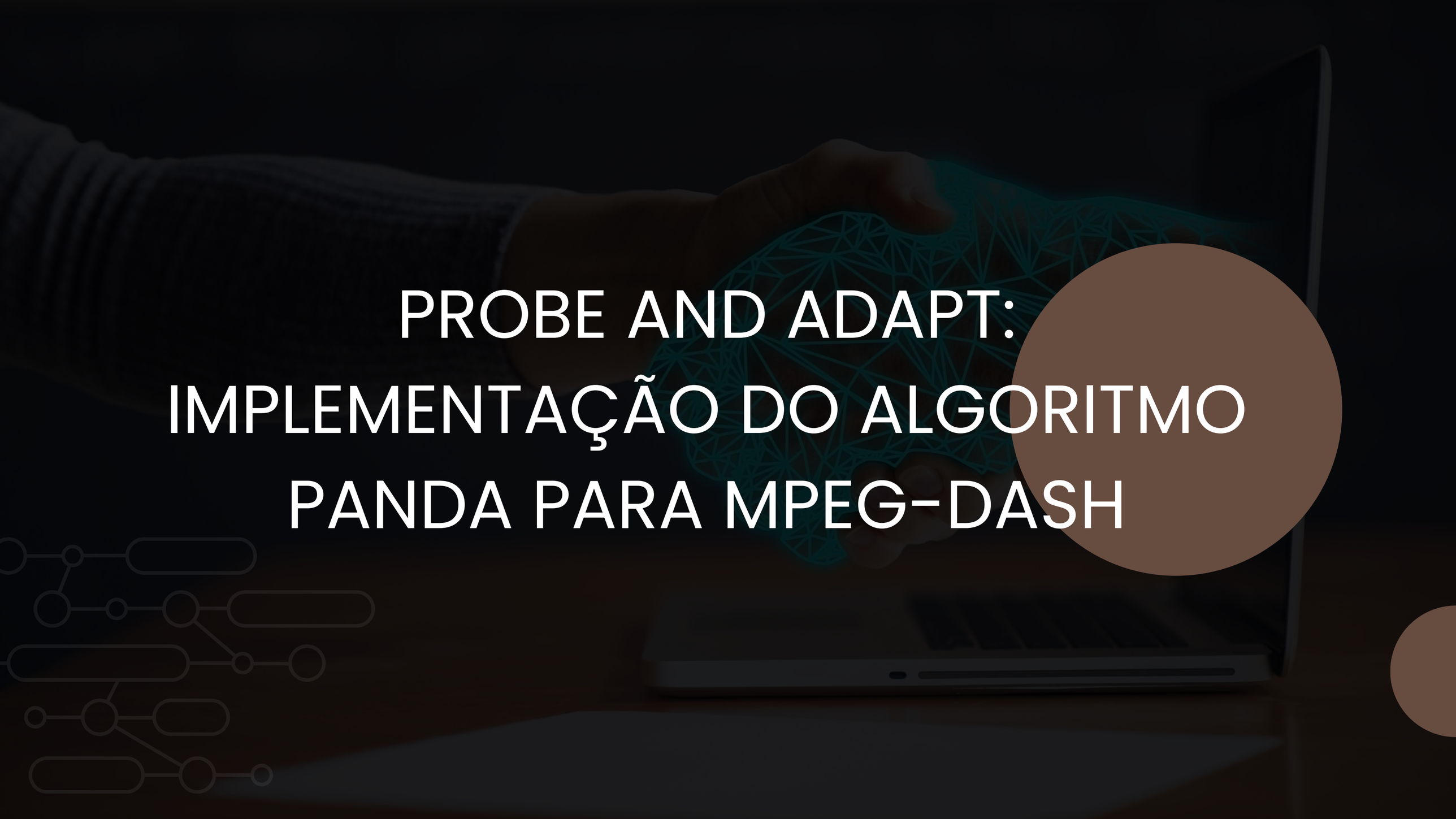
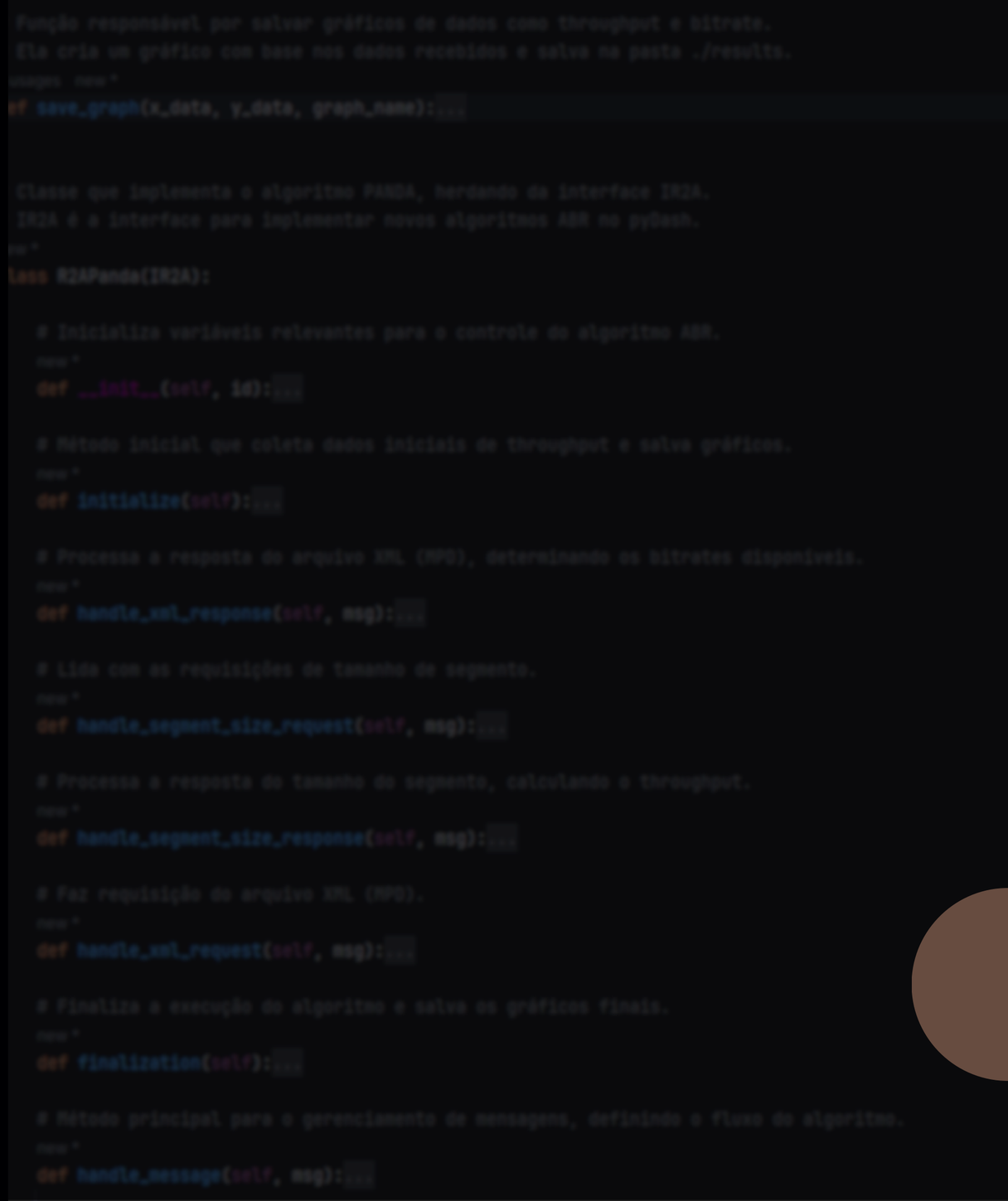


PROBE AND ADAPT: IMPLEMENTAÇÃO DO ALGORITMO PANDA PARA MPEG-DASH

INTRODUÇÃO AO CÓDIGO

- O que é o PANDA?
  - PANDA é um algoritmo de Adaptação de Taxa de Bits (ABR) que ajusta a qualidade do vídeo durante o streaming baseado nas condições da rede.
  - Ele visa minimizar pausas na reprodução do vídeo (rebuffering), ajustando o bitrate de acordo com o throughput disponível.

- Principais Objetivos:
  - Manter a melhor qualidade de vídeo possível.
  - Evitar que o buffer esgote e cause pausas.
  - Ajustar o throughput dinamicamente.



VISÃO GERAL DO CÓDIGO

- Estrutura:
  - O código implementa a classe R2APanda, que herda de IR2A (interface de algoritmos ABR).
  - Implementação dos métodos obrigatórios da interface para lidar com requisições e respostas durante o streaming.

- Pacotes Importados:
  - requests: Para fazer requisições HTTP (usado para obter o MPD).
  - psutil: Coleta dados de throughput da rede.
  - xml.etree.ElementTree as ET: Para processar o arquivo MPD em formato XML.
  - matplotlib.pyplot: Gera gráficos do desempenho do algoritmo.

```
import requests
import psutil
from xml.etree import ElementTree as ET
import matplotlib.pyplot as plt
```

```python
def save_graph(x_data, y_data, graph_name):
    print(f"Salvando gráfico: {graph_name}")
    print(f"x_data: {x_data}")
    print(f"y_data: {y_data}")

    if len(x_data) == 0 or len(y_data) == 0:
        # Se não houver dados suficientes, o gráfico não será criado.
        print(f"Nenhum dado disponível para {graph_name}, gráfico não será criado.")
        return

    if len(y_data) < len(x_data):
        # Se houver menos dados em y_data do que x_data, preenche com o último valor disponível.
        y_data.extend([y_data[-1] if y_data and y_data[-1] is not None else 0] * (len(x_data) - len(y_data)))

    # Configura e gera o gráfico com os dados fornecidos.
    plt.figure()
    plt.plot( *args: x_data, y_data, marker='o', linestyle='-', label=graph_name)
    plt.title(f'{graph_name} over time')
    plt.xlabel('Time (s)')
    plt.ylabel(graph_name)
    plt.grid(True)
    file_path = os.path.join('./results', f'{graph_name}.png')

    # Garante que o diretório exista e salva o gráfico.
    if not os.path.exists('./results'):
        print("Diretório './results' não existe. Criando diretório.")
        os.makedirs('./results')

    plt.savefig(file_path)
    plt.close()
    print(f'Gráfico salvo em: {file_path}')
```

MÉTODO SAVE_GRAPH

- Recebe dois vetores: x_data (tempo) e y_data (valor, como throughput ou bitrate).
- Cria o gráfico utilizando matplotlib e salva no diretório ./results.
- Verifica se o diretório existe e, se não, o cria.
- Importância: Esses gráficos mostram como o algoritmo ajusta o bitrate e throughput ao longo do tempo.

- **Motivação:** Função criada para suprir as dificuldades que o grupo estava encontrando em gerar os gráficos apenas herdando a classe IR2A.
- **Importância:** Esses gráficos permitem visualizar como o algoritmo ajusta o bitrate e o throughput ao longo do tempo.

INICIALIZAÇÃO (INITIALIZE)

Inicializa os dados e gráficos

- Coleta o throughput inicial usando psutil.net_io_counters(), que captura a quantidade de dados enviados e recebidos.
- Adiciona os valores de throughput e bitrate inicial nas listas correspondentes.
- Chama a função save_graph para gerar os gráficos iniciais de throughput e bitrate.

- **Importância:** Essa fase inicial é crítica para começar o processo de adaptação da taxa de bits.

```python
def initialize(self):
    print("Inicializando execução do algoritmo PANDA.")
    current_time = time.time()
    self.time_data.append(current_time)

    # Coleta os dados de throughput da rede utilizando a biblioteca psutil.
    net_io = psutil.net_io_counters()
    throughput = net_io.bytes_sent + net_io.bytes_recv

    # Inicializa com valores de throughput e bitrate.
    self.throughput_data.append(throughput / 1000)  # Em Kbps
    self.bitrate_data.append(self.target_rate if self.target_rate else 0)

    # Salva os gráficos iniciais.
    save_graph(self.time_data, self.throughput_data, graph_name: 'Initial Throughput')
    save_graph(self.time_data, self.bitrate_data, graph_name: 'Initial Bitrate')
```

PROCESSAMENTO DO XML (MPD)

```
def handle_xml_response(self, msg):
    self.parsed_mpd = parse_mpd(msg.get_payload())
    self.qi = self.parsed_mpd.get_qi()  # Obtém os bitrates disponíveis (Quality Index).
    self.target_rate = min(self.qi)  # Define o bitrate inicial como o menor disponível.
    self.current_bitrate = self.target_rate

    # Atualiza o tempo e bitrate atual.
    self.time_data.append(time.time())
    self.bitrate_data.append(self.current_bitrate)

    print(f"Resposta XML processada. Bitrates disponíveis: {self.qi}")
    print(f"Taxa alvo inicial: {self.target_rate}")
    self.send_up(msg)  # Envia a mensagem para a camada superior (Player).
```

Processa a resposta do MPD (Media Presentation Description)

- parse_mpd: Extrai informações do MPD (como os bitrates disponíveis).
- Define a taxa alvo inicial como o menor bitrate disponível.
- Adiciona os dados de tempo e bitrate para gerar gráficos.

- **Importância:** O MPD contém as informações cruciais para o algoritmo decidir qual qualidade escolher. Aqui, o algoritmo começa a definir os parâmetros de bitrate com base no conteúdo disponível.

SOLICITAÇÃO DE SEGMENTOS

Realiza a requisição de um novo segmento:

- Simula um atraso com time.sleep(1) para replicar o tempo de download.
- Adiciona o bitrate atual à solicitação do segmento.
- Salva gráficos com os dados de throughput e bitrate antes da requisição.

- **Importância:** Esse método controla a lógica de solicitar o próximo segmento de vídeo com a qualidade que o algoritmo decide ser a mais apropriada.

```
def handle_segment_size_request(self, msg):
    print("handle_segment_size_request foi chamado")
    time.sleep(1)  # Simula um pequeno atraso.
    self.last_download_time = time.time()
    self.time_data.append(self.last_download_time)

    # Sincroniza os dados de bitrate e tempo.
    self.bitrate_data.append(self.current_bitrate)

    # Define o bitrate para a solicitação e envia para a camada inferior.
    msg.add_quality_id(self.current_bitrate)
    self.send_down(msg)
    print(f"Solicitação de segmento com bitrate atual: {self.current_bitrate}")

    # Salva os gráficos após a requisição.
    save_graph(self.time_data, self.throughput_data, graph_name: 'Pre-request Throughput')
    save_graph(self.time_data, self.bitrate_data, graph_name: 'Pre-request Bitrate')
```

RESPOSTA DO SEGMENTO

```python
def handle_segment_size_response(self, msg):
    current_time = time.time()
    download_duration = current_time - self.last_download_time
    segment_size = msg.get_payload_size()  # Obtém o tamanho do payload.

    # Calcula o throughput baseado no tamanho do segmento e no tempo de download.
    if download_duration > 0 and segment_size > 0:
        throughput = (segment_size / download_duration) / 1000
    else:
        throughput = 0

    # Atualiza os dados de tempo, throughput e bitrate.
    self.time_data.append(current_time)
    self.throughput_data.append(throughput)
    self.bitrate_data.append(self.current_bitrate if self.current_bitrate is not None else 0)

    # Verifica se os dados estão sincronizados.
    assert len(self.time_data) == len(self.throughput_data), "Descompasso entre time_data e throughput_data"
    assert len(self.time_data) == len(self.bitrate_data), "Descompasso entre time_data e bitrate_data"

    # Salva os gráficos de throughput e bitrate.
    save_graph(self.time_data, self.throughput_data, graph_name: 'Throughput')
    save_graph(self.time_data, self.bitrate_data, graph_name: 'Bitrate')
```

Processa a resposta do segmento e ajusta o throughput

- Calcula o throughput com base no tamanho do segmento baixado e o tempo de download.
- Atualiza as listas de tempo, throughput, e bitrate.
- Salva gráficos com o throughput e o bitrate após a resposta.

- **Importância:** Aqui, o algoritmo mede o desempenho real da rede e ajusta o bitrate para as futuras requisições de segmentos.

REQUISIÇÃO XML (MPD)

Solicita o arquivo MPD e suas opções de bitrate e segmentos.o do Arquivo MPD

- Verifica se a URL do MPD é válida.
- Faz uma requisição HTTP para baixar o MPD.
- Faz o parsing do arquivo XML recebido para extrair as informações de segmentos e bitrates.
- Registra o tempo da requisição e atualiza os dados de bitrate.

- **Importância:** A obtenção do MPD é crítica para a adaptação da taxa de bits, pois permite ao algoritmo conhecer as opções de qualidade disponíveis para os segmentos futuros.

```python
def handle_xml_request(self, msg):
    xml_url = msg.get_payload()
    print(f">>>> Requisição XML recebida: {xml_url}")

    # Verifica se a URL é válida e faz a requisição do MPD.
    if not xml_url.startswith("http"):
        print("Erro: O conteúdo recebido não é uma URL válida de XML.")
        return

    try:
        request_time = time.time()
        self.time_data.append(request_time)
        self.bitrate_data.append(self.current_bitrate if self.current_bitrate is not None else 0)

        # Faz a requisição HTTP e processa o conteúdo XML.
        response = requests.get(xml_url)
        response.raise_for_status()
        xml_content = response.text
        xml_tree = ET.fromstring(xml_content)

        for element in xml_tree:
            print(f"Tag: {element.tag}, Atributos: {element.attrib}, Texto: {element.text}")

        self.send_up(msg)  # Envia a resposta para a camada superior (Player).

    except requests.exceptions.RequestException as e:
        print(f"Erro ao fazer requisição HTTP: {e}")
    except ET.ParseError as e:
        print(f"Erro ao parsear o XML: {e}")
```

```python
def finalization(self):
    print("Finalizando execução do algoritmo PANDA.")
    final_time = time.time()
    self.time_data.append(final_time)
    self.bitrate_data.append(self.current_bitrate if self.current_bitrate is not None else 0)

    save_graph(self.time_data, self.throughput_data, graph_name: 'Final Throughput')
    save_graph(self.time_data, self.bitrate_data, graph_name: 'Final Bitrate')
```

FINALIZAÇÃO DO ALGORITMO

Finaliza a execução do algoritmo e gera os gráficos finais.

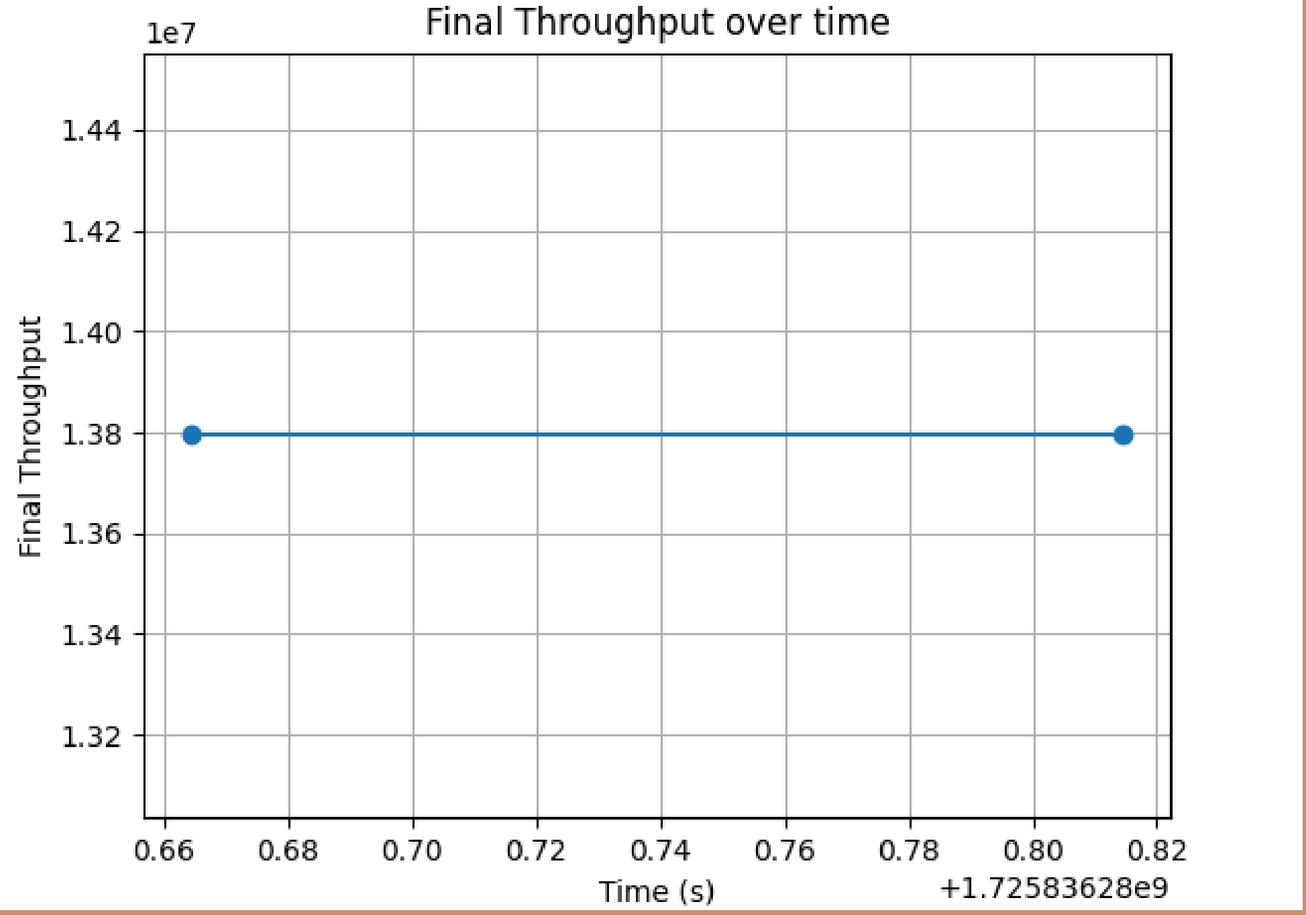
- Registra o tempo de finalização do algoritmo.
- Atualiza os dados de bitrate e throughput.
- Salva os gráficos de throughput e bitrate acumulados ao longo da execução.

- **Importância:** A finalização documenta o comportamento do algoritmo, permitindo uma análise clara do desempenho em termos de variação de taxa de bits e condições da rede.

TESTE 1:

Variação no Tamanho do Buffer

Esse gráfico mostra a estabilidade da rede durante o processo de download do segmento do vídeo. Não houve variação significativa no throughput entre os dois pontos medidos, o que indica que, nesse experimento, a largura de banda disponível para o algoritmo R2APanda foi constante.



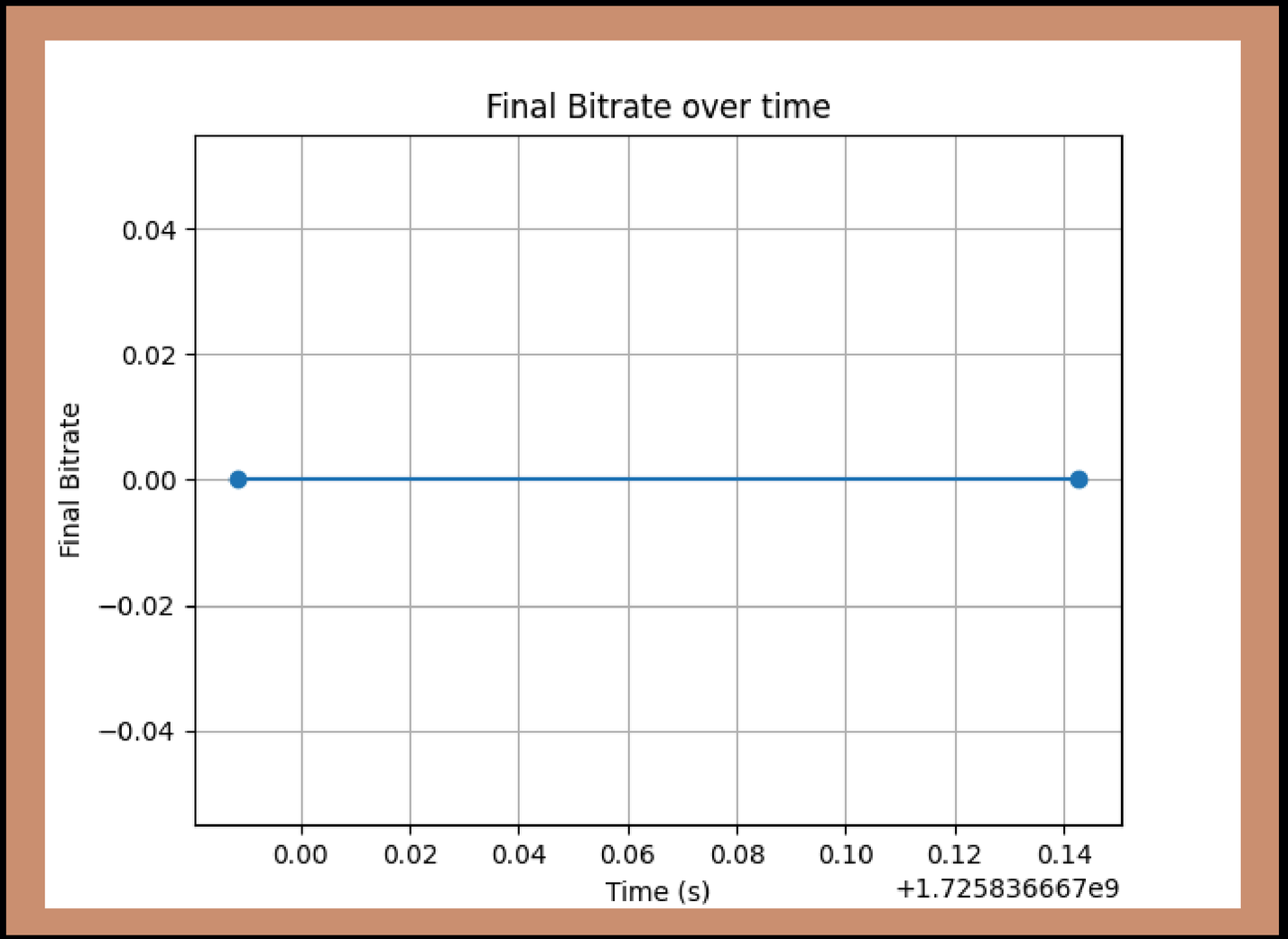
Observações

- **Eixo Y (Final Throughput):** Mostra o throughput em uma unidade de medida muito alta (por volta de 13.794.410 bits/s ou aproximadamente 13,8 Mbps).
- **Eixo X (Tempo):** Mostra dois pontos de tempo separados por aproximadamente 0,15 segundos.
- **Comportamento:** O throughput permanece constante durante esse pequeno intervalo de tempo, sugerindo que a rede manteve uma taxa de transferência estável.

TESTE 2:

Mudança no Perfil de Traffic Shaping

Logo no início da execução, o algoritmo PANDA não conseguiu definir um valor de bitrate adequado. Isso pode ser devido a vários fatores, como a indisponibilidade de informações sobre a qualidade do vídeo (MPD) ou o throughput da rede no momento inicial estar muito baixo ou inconsistente.



Observações

- **Eixo X (Tempo):** Representa o tempo de execução do algoritmo PANDA. Vemos dois pontos ao longo da execução (aproximadamente 0.14 segundos no intervalo total).
- **Eixo Y (Bitrate):** Representa a taxa de bits (bitrate) utilizada para transmitir o vídeo. Aqui, observamos que o bitrate permaneceu constante e igual a 0 durante todo o período medido.

GRÁFICOS SEMPRE IGUAIS: POSSÍVEIS CAUSAS

A razão pela qual os gráficos podem estar sempre apresentando o mesmo resultado, independentemente das mudanças no JSON, pode estar relacionada aos seguintes fatores:

01

Falta de Atualização dos Dados:

O código pode estar coletando e salvando os gráficos com base em dados que não estão sendo atualizados corretamente durante o processo. Isso pode causar a geração de gráficos idênticos.

02

Gravação dos Gráficos Apenas no Final:

O código utiliza a função save_graph várias vezes, mas é possível que os gráficos intermediários não estejam sendo salvos corretamente. Isso pode ocorrer devido a condições específicas no fluxo do código, fazendo com que apenas os gráficos finais sejam gerados.

03

Dados de Tempo (time_data) Incompletos:

O time_data pode não estar sendo preenchido adequadamente entre as requisições e respostas, fazendo com que os gráficos tenham sempre a mesma estrutura.

04

Divergência Entre o Tamanho dos Dados:

Se o comprimento de x_data (tempo) e y_data (throughput ou bitrate) não estiver sincronizado, os gráficos podem ser distorcidos ou incompletos.

05

Sincronização de Variáveis Não Garantida:

A sincronização entre as variáveis de bitrate, throughput e tempo pode estar falhando em partes do código, resultando em dados não variados.

06

Execução Sequencial:

O código pode estar salvando gráficos somente em eventos finais, sem garantir que as mudanças nos dados ocorram adequadamente entre uma solicitação e outra.

CONCLUSÃO

O código implementa o algoritmo PANDA para controle adaptativo de bitrate (ABR) em redes de transmissão de vídeo. Ele:

- Coleta e processa dados de throughput, bitrate e tempo durante a execução;
- Gera e salva gráficos para visualizar a variação dessas métricas ao longo do tempo;
- Utiliza uma estrutura modular para gerenciar as comunicações entre as camadas de Player, Connection Handler e R2A (PANDA);
- A sincronização inadequada de dados pode resultar na repetição de gráficos ou ausência de gráficos intermediários.

O foco do código está em ajustar dinamicamente o bitrate para otimizar a experiência de reprodução de vídeo conforme as condições da rede.

OBRIGADO!

GRUPO 15
GIULIA MOURA FERREIRA, 20/00018795
JOÃO VITOR VIEIRA, 22/1022023

CIC0124 - REDES DE COMPUTADORES - TURMA 03 - 2024/1