última versão dos parâmetros aceitos pela plataforma.

³Supondo que a sua largura de banda seja suficiente para que você consiga recuperar o vídeo na melhor qualidade. Exemplo: sua largura de banda é 100 Mbps e o segmento de vídeo solicitado foi codificado a 5 Mbps.



- **High (H)** – perfil com a maior restrição de banda, fornecendo taxa de transferência suficiente para que a menor qualidade de vídeo seja recuperada sem pausas no `Player`.

Os valores de vazão, definidos pelos perfis acima e carregados do arquivo `mpd`, servem como valor da média na configuração da Distribuição de Probabilidade Exponencial utilizada para definir o valor real de banda que será aplicado à restrição de TF. Este parâmetro aceita uma sequência de letras: “L”, “M”, e “H”, podendo apresentar sequência e quantidade variável. Exemplo de sequências válidas: `LLLLMMMHMMHH`, `LL`, `HMHLH`. A aplicação dos perfis de TF segue o padrão de acesso de uma lista circular, permanecendo pelo tempo `traffic_shaping_profile_interval` em cada elemento da lista (perfil de TF).

- `traffic_shaping_seed` – para permitir a comparação entre os algoritmos ABR, a sequência de restrição de banda aplicada deve ser a mesma para todos os algoritmos avaliados. Sendo assim, este parâmetro define a semente utilizada pela Distribuição de Probabilidade Exponencial na geração da sequência de larguras de banda aplicada. Este deve ser um valor inteiro.
- `url_mpd` – define a `url` completa para o arquivo `mpd`. Esta informação permite à classe `Player` recuperar a lista de qualidades existentes para o vídeo alvo, bem como, iniciar o processo de obtenção dos segmentos de vídeo.
- `r2a_algorithm` – Este campo define o nome da classe, referente ao algoritmo ABR, que será carregada dinamicamente pela plataforma `pyDash`.
 - **Observação:** é mandatório que a classe que implementa o novo algoritmo ABR esteja armazenada na pasta `r2a`. Tanto o nome da classe quanto o nome do arquivo sejam os mesmos. Como exemplo, a classe `R2ANewAlgorithm1` dever estar contida no arquivo `r2a/r2anewalgorithm1.py`.

5 Avaliação

Cada grupo deverá entregar via Moodle o código da classe que implementa o algoritmo ABR (observar a data de entrega na página da sua turma) e um relatório contendo as seguintes informações:



```
{  
    "buffering_until": 5,  
    "max_buffer_size": 60,  
    "playbak_step": 1,  
    "traffic_shaping_profile_interval": "1",  
    "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": "R2ARandom"  
}
```

Figura 13: Arquivo de configuração `dash_client.json`.

1. Capa: contendo a identificação da disciplina, turma e dos membros do grupo (nome completo + matrícula);
2. Introdução (1 página): apresentação do problema a ser resolvido e da proposta de solução (passar a intuição).
3. Algoritmo ABR: explicação detalhada da solução proposta incluindo a descrição dos trechos de código.
4. Incluir na explicação do algoritmo ABR um exemplo de funcionamento do protocolo. Adicionar figuras para melhorar o entendimento.
5. Adicionar avaliação do seu algoritmo ABR, apresentando gráficos e análise explicando o comportamento obtido.
6. Conclusão (1/2 página): resumo do trabalho realizado e comentários sobre o que foi aprendido.
7. Referências: lista de todas as referências citadas no relatório.

Adicionalmente, uma apresentação síncrona será marcada, na qual, cada grupo defenderá o algoritmo ABR proposto e informará como foi feita a distribuição do trabalho entre seus membros.

5.1 Avaliação do algoritmo ABR

O processo de avaliação da qualidade do algoritmo ABR proposto considerará a inspeção do código fornecido, juntamente com a análise de métricas de desempenho, que são compostas basicamente da sequência de qualidades dos segmentos baixados (maior melhor), e também da quantidade e do tamanho das interrupções ocorrida no vídeo (menor melhor).

5.2 Cenários de teste

Considerando as 6 versões disponíveis do vídeo (vide Fig.8), serão avaliados 3 cenários de tráfego distintos, compostos por diferentes combinações de TF, buscando emular uma conexão com baixa, média e alta variação da latência.

Referências

- [1] Ana Carolina Menezes William Macedo. mpeg-DASH. <https://www.gta.ufrj.br/ensino/eel879/vf/mpeg-dash/>, 2019.
- [2] Marcos Caetano. Redes de Computadores: Camada Aplicação - DASH & CDN / Seção 2.6. <https://youtu.be/Tzm9uDi8ZG0>, 2020.
- [3] DASH Industry Forum (DASH-IF). ABR Logic. <https://github.com/Dash-Industry-Forum/dash.js/wiki/ABR-Logic>, 2018.
- [4] DASH Industry Forum (DASH-IF). Guidelines for Implementation: DASH-IF Interoperability Points. <https://dashif.org/docs/DASH-IF-IOP-v4.3.pdf>, 2018.
- [5] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.
- [6] Wikipedia. Dynamic Adaptive Streaming over HTTP. https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP, 2020.