Transport SDK Programming Guide

Loongtek.com

SDK Features

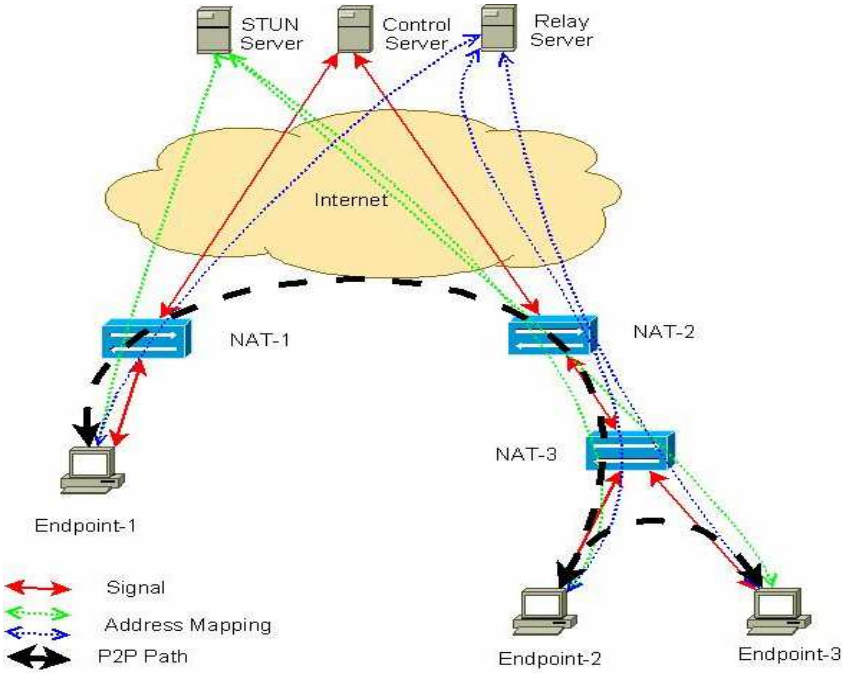
Transport SDK is a P2P tunnel library, following is the main features:

- **Support ICE, STUN and Relay protocols for traversing NATs.**
- **Embedded STUN and Relay server**
- **Select the best connection from multiple available paths between two endpoints.**
- **Provide reliable packets delivery mechanism based on ICE connections**
- **Encrypt/Decrypt packets using Blowfish algorithm.**

Install

This library build using VC2005, if your host does not install .net framework please firstly run vcredist_x86.exe, can download from
http://www.microsoft.com/downloads/details.aspx?familyid=32bc1bee-a3f9-4c13-9c99-220b62a191ee&displaylang=en
Your Apps should include header file transport.h and link transport.lib.

Brief Introduce



As above figure showing there are two endpoint behided NATs, STUN/Relay servers are on the Internet, and a central server is also required to control connecting signals. Every endpoint has many IP addresses:

1. Local IP address based on host network interfaces, every network interface have a private or public IP, which is the prime address of the endpoint.
2. If the endpoint behided NATs there exist a mapping IP address from WAN side view. Clients can know their mapping IP address by communicating with STUN server. These IP addresses sometimes may be changed and have different restrictions depending on NATs type which will be traversed.
3. In some serious occasions No any directly path is available between two endpoints, it need help of relay servers, the relay server create a port to receive and send packets for that client.

Before beginning to establish a connection the clients don't know remote endpoint IP addresses and can't know whether remote endpoint behided NATs or not. But all endpoints are connected to the Control Server, so the server as a controller to start/destroy connecting and transfer endpoints information such as IP addresses among all clients. This SDK don't take care of these control signals which are designed by the App developers.

Main classes and APIs

1. `char* Tr_GetSDKInfo()`
   `Function :  Tr_GetSDKInfo`
   `Parameters :`

Description : Get SDK information
Return : info string

2. int Tr_Init(TrSocketAddress *stun_server, TrSocketAddress *relay_server)
Function : Tr_Init
Parameters :
TrSocketAddress * stun_server - STUN server ip address
TrSocketAddress * relay_server - Relay server ip address
Description : Initialize this SDK, if App dont want to connect to remote stun server or relay server NULL is acceptable
Return : int
T_EROOR - On Failure
T_SUCCESS - On Success
Note : This function must be called before others APIs

3. void Tr_Destroy()
Function : Tr_Destroy
Parameters :
Description : release SDK resources
Return : void

4. int Tr_CreateICESocket(TrICESocket *socket)
Function : Tr_CreateICESocket
Parameters :
TrICESocket *socket: -- ICE session
Description : Create a ICE socket (session)
Return : int
T_EROOR - On Failure
T_SUCCESS - On Success

5. void Tr_DestroyICESocket(TrICESocket *socket)
Function : Tr_DestroyICESocket
Parameters :
TrICESocket *socket: -- ICE session
Description : Destory a ICE socket (session)
Return : void

6. int Tr_StartStunServer(unsigned short port)
Function : Tr_StartStunServer
Parameters :
unsigned short port: -- Listen on this port
Description : Start STUN server on this machine
Return : int
T_EROOR - On Failure

T_SUCCESS - On Success

7. int Tr_StopStunServer()

Function : Tr_StopStunServer

Parameters :

Description : Stop STUN server on this machine

Return : int ;

T_EROOR - On Failure

T_SUCCESS - On Success

8. int Tr_StartRelayServer(unsigned short port)

Function : Tr_StartRelayServer

Parameters :

Description : Start Relay server on this machine

Return : int

T_EROOR - On Failure

T_SUCCESS - On Success

9. int Tr_StopRelayServer()

Function : Tr_StartRelayServer

Parameters :

Description : Stop Relay server on this machine

Return : int ;

T_EROOR - On Failure

T_SUCCESS - On Success

10. class TrICESocket

A) TrICESocket(bool reliable, unsigned char *key, int keylen)

Parameters:

reliable -- if true the TrICESocket will support
restransmiting and reordering mechanism

key -- if keylen > 0 then this point to the
security key

keylen -- key length, if greater than 0 means the
TrICESocket should support encrypt/decrypt
packets

B) virtual void OnReceivePacketData(const char *data, int len) = 0

Parameters:

data -- point to packet buffer header

len -- received packet length

C) virtual void OnSessionState(TrSessionState state) = 0

Parameters:

state -- TrICESocket state event

D) virtual void OnCandidatesReady(TrCandidate *candidates, int count) = 0

When one or more ports are ready for receiving data then the
TrICESocket send this msg to App
Parameters:
candidates -- point to the header of the candidate array
count -- candidate array size

E) void AddRemoteCandidates(TrCandidate *candidates, int count)
When App receive remote endpoint addresses then call
this interface to notify TrICESocket
Parameters:
candidates -- point to the header of the candidate array
count -- candidate array size

F) void SendPacketData(const char *data, int data_len)
App send packets to SDK
Parameters:
data -- packet buffer
data_len -- packet length

11. class TrAsyncSocket
This class represent a asynchronous socket, not necessary but maybe useful for you

A) TrAsyncSocket(int socket_type)
socket_type -- TR_SOCK_STREAM or TR_SOCK_DGRAM

B) TrAsyncSocket(TrAsyncSocketHelper *helper)
For accepting new connection, TrAsyncSocketHelper getted from SDK and must pass back to the SDK.

C) void Bind(const TrSocketAddress* addr)

D) void Bind(unsigned short port)

E) void Listen(int backlog)

F) void Connect(TrSocketAddress* addr)

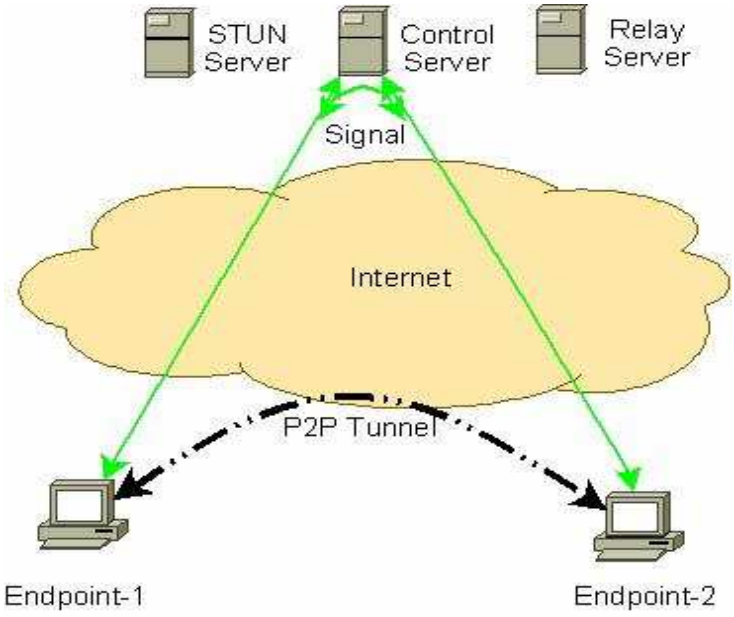
G) virtual void OnRead(char * data, int len)

H) void Write(const char * data, int len)

I) void Close()

J) virtual void OnAccept(TrAsyncSocketHelper *helper, TrSocketAddress remote_addr)
Called under server mode
If receive a connect msg SDK call this function to notify the App
helper -- for SDK, the App must pass this return to SDK
remote_addr -- remote endpoint address

Demo App

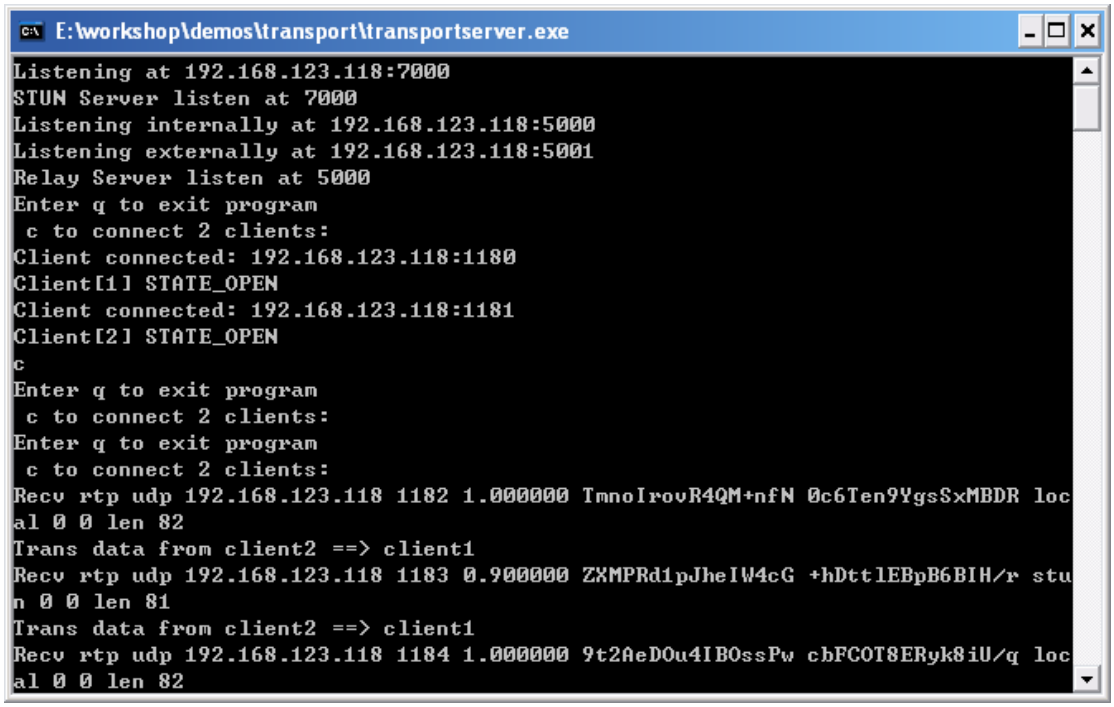


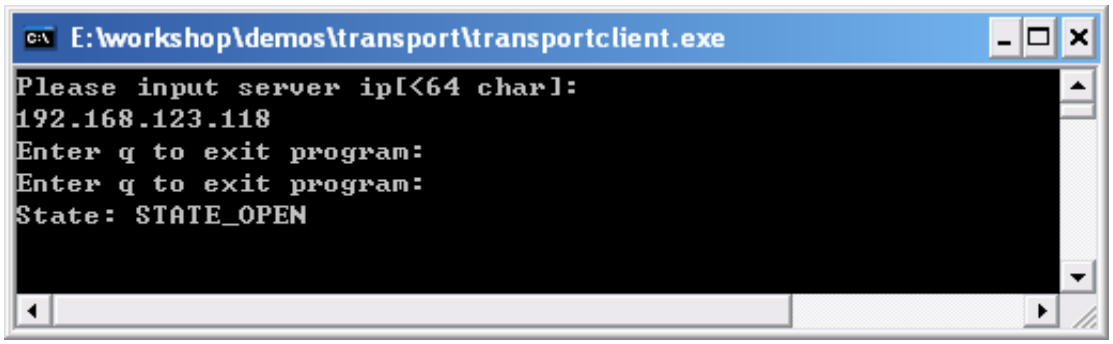
This sample shows how to create a P2P tunnel between two endpoints using this library. It includes a STUN Server, Relay Server, Control Server and two clients. Basic flows as bellowed:

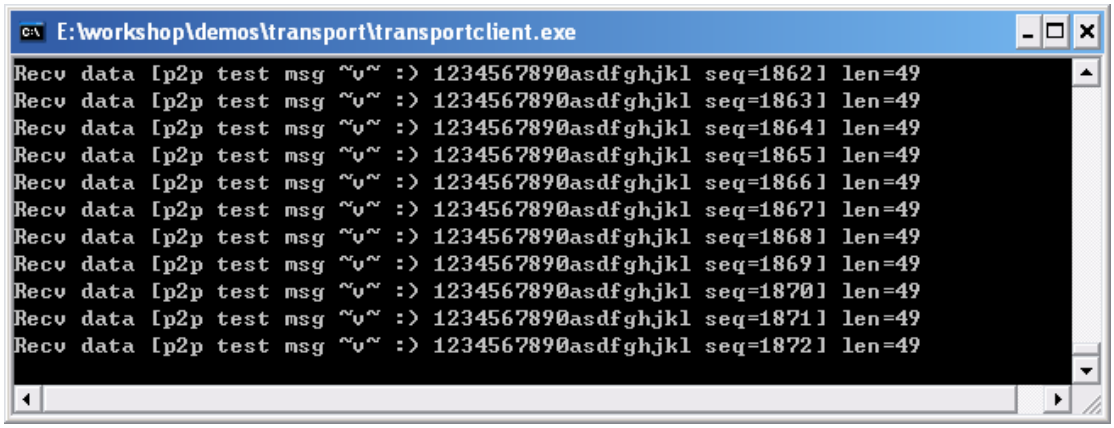
1) Start control server and STUN server and Relay server
2) Run first endpoint and connect to the control server
3) Run second endpoint and connect to the control server
4) After two clients all connect to the control server then the server send START command to clients
5) Clients create TrICESocket and begin to traverse NATs
6) If the TrICESocket create UDP/TCP/Relay ports succeeded then call **OnCandidatesReady** to tell the App
7) The App send candidates(IP addresses) to the control server
8) The control server forward these candidates messages to another client
9) When the client receive candidates messages then call **AddRemoteCandidates** to tell SDK
10) TrICESockets try to connect to remote endpoint and choice the best connection to transmit and receive packets
11) If establish connection fail then will repeat 6)-10)
12) After P2P connection has setup the SDK call **OnSessionState** to tell the App and TrICESockets can receive and send packets

Start demo app:

1) run transportserver.exe
2) run transportclient.exe
3) enter server ip address in client's consol
4) repeat 2-3
5) after two clients all connect to the server, enter 'c' in server's consol







Server App

```
======== Init and start servers ========
/*
 * Before do anything by the SDK
 * Firstly please initialize it
 */
```

```
Tr_Init(NULL, NULL);

/*
 * Start STUN server on local host and listen to port 7000
 */
if(Tr_StartStunServer(7000)==T_ERROR)
     printf("Start STUN Server Failed");
else
     printf("STUN Server listen at 7000\n");

/*
 * Start Relay server on local host and listen to port 5000
 */
if(Tr_StartRelayServer(5000)==T_ERROR)
     printf("Start Relay Server Failed");
else
     printf("Relay Server listen at 5000\n");

/*
 * Start controling server and listen to port 9000
 * Waiting clients to connect to here
 */
svr_socket = new TrAsyncSocketServer();
svr_socket->Bind(9000);
svr_socket->Listen(5);
```

======== Accept new connection ========

```
virtual void OnAccept(TrAsyncSocketHelper *helper, TrSocketAddress remote_addr){

     printf("Client connected: %s:%d\n", remote_addr.host, remote_addr.port);

     /*
      * A client want to connect to the server and SDK report this event to App
      * App must create a new TrAsyncSocket and pass the helper return to SDK
      * the helper object is only useful for the SDK
      */
     if(!client1){
          client1 = new TrAsyncSocketClient(helper);
          client1->cid_ = 1;
     }else if(!client2){
          client2 = new TrAsyncSocketClient(helper);
          client2->cid_ = 2;
     }
}
```

```
======== Send START command ========
    }else if (ch == 'c'){
        char cmd[32] = {0};

        sprintf_s(cmd, "%s", "START");

        if(!client1 || !client2) continue;

        /*
         * Very simple control signal only one msg: START
         * Tell two clients to begin connecting to each other
         */
        client1->Write(cmd, (int)strlen(cmd)+1);
        client2->Write(cmd, (int)strlen(cmd)+1);
    }

======== Transfer candidates messages ========
void OnRead(char * data, int len){
    if(len < 0) return;

    printf("Recv %s len %d\n", data, len);

    /*
     * Transfer candidates info between two clients
     */
    if(cid_ == 1 && client2 && client2->connected_){
        printf("Trans data from client1 ==> client2\n");
        client2->Write(data, len);
    }
    else if(cid_ == 2 && client1 && client1->connected_){
        printf("Trans data from client2 ==> client1\n");
        client1->Write(data, len);
    }

    return;

}
```

Client App

```
======== Init and connect to the controlling server ========
/*
 * Input the server IP address
 */
```

```
memset(server_ip, 0, MAX_HOST_NAME_LENGTH);
printf("Please input server ip[<%d char]:\n", MAX_HOST_NAME_LENGTH);
fscanf(stdin, "%s", server_ip);

/*
 * This client want to connect to STUN and Relay servers
 * These servers help clients to traverse NATs
 */
strncpy(stun_addr.host, server_ip, MAX_HOST_NAME_LENGTH);
stun_addr.port = 7000;

strncpy(relay_addr.host, server_ip, MAX_HOST_NAME_LENGTH);
relay_addr.port = 5000;

/*
 * Firstly initialize the SDK
 */
Tr_Init(&stun_addr, &relay_addr);

/*
 * Create a new client socket
 */
g_client_socket = new TrAsyncSocketClient(TR_SOCK_STREAM);

/*
 * The client connect to the server
 */
strncpy(svr_addr.host, server_ip, MAX_HOST_NAME_LENGTH);
svr_addr.port = 9000;
g_client_socket->Connect(&svr_addr);
```

======== Received START command and create ICE session ========

```
/*
 * In this simple demo, this string must be the control command
 * sended by the server, so the client must quickly create a ICE
 * socket for Peer to Peer connecting
 */
if(len==(strlen("START")+1) && !strcmp(data, "START")){

    printf("Creating ICESocket...\n");
    g_ice_socket = new ICESocket();
    Tr_CreateICESocket(g_ice_socket);

    return;
```

```
    }
```

======== Local candidates are ready and send to the server ========

```
/*
 * This ICE socket has create a port(UDP/TCP/STUN/RELAY) and is ready
 * for remote connecting
 * SDK report this msg to the App, App must through control signal channel
 * send it to remote endpoint
 */
void
ICESocket::OnCandidatesReady(TrCandidate *candidates, int count)
{
    char pktbuf[1024];
    int  i;

    printf("=====Candidates Ready====\n");

    for(i=0; i<count; i++){
        TrCandidate *c = (TrCandidate *)candidates+i;

        memset(pktbuf, 0, sizeof(pktbuf));

        sprintf(pktbuf, "%s %s %s %d %f %s %s %s %s %d",
            c->name,
            c->protocol,
            c->address.host,
            c->address.port,
            c->preference,
            c->username,
            c->password,
            c->type,
            c->network_name,
            c->generation);
        printf("candidate[%d]:%s\n", i, pktbuf);

        /*
         * HERE!! Send candidate info to the server
         */
        g_client_socket->Write(pktbuf, strlen(pktbuf)+1);
    }

    return;
}
```

```
======== Receive remote candidates information and pass into SDK ========
/*
 * If the msg is not start command it must be a candidate info
 * forward by the server from the remote endpoint
 */
sscanf(data, "%s%s%s%s%s%s%s%s%s%s",
          name,
          protocol,
          address.host,
          str_port,
          str_preference,
          username,
          password,
          type,
          network_name,
          str_generation);

c.name = name;
c.protocol = protocol;
strcpy(c.address.host, address.host);
c.address.port = atoi(str_port);
c.preference = atof(str_preference);
c.username = username;
c.password = password;
c.type = type;
c.network_name = network_name;
c.generation = atoi(str_generation);

/*
 * Tell the SDK and the ICE socket know where
 * it will connect to
 */
g_ice_socket->AddRemoteCandidates(&c, 1);

======== Create P2P connection successful ========
case TR_STATE_CONNECTED:
     printf("State:TR_STATE_CONNECTED\n");
     /*
      * ICE socket connect to remote endpoint succefull
      * this means two clients behinded NATs have connected by UDP or STUN UDP or ...
      */
     hHandle = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ProcMsgThreadEntry,
(LPVOID)NULL, 0, &dwThreadId);
     CloseHandle(hHandle);
```

```
            break;

    ======== Send test packets between two clients ========
/*
 * After the connection established between two clients
 * send test packets through ICE session each other
 */
unsigned int ProcMsgThreadEntry(LPVOID lpParam)
{
    char pktbuf[1024];
    int  seq = 1;

    while(!exit_flag)
    {
        sprintf(pktbuf, "p2p test msg ~v~ :) 1234567890asdfghjkl seq=%d", seq++);
        g_ice_socket->SendPacketData(pktbuf, strlen(pktbuf)+1);

        Sleep(30);
    }

    return 0;
}
```