Universidade Regional de Blumenau
Centro de Ciências Exatas e Naturais
Curso de Ciências da Computação

CRIAÇÃO DE STREAMING DE VÍDEO PARA TRANSMISSÃO DE SINAIS DE VÍDEO EM TEMPO REAL PELA INTERNET

Acadêmico: Clodoaldo Tschöke
Orientador: Francisco Adell Péricas

Roteiro da Apresentação

- **Introdução**
- **Objetivo**
- **Arquitetura Internet**
- ***Multimedia Networking***
  - **Serviços de vídeo**
  - **Streaming de vídeo na Internet**
  - **Protocolos de Tempo Real**
- **Técnicas e ferramentas utilizadas**
- **Desenvolvimento do protótipo**
  - **Especificação e Implementação do protótipo**
- **Operacionalidade do protótipo**
- **Conclusão**
  - **Extensões**

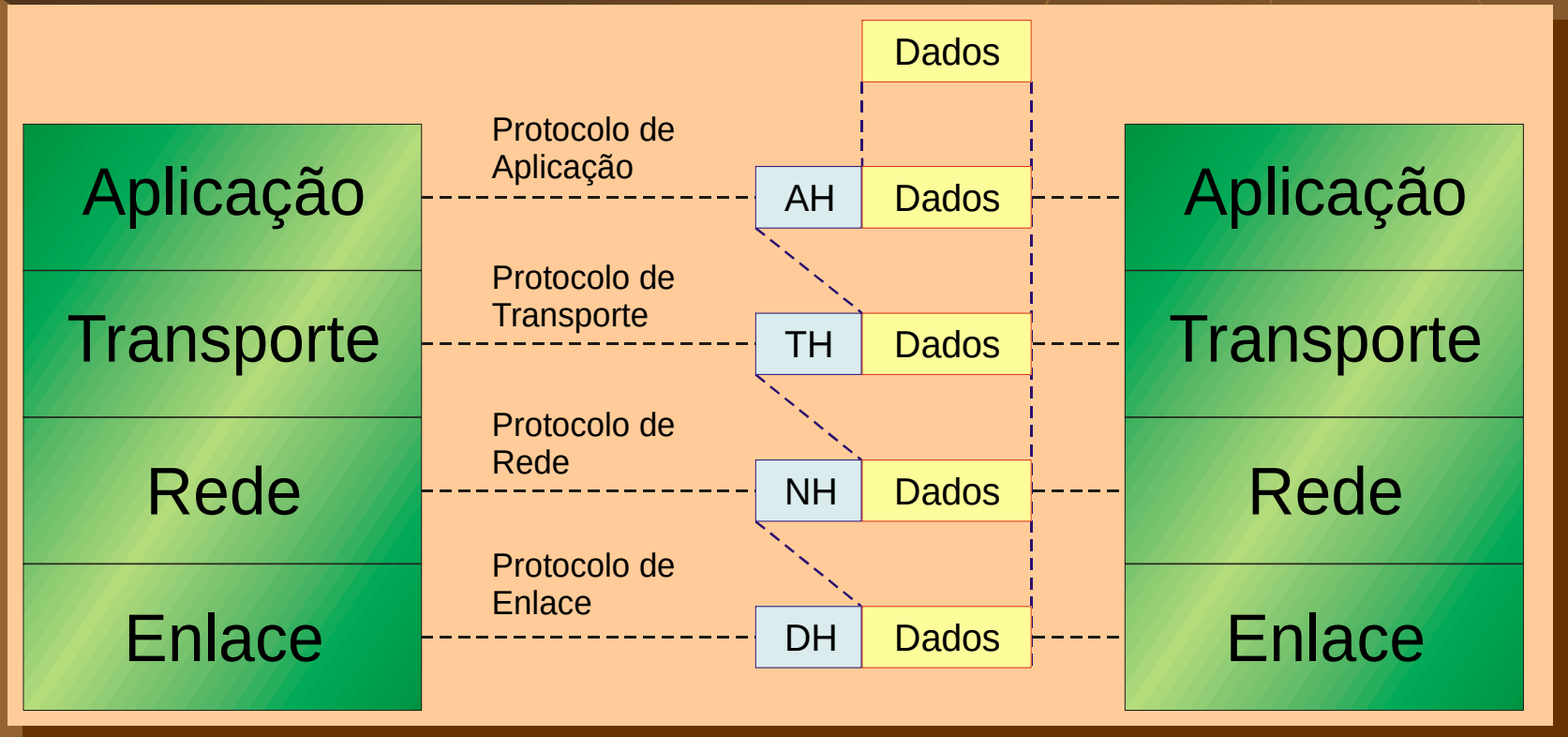
Introdução

- **Tráfego de informação multimídia na rede vem crescendo a cada dia**
- ***Streaming media* é uma técnica para transferência de áudio/vídeo pela Internet em tempo real**
- **Aplicações:**
  - **videoconferência**
  - **vídeo sob demanda**
  - **Web TV**
  - **jogos interativos**

Objetivo

Transformar um sinal de vídeo digital externo capturado por uma placa de vídeo, em um sinal de *streaming media* para a transmissão em tempo real pela Internet

Arquitetura Internet



Multimedia Networking

- **Infra-estrutura de hardware e software para suportar o transporte de informação multimídia na rede**
- **Características das aplicações:**
  - **considerações com o tempo**
  - **tolerância a perda**

Serviços de vídeo

- ***Streaming* de vídeo armazenado**
- ***Streaming* de vídeo ao vivo**
- **Vídeo interativo em tempo real**

Streaming de vídeo na Internet

- **Multimídia implica em um intenso tráfego de dados**
- **Agregar recursos de rede torna-se impraticável**
- **A Internet não pode garantir a chegada ordenada dos dados ao seu destino**
- **Os protocolos desenvolvidos precisam considerar o *multicast* para reduzir o tráfego**
- **Operações padrão para as aplicações gerenciarem o transporte e apresentação de dados multimídia**

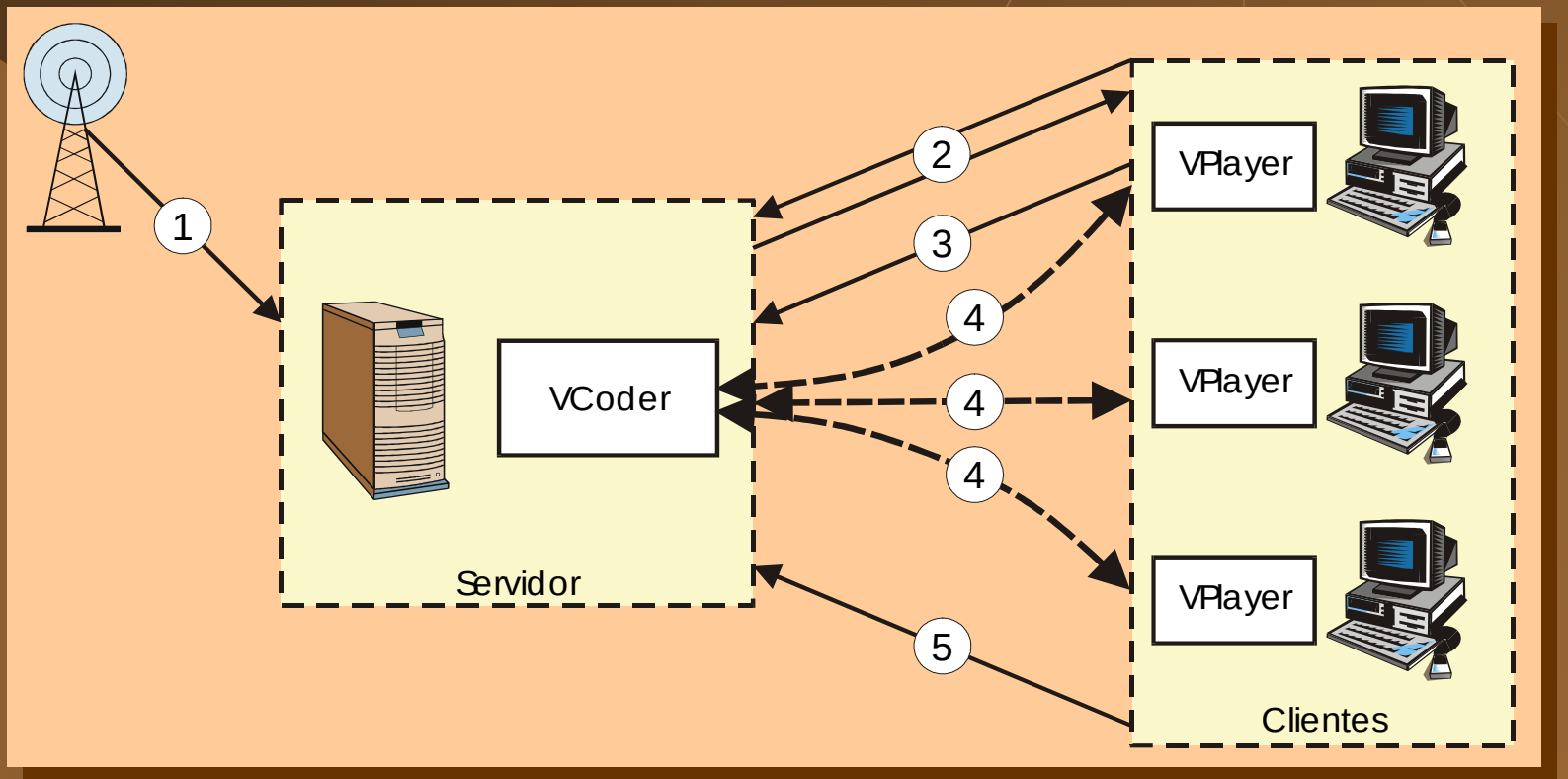
Protocolos de Tempo Real

- **Real-Time Streaming Procotol (RTSP)**
  - **controle da transmissão**
- **Real-Time Protocol (RTP)**
  - **transmissão do dado de áudio e vídeo**
- **Real-Time Control Protocol (RTCP)**
  - **monitoramento da entrega dos dados**
- **Resource ReServation Protocol (RSVP)**
  - **reserva de recursos**

Técnicas e ferramentas utilizadas

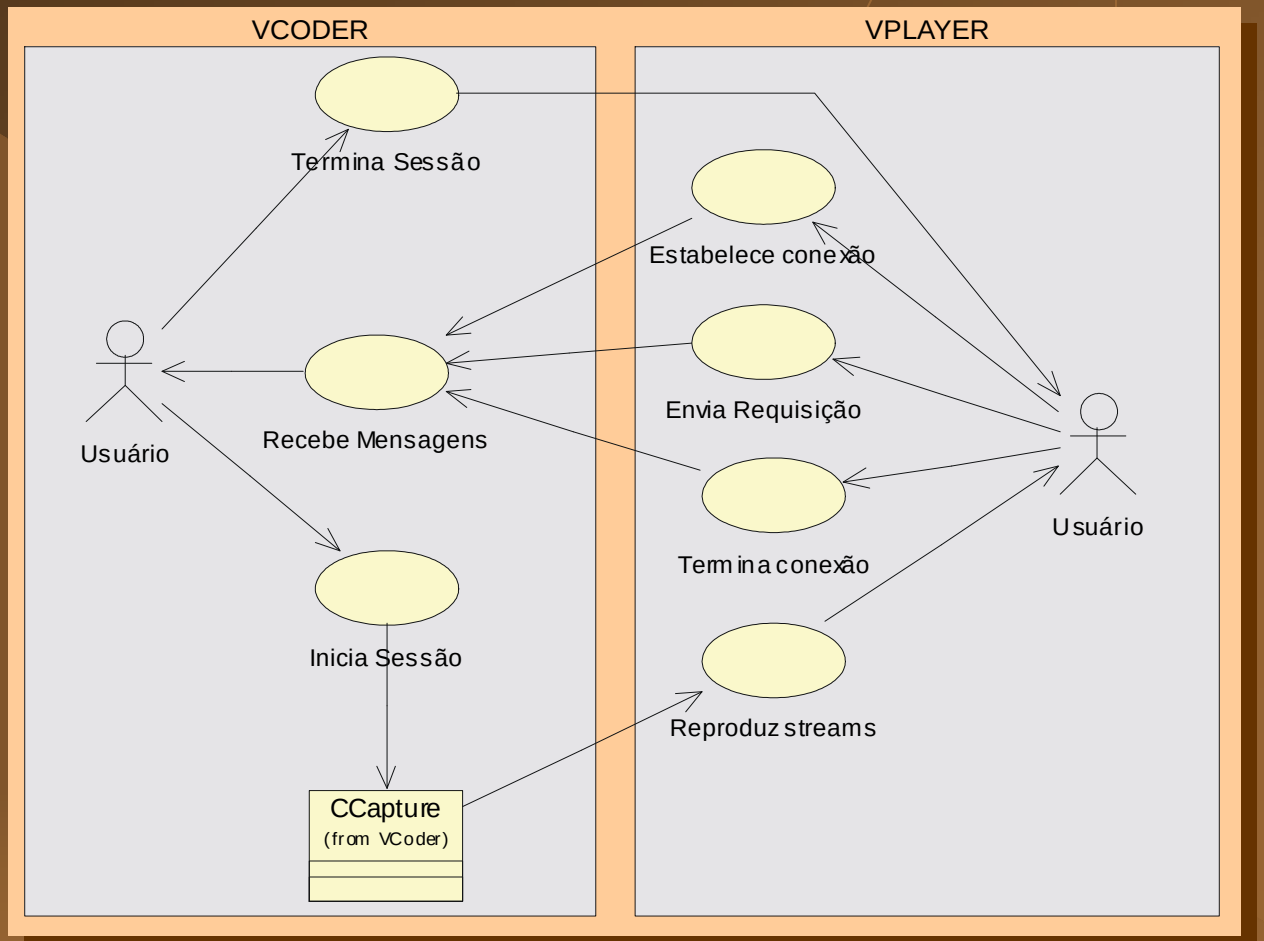
- **Orientação a objetos**
  - **UML (Unified Modeling Language)**
  - **Ferramenta Rational Rose**
- **Windows sockets API**
- **Biblioteca JRTPLib**
- **Microsoft Visual Studio**
  - **Visual C++ 6.0**
  - **MFC (Microsoft Foundation Classes)**

Desenvolvimento do protótipo de software



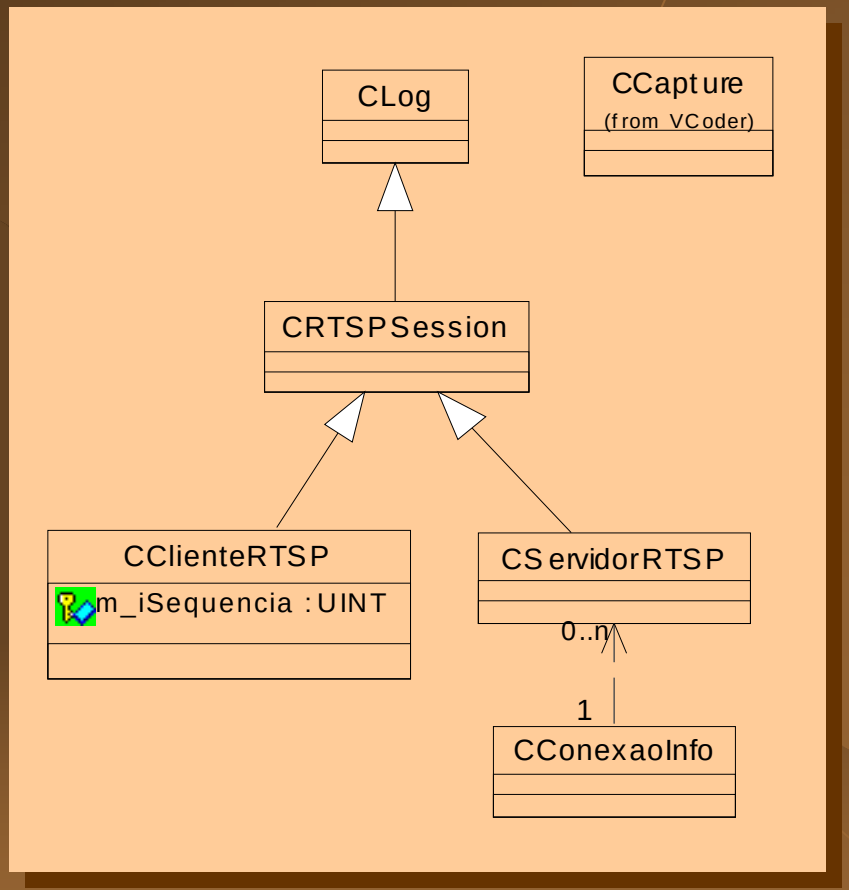
Especificação do protótipo

- **Diagrama de casos de uso**



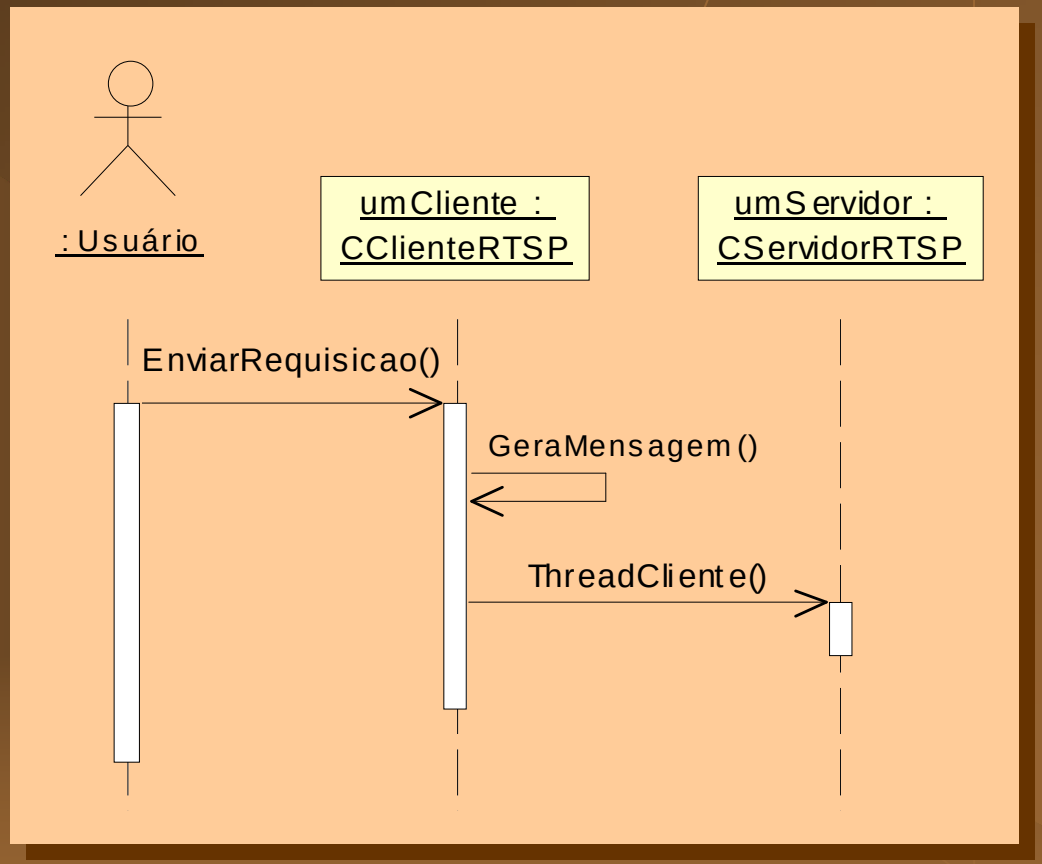
Especificação do protótipo

- **Diagrama de classes**



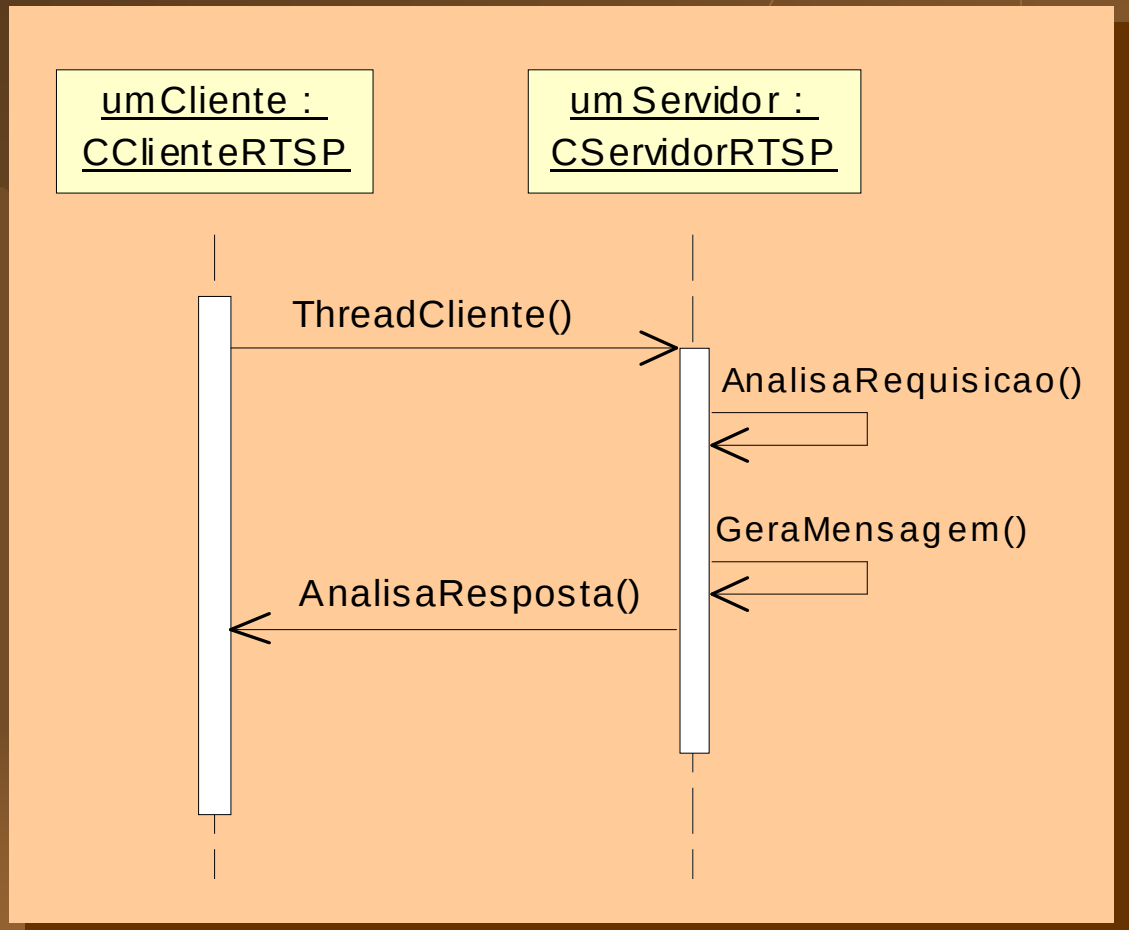
Especificação do protótipo

- **Diagrama de seqüência ENVIA REQUISIÇÃO**



Especificação do protótipo

- **Diagrama de seqüência RECEBE MENSAGENS**



Implementação do protótipo

```
int CClienteRTSP::EnviarRequisicao(SOCKET s, int iMensagem)
{
    if (s == INVALID_SOCKET)
    {
        WSASetLastError(WSAENOTSOCK);
        return SOCKET_ERROR;
    }

    int result;
    string szRequest, szResponse;
    WSABUF Buffer;
    DWORD dwNumeroBytesEnviados = 0;
    DWORD dwBytesEnviados = 0;

    result = GeraMensagem(iMensagem, m_iSessao, ++m_iSequencia, szRequest);
    if (result == SOCKET_ERROR)
        return result;

    //envia a mensagem
    do
    {
        Buffer.len = (szRequest.size() - dwBytesEnviados) >= SENDBLOCK ? SENDBLOCK : szRequest.size() - dwBytesEnviados;
        Buffer.buf = (char*)((DWORD)szRequest.c_str() + dwBytesEnviados);

        result = WSASend(s, &Buffer, 1, &dwNumeroBytesEnviados, 0, 0, NULL);
        if(result != SOCKET_ERROR)
            dwBytesEnviados += dwNumeroBytesEnviados;
    }
    while((dwBytesEnviados < (UINT)szRequest.size()) && SOCKET_ERROR != result);

    //
    // INFORMACOES ESTATISTICAS
    //
    m_estatisticas.nRTSPByteEnviado += (double)dwBytesEnviados;
    m_estatisticas.nRTSPEnviado++;
```

Implementação do protótipo

```
unsigned __stdcall CRTSPSession::ThreadCliente(LPVOID pParam)
{
    CConexaoInfo *pConexao = (CConexaoInfo*)pParam;
    CRTSPSession *pSession = pConexao->GetServidor();
    SOCKET s = pConexao->GetSocketRTSP();
                    .
                    .
                    .
    //
    // Processa eventos da rede
    //
    if(NetworkEvents.lNetworkEvents & FD_READ)
    {
        DWORD NumberOfBytesRecvd;
        WSABUF Buffers;
        DWORD dwBufferCount = 1;
        char szBuffer[MAX_BUFFER];
        DWORD Flags = 0;
        UINT iRequisicao = 0;
        Buffers.buf = szBuffer;
        Buffers.len = MAX_BUFFER;

        result = WSARecv(s, &Buffers, dwBufferCount, &NumberOfBytesRecvd, &Flags, NULL, NULL);
        if(result != SOCKET_ERROR)
        {
            pSession->m_estatisticas.nRTSPRecebido++;
            pSession->m_estatisticas.nRTSPByteRecebido += (double)NumberOfBytesRecvd;

            //
            // Verifica se a requisicao esta completa
            //
            szRequest += string(szBuffer, NumberOfBytesRecvd);
            if (!pSession->EstaCompleta(szRequest))
                continue;

            result = pSession->AnalisaRequisicao(s, szRequest, szResponse, iRequisicao);
                    .
                    .
                    .
```

Implementação do protótipo

```
//
// Envia resposta ao cliente
//
NumberOfBytesSent = 0;
dwBytesSent = 0;
do
{
    Buffer.len = (szResponse.size() - dwBytesSent) >= SENDBLOCK ? SENDBLOCK : szResponse.size() - dwBytesSent;
    Buffer.buf = (char*)((DWORD)szResponse.c_str() + dwBytesSent);

    result = WSASend(s, &Buffer, 1, &NumberOfBytesSent, 0, 0, NULL);
    if(SOCKET_ERROR != result)
        dwBytesSent += NumberOfBytesSent;
}
while((dwBytesSent < szResponse.size()) && SOCKET_ERROR != result);

if(WSAGetLastError() == WSAEWOULDBLOCK)
{
    bResend = TRUE;
    continue;
}

if(result != SOCKET_ERROR)
{
    pSession->m_estatisticas.nRTSPByteEnviado += (double)dwBytesSent;
    pSession->m_estatisticas.nRTSPEnviado++;
}
else
    pSession->GravaLog("WSASend(...) falhou", "ThreadCliente, primeiro envio", WSAGetLastError());

szRequest.erase(0, string::npos);
```

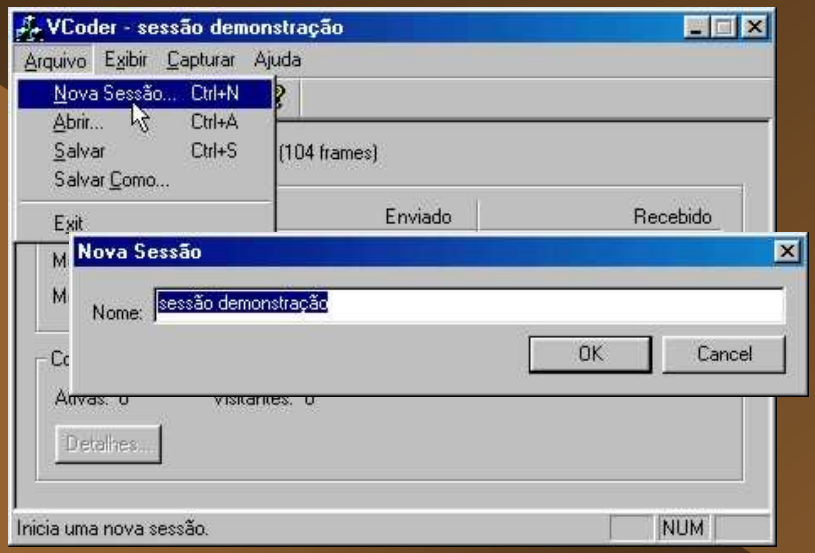
Implementação do protótipo

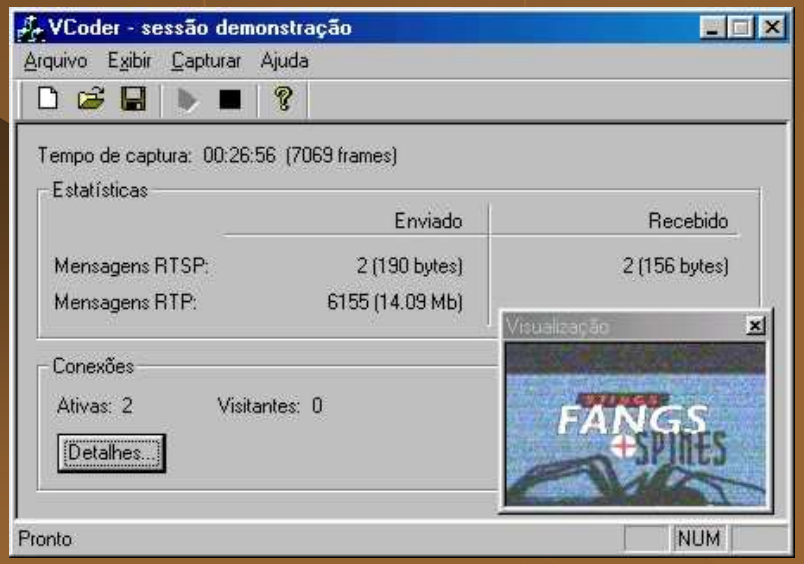
```
int CServidorRTSP::AnalisaRequisicao(SOCKET s, string szRequisicao, string &szResposta, UINT &iRequisicao)
{
          .
          .

   if (szOperacao == "PLAY" || szOperacao == "PAUSE" || szOperacao == "TEARDOWN")
   {
       //
       // Pega o endereco relacionado a sessao
       //
       struct sockaddr_in addr;
       if (!m_clientes.Lookup(iSessao, addr))
       {
           GeraMensagem(RTSP_RESPONSE_400, iSessao, iSeq, szResposta);
           return SOCKET_ERROR;
       }
       if (szOperacao == "PLAY")
       {
           iRequisicao = RTSP_PLAY;

           result = m_rtp.AddDestination(ntohl(addr.sin_addr.S_un.S_addr), ntohs(addr.sin_port));
           if (result < 0)
           {
               GravaLog(RTPGetErrorString(result));
               GeraMensagem(RTSP_RESPONSE_400, iSessao, iSeq, szResposta);
           }
           else
               GeraMensagem(RTSP_RESPONSE_OK, iSessao, iSeq, szResposta);
       }
       if (szOperacao == "PAUSE")
       {
           iRequisicao = RTSP_PAUSE;

           result = m_rtp.DeleteDestination(ntohl(addr.sin_addr.S_un.S_addr), ntohs(addr.sin_port));
           if (result < 0)
           {
               GravaLog(RTPGetErrorString(result));
               GeraMensagem(RTSP_RESPONSE_400, iSessao, iSeq, szResposta);
           }
           else
               GeraMensagem(RTSP_RESPONSE_OK, iSessao, iSeq, szResposta);
       }
          .
          .
          .
```

Implementação do protótipo

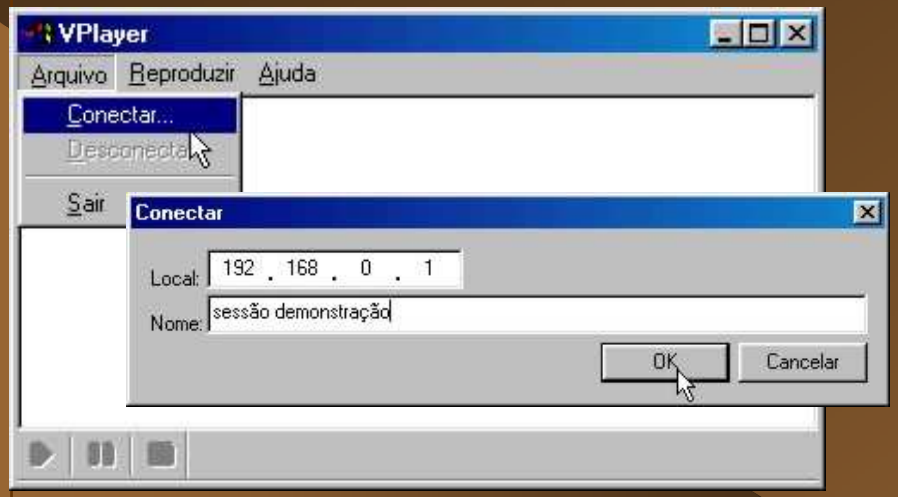
```
//mensagens enviadas pelo cliente
const char setup[]          = "SETUP %s/%s RTSP/1.0\nCSeq: %d\nTransport: RTP %d\r\n\r\n";
const char play[]           = "PLAY %s/%s RTSP/1.0\nCSeq: %d\nSession: %d\r\n\r\n";
const char pause[]          = "PAUSE %s/%s RTSP/1.0\nCSeq: %d\nSession: %d\r\n\r\n";
const char teardown[]       = "TEARDOWN %s/%s RTSP/1.0\nCSeq: %d\nSession: %d\r\n\r\n";


//mensagens enviadas pelo servidor
const char response_ok[]    = "RTSP/1.0 200 OK\nCSeq: %d\nSession: %d\nDate: %s\r\n\r\n";
const char response_setup[] = "RTSP/1.0 200 OK\nCSeq: %d\nSession: %d\nTransport: RTP %d\nDate: %s\r\n\r\n";
const char response_400[]   = "RTSP/1.0 400 Bad Request\nCSeq: %d\nSession: %d\nDate: %s\r\n\r\n";
const char response_454[]   = "RTSP/1.0 454 Session not found\nCSeq: %d\nDate: %s\r\n\r\n";
const char response_501[]   = "RTSP/1.0 501 Not Implemented\nCSeq: %d\nSession: %d\nDate: %s\r\n\r\n";
const char response_551[]   = "RTSP/1.0 551 Option not supported\nCSeq: %d\nSession: %d\nDate: %s\r\n\r\n";
```

Operacionalidade do protótipo





Operacionalidade do protótipo



Conclusão

- **Constitui-se tema relativamente novo e possuidor de problemas a serem solucionados**
  - **A estrutura atual da rede não atende aos propósitos da transmissão de informação em tempo real**
- **Propostas têm sido elaboradas**
  - **Protocolos de tempo real**

Extensões

- **Compressão de vídeo**
- **Aprimoramento da implementação do protocolo RTSP**
- ***Streaming* de áudio**

DEMONSTRAÇÃO...

Implementação do protótipo

```
int CClienteRTSP::GeraMensagem(int iMensagem, int iSessao, int iCSeq, string &szMensagem)
{
    char pMens[2048];
    char szDT[128];
    struct tm *newtime;
    long ltime;

    //
    // Gera string de data GMT atual
    //
    time(&ltime);
    newtime = gmtime(&ltime);
    strftime(szDT, 128, "%a, %d %b %Y %H:%M:%S GMT", newtime);

    switch (iMensagem)
    {
        case RTSP_SETUP:
            sprintf(pMens, setup, m_szDestino.c_str(), m_szNomeSessao.c_str(), iCSeq, m_iPortaRTP);
            break;

        case RTSP_PLAY:
            sprintf(pMens, play, m_szDestino.c_str(), m_szNomeSessao.c_str(), iCSeq, iSessao);
            break;

        case RTSP_PAUSE:
            sprintf(pMens, pause, m_szDestino.c_str(), m_szNomeSessao.c_str(), iCSeq, iSessao);
            break;

        case RTSP_TEARDOWN:
            sprintf(pMens, teardown, m_szDestino.c_str(), m_szNomeSessao.c_str(), iCSeq, m_iSessao);
            break;

        case RTSP_RESPONSE_OK:
        case RTSP_RESPONSE_OKS:
        case RTSP_RESPONSE_400:
        case RTSP_RESPONSE_454:
        case RTSP_RESPONSE_501:
        case RTSP_RESPONSE_551:
        default:
            ASSERT(FALSE);
            WSASetLastError(WSAENOBUFS);
            return SOCKET_ERROR;
    }

    szMensagem = pMens;
    return 0;
}
```

Implementação do protótipo

```
int CServidorRTSP::GeraMensagem(int iMensagem, int iSessao, int iCSeq, string &szMensagem)
{
        .
        .
        .
    switch (iMensagem)
    {
        case RTSP_RESPONSE_OK:
            sprintf(pMens, response_ok, iCSeq, iSessao, szDT);
            break;

        case RTSP_RESPONSE_OKS:
            sprintf(pMens, response_setup, iCSeq, iSessao, m_iPortaRTP, szDT);
            break;

        case RTSP_RESPONSE_400:
            sprintf(pMens, response_400, iCSeq, iSessao, szDT);
            break;

        case RTSP_RESPONSE_454:
            sprintf(pMens, response_454, iCSeq, szDT);
            break;

        case RTSP_RESPONSE_501:
            sprintf(pMens, response_501, iCSeq, iSessao, szDT);
            break;

        case RTSP_RESPONSE_551:
            sprintf(pMens, response_551, iCSeq, iSessao, szDT);
            break;

        case RTSP_SETUP:
        case RTSP_PLAY:
        case RTSP_PAUSE:
        case RTSP_TEARDOWN:
        default:
            ASSERT(FALSE);
            WSASetLastError(WSAENOBUFS);
            return SOCKET_ERROR;
    }

    szMensagem = pMens;
    return 0;
}
```