Design of the Project

The project intents to implement a prototype of emulator of multiple network sessions, like PPP and DHCP. It will include 3 component – GUI, server and client.

Server and client emulate the behavior of network sessions, basically client try to connect to server and setup sessions simultaneously. It may emulates the behavior of a state machine.

GUI is responsible for configuration the settings and control the activity like start, stop, reset. And it will also show the data of successful connected sessions and performance data like connection rate and runtime go-routine numbers.

The server and client will be implemented by GO with concurrency. The implementation of the GUI will be TBD - may be QtPython or even Tk. The communication between GUI and Server/client will be protocol buffer (or XML RPC but I prefer protocol buffer now). The communication between Server and Client will be udp socket.

# Phase1:

I will implement a server and client with one session at first. It's basically a prototype of the protocol which I create for the emulator.

The protocol includes 2 component – Server and Client. Server listens on a UDP socket. Client try to send out a hello packet with a magic number (a random integer of 48 bits). Server get the hello and send a hello packet back to client with the magic number as well as server's id (a maximal 32 bytes string). So the client get the hello packet and know a server responses his hello packets. And it then sent out a session request to server with the server's id. The server get the request and identify it's a request to him so it will assign a session id to client and include the session id in the reply to client. Then the client will reply with a confirm to server which mean the session is established.

Client

Server

Hello

Hello with server id

Session Request with SID

Session Reply with session id

Session Confirm with session id

Hello packets format:

{

Protocol name : 16 bytes - ChrisGoPrototype

Message name: 8 bytes - “hello”

End point magic number: 8 bytes

Server ID: 32 bytes

}

Session request format:

{

Protocol name : 16 bytes - ChrisGoPrototype

Message name: 8 bytes - “request”

End point magic number: 8 bytes

Server ID: 32 bytes

}

Session reply format:

{

Protocol name : 16 bytes - ChrisGoPrototype

Message name: 8 bytes - “reply”

End point magic number: 8 bytes

Server ID: 32 bytes

Session ID: 4 bytes

}

Session Confirm format:

{

Protocol name : 16 bytes - ChrisGoPrototype

Message name: 8 bytes - “confirm”

End point magic number: 8 bytes

Server ID: 32 bytes

Session ID: 4 bytes

}

Client State machine:

Initial → Requesting → Established

Server Session State machine:

Available → Allocating → Established