Video Streaming App


```
Instituto Federal de Minas Gerais - Campus Formiga
06/05/2018
José Luiz
Jonathan Arantes
Vinícius Araújo
```


Este aplicativo de streaming de vídeo provê um controle de transmissão de dados de modo a não desperdiçar a rede (banda larga, wifi, 4g, etc) do usuário ao transmitir um arquivo de vídeo, enquanto mantém a bufferização do arquivo ao player de vídeo.

Até este ponto do projeto, os usos dos datagramas *UDP* são para transferência de mensagens entre Cliente e Servidor para comunicar a lista de arquivos de vídeo que podem ser transmitidos e o *stream* de dados partindo do servidor para o cliente.

Segue abaixo a descrição das funções implementadas, separadas por arquivo:

Servidor

- Inicialização `__init__`:

  Inicialização do objeto Servidor;

  - `self.diretorio`: Diretório da pasta *streamer*, onde ficam localizados os arquivos de vídeo a serem transmitidos;
  - `self.poolThreads`: *Pool* de *threads* para processamento simultâneo das funções;
  - `self.udp`: *Socket UDP* utilizado na trasmissão de dados;
  - `HOST` e `PORT`: Endereço IP e porto de acesso do servidor;
  - `self.arquivolog`: Arquivo para registro de *log*;
  - `self.conteudo`: Conteudo anterior do arquivo de *log*;
  - `orig`: Endereço de origem do servidor (IP:Porto);
  - `self.diretorio_arquivos`: Diretorio dos arquivos para transmissão;

- `build_thread_control(self)`: Função para travamento (*lock*) da *thread* e execução desta;
- `build_threads(self)`: Cria a *pool* de *threads* do servidor;
- `wait(self)`: Espera a requisição de um cliente, um loop infinito que continua a receber pacotes do *socket* e aceita a requisição (se houver) do cliente;

- `accept(self, msg, cliente)`: Envia a requisição para uma das *threads* em espera na *pool*, em seguida retorna para a função `wait(self)`;
- `checkmsg(self, msg)`: Trata a mensagem recebida pelo *socket* para o padrão de `cabecario` (*header*) e `dados` (*data*);

Cliente

- Inicialização `__init__`:

  Inicialização do objeto Cliente;

  - `self.udp`: *Socket UDP* do cliente para recebimento dos dados;
  - `self.meuip`: IP do cliente;
  - `orig`: IP:Porto da aplicação do cliente (porto é definido como 0 para ser decidido pela função `self.udp.bind`);
  - `self.udp.bind`: Aloca um porto à aplicação cliente para recebimento de dados;
  - `self.meuporto`: Porto de acesso da aplicação para a rede;
  - `self.servidor`: IP:Porto do servidor que irá transmitir os dados;
  - `self.controle`: Objeto `ControleEnvio()`;
  - `self.udp.settimeout(10)`: Configura o tempo de *timeout* da transmissão *UDP*;
  - `self.buffer`: Vetor de *buffer* do cliente;
  - `self.arquivo`: Arquivo que será gravado em disco;
  - `self.pacotes_recebidos`: Vetor de buffer do *socket*;
  - `self.num_pacotes`: Número de pacotes recebidos;
  - `self.video`: Objeto `Video()`;

- `requisita_servidor(self)`: Envia requisição para o servidor iniciar a comunicação, inicia o objeto `Video()` e entra em loop para receber as mensagens do servidor;
- `recebermsg(self)`: Recebe o pacote do *socket* e passa esta para o padrão `mensagem` e `srvenvio`;
- `desmonta_pacote(self, msg)`: Passa o pacote para o padrão *header:data*;
- `checkmsg(self, msg, srvenvio)`: Executa `desmonta_pacote(self, msg)` e converte os bytes para leitura;
- `tratamento(self, msg, srv)`:
- `worker()`: Chama a execução do tocador de vídeo MPV;

ControleEnvio

- Inicialização `__init__`:

  Inicialização do objeto ControleEnvio;

  - `self.buffersize`: Tamanho do buffer (8mb);
  - `self.windowsize`: Tamanho da janela (ainda não definida, testes são necessários);
  - `self.unidadecontrole`: Objeto `UnidadeControle()` recebida via parâmetro;

- `sendmsg(self, msg, cliente, udp, tipomsg, usounidadecontrole=False, seq_inicial = 0)`: Realiza a formatação da mensagem para ser enviada;
- `fragmenta(self, msg)`: Fragmenta a mensagem se esta for maior que o tamanho máximo para empacotamento (1016kb) e retorna uma lista com os fragmentos ordenados da mensagem;
- `adiciona_cabecalho(self, msg, numero_sequencia, tipomsg)`: Empacota a mensagem para envio, indexando o *header* do pacote no início da mensagem;

Pacote

- Inicialização `__init__`:

  Inicialização do objeto Pacote;

  - `self.dados`: Dados armazenados;
  - `self.time`: *Timeout* do pacote;
  - `self.numseq`: Número de sequência (para controle em caso de fragmentação);

Transferencia

- Inicialização `__init__`:

  Inicialização do objeto Transferencia;

  - `Thread.__init__(self)`: Inicia uma thread para executar este objeto;
  - `self.dest`: Endereço de destino dos dados;
  - `self.arquivo`: Arquivo de vídeo a ser lido;
  - `self.unidadecontrole`: Objeto `UnidadeControle()` recebido por parâmetro;
  - `self.controle`: Objeto `ControleEnvio()` que recebe o objeto `UnidadeControle()`;
  - `self.caixadeaviso`:
  - `self.lock`: Trava de *threads*;

- `run(self)`: Função de execução da *thread*;
- `leitura_arquivo(self)`: Realiza a leitura de *bytes* no arquivo, lendo a quantidade de *bytes* do tamanho do *buffer*;

- `fechar_arquivo(self)`: Fecha o arquivo;
- `checkavisos(self)`: Checa as mensagens da unidade de controle;

UnidadeControle

- Inicialização `__init__`:

  Enumeração com a ordem:

  ```
  1: PACOTE,
  2: ACK,
  3: TIME;
  ```

  Inicialização do objeto UnidadeControle;

  - `self.listaClientes`: Lista de clientes conectados ao servidor;
  - `self.threadsusadas`: Lista de *threads* em uso;
  - `self.lock`: Trava das *threads*;
  - `self.listaPortos`: Lista de portos utilizados para comunicação com os clientes;
  - `self.udp`: *Socket* de comunicação da unidade de controle;
  - `self.udp.settimeout(1)`: Configura o tempo de *timeout* para 1ms;

- `run(self)`:
- `add_buffer(self, cliente, threferente)`: Buffer do controle de envio;
- `add_pacote(self, cliente, pacote, numseq, valor`: Adiciona um pacote ao buffer de envio;
- `add_porto(self, udp)`: Adiciona o porto à lista para recebimento dos *ACKs* da transferência;
- `remover_cliente(self, cliente, udp)`: Termina a comunicação com o cliente, removendo-o da lista de clientes, das *threads* utilizadas e da lista de portos;
- `avisar_thread(self, th, aviso)`: Manda um aviso para a *thread* de se ela pode terminar a execução, se ela deve reenviar algum arquivo, ou se ela pode continuar executando;
- `tratamsg(self, msg, cliente)`: Desmonta o pacote e verifica o *ACK* recebido;
- `desmonta_pacote(self, msg)`: Desmonta o pacote para o padrão *header:data*;
- `verifica_timeout_pacote(self)`: Verifica se o pacote chegou ao seu *timeout*, se o *timeout* do pacote chegou a 0, reenvia o pacote e reseta o *timeout*;
- `verifica_liberacao_thread(self, cliente)`: Envia um aviso à thread para parar a execução;

Pontos fortes

Por trabalhar com uma pool threads, o servidor pode responder a mais de um cliente, porem deve respeitar um numero maximo de 10 conexoes que e o numero de threads criadas.

O envio e feito de forma sequencial a uma taxa de aproximadamente 1MegaByte por segundo, e os ACKs sao recebidas pela unidade de controle controlada por uma outra thread reeviando caso atingir um timeout (cada pacote enviado gera um time propio), entao a thread que trata a transferencia do arquivo nao sofre interrupcoes na sua transferencia.

Pontos fracos

Ainda nao foi inserido uma janela de transferencia, entao ao iniciar uma transferencia se um pacote for perdido, mesmo tendo retransmissao, o cliente so recebera o arquivo depois de um certo tempo pois havera muitos pacotes a sua frente o que acasionara na interrupção do video.

O limite da velocidade de transferencia é proximo de 1MB, gerando problemas no stremer de videos com a resolução muito elevada.