EWF P2P APPLICATION DESIGN

Introduction

This project aims to build build a direct (TCP/UDP) connection between two applications which are on different hosts. It requires an intermediate host with public IP address to help to build the connection but not to keep the connection. Since hosts may be behind NAT, STUN(RFC5389) is utilized to help to do the NAT traversal. Therefore, this project will be a usage of STUN and several new methods will be added to STUN protocol. As this is just a preliminary design, it may changes when it comes to implementation.

Main procedure

- ### Resister
  1. Client binds the port number, P, to itself.
  2. Client generates a 128-bit random number as its ID.
  3. Client sends **REQUEST** with method **REGISTER** to server.
  4. Server sends **RESPONSE** to client.
  5. If client gets success **RESPONSE**, continue, else, repeat 2-4.

- ### Bind
  1. Client sends **STUN REQUEST** with method **BIND** to server.
  2. Server sends **STUN RESPONSE** to client.
  3. Client sends **REQUEST** with method **BIND** to server.
  4. Server sends **RESPONSE** to client.

- ### Query
  1. Client sends **REQUEST** with method **QUERY** to server.
  2. Server sends **RESPONSE** to client.

- ### Connection
  1. Client initiates connection with peer host.

Protocol defination

STUN BIND method will follow RFC5389. Methods to be defined are **REGISTER**, **BIND** and **QUERY**. To make reference simple, **REGISTER**, **BIND** and **QUERY** are methods of protocol **X**.

Header

All **X** messages MUST start with a 20-byte header. This is to make proccessing simple since **X** protocol is multiplexed with **STUN** protocol on the same port. Format of the **X** message header:

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |1 1|     X Message Type        |         Message Length        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                             All 0s                            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
 |                     Transaction ID (96 bits)                  |
 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The most significant 2 bits of every **X** message MUST be ones. This is used to differentiate **X** packets from **STUN** packets.

The message type field is decomposed further into the following structure:

```
  0                 1
  2  3  4 5 6 7 8 9 0 1 2 3 4 5
 +--+--+-+-+-+-+-+-+-+-+-+-+-+-+
 |M |M |M|M|M|C|M|M|M|C|M|M|M|M|
 |11|10|9|8|7|1|6|5|4|0|3|2|1|0|
 +--+--+-+-+-+-+-+-+-+-+-+-+-+-+
```

This follows the defination in **STUN** protocol with extension to methods:
REGISTER: method = 0b000000000010
BIND: method = 0b000000000011
QUERY: method = 0b000000000100

The message with invalid header will be simply ignored with no response.

Body

Body format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Attribute name         |        Attribute length       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                       Attribute content                       |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Attribute name:
ERROR_REASON: 0b0000000000000000 or 0x0000
ID: 0b0000000000000001 or 0x0001
NETWORK_ADDRESS: 0b0000000000000010 or 0x0002
KEY: 0b0000000000000011 or 0x0003

Attribute length:

Since the attribute content's length MUST be multiple of 4 bytes, the attribute length field will be the atual number of bytes/ 4.

ERROR_REASON:

Always 4 bytes.

Unknown method: 0x00000000

Invalid attributes: 0x00000001

ID conflict: 0x00000001

ID not registered: 0x00000002

ID and KEY not matched: 0x00000003

Unknown reason: 0x11111111(For debugging purpose).

Methods

REGISTER:

Client sends its generated ID to server. Server checks the availability of the ID, and sends the response back.

Request attributes: ID.

Success response attributes: KEY.

Error response attributes: ERROR_REASON.

BIND:

Client sends its ID, KEY and public network address to server. Server validates ID and KEY, and stores the network address to the ID if validation succeeded, and sends back the response.

Request attributes: ID, KEY, NETWORK_ADDRESS.

Success response attributes: Nil.

Error response attributes: ERROR_REASON

QUERY:

Client sends the ID of the host to whch it wants to connect to server. Server check whether the ID has been registered and get the network address of the ID if it is registered and send back the response.

Request attributes: ID.

Success response attributes: NETWORK_ADDRESS.

Error response attributes: ERROR_REASON.

Security

Currently the only security is to prevent attackers from changing the network address of a random ID. This is ensured by an ID-KEY validation.

Further security protection will be added in the future.

Other Issues

Currently it only considers the success senario. Further details to handle failure senario will be added in the future, for example, after user get the network address of target ID, how to handle when it fails to connect to the target host.

More issues will be found during implementation.