

**1. General Information and specification**

The product is an intermediate echo server and client that covers the following features and specifications.

1. An Echo server that echoes back the message sent to the server.
2. Non GUI implementation in C++. This project utilizes C++ 14.
3. Not using 3rd party components such as Boost ASIO. This product can be built from the get go.
4. Works on Windows and can be open with Visual Studio 2017 as well as Visual Studio 2019 (Community Edition).
5. Can be built in Linux-like platform. Utilizes CMake 3.10 for build (default with Ubuntu 18.04)
6. Supports binary based protocol with mandatory header and variable sized body.
7. Server supports multiple client sessions.
8. Server utilizes thread pool.
9. Server supports multiple instances and connects with each other.
10. Server messages are broadcasted to all clients connected and to the clients connected to its peers as well.
11. A very flexible and configurable Client application.
12. Client can connect to any server by providing a server list thru a file as argument or as a command line parameter.
13. Client reconnects to a server when connection is lost.

**2. Build and environment**

For Windows, Using Visual Studio 2017 or 2019 Community Edition;

1. Double click the solution file CrytekCppTest.sln
2. In the Solution Explorer Window, right click the Solution and select ‘Build’ or ‘Rebuild Solution’
3. Build result can be found in Demo/Windows folder in the main directory in preparation for demo run.

For Linux-like platform,

1. Install CMake 3.10 or higher, suggestion is to use Ubuntu 18.04 and install the necessary build-tools.
2. In the main project directory, create the build folder and go inside it

*> mkdir build && cd build*

1. Configure cmake files by running

*> cmake ...*

1. Build the project

> make

1. Binaries are generated in build/Client and build/Server folders

:: Furthermore, IDE such as CLion and VSCode can be used to build the project in Linux, CLion automatically installs the necessary build-tools during install in Linux systems.

**3. Run time configuration**

When built properly, the project generates two files

*./Client.exe* or *./Client* in Linux-like platform

./Server.exe or ./Server in Linux-like platform

Each program provides a customizable options for its run time execution. These options are settable thru its command line.

**Server Command Line Options**

|  |  |
| --- | --- |
| -h,--help | Display Help (this information) |
| --peers <filename> | loads the peers server list from file (optional)  Server can run even without specifying peers. |
| -i,--id <string> | Server identity (REQUIRED) |
| -p,--port <number> | Server port to listens to (required) |
| -thread-count <number> | Max number of threads in the thread pool to. (default 100) |
| --expand <true/false> | Will the server expand and create more thread if it runs out (default false). |

Example: *./Server.exe –id server1 –port 9000 –peers peersconfig.txt*

**Client Command Line Options**

|  |  |
| --- | --- |
| -h,--help | Display Help (this information) |
| --frequency <number> | Number of times this client will send the message, Set to (-1) to continuously sends forever, (default is 1). |
| --reconnect-retry <number> | Number of times to retry reconnect when disconnected from server (default is 3) |
| --reconnect-delay <number> | Number of seconds to wait before next reconnect attempt in seconds (default is 5 seconds) |
| --message <string> | Set the Message to send to the server (default is 'Hello World'). Note to place the value in between quotes (“) |
| --delay <integer> | Delay in sending the next message in seconds (default is 5 seconds). |
| --host <address> | Ip address or host name of the server to connect to |
| --port <number> | Port of the host name of the server to connect to |
| --servers <filename> | Specify the file that contains the list of servers to connect to. The client connects to the first address in the list, if connection fails then it will move to the next one. |
| -i, --id <string> | Client identity in the network (REQUIRED) |

Guideline for setting up server list:

When --host and --port is specified, it takes higher precedence than --servers. That means the client will attempt to connect to the specified host first, if connection fails then it will go thru the list provided in the server list file.

Example: *./Client.exe –id client\_1 –host localhost –port 9000 —frequency -1 #spams forever*

**Config files**

Both the Server and Client applications accepts files as configuration especially for peers (Server) and server list to connect to (Client).

This application implemented a very basic configuration file loader and parser where values are separated via a white space and loaded each line.  
See bellow information for each configuration file.

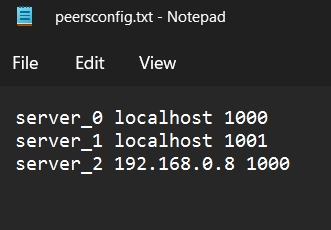
Server Peers list (--peers)

Contains the list of peers to connect to

Syntax:

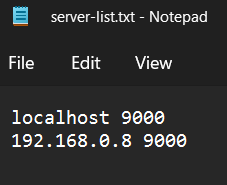
*ServerID(space)address(space)port(newline)*

***TOBE UPDATED WITH PEERS PORT at +1***

**

Client’s Server List (--servers)

Contains the list of server to connect to

 Syntax

Host(space)port(newline)

**4. Demo**

The package together with this documentation contains a demo environment, to run just execute the batch file

./ run-demo.bat

**Setup**

Running 3 Servers, 1 node connected to each servers

1. Server 1

- No peers connected at boot

- 100 thread count in pool

- Not expanding

1. Server 2

- Connect to Server 1 and Server 3 at boot

- 100 thread count in pool

- Not expanding

1. Server 3

- Connect to Server 1 and Server 2 at boot

- 100 thread count in pool

- Not expanding

1. Client 1

- Frequency -1 (sends forever)

- connects to Server 1

- send delay 1 seconds

- sends message “Hello from client 1”

1. Client 2

- Frequency -1 (sends forever)

- connects to Server 2

- send delay 1 seconds

- sends message “Hello from client 2”

1. Client 3

- Frequency 10

- connects to Server 3

- send delay 1 seconds

- sends message “Hello from client 3”

Expected Result

1. Each client will receive their own message sent to the server they are connected to
2. Each client will receive the messages from other clients.
3. Client 3 will disconnects and close after 10th message sent.

By editing the batch file provided on the demo, an experiment can be made to test various features of the programs.

Reconnection Test

1. Run 1 server and 1 client with client sending forever (frequency -1), after few seconds, close the server window. Client will detect disconnection, it will pause the Send thread and will attempt reconnection.
2. Run the server again (using same port), unless the retry attempt in the client side runs out, it should reconnect and resume send operation.

Thread pool expansion test.

1. Run 1 server and set number of thread to only 2 (numOfThreads 2) in the server config file.
2. Run client connecting to this server. Server should run properly and client should be receiving messages as expected.
3. Run another client that connects to the same server, Server should show Constant creation and deletion of new thread, both clients should work normally.
4. Close either one of the Client. Server should stop expanding/de allocating thread and both application behaves normally.

**4. Design**

**General Setup Overview**

**(drawing here)**

**Code Structure**

**Server**

The core of the server design is the thread pool.

Show flow chart.

**TODO: ID NAMING**

**server\_X**

**client\_X**

**TALK ABOUT BASIC HANDSHAKE**

**PEERS connect using different port**

**BUILD IN LINUX!!**