**CU30 Negotiation Protocol**

Port Numbers:
Maybe `#define` now, entered by the user or selected by OS later
NEG_O_PORT Port used for negotiation, default = 9000
RECV_PORT Port used for receiving video stream, default = 9002

```c
struct NegoMsg {
    bool fHost; // identify whether the machine is the Host, see below
    bool fCanSend; // identify whether the machine can send video
    bool fCanRecv; // identify whether the machine can receive video
    int iPortRecv; // valid only if fCanRecv, the port on which the video will be sent
}
```

Initialization:

At the beginning the machine is assumed to be the host. We first get the current time, which is going be set as the time stamp for the RtpPackets. The machine with the “earlier” timestamp will be the host. On windows this is done with:

```c
FILETIME ft;
GetSystemTimeAsFileTime(&ft); // get the 64 bits filetime
DWORD dwtime = ft.dwLowDateTime // get the least significant 32 bits
```

Then set the timestamp of the RtpPacket to be dwtime.

**However, since Linux and Windows differs from calculation of system time, the timestamp when doing negotiating between two OS can’t determine actual sequence, but this won’t affect the negotiation.**

With the input of HOSTNAME from the user, we keep doing the following at intervals of 1 second:

1. Send an RtpPacket containing NegoMsg to the HOSTNAME machine on its NEG_O_PORT, which is the machine we want to connect to. fHost is set to true.
2. Try to receive a RtpPacket on our NEGO_PORT.

If we receive an RtpPacket, we first check if it is valid by checking whether its content size is equal to the size of NegoMsg struct.

Then we check the value of fHost. If fHost is false, that means the other machine has received our first RtpPacket, and we can take up the “host” part. If fHost is true, then we compare the timestamp of the packet to our timestamp set at the beginning. If our timestamp is “earlier”, then we do nothing, else we will take up the “client” part.

**Host**

When we receive the reply message from the “client”, we first see whether we can start sending video now. If we “can send” and the other machine “can receive”, then we can simply start capturing and sending. Then we check to see if we “can receive” and the other machine “can send”. If so, then we will start listening for incoming video on PORT_RECV. We will then send confirmation message to the other machine’s NEGO_PORT. The confirmation message should just be a simple text message “CANSENDNOW” contained in an RtpPacket. It should be sent continuously every second, until we receive the video stream.

**Client**

When we take up the client part, we first check to see if both we “can receive” and the other machine “can send”. If so, we start listening for incoming video on PORT_RECV. We also check to see if both we “can send” and the other machine “can receive”. If so, we connect our sending port to the other machine’s receiving port to prepare for sending video. On windows, this is done by calling bind and connect. We then do the following every second, in case of packet loss.

1. Send a NegoMsg to the other machine’s EGO_PORT with fHost set to false, and let the other machine know our RECV_PORT.
2. Check to see if we can receive any confirmation message.

After receiving the confirmation, we can start capturing and sending video. A duplex video-conferencing will be established.