GUIA DE INTEGRAÇÃO

P2P Intelbras para VMS

Solução Completa para Portaria Remota

Informação	Valor
Data	11/01/2026
Versão	1.0 - FINAL
Status	■ TESTADO E FUNCIONANDO
Compatibilidade	DVRs/NVRs Intelbras e Dahua

ÍNDICE

1. VISÃO GERAL DA SOLUÇÃO

Esta solução permite conectar a DVRs/NVRs Intelbras e Dahua através da infraestrutura P2P cloud, sem necessidade de IP público, DDNS ou liberação de portas no roteador do cliente.

Como Funciona:

1. O DVR mantém conexão permanente com servidores P2P da Intelbras
2. Seu VMS se conecta aos mesmos servidores usando o número de série do DVR
3. Um túnel TCP local é criado (ex: 127.0.0.1:17777)
4. O SDK Dahua conecta neste túnel como se fosse conexão direta
5. Os dados fluem: VMS → Túnel Local → Servidor P2P → DVR

Vantagens:

- ■ Funciona com clientes Starlink, NAT, CGNAT
- ■ Não precisa de IP público ou DDNS
- ■ Não precisa liberar portas no roteador
- ■ Não precisa de conta Intelbras Cloud
- ■ Compatível com SDK Dahua existente
- ■ Mesma qualidade de conexão direta

2. ARQUIVOS NECESSÁRIOS

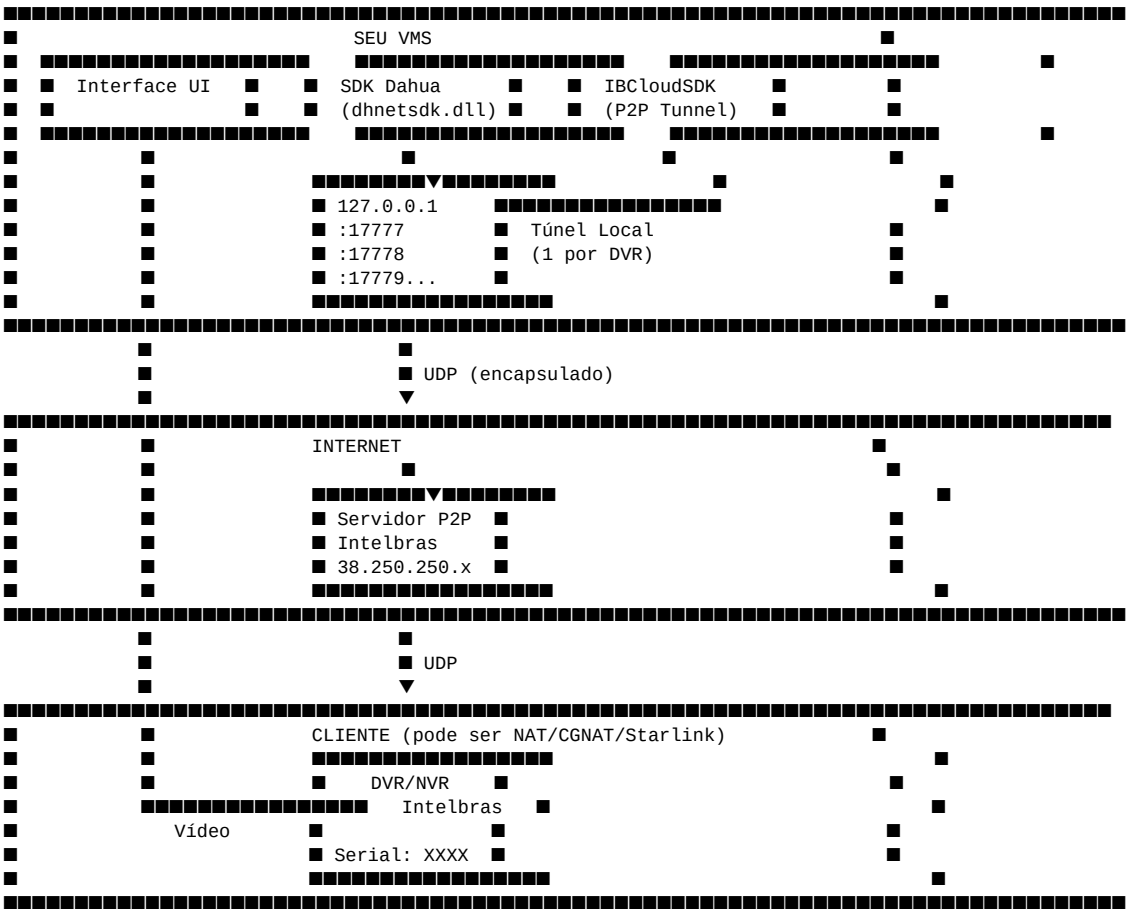
■■ IMPORTANTE: Todas as DLLs devem estar na mesma pasta do executável!

Arquivo	Tamanho	Obrigatório	Descrição
IBCloudSDK.dll	109 KB	SIM	SDK principal P2P (32-bit)
libt2u.dll	237 KB	SIM	Core do túnel P2P
dhnetsdk.dll	18 MB	SIM	SDK Dahua para vídeo
dhplay.dll	807 KB	SIM	Player Dahua
Infra.dll	1.1 MB	SIM	Infraestrutura
NetFramework.dll	590 KB	SIM	Framework de rede
Stream.dll	295 KB	SIM	Streaming
StreamSvr.dll	1.2 MB	SIM	Servidor de stream
avnetsdk.dll	3.2 MB	SIM	SDK de rede AV
dhconfigsdk.dll	3.2 MB	Opcional	Configuração
h264dec.dll	567 KB	SIM	Decodificador H.264
hevcdec.dll	778 KB	SIM	Decodificador H.265

Arquitetura: 32-bit

A IBCloudSDK.dll é 32-bit. Seu VMS deve ser compilado como 32-bit ou usar um processo auxiliar 32-bit para gerenciar os túneis.

3. ARQUITETURA DO SISTEMA



4. API IBCloudSDK - REFERÊNCIA COMPLETA

Funções Principais:

Função	Parâmetros	Retorno	Descrição
IBCloud_init	(void)	int	Inicializa SDK. Retorna 0 = sucesso
IBCloud_exit	(void)	void	Finaliza SDK e fecha túneis
IBCloud_status	(void)	int	Status: 0=conectado, -1=erro
IBCloud_query	(char* serial)	int	1=encontrado, -1=não encontrado
IBCloud_add_port	(serial, remote, local)	int	Porta local ou -1=erro
IBCloud_del_port	(ushort port)	void	Remove túnel da porta
IBCloud_automatic_shutdown	(int enable)	int	0=desabilita shutdown
IBCloud_connection_timeout	(int ms)	int	Define timeout em ms

■■ ORDEM DE CHAMADA OBRIGATÓRIA:

1. IBCloud_automatic_shutdown(0) ← PRIMEIRO! Desabilita shutdown
2. IBCloud_connection_timeout(60000) ← Timeout de 60 segundos
3. IBCloud_init() ← Inicializa conexão com servidor P2P
4. Aguardar 2-3 segundos
5. IBCloud_query(serial) ← Verifica se DVR está online
6. IBCloud_add_port(serial, 37777, porta_local) ← Cria túnel
7. CLIENT_Login('127.0.0.1', porta_local, ...) ← Conecta SDK Dahua

5. CÓDIGO DELPHI - CLASSE TIntelbrasP2P

```
unit IntelbrasP2P;

interface

uses
  System.SysUtils, System.Classes, Winapi.Windows, System.SyncObjs;

type
  TIntelbrasP2P = class
  private
    FDLLHandle: THandle;
    FInitialized: Boolean;
    FTunnels: TStringList;  // Serial=Porta
    FLock: TCriticalSection;

    // Ponteiros para funções da DLL
    FIBCloud_init: function: Integer; cdecl;
    FIBCloud_exit: procedure; cdecl;
    FIBCloud_status: function: Integer; cdecl;
    FIBCloud_query: function(uuid: PAnsiChar): Integer; cdecl;
    FIBCloud_add_port: function(uuid: PAnsiChar; remote_port, local_port: Word): Integer; cdecl;
    FIBCloud_del_port: procedure(port: Word); cdecl;
    FIBCloud_automatic_shutdown: function(enable: Integer): Integer; cdecl;
    FIBCloud_connection_timeout: function(timeout_ms: Integer): Integer; cdecl;

    function GetNextLocalPort: Word;
  public
    constructor Create;
    destructor Destroy; override;

    function Initialize: Boolean;
    procedure Finalize;

    function IsDeviceOnline(const Serial: string): Boolean;
    function CreateTunnel(const Serial: string; RemotePort: Word = 37777): Integer;
    procedure CloseTunnel(const Serial: string);
    procedure CloseAllTunnels;

    function GetTunnelPort(const Serial: string): Integer;

    property Initialized: Boolean read FInitialized;
  end;

var
  P2PManager: TIntelbrasP2P;

implementation

const
  BASE_LOCAL_PORT = 17777;
  MAX_TUNNELS = 100;

constructor TIntelbrasP2P.Create;
begin
  inherited;
  FInitialized := False;
  FTunnels := TStringList.Create;
  FLock := TCriticalSection.Create;
  FDLLHandle := 0;
end;

destructor TIntelbrasP2P.Destroy;
begin
  Finalize;
  FTunnels.Free;
  FLock.Free;
  inherited;
end;

function TIntelbrasP2P.GetNextLocalPort: Word;
var
  I: Integer;
  Port: Word;
  InUse: Boolean;
begin
  for Port := BASE_LOCAL_PORT to BASE_LOCAL_PORT + MAX_TUNNELS do
```

```
  begin
    InUse := False;
    for I := 0 to FTunnels.Count - 1 do
    begin
      if StrToIntDef(FTunnels.ValueFromIndex[I], 0) = Port then
      begin
        InUse := True;
        Break;
      end;
    end;
    if not InUse then
      Exit(Port);
  end;
  Result := 0;
end;

function TIntelbrasP2P.Initialize: Boolean;
var
  InitResult: Integer;
begin
  Result := False;
  FLock.Enter;
  try
    if FInitialized then
      Exit(True);

    // Carregar DLL
    FDLLHandle := LoadLibrary('IBCloudSDK.dll');
    if FDLLHandle = 0 then
    begin
      raise Exception.Create('Erro ao carregar IBCloudSDK.dll');
    end;

    // Obter ponteiros das funções
    @FIBCloud_init := GetProcAddress(FDLLHandle, 'IBCloud_init');
    @FIBCloud_exit := GetProcAddress(FDLLHandle, 'IBCloud_exit');
    @FIBCloud_status := GetProcAddress(FDLLHandle, 'IBCloud_status');
    @FIBCloud_query := GetProcAddress(FDLLHandle, 'IBCloud_query');
    @FIBCloud_add_port := GetProcAddress(FDLLHandle, 'IBCloud_add_port');
    @FIBCloud_del_port := GetProcAddress(FDLLHandle, 'IBCloud_del_port');
    @FIBCloud_automatic_shutdown := GetProcAddress(FDLLHandle, 'IBCloud_automatic_shutdown');
    @FIBCloud_connection_timeout := GetProcAddress(FDLLHandle, 'IBCloud_connection_timeout');

    // Verificar funções essenciais
    if not Assigned(FIBCloud_init) or not Assigned(FIBCloud_add_port) then
    begin
      FreeLibrary(FDLLHandle);
      FDLLHandle := 0;
      raise Exception.Create('Funções não encontradas na DLL');
    end;

    // IMPORTANTE: Desabilitar shutdown automático ANTES de init
    if Assigned(FIBCloud_automatic_shutdown) then
      FIBCloud_automatic_shutdown(0);

    // Configurar timeout (60 segundos)
    if Assigned(FIBCloud_connection_timeout) then
      FIBCloud_connection_timeout(60000);

    // Inicializar
    InitResult := FIBCloud_init();

    // Aguardar conexão
    Sleep(3000);

    FInitialized := True;
    Result := True;
  finally
    FLock.Leave;
  end;
end;

procedure TIntelbrasP2P.Finalize;
begin
  FLock.Enter;
  try
    if not FInitialized then
      Exit;
```

```
    CloseAllTunnels;

    if Assigned(FIBCloud_exit) then
      FIBCloud_exit();

    if FDLLHandle <> 0 then
    begin
      FreeLibrary(FDLLHandle);
      FDLLHandle := 0;
    end;

    FInitialized := False;
  finally
    FLock.Leave;
  end;
end;

function TIntelbrasP2P.IsDeviceOnline(const Serial: string): Boolean;
begin
  Result := False;
  if not FInitialized or not Assigned(FIBCloud_query) then
    Exit;

  Result := FIBCloud_query(PAnsiChar(AnsiString(Serial))) = 1;
end;

function TIntelbrasP2P.CreateTunnel(const Serial: string; RemotePort: Word): Integer;
var
  LocalPort: Word;
  PortResult: Integer;
  Idx: Integer;
begin
  Result := -1;

  FLock.Enter;
  try
    if not FInitialized then
      Exit;

    // Verificar se já existe túnel para este serial
    Idx := FTunnels.IndexOfName(Serial);
    if Idx >= 0 then
    begin
      Result := StrToIntDef(FTunnels.ValueFromIndex[Idx], -1);
      Exit;
    end;

    // Verificar se dispositivo está online
    if not IsDeviceOnline(Serial) then
      Exit;

    // Obter próxima porta disponível
    LocalPort := GetNextLocalPort;
    if LocalPort = 0 then
      Exit;

    // Criar túnel
    PortResult := FIBCloud_add_port(PAnsiChar(AnsiString(Serial)), RemotePort, LocalPort);

    if PortResult > 0 then
    begin
      FTunnels.Add(Serial + '=' + IntToStr(PortResult));
      Result := PortResult;
    end;
  finally
    FLock.Leave;
  end;
end;

procedure TIntelbrasP2P.CloseTunnel(const Serial: string);
var
  Idx: Integer;
  Port: Word;
begin
  FLock.Enter;
  try
    Idx := FTunnels.IndexOfName(Serial);
    if Idx >= 0 then
    begin
```

```
      Port := StrToIntDef(FTunnels.ValueFromIndex[Idx], 0);
      if (Port > 0) and Assigned(FIBCloud_del_port) then
        FIBCloud_del_port(Port);
      FTunnels.Delete(Idx);
    end;
  finally
    FLock.Leave;
  end;
end;

procedure TIntelbrasP2P.CloseAllTunnels;
var
  I: Integer;
  Port: Word;
begin
  FLock.Enter;
  try
    for I := FTunnels.Count - 1 downto 0 do
    begin
      Port := StrToIntDef(FTunnels.ValueFromIndex[I], 0);
      if (Port > 0) and Assigned(FIBCloud_del_port) then
        FIBCloud_del_port(Port);
    end;
    FTunnels.Clear;
  finally
    FLock.Leave;
  end;
end;

function TIntelbrasP2P.GetTunnelPort(const Serial: string): Integer;
var
  Idx: Integer;
begin
  Result := -1;
  Idx := FTunnels.IndexOfName(Serial);
  if Idx >= 0 then
    Result := StrToIntDef(FTunnels.ValueFromIndex[Idx], -1);
end;

initialization
  P2PManager := TIntelbrasP2P.Create;

finalization
  P2PManager.Free;

end.
```

6. CÓDIGO C# - CLASSE IntelbrasP2P

```
using System;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System.Threading;

namespace VMS.P2P
{
    public class IntelbrasP2P : IDisposable
    {
        #region DLL Imports
        [DllImport("IBCloudSDK.dll", CallingConvention = CallingConvention.Cdecl)]
        private static extern int IBCloud_init();

        [DllImport("IBCloudSDK.dll", CallingConvention = CallingConvention.Cdecl)]
        private static extern void IBCloud_exit();

        [DllImport("IBCloudSDK.dll", CallingConvention = CallingConvention.Cdecl)]
        private static extern int IBCloud_status();

        [DllImport("IBCloudSDK.dll", CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Ansi)]
        private static extern int IBCloud_query(string uuid);

        [DllImport("IBCloudSDK.dll", CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Ansi)]
        private static extern int IBCloud_add_port(string uuid, ushort remote_port, ushort local_port);

        [DllImport("IBCloudSDK.dll", CallingConvention = CallingConvention.Cdecl)]
        private static extern void IBCloud_del_port(ushort port);

        [DllImport("IBCloudSDK.dll", CallingConvention = CallingConvention.Cdecl)]
        private static extern int IBCloud_automatic_shutdown(int enable);

        [DllImport("IBCloudSDK.dll", CallingConvention = CallingConvention.Cdecl)]
        private static extern int IBCloud_connection_timeout(int timeout_ms);
        #endregion

        private const ushort BASE_PORT = 17777;
        private const int MAX_TUNNELS = 100;

        private bool _initialized = false;
        private readonly Dictionary<string, ushort> _tunnels = new Dictionary<string, ushort>();
        private readonly object _lock = new object();

        private static IntelbrasP2P _instance;
        public static IntelbrasP2P Instance => _instance ?? (_instance = new IntelbrasP2P());

        public bool IsInitialized => _initialized;

        public bool Initialize()
        {
            lock (_lock)
            {
                if (_initialized) return true;

                try
                {
                    // IMPORTANTE: Desabilitar shutdown ANTES de init
                    IBCloud_automatic_shutdown(0);

                    // Configurar timeout (60 segundos)
                    IBCloud_connection_timeout(60000);

                    // Inicializar
                    int result = IBCloud_init();

                    // Aguardar conexão
                    Thread.Sleep(3000);

                    _initialized = true;
                    return true;
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Erro ao inicializar P2P: {ex.Message}");
                    return false;
                }
            }
```

```
}

public void Shutdown()
{
    lock (_lock)
    {
        if (!_initialized) return;

        CloseAllTunnels();
        IBCloud_exit();
        _initialized = false;
    }
}

public bool IsDeviceOnline(string serial)
{
    if (!_initialized) return false;
    return IBCloud_query(serial) == 1;
}

public int CreateTunnel(string serial, ushort remotePort = 37777)
{
    lock (_lock)
    {
        if (!_initialized) return -1;

        // Já existe túnel?
        if (_tunnels.ContainsKey(serial))
            return _tunnels[serial];

        // Dispositivo online?
        if (!IsDeviceOnline(serial))
            return -1;

        // Próxima porta disponível
        ushort localPort = GetNextPort();
        if (localPort == 0) return -1;

        // Criar túnel
        int result = IBCloud_add_port(serial, remotePort, localPort);

        if (result > 0)
        {
            _tunnels[serial] = (ushort)result;
            return result;
        }

        return -1;
    }
}

public void CloseTunnel(string serial)
{
    lock (_lock)
    {
        if (_tunnels.TryGetValue(serial, out ushort port))
        {
            IBCloud_del_port(port);
            _tunnels.Remove(serial);
        }
    }
}

public void CloseAllTunnels()
{
    lock (_lock)
    {
        foreach (var kvp in _tunnels)
        {
            IBCloud_del_port(kvp.Value);
        }
        _tunnels.Clear();
    }
}

public int GetTunnelPort(string serial)
{
    lock (_lock)
    {
```

```
                return _tunnels.TryGetValue(serial, out ushort port) ? port : -1;
            }
        }

        private ushort GetNextPort()
        {
            for (ushort p = BASE_PORT; p < BASE_PORT + MAX_TUNNELS; p++)
            {
                if (!_tunnels.ContainsValue(p))
                    return p;
            }
            return 0;
        }

        public void Dispose()
        {
            Shutdown();
        }
    }
}
```

7. FLUXO DE INTEGRAÇÃO NO VMS

Inicialização do Sistema (uma vez ao iniciar o VMS):

```
// Delphi
procedure TFormMain.FormCreate(Sender: TObject);
begin
  // Inicializar P2P uma única vez
  if not P2PManager.Initialize then
    ShowMessage('Erro ao inicializar módulo P2P');
end;

procedure TFormMain.FormDestroy(Sender: TObject);
begin
  // Finalizar P2P
  P2PManager.Finalize;
end;
```

Conexão a um DVR P2P:

```
// Delphi - Conectar a DVR via P2P
function TFormMain.ConectarDVR_P2P(const Serial, Usuario, Senha: string): Boolean;
var
  LocalPort: Integer;
  LoginID: Integer;
begin
  Result := False;

  // Criar túnel P2P
  LocalPort := P2PManager.CreateTunnel(Serial);
  if LocalPort < 0 then
  begin
    ShowMessage('DVR não encontrado ou offline: ' + Serial);
    Exit;
  end;

  // Conectar SDK Dahua no túnel local
  LoginID := CLIENT_Login(
    PAnsiChar(AnsiString('127.0.0.1')),  // IP local
    LocalPort,                           // Porta do túnel
    PAnsiChar(AnsiString(Usuario)),
    PAnsiChar(AnsiString(Senha)),
    @DeviceInfo,
    @ErrorCode
  );

  if LoginID > 0 then
  begin
    // Sucesso! Salvar LoginID para operações futuras
    FLoginID := LoginID;
    FSerial := Serial;
    Result := True;
  end
  else
  begin
    // Falha - fechar túnel
    P2PManager.CloseTunnel(Serial);
    ShowMessage('Falha ao conectar. Erro: ' + IntToStr(ErrorCode));
  end;
end;
```

Desconexão:

```
// Delphi - Desconectar DVR
procedure TFormMain.DesconectarDVR;
begin
  if FLoginID > 0 then
  begin
    CLIENT_Logout(FLoginID);
    FLoginID := 0;
  end;

  if FSerial <> '' then
```

```
  begin
    P2PManager.CloseTunnel(FSerial);
    FSerial := '';
  end;
end;
```

8. EXEMPLO PRÁTICO - CONEXÃO COMPLETA

Código completo para conectar e visualizar câmera:

```
// C# - Exemplo completo
public class CameraP2P
{
    private int _loginId = 0;
    private string _serial = "";

    public bool Conectar(string serial, string usuario, string senha)
    {
        // 1. Inicializar P2P (se ainda não foi)
        if (!IntelbrasP2P.Instance.IsInitialized)
        {
            if (!IntelbrasP2P.Instance.Initialize())
            {
                Console.WriteLine("Erro ao inicializar P2P");
                return false;
            }
        }

        // 2. Criar túnel
        int porta = IntelbrasP2P.Instance.CreateTunnel(serial);
        if (porta < 0)
        {
            Console.WriteLine($"DVR {serial} offline ou não encontrado");
            return false;
        }

        Console.WriteLine($"Túnel criado: 127.0.0.1:{porta}");

        // 3. Conectar SDK Dahua
        NET_DEVICEINFO deviceInfo = new NET_DEVICEINFO();
        int error = 0;

        _loginId = NETClient.Login(
            "127.0.0.1",    // Sempre localhost
            (ushort)porta,   // Porta do túnel
            usuario,
            senha,
            out deviceInfo,
            out error
        );

        if (_loginId > 0)
        {
            _serial = serial;
            Console.WriteLine($"Conectado! Canais: {deviceInfo.byChanNum}");
            return true;
        }
        else
        {
            IntelbrasP2P.Instance.CloseTunnel(serial);
            Console.WriteLine($"Erro de login: {error}");
            return false;
        }
    }

    public void IniciarVideo(int canal, IntPtr handle)
    {
        if (_loginId <= 0) return;

        // Iniciar preview no handle (PictureBox, Panel, etc)
        int playHandle = NETClient.RealPlay(_loginId, canal, handle);

        if (playHandle > 0)
            Console.WriteLine($"Vídeo iniciado no canal {canal}");
    }

    public void Desconectar()
    {
        if (_loginId > 0)
        {
            NETClient.Logout(_loginId);
            _loginId = 0;
```

```
        }

        if (!string.IsNullOrEmpty(_serial))
        {
            IntelbrasP2P.Instance.CloseTunnel(_serial);
            _serial = "";
        }
    }
}

// Uso:
var camera = new CameraP2P();
if (camera.Conectar("OJHL0700323ZS", "admin", "aj213141"))
{
    camera.IniciarVideo(0, pictureBox1.Handle);
}
```

9. GERENCIAMENTO DE MÚLTIPLOS DISPOSITIVOS

O sistema suporta múltiplos túneis simultâneos. Cada DVR recebe uma porta local diferente:

DVR Serial	Porta Remota	Porta Local	Status
OJHL0700323ZS	37777	17777	Conectado
1ZRI1004554LZ	37777	17778	Conectado
DOJ0002540617	37777	17779	Conectado
ABC123456789	37777	17780	Disponível

Dicas para múltiplos dispositivos:

• Inicialize o P2P uma única vez ao iniciar o VMS

• Crie túneis sob demanda (quando usuário abrir câmera)

• Feche túneis quando não estiverem em uso (economia de recursos)

• Use thread separada para criar túneis (não bloquear UI)

• Implemente reconexão automática se túnel cair

10. TROUBLESHOOTING

Erro	Causa	Solução
BadImageFormatException	Arquitetura 32/64 bit	Compilar VMS como 32-bit
DllNotFoundException	DLL não encontrada	Copiar TODAS DLLs para pasta do EXE
IBCloud_query retorna -1	DVR offline	Verificar se DVR está conectado à internet
IBCloud_add_port retorna -1	Túnel já existe	Usar GetTunnelPort() primeiro
CLIENT_Login falha	Credenciais erradas	Verificar usuário/senha do DVR
Túnel cai após segundos	Shutdown automático	Chamar IBCloud_automatic_shutdown(0)
Timeout de conexão	Timeout curto	Chamar IBCloud_connection_timeout(60000)
Vídeo não aparece	Handle inválido	Verificar se Handle do controle é válido

Checklist de Integração:

■ Todas as DLLs copiadas para pasta do executável

■ Projeto compilado como 32-bit (x86)

■ IBCloud_automatic_shutdown(0) chamado ANTES de init

■ IBCloud_connection_timeout(60000) configurado

■ IBCloud_init() chamado uma única vez

■ Aguardar 2-3 segundos após init

■ IBCloud_query() para verificar se DVR está online

■ IBCloud_add_port() para criar túnel

■ CLIENT_Login() com IP 127.0.0.1 e porta do túnel

Documento gerado em: 11/01/2026 10:02:24

Solução testada e funcionando com SIMNext e DVR iNVD 1016