

Document

XSTUNT Reference 1.0

Author

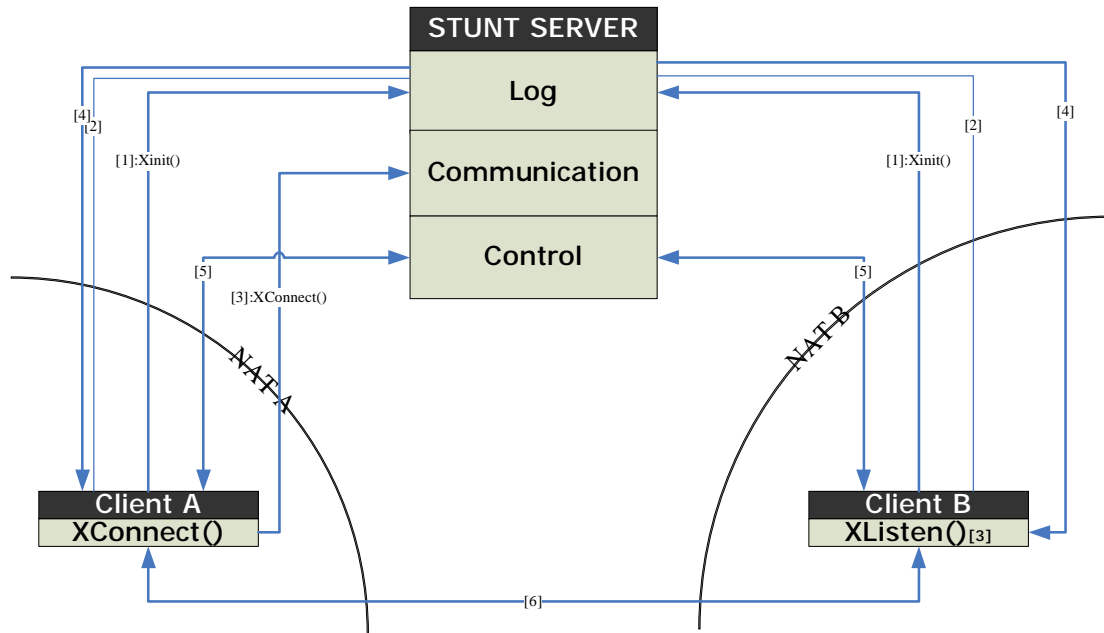
Kuanyu Chou <cky@cis.nctu.edu.tw>

Last Modified Date

2006/4/19

Document	1
Author	1
Last Modified Date	1
XSTUNT Architecture and Procedure	2
XSTUNT LIBRARY	3
XSTUNT External Specification	3
XInit	4
XListen	5
XConnect	6
XDeRegister	7
XSTUNT Internal Specification	8
XAsymServer	12
XAsymClient	12
XPredict	13
XTryConnect	14
XProbe	14
XProbeNAT	15
XCheckConeTCP	15
XCheckConeTCPProbe	16
XAnalyzeNature	17
XGenFingerprint	17
XWriteFingerprint	18
XReadFingerprint	18
XInitSockAddr	19
XSleep	19
XGetErrno	19
Appendix	21

XSTUNT Architecture and Procedure



Fig[1]: XSTUNT Architecture and Procedure

The goal of XSTUNT is to provide a solution to accomplish TCP NAT traversal. The XSTUNT architecture has three roles as above figure: one stunt server and two clients. The STUNT server is in the public internet but the two clients are behind NATs. The STUNT server opens 3 ports for listening: Log, communication, and control. Log is responsible for client registration, connection notifications, and error log. This is a long term connection from registering to deregistering. Communication is responsible for connection requests, deregistering, and query function of management. Control channel is a shunt mechanism when two clients begin to exchange their information.

Here we illustrate the XSTUNT procedure by the following example: Client A wants to make a direct TCP connection to Client B-

- [1]. Client A and Client B must register their IDs, probe NAT type, get the server data on the Log port of STUNT SERVER by XInit().
- [2]. The connections between Clients and STUNT SERVER are established.
- [3]. Client A send a connection request to port Communication of STUNT SERVER by XConnect() and Client B is in the listen state by XListen().
- [4]. STUNT SERVER accepts the request, validates it, and informs the 2 clients by the connection established in [2].
- [5]. Client A and Client B begin to exchange their information through the control channel of STUNT SERVER.
- [6]. Client A finally makes a direct TCP connection through NATs to B.

Following library is an implementation of the above architecture. The method to do the TCP NAT Traversal can be referred to "Characterization and Measurement of TCP Traversal through NAT and Firewalls" addressed by Saikat and Paul. The library implements "STUNT #2" approach in above paper. This program should cooperate with a particular STUNT server for function.

About STUNT, please read: <http://nutss.gforge.cis.cornell.edu/stunt.php> for more information.

XSTUNT LIBRARY

This library provides programmers with a set of simple functions to create STUNT connections through cooperation with a XSTUNT server. Those functions implement the architecture and procedure mentioned above for the client side. Following table lists its abilities in different conditions: Source is the client side who attempts to create a connection to the Destination. Public IP means the client is in the public internet and Private IP means the client is behind NATs.

Source \ Destination	Public IP	Private IP
Public IP	X (Use plain TCP connection)	V
Private IP	X (Use plain TCP connection)	V (Different NAT)/ X(Same NAT)

Obviously, if the destination is in the public internet, we do not use this library to create a TCP connection because a plain TCP socket connection should definitely work well. This library should be used when the destination is behind NAT. A hairpin problem will happen if two clients are behind the same NAT. In this situation, this API returns a local IP address to the Source and then it can create a direct TCP connection in LAN. Following topics will guide you to understand how to use this library.

XSTUNT External Specification

XSTUNT Type Convention

Basic Type	XSTUNT	Size
char	CHAR	1 byte
unsigned char	UCHAR	1 byte
signed char	INT8	1 byte
signed short	INT16	2 bytes
unsigned short	UINT16	2 bytes
signed int	INT32	4 bytes
unsigned int	UINT32	4 bytes

XSTUNT Structures

N/A

XSTUNT Constants

E_ErrorStat

E_ErrorStat defines all errors in XSTUNT library. The meaning of every error will be mentioned in the XSTUNT Functions.

Error Code	Value	Description
ERR_NONE	0	Successful
ERR_FAIL	-1	Fail
ERR_CREATE_SOCKET	-4000	Fail to create socket
ERR_CONNECT	-3999	Fail to connect
ERR_VERSION	-3998	Versions of server and client are mismatched
ERR_DUPLICATE_ID	-3997	The ID is already registered on the server
ERR_PROBE	-3996	Failed during probing
ERR_TIMEOUT	-3995	Timeout
ERR_COMM_TIMEOUT	-3994	Timeout during reading response from communication service
ERR_ECHO_TIMEOUT	-3993	Timeout during reading echo
ERR_SELECT	-3992	Select error
ERR_RECEIVE	-3991	Fail to receive
ERR_SYN_RECEIVE	-3990	Failed during receiving proprietary SYN
ERR_SYN_SEND	-3989	Failed during sending proprietary SYN
ERR_ADDRPORT_RECEIVE	-3988	Failed during receiving address and port
ERR_ADDRPORT_SEND	-3987	Failed during sending address and port
ERR_ASYMSERVER	-3986	Failed in XAsymServer()
ERR_ASYMCLIENT	-3985	Failed in XAsymClient()
ERR_PREDICT	-3984	Failed during port predicting
ERR_SEND	-3983	Failed to send
ERR_TRYCONNECT	-3982	Failed in XTryConnect()
ERR_SETSOCKOPT	-3981	Failed to set socket option
ERR_BIND	-3980	Failed to bind
ERR_LISTEN	-3979	Failed to listen

ERR_ASYM_TIMEOUT	-3978	Timeout during waiting the connection from AsymServer
ERR_ACCEPT	-3977	Failed to accept
ERR_MATCH	-3976	Failed to find the destination peer on the server
ERR_SAME_NAT	-3975	Peer client is in the same NAT
ERR_HAIRPIN	-3974	Hairpin problem

XSTUNT Variables

N/A

XSTUNT Functions

XInit

Purpose

Given two IPs of the STUNT server and a client ID, the function will probe the NAT type, register the specified ID and then return a log socket of STUNT server if the process works successfully.

Declared In

ClientAPI.h

Prototype

```
INT32 XInit(const CHAR* pchIP1, const CHAR* pchIP2, SOCKET* psServerLog, CHAR* pchID, INT32 *pnErrCode);
```

Parameters

-> pchIP1 The 1st IP of the STUNT server. Pass NULL to use the default STUNT server.

-> pchIP2 The 2nd IP of the STUNT server. Pass NULL to use the default STUNT server.

<-> psServerLog The STUNT server socket

-> pchID The client ID which will be registered to the STUNT server. The length of the ID must shorter than or equal to the value of MAX_ID_LEN and not an empty string.

<-> pnErrCode Error code.

Result

ERR_NONE Successful.

ERR_CREATE_SOCKET Fail to create a socket.

ERR_CONNECT Fail to connect to the STUNT server.

ERR_RECEIVE Fail to send data.

ERR_SEND Fail to receive data.

ERR_VERSION The required client version mismatch.

ERR_PROBE Fail during probing the NAT type.

ERR_DUPLICATE_ID The specified ID is already registered in the STUNT server.

Comments

Programmers must first use this function to initialize the XSTUNT and get a STUNT server socket, then use the socket in the functions listed below to finish the direct TCP connections. Notice that if an empty ID string is passed in then ERR_DUPLICATE_ID will be returned.

Programmers may need to check the error reason by pnErrCode when the result is ERR_CREATE_SOCKET, ERR_CONNECT, ERR_RECEIVE, or ERR_SEND.

See Also

N/A

Example Code

```
SOCKET sServer = (SOCKET)-1;
int nErr = 0, nRet = 0;

if ((nRet = XInit("59.124.31.21", "59.124.31.19", &sServer, "listener", &nErr)) == ERR_NONE)
{
```

```

    printf("Initialize successfully!\n");
    //do the XSTUNT processes
}
else
{
    printf("Initialization failed. ErrType(%d) ErrCode(%d)\n", nRet, nErr);
}

```

XListen

Purpose

Listen on a specified socket through the help of a STUNT server.

Declared In

ClientAPI.h

Prototype

```

INT32 XListen(SOCKET sServerLog, SOCKET* psListen, struct sockaddr_in *pAddrPeer, INT32
nTimeoutSec, INT32* pnErrCode);

```

Parameters

-> sServerLog	The STUNT server socket. This socket must be gotten from XInit() and be valid.
<-> psListen	User specified listen socket. User should access this socket to read/ write data if the process works successfully.
<-> pAddrPeer	The address information of the connecting peer. Pass NULL if user does not need this information.
-> nTimeoutSec	The timeout is used when another peer is attempting to connect to. It's not the listen waiting time.
<-> pnErrCode	Error code.

Result

ERR_NONE	Successful.
ERR_TIMEOUT	Timeout during waiting the connection request from STUNT server.
ERR_SELECT	SELECT fail during waiting the connection request from STUNT server.
ERR_RECEIVE	Fail during receiving control channel data from STUNT server.
ERR_CREATE_SOCKET	Fail to create a socket.
ERR_CONNECT	Fail to connect to the control channel of STUNT server.
ERR_ECHO_TIMEOUT	Timeout during reading sync-echo
ERR_SYN_RECEIVE	Fail to receive echo
ERR_SYN_SEND	Fail to send echo.
ERR_ASYMSERVER	Fail during being an asymmetric server.

Comments

Use the function to accept STUNT connections from the other peer. If the function returns ERR_NONE, programmers can access this listen socket for reading or writing data and close the socket to disconnect. The function is non-blocking. Therefore programmers should use the method like loops to wait forever. The nTimeoutSec parameter is functioned during another waiting procedure when a connection is attempting to create. This value is strongly recommend to set as the same as it set in XConnect() or it will probably cause a synchronization problem. Notice that when the process is successfully completed, the listen socket is set to non-blocking, so programmer should use select to block it with a specified timeout.

Programmers may need to check the error reason by pnErrCode for all errors listed above.

See Also

[XInit](#), [XConnect](#)

Example Code

```

//sServer is a valid STUNT socket initialized by XInit()...

```

```

while (true)
{
    char chStr[32];

```

```

int nRcv = 0, nErr = 0;
SOCKET sListen = (SOCKET) -1;
fd_set Socks;
if (XListen(sServer, &sListen, NULL, 12, &nErr) == ERR_NONE)
{
    FD_ZERO(&Socks);
    FD_SET(sListen, &Socks);
    //sListen is changed to a non-blocking socket, so we need to block it.
    select(((INT32)sListen) + 1, &Socks, NULL, NULL, NULL);
    nRcv = recv(sListen, chStr, 32, 0); //receiving 32 bytes from the connector side
    if (nRcv > 0)
        printf("Msg>> %s\n", chStr);
    send(sListen, chStr, strlen(chStr), 0);
    //other processes...
}
} //end while

```

XConnect

Purpose

Create a STUNT connection to a specified peer through the help of a STUNT server

Declared In

ClientAPI.h

Prototype

```

INT32 XConnect(SOCKET sServerLog, CHAR *pchPeerID, SOCKET *psConnect, struct sockaddr_in
*pAddrPeer, INT32 nTimeoutSec, INT32* pnErrCode);

```

Parameters

-> sServerLog	The STUNT server socket. This socket must be gotten from XInit() and be valid.
<-> pchPeerID	The ID of the destination peer who will be connected to. The length of the ID must shorter than or equal to the value of MAX_ID_LEN and not an empty string.
<-> psConnect	User specified connection socket.
<-> pAddrPeer	The address information of the connecting peer. Pass NULL if user does not need this information.
-> nTimeoutSec	The timeout is used when the function is attempting to connect to.
<-> pnErrCode	Error code.

Result

ERR_NONE	Successful.
ERR_CREATE_SOCKET	Fail to create the communication/ control channel socket.
ERR_CONNECT	Fail to connect to the communication/ control channel of the STUNT server.
ERR_MATCH	The matching process is failed. (The reason will be shown on the STUNT server.)
ERR_SAME_NAT	The destination peer is behind the same NAT. The local address of the peer will be returned through pnErrCode. Programmers should try the direct connection in LAN by using this address.
ERR_TIMEOUT	Timeout during waiting receiving peer data from STUNT server.
ERR_RECEIVE	Fail to receive data.
ERR_CREATE_SOCKET	Fail to create a socket.
ERR_ECHO_TIMEOUT	Timeout during reading sync-echo.
ERR_SYN_RECEIVE	Fail to receive echo.
ERR_SYN_SEND	Fail to send echo.
ERR_COMM_TIMEOUT	Timeout during reading response from communication service.
ERR_ASYMCLIENT	Fail during being an asymmetric client.

Comments

Use the function to create a STUNT connection to a specified peer. After successful connection, access the connection socket to read or write data and disconnect that by closing this socket. This value is strongly recommend to set as the same as it in XListen() or it will probably cause a synchronization problem. There are some reasons for ERR_MATCH: destination ID not found, ID is identical to the ID

itself or the peer's state is DEAD. ERR_SAME_NAT means the destination peer and the source peer are behind the same most outside NAT (with the same public IP address). Programmers can try to create a direct connection in the local LAN and it may be successful if they are behind the same most inside NAT. Notice that if a connection is established successfully, it is recommended that any size of data should be sent in 60 seconds or the connection will be closed in some kinds of occasions. Programmers may need to check the error reason by pnErrCode for all errors listed above except ERR_MATCH and ERR_SAME_NAT.

See Also

[XInit](#), [XListen](#)

Example Code

```
//sServer is a valid STUNT socket initialized by XInit()...
//1234 is a listening port number of the destination client.

char chStr[32] = "Data";
int nRet = 0, nErr = 0;
SOCKET sConnect = (SOCKET) -1;
struct sockaddr_in AddrPeer;
nRet = XConnect(sServer, (char*)"connector", &sConnect, NULL, 12, &nErr);
if (nRet == ERR_NONE)
{
    //send or receive
    send(sConnect, chStr, 32, 0);
    recv(sConnect, chStr, 32, 0);
    //other processes...
}
else if (nRet == ERR_SAME_NAT) //Behind the same NAT
{
    AddrPeer.sin_family = AF_INET;
    AddrPeer.sin_addr.s_addr = nErr; //The error code contains the private IP. Use it!
    AddrPeer.sin_port = htons(1234);

    //Try to make a plain TCP connection in LAN
    sConnect = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    nRet = connect(sConnect, (struct sockaddr *)&AddrPeer, sizeof(AddrPeer));
    if (nRet == 0)
    {
        send(sConnect, chStr, 32, 0);
        recv(sConnect, chStr, 32, 0);
        //other processes...
    }
    else //fail to connect in LAN
        printf("Fail to connect\n");
}
else //fail to connect by XSTUNT
    printf("Fail to connect. ErrType(%d) ErrCode(%d)\n", nRet, nErr);
```

XDeRegister

Purpose

Deregister the client on and close the socket of the connection to the STUNT server

Declared In

ClientAPI.h

Prototype

```
INT32 XDeRegister(SOCKET sServerLog, CHAR* pchID, INT32* pnErrCode);
```

Parameters

-> sServerLog	The STUNT server socket. This socket must be gotten from XInit() and be valid.
-> pchID	The client ID which will be deregistered from the STUNT server.
<-> pnErrCode	Error code.

Result

ERR_NONE	Successful.
ERR_CREATE_SOCKET	Fail to create a socket to the communication port.
ERR_CONNECT	Fail to connect to the communication port of STUNT server.

Comments

Use the function to deregister the client on the STUNT server after finish the whole STUNT procedure. Programmers may need to check the error reason by pnErrCode for all errors listed above.

See Also

[XInit](#)

XSTUNT Internal Specification

XSTUNT Type Convention

Basic Type	XSTUNT	Size
char	CHAR	1 byte
unsigned char	UCHAR	1 byte
signed char	INT8	1 byte
signed short	INT16	2 bytes
unsigned short	UINT16	2 bytes
signed int	INT32	4 bytes
unsigned int	UINT32	4 bytes

XSTUNT Structures

ServerInfo

```
typedef struct _ServerInfo
{
    CHAR        chID[MAX_ID_LEN + 1];
    INT32       nIP1;
    INT32       nIP2;
    INT16       wPort1;
    INT16       wPort2;
    INT16       wPort3;
    INT16       wPort4;
    INT16       wPort5;
    INT16       wPort6;
    INT16       wPort7;
    INT16       wPort8;
} ServerInfo;
```

Field Descriptions

chID	Client ID
nIP1	The 1 st IP of the STUNT server
nIP2	The 2 nd IP of the STUNT server
wPort1	Echo service port1 number
wPort2	Echo service port2 number
wPort3 ~ wPort8	Reserved

Echo

```
typedef struct _Echo
{
    INT32       nIP;
    INT16       wPort;
    INT16       wPadding;
    union
    {
        CHAR chData[24];
        struct
        {
            INT32 nSeq;
            CHAR chResponse;
        }Tcp;
    } struct
}
```

```

        {
            INT32 nConnID;
            CHAR chPeerID[MAX_ID_LEN + 1];
            INT32 nPeerIP;
            CHAR chRole;
        }Conn;
    }Data
}Echo;

```

Field Descriptions

chID	Client ID
nIP	IP with of the STUNT server
wPort	The port number
wPadding	Reserved
Data:chData	Reserved
Data:Tcp:nSeq:	Reserved
Data:Tcp:chResponse	Reserved
Data:Conn:nConnID	Reserved
Data:Conn:chPeerID	Reserved
Data:Conn:nPeerIP	Reserved
Data:Conn:chRole	Reserved

Msg

```

typedef struct _Msg
{
    INT32 nType;
    union
    {
        struct
        {
            CHAR chSrcID[MAX_ID_LEN + 1];
            CHAR chDstID[MAX_ID_LEN + 1];
        } Connection;
        CHAR chID[MAX_ID_LEN + 1];
    } Data;
} Msg;

```

Field Descriptions

nType	Message type: Defined by E_Message.
Data.Connection.chSrcID	Source ID used in nType = MSG_CONNECT
Data.Connection.chDstID	Destination ID used in nType = MSG_CONNECT
Data.chID	Client ID used in MSG_DEREGISTER and MSG_QUERY.

Fingerprint

```

typedef struct _Fingerprint
{
    INT32 nClientVer;
    INT32 nServerVer;
    CHAR chID[MAX_ID_LEN + 1];
    INT32 nDone;
    INT32 nGAddr;
    struct
    {
        INT32 nIncrement;
        INT8 bPortPreserving;
    } Connection;
} Fingerprint;

```

Field Descriptions

nClientVer	Client build version
nServerVer	Server version
chID	Client ID
nDone	Indicate if the fingerprint is done or not
nGAddr	Global address of the client

Connection.nIncrement Port incremental value of NAT for next connection
 Connection.bPortPreserving Port preservation characteristic of the NAT

XSTUNT Constants

Constant	Value	Description
BUILD	"BUILD \$VER:10\$"	Build version string of this library.
BUILDVER	10	This build version must match the server's or will cause version mismatch error during initialization.
TEST_TIME	N/A	Debug constant for watching time consuming for some procedures.
TEST_DEBUG	0	Debug message switcher. See ClientMacro.h
DEF_SERVER_IP1	"59.124.31.21"	The first default IP of STUNT server
DEF_SERVER_IP2	"59.124.31.19"	The second default IP of STUNT server
SERVER_LOG_PORT	8123	Log service port number of STUNT server
SERVER_COMM_PORT	8132	Communication service port number of STUNT server
MAX_IP_STR_LEN	20	Maximal IP string length
MAX_ID_LEN	32	Maximal Client ID length
MAX_FINGERPRINT_LEN	1024	Maximal fingerprint length
COMM_TIMEOUT	2	Suggested timeout of receiving communication message.
RCV_PEER_DATA_TIMEOUT	2	Suggested timeout of receiving peer information from control channel service.
RCV_ECHO_TIMEOUT	7	Suggested timeout of synchronization between peers by STUNT server
ASYM_TIMEOUT	3	Suggested timeout of directly connecting to the peer
FINGERPRINT_FILE	"fingerprint.dat"	Fingerprint file name on local side.
PROBE_RETRY_TIMES	3	NAT probing maximal retry times
PROBE_FAIL_SLEEP_SEC	10	Sleeping time in second if failed in probing TCP cone type.
RANDOM_INCREMENT	0x10000000	Random increment of symmetric NAT

E_TagType

E_TagType defines all tags used in synchronization with help by STUNT server.

```
typedef enum
{
    TAG_OK = 1,
    TAG_ABORT,
    TAG_ACKABORT,
}E_TagType;
```

E_StateType

E_ErrorStat defines all possible states of a client.

```
typedef enum
{
    CSTATE_PROBE = 0,
    CSTATE_IDLE,
    CSTATE_BUSY,
    CSTATE_DEAD,
    CSTATE_TYPE_AMT
}E_StateType;
```

E_Protocol

E_ErrorStat defines all protocol types used in this library. UDP is not available.

```
typedef enum
{
    PRO_UDP = 0,
    PRO_TCP,
```

```

    PRO_AMT
} E_Protocol;

```

E_ErrorStat

E_ErrorStat defines all message types for communication service of the STUNT server.

```

typedef enum
{
    MSG_CONNECT = 10001,
    MSG_DEREGISTER,
    MSG_QUERY,
    MSG_AMT
} E_Message;

```

E_ErrorStat

E_ErrorStat defines all errors in XSTUNT library. The meaning of every error will be mentioned in the XSTUNT Functions.

Error Code	Value	Description
ERR_NONE	0	Successful
ERR_FAIL	-1	Fail
ERR_CREATE_SOCKET	-4000	Fail to create socket
ERR_CONNECT	-3999	Fail to connect
ERR_VERSION	-3998	Versions of server and client are mismatched
ERR_DUPLICATE_ID	-3997	The ID is already registered on the server
ERR_PROBE	-3996	Failed during probing
ERR_TIMEOUT	-3995	Timeout
ERR_COMM_TIMEOUT	-3994	Timeout during reading response from communication service
ERR_ECHO_TIMEOUT	-3993	Timeout during reading echo
ERR_SELECT	-3992	Select error
ERR_RECEIVE	-3991	Fail to receive
ERR_SYN_RECEIVE	-3990	Failed during receiving proprietary SYN
ERR_SYN_SEND	-3989	Failed during sending proprietary SYN
ERR_ADDRPORT_RECEIVE	-3988	Failed during receiving address and port
ERR_ADDRPORT_SEND	-3987	Failed during sending address and port
ERR_ASYMSERVER	-3986	Failed in XAsymServer()
ERR_ASYMCLIENT	-3985	Failed in XAsymClient()
ERR_PREDICT	-3984	Failed during port predicting
ERR_SEND	-3983	Failed to send
ERR_TRYCONNECT	-3982	Failed in XTryConnect()
ERR_SETSOCKOPT	-3981	Failed to set socket option
ERR_BIND	-3980	Failed to bind
ERR_LISTEN	-3979	Failed to listen
ERR_ASYM_TIMEOUT	-3978	Timeout during waiting the connection from AsymServer
ERR_ACCEPT	-3977	Failed to accept
ERR_MATCH	-3976	Failed to find the destination peer on the server
ERR_SAME_NAT	-3975	Peer client is in the same NAT
ERR_HAIRPIN	-3974	Hairpin problem

XSTUNT Variables

XSTUNT global variables

Type	Name	Description
FILE*	g_pDbgFile	Destination debugging message output file name
CHAR	g_chClientID[MAX_ID_LEN + 1]	Client ID
INT32	g_nClientIP	Client IP
INT32	g_nServerVersion	Server version
CHAR	g_szServerIP1[MAX_IP_STR_LEN + 1]	The first IP of the STUNT server
CHAR	g_szServerIP2[MAX_IP_STR_LEN + 1]	The second IP of the STUNT server
Fingerprint	g_Fingerprint	Fingerprint
ServerInfo	g_ServerInfo	A buffer for exchanging server and client information.

XSTUNT Functions

XAsymServer

Purpose

This function predicts the global port, exchanges the IP/PORT information through STUNT server, and finally connects to the Asymmetric Client.

Declared In

Client.h

Prototype

```
INT32 XAsymServer(SOCKET sServerLog, SOCKET sCtrl, SOCKET *psListen, struct sockaddr_in *pAddrPeer, INT32 nTimeoutSec, INT32* pnErrCode);
```

Parameters

-> sServerLog	The STUNT server socket. This socket must be gotten from XInit() and be valid.
-> sCtrl	A valid Control service socket of the STUNT server.
<-> psListen	User specified listen socket. User should access this socket to read/ write data if the process works successfully.
<-> pAddrPeer	The address information of the connecting peer. Pass NULL if user does not need this information.
-> nTimeoutSec	The timeout will be set in all of the waiting procedures in this function.
<-> pnErrCode	Error code.

Result

ERR_NONE	Successful.
ERR_PREDICT	Fail during port prediction. Check in XPredict().
ERR_SEND	Fail to send public IP/PORT of the client.
ERR_TIMEOUT	Timeout during waiting reading data from STUNT server.
ERR_SELECT	SELECT fail during connecting to another peer.
ERR_RECEIVE	Fail during receiving peer information or sync data from STUNT server.
ERR_CONNECT	Fail to connect to the destination peer.
ERR_HAIRPIN	The public IP addresses of source and destination are the same.

Comments

This is an important function to start a simultaneous TCP connection with the other peer. Therefore, the function and XAsmClient() must be launched in the meantime on two different machines behind different NATs. Notice that *psListen will be set to non-blocking I/O in this function to achieve a connection timeout and never set back to the original value throughout. Timeout in this function should be set the same as that set in XAsmServer() to make synchronization well. An experimental suggested value is defined by ASYM_TIMEOUT. Check the error reason by pnErrCode for all errors listed above except ERR_PREDICT and ERR_HAIRPIN.

See Also

[XListen](#), [XAsymClient](#), [XPredict](#)

XAsymClient

Purpose

This function predicts the global port, exchanges the IP/PORT information through STUNT server, tries to connect to the destination peer, theoretically gets a failure, listens to the Asymmetric Server and finally establishes the connection.

Declared In

Client.h

Prototype

```
INT32 XAsymClient(SOCKET sServerLog, SOCKET sCtrl, SOCKET *psConnect, struct sockaddr_in *pAddrPeer, INT32 nTimeoutSec, INT32* pnErrCode);
```

Parameters

-> sServerLog	The STUNT server socket. This socket must be gotten from XInit() and be valid.
-> sCtrl	A valid Control service socket of the STUNT server.
<-> psConnect	User specified connection socket.
<-> pAddrPeer	The address information of the connecting peer. Pass NULL if user does not need this information.
-> nTimeoutSec	The timeout will be set in all of the waiting procedures in this function.
<-> pnErrCode	Error code.

Result

ERR_NONE	Successful.
ERR_PREDICT	Fail during port prediction. Check in XPredict().
ERR_SEND	Fail to send public IP/PORT of the client.
ERR_TIMEOUT	Timeout during waiting reading data from STUNT server.
ERR_SELECT	SELECT fail during connecting to another peer.
ERR_RECEIVE	Fail during receiving peer information or sync data from STUNT server.
ERR_CONNECT	Fail to connect to the destination peer.
ERR_HAIRPIN	The public IP addresses of source and destination are the same.
ERR_TRYCONNECT	XTryConnect() returned ERR_NONE, but it represents that the function should return back failure.
ERR_CREATE_SOCKET	Fail to create a new socket.
ERR_SETSOCKOPT	Fail to set socket option.
ERR_BIND	Fail to bind a new socket.
ERR_LISTEN	Fail to listen.
ERR_ASYM_TIMEOUT	Timeout during waiting the connection from AsymServer.
ERR_ACCEPT	Fail to accept.

Comments

This is an important function to start a simultaneous TCP connection with the other peer. Therefore, the function and XAsmServer() must be launched in the meantime on two different machines behind different NATs. Timeout in this function should be set the same as that set in XAsmClient() to make synchronization well. An experimental suggested value is defined by ASYM_TIMEOUT.

Check the error reason by pnErrCode for all errors listed above except ERR_ASYM_PREDICT, ERR_HAIRPIN, and ERR_ASYM_TIMEOUT.

See Also

[XConnect](#), [XAsymClient](#), [XPredict](#), [XTryConnect](#)

XPredict

Purpose

This function predicts the global port that will be used in the next connection and return the socket with that global port number on NAT.

Declared In

Client.h

Prototype

```
INT32 XPredict(SOCKET sServerLog, SOCKET *psAux, struct sockaddr_in *pAddrGlobal, struct sockaddr_in *pAddrLocal);
```

Parameters

-> sServerLog	The STUNT server socket. This socket must be gotten from XInit() and be valid.
<-> psAux	A socket descriptor. It will be set to a valid socket bound on the local address: pAddrLocal and with a predicted port: pAddrGlobal on NAT.
<-> pAddrGlobal	The function will write the address with global predicted port for next connection.
<-> pAddrLocal	The function will randomly assign an address to this variable and bind it on psAux.

Result

ERR_NONE Successful.
ERR_FAIL Failed in this function.

Comments

The function will bind a random address on psAux twice to determine the predicted global port using XCheckConeTCPProbe(). Notice that the increment data of fingerprint should also be ready for the prediction.

See Also

[XCheckConeTCPProbe](#), [XAsymServer](#), [XAsymClient](#), [XAnalyzeNature](#)

XTryConnect

Purpose

This function tries to make a connection to the destination peer.

Declared In

Client.h

Prototype

```
INT32 XTryConnect(SOCKET sServerLog, SOCKET sAuxServer, struct sockaddr *pAddrPeer, INT32 nAddrPeerLen, INT32 nMiliSec);
```

Parameters

-> sServerLog The STUNT server socket. This socket must be gotten from XInit() and be valid.
-> sAuxServer The socket for creating this connection.
-> pAddrPeer Destination peer address.
-> pAddPeerLen Length of pAddrPeer
-> pMiliSec Timeout of this non-blocking connection

Result

ERR_NONE Successful.
ERR_FAIL Failed in this function.

Comments

Notice that the normal result should be ERR_FAIL but not ERR_NONE. The goal of this function is to punch a hole on the NAT. pMiliSec should be large enough for the TCP-SYN to leave the host and hit the NAT(s) it is behind. In this library, the value is 1000ms but even 500ms or 100ms would work just fine.

This function try to create a connection to the destination peer and fails immediately and we observe that some recipient NATs will send the RST, but most NATs will silently discard the SYN and not bother to send a RST. That's what we observed in our measurement study. Even if the recipient's NAT does sent a RST (a small percent of NATs), it turns out that the RST won't affect anything since not all senders' NATs will close the hole.

More on the SYN-RST numbers here:

<https://www.guha.cc/saikat/stunt-results.php?>

In the column: 'Incoming SYN Unsolicited', only 21 out of 141 models tested respond with RST packets. Re-normalized with each brand's market share, that is less than 10%. In the column 'Incoming SYN after RST', 66 out of 141 NATs will accept the inbound SYN after the RST (i.e. do not close the hole). Renormalized, that number is around 75%.

So overall, only 2.5% of the time will there be a RST packet that will close the hole. That is our estimate.

See Also

[XAsymClient](#)

XProbe

Purpose

This function will fetch the OS type, probe the NAT type of the client, generate/write the fingerprint, and send the result to the STUNT server.

Declared In

Client.h

Prototype

```
INT32 XProbe(SOCKET sServerLog);
```

Parameters

-> sServerLog The STUNT server socket. This socket must be gotten from XInit() and be valid.

Result

ERR_NONE Successful.
ERR_FAIL Failed.

Comments

The function reports the client's information to the STUNT server logger with ID, OS type, build version, and NAT fingerprint. Possible OS type strings are "WIN32" and "NON_WIN32". If OS type is WIN32, the OS version (%PlatformID%. %MajorVersion%. %MinorVersion%. %ServicePackMajor%. %ServicePackMinoris%) is also returned. Build version is the build string defined by constant BUILD. For the fingerprint, please refer to XGenFingerprint for more reference.

See Also

[XProbeNAT](#), [XGenFingerprint](#), [XWriteFingerprint](#)

XProbeNAT

Purpose

This is a wrap of XCheckConeTCP(). It just controls the retry times and sleeping time.

Declared In

Client.h

Prototype

```
INT32 XProbeNAT(SOCKET sServerLog);
```

Parameters

-> sServerLog The STUNT server socket. This socket must be gotten from XInit() and be valid.

Result

ERR_NONE Successful.
ERR_FAIL Failed after all retries.

Comments

Adjust the retry times and sleeping time by modifying PROBE_RETRY_TIMES and PROBE_FAIL_SLEEP_SEC.

See Also

[XCheckConeTCP](#)

XCheckConeTCP

Purpose

This function checks NAT's TCP port mapping characteristics. It's a wrap of XCheckConeTCPProbe() and XAnalyzeNature().

Declared In

Client.h

Prototype

```
INT32 XCheckConeTCP(SOCKET sServerLog);
```

Parameters

-> sServerLog The STUNT server socket. This socket must be gotten from XInit() and be valid.

Result

ERR_NONE Successful.
ERR_FAIL Failed.

Comments

The function binds 1 port for XCheckConeTCPProbe() to detect characteristics of the NAT and then passes the result to XAnalyzeNature().

See Also

[XCheckConeTCPProbe](#), [XAnalyzeNature](#)

XCheckConeTCPProbe

Purpose

This function try to connect to the echo service with different IP-PORT combinations and then get the public IP and port of the client.

Declared In

Client.h

Prototype

```
void XCheckConeTCPProbe(SOCKET sServerLog, SOCKET sEcho, INT32 nSeq, UINT32 *punResAddr, UINT16 *puwResPort);
```

Parameters

-> sServerLog The STUNT server socket. This socket must be gotten from XInit() and be valid.
-> sEcho A echo socket created in XCheckConeTCP(). It will be used to connect to the echo service of STUNT server.
-> nSeq Test sequence number (0 ~ 4).
<-> *punResAddr The public address of the client returned from the STUNT server.
<-> *puwResPort The public port of the client returned from the STUNT server.

Result

N/A

Comments

The function will be triggered 4 times with sequence number 0 to 4 to determine the IP:PORT combination of the echo service provided by the STUNT server. 4 combinations are IP1:PORT1, IP1:PORT2, IP2:PORT1, IP2:PORT2. The echo service will return the public IP:PORT of the client every time.

See Also

[XCheckConeTCPProbe](#), [XAnalyzeNature](#)

XAnalyzeNature

Purpose

This function analyzes the passed in IP:PORT data and write the result on fingerprint.

Declared In

Client.h

Prototype

```
void xAnalyzeNature(UINT32 *punAddrGlobal, UINT16 *puwPortGlobal, UINT16 *puwPortLocal,
INT32 nTimes, E_Protocol nProtocol);
```

Parameters

->	*punAddrGlobal	The pointer which points to a global (public) IP array.
->	*puwPortGlobal	The pointer which points to a global (public) Port array
->	*puwPortLocal	The pointer which points to a local (private) Port array
<->	nTimes	Test times. The value is also the size of the above arrays
<->	nProtocol	Always PRO_TCP in this library.

Result

N/A

Comments

The function is called by XCheckConeTCP(). XCheckConeTCP will pass in the testing result got from XCheckConeTCPProbe(): global IP:PORT and local PORT array. The function will determine the NAT characteristics such as Port-Preserving and Cone/Symmetric and write it on the fingerprint. Notice that only TCP analyzing is used in this library.

See Also

[XCheckConeTCPProbe](#), [XCheckConeTCP](#)

XGenFingerprint

Purpose

This function interprets the binary fingerprint data to a readable string.

Declared In

Client.h

Prototype

```
void XGenFingerprint(CHAR *pchPrint, INT32 nSize);
```

Parameters

<->	*pchPrint	A buffer for storing the readable fingerprint string.
->	nSize	Size of the above string

Result

N/A

Comments

The function will print the following TCP NAT characteristic string:
[Does the NAT preserve the port number on the local machine?]
PORT-PRESERVING: Yes, the NAT will preserve the port number as the same as local port.
NOT-PORT-PRESERVING: No, the NAT won't preserve the port number.
[Does the NAT change the mapping port when the local machine tries to create another connection?]
CONE: The NAT will use a port which is opened before for this connection.

RANDOM-SYMMETRIC: The NAT will randomly open a port even if the machine is already opening a port on NAT.
DELTA-N-SYMMETRIC: The NAT will open a port of number with a number-N increment to the previous connection.

See Also

[XProbe](#)

XWriteFingerprint

Purpose

This function writes the NAT fingerprint to specified file on the local machine.

Declared In

Client.h

Prototype

```
INT32 XWriteFingerprint(SOCKET sServerLog);
```

Parameters

-> sServerLog The STUNT server socket. This socket must be gotten from XInit() and be valid.

Result

ERR_NONE Successful.
ERR_FAIL Failed.

Comments

The file name is defined by FINGERPRINT_FILE.

See Also

[xGenFingerprint](#), [XReadFingerprint](#)

XReadFingerprint

Purpose

This function read fingerprint file and write the data on the global fingerprint variable.

Declared In

Client.h

Prototype

```
INT32 XReadFingerprint(void);
```

Parameters

N/A

Result

ERR_NONE Successful.
ERR_FAIL Failed.

Comments

The file name is defined by FINGERPRINT_FILE. Not only I/O error but also build version and client ID will be checked in this function.

See Also

[XGenFingerprint](#), [XReadFingerprint](#)

XInitSockAddr

Purpose

This function read fingerprint file and write the data on the global fingerprint variable.

Declared In

Client.h

Prototype

```
INT32 XInitSockAddr(struct sockaddr_in *pSockAddr, INT16 wFamily, const CHAR * pchAddr,
UINT16 uwPort, UINT32 unConvertedAddr, UINT16 uwConvertedPort);
```

Parameters

<->	pSockAddr	The socket address needed to set.
->	wFamily	Net family.
->	pchAddr	Socket address in string. Pass NULL if pass in converted data
->	uwPort	Socket port
->	unConvertedAddr	Converted address
->	uwConvertedPort	Converted port

Result

ERR_NONE	Successful.
----------	-------------

Comments

A utility to set address data for 2 different forms.

See Also

N/A

XSleep

Purpose

A delay function.

Declared In

Client.h

Prototype

```
void XSleep(INT32 nSec, INT32 nUSec);
```

Parameters

->	nSec	Time value, in second.
->	nUSec	Time value, in microsecond.

Result

N/A

Comments

Implement of sleep in linux.

See Also

N/A

XGetErrno

Purpose

This function gets errno in windows and linux.

Declared In

Client.h

Prototype

```
INT32 XGetErrno(void);
```

Parameters

N/A

Result

For windows, this function returns winsock error code but errno otherwise.

Comments

Use this function to get error code in windows or linux.

See Also

N/A

Appendix

NAT Solutions

UDP

- I Use STUN to detect NAT type (source code is available: <http://sourceforge.net/projects/stun>)
- I Full Cone/ Restricted Cone/ Port Restricted are solved by a NAT probe. (NAT probe sample code is available: <http://www.ppcn.net/n1306c2.aspx>)
- I NAT probe: Client A and Client B are all registered in this server. The server knows the public IP:port of both clients registered in. If client A wants to talk to client B, client A will first send a message to client B but abandoned by NAT B. client A then tells NAT probe to inform client B. Client B will send a trash message to client A and then Client A knows it's possible to talk.
- I In symmetric NAT, we must guess the port NAT assigned.

TCP

- I TCP NAT Traversal is also available:
<http://zgp.org/pipermail/p2p-hackers/2005-September/002979.html>
- I This library (Java) is based on this research paper:
<http://nutss.net/pub/imc05-tcpnat.pdf>
- I The key result of the paper is: TCP NAT Traversal can work 85%-90% of the time today (without any special assumptions on NATs), and 100% of the time between pairs of certain well-behaved NATs.