Streaming de Vídeo com RTSP e RTP

Neste trabalho, cada equipe deverá implementar um servidor e um cliente de transmissão de vídeo que se comunicam utilizando o protocolo RTSP (Real-Time Streaming Protocol) e enviam dados por meio do protocolo RTP (Real-time Transfer Protocol). Sua tarefa será implementar o protocolo RTSP no cliente e realizar o empacotamento RTP no servidor.

Forneceremos o código base que implementa o protocolo RTSP no servidor, a descompactação de pacotes RTP no cliente e o gerenciamento da exibição do vídeo transmitido. Cada equipe precisará **atualizar o código Python** fornecido e escrever o código restante.

Código

Client, ClientLauncher
O módulo *ClientLauncher* inicia o cliente e a interface do usuário, que será utilizada para enviar comandos RTSP e exibir o vídeo. Na classe *Client*, você precisará implementar as ações correspondentes aos botões pressionados. Não é necessário modificar o módulo *ClientLauncher*.

ServerWorker, Server
Esses dois módulos implementam o servidor que responde às solicitações RTSP e transmite o vídeo. A interação RTSP já está implementada, e o módulo *ServerWorker* chama métodos da classe *RtpPacket* para realizar a pacotização dos dados de vídeo. Não é necessário modificar esses módulos.

RtpPacket
Essa classe é usada para manipular os pacotes RTP. Ela possui métodos separados para lidar com os pacotes recebidos no lado do cliente, e você não precisará modificá-los. O cliente também realiza a descompactação (decodificação) dos dados, e você não precisará modificar esse método. Sua tarefa será completar a implementação da pacotização de dados de vídeo em RTP, utilizada pelo servidor.

VideoStream
Essa classe é responsável por ler os dados de vídeo de um arquivo no disco. Não é necessário modificar esta classe.

Executando o código
Depois de completar o código, você pode executá-lo da seguinte forma:

1. Primeiro, inicie o servidor com o comando:

```
bash
Copiar código
python Server.py server_port
```

onde `server_port` é a porta na qual o servidor escuta conexões RTSP. A porta padrão do RTSP é 554, mas você precisará escolher um número de porta maior que 1024.

2. Em seguida, inicie o cliente com o comando:

```
bash
Copiar código
python ClientLauncher.py server_host server_port RTP_port
video_file
```

onde:

- `server_host` é o nome da máquina onde o servidor está em execução,
- `server_port` é a porta na qual o servidor está escutando,
- `RTP_port` é a porta na qual os pacotes RTP serão recebidos, e
- `video_file` é o nome do arquivo de vídeo que você deseja solicitar (fornecemos um arquivo de exemplo chamado *movie.Mjpeg*). O formato do arquivo é descrito na seção do Apêndice.

O cliente abrirá uma conexão com o servidor e exibirá uma janela semelhante a esta:

Você pode enviar comandos RTSP ao servidor pressionando os botões.
Uma interação normal com o protocolo RTSP ocorre da seguinte maneira:

1. O cliente envia o comando **SETUP**. Esse comando é usado para configurar a sessão e os parâmetros de transporte.
2. O cliente envia o comando **PLAY**. Esse comando inicia a reprodução.
3. O cliente pode enviar o comando **PAUSE** caso deseje pausar a reprodução.
4. O cliente envia o comando **TEARDOWN**. Esse comando encerra a sessão e fecha a conexão.

O servidor sempre responde a todas as mensagens enviadas pelo cliente. O código 200 significa que a solicitação foi bem-sucedida, enquanto os códigos 404 e 500 representam, respectivamente, os erros de *FILE_NOT_FOUND* e erro de conexão. Neste laboratório, você não precisa implementar outros códigos de resposta. Para mais informações sobre RTSP, consulte o RFC 2326.

O Cliente

Sua primeira tarefa é implementar o protocolo RTSP no lado do cliente. Para isso, você deve completar as funções chamadas quando o usuário clica nos botões da interface. Será necessário implementar as ações para os seguintes tipos de solicitação:

Quando o cliente inicia, ele também abre o socket RTSP para o servidor. Use este socket para enviar todas as solicitações RTSP.

SETUP

- Envie a solicitação **SETUP** para o servidor. Será necessário inserir o cabeçalho *Transport*, especificando a porta do socket de dados RTP que você acabou de criar.
- Leia a resposta do servidor e analise o cabeçalho *Session* (da resposta) para obter o ID da sessão RTSP.
- Crie um socket de datagrama para receber os dados RTP e defina o tempo limite no socket para 0,5 segundos.

PLAY

- Envie a solicitação **PLAY**. Você deve inserir o cabeçalho *Session* e usar o ID da sessão retornado na resposta do comando **SETUP**. Não inclua o cabeçalho *Transport* nesta solicitação.
- Leia a resposta do servidor.

PAUSE

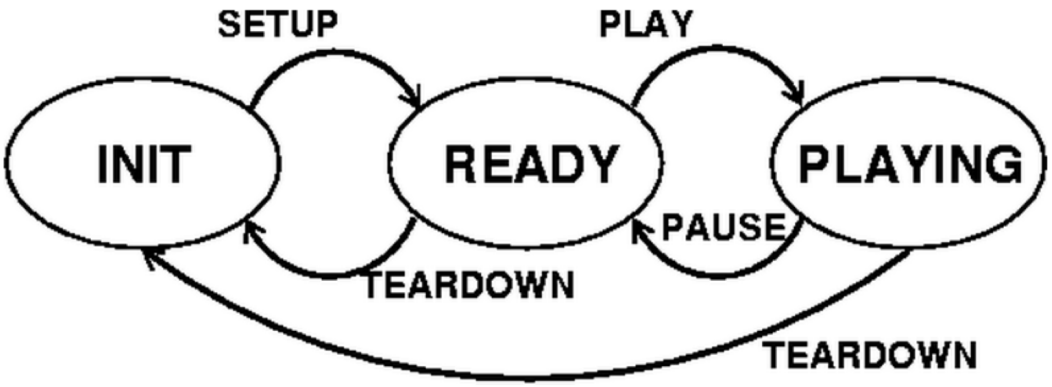
- Envie a solicitação **PAUSE**. Você deve inserir o cabeçalho *Session* e usar o ID da sessão retornado na resposta do comando **SETUP**. Não inclua o cabeçalho *Transport* nesta solicitação.
- Leia a resposta do servidor.

TEARDOWN

- Envie a solicitação **TEARDOWN**. Você deve inserir o cabeçalho *Session* e usar o ID da sessão retornado na resposta do comando **SETUP**. Não inclua o cabeçalho *Transport* nesta solicitação.
- Leia a resposta do servidor.

Nota: Você deve inserir o cabeçalho *CSeq* em todas as solicitações que enviar. O valor do cabeçalho *CSeq* é um número que começa em 1 e é incrementado em um para cada solicitação enviada.

Estado do Cliente
Uma das principais diferenças entre HTTP e RTSP é que, no RTSP, cada sessão possui um estado. Neste laboratório, você precisará manter o estado do cliente atualizado. O estado do cliente muda quando ele recebe uma resposta do servidor, de acordo com o seguinte diagrama de estados.

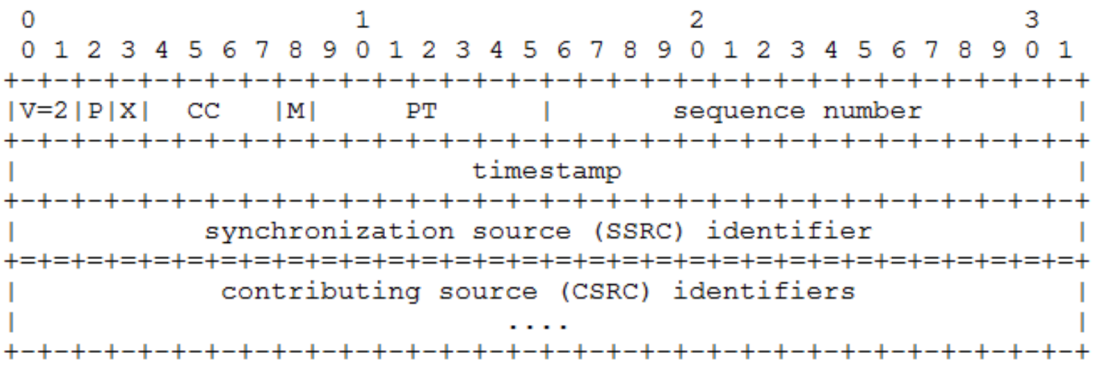


O Servidor
No lado do servidor, será necessário implementar a empacotamento dos dados de vídeo em pacotes RTP. Você precisará criar o pacote, configurar os campos no cabeçalho do pacote e copiar a carga útil (ou seja, um quadro de vídeo) para o pacote.

Quando o servidor recebe a solicitação **PLAY** do cliente, ele lê um quadro de vídeo do arquivo e cria um objeto *RtpPacket*, que encapsula o quadro de vídeo em formato RTP. Em seguida, o servidor envia o quadro ao cliente via UDP a cada 50 milissegundos.

Para a encapsulação, o servidor chama a função *encode* da classe *RtpPacket*. Sua tarefa é implementar essa função. Você precisará realizar o seguinte:
(as letras entre parênteses se referem aos campos no formato do pacote RTP mostrado abaixo).

- Definir o campo de versão RTP (*V*). Você deve configurá-lo como 2.
- Configurar os campos de preenchimento (*P*), extensão (*X*), número de fontes contribuintes (*CC*) e marcador (*M*). Todos esses campos são configurados como zero neste laboratório.
- Definir o campo de tipo de carga útil (*PT*). Neste laboratório, usamos MJPEG, cujo tipo é 26.
- Definir o número de sequência. O servidor fornece esse número como argumento *frameNbr* para a função *encode*.
- Configurar o carimbo de tempo (*timestamp*) usando o módulo *time* do Python.
- Definir o identificador de fonte (*SSRC*). Este campo identifica o servidor. Você pode escolher qualquer valor inteiro para isso.
- Como não há outras fontes contribuintes (campo *CC == 0*), o campo *CSRC* não existe. O comprimento do cabeçalho do pacote é, portanto, de 12 bytes, ou as três primeiras linhas do diagrama abaixo.



Manipulando os Bits
Aqui estão alguns exemplos de como configurar e verificar bits individuais ou grupos de bits. Observe que, no formato do cabeçalho do pacote RTP, os números de bits menores correspondem a bits de ordem superior, ou seja, o bit número 0 de um byte equivale a 272^727 e o bit número 7 equivale a 1 (ou 202^020). Nos exemplos abaixo, os números de bits referem-se aos números no diagrama mencionado anteriormente.

Como o campo de cabeçalho da classe *RtpPacket* é do tipo *bytearray*, você precisará configurar o cabeçalho byte por byte, ou seja, em grupos de 8 bits. O primeiro byte contém os bits de 0 a 7, o segundo byte contém os bits de 8 a 15, e assim por diante.

- Para configurar o bit número nnn na variável *mybyte* do tipo *byte*:

  ```
  python
  Copiar código
  mybyte = mybyte | 1 << (7 - n)
  ```

- Para configurar os bits nnn e n+1n + 1n+1 com o valor de *foo* na variável *mybyte*:

  ```
  python
  Copiar código
  mybyte = mybyte | foo << (7 - n)
  ```

  Note que *foo* deve ter um valor que possa ser expresso com 2 bits, ou seja, 0, 1, 2 ou 3.

- Para copiar um inteiro de 16 bits (*foo*) em 2 bytes (*b1* e *b2*):

  ```
  python
  Copiar código
  b1 = (foo >> 8) & 0xFF
  b2 = foo & 0xFF
  ```

  Após isso, *b1* conterá os 8 bits mais significativos de *foo* e *b2* conterá os 8 bits menos significativos de *foo*.

Você pode copiar um inteiro de 32 bits em 4 bytes de forma semelhante.

Tarefa opcional com bonificação extra

Atualmente, o cliente e o servidor implementam apenas as interações RTSP mínimas necessárias e o comando PAUSE. Implemente o método DESCRIBE, que é usado para fornecer informações sobre o fluxo de mídia. Quando o servidor recebe uma solicitação DESCRIBE, ele envia de volta um arquivo de descrição de sessão, que informa ao cliente quais tipos de fluxos estão na sessão e quais codificações são utilizadas.