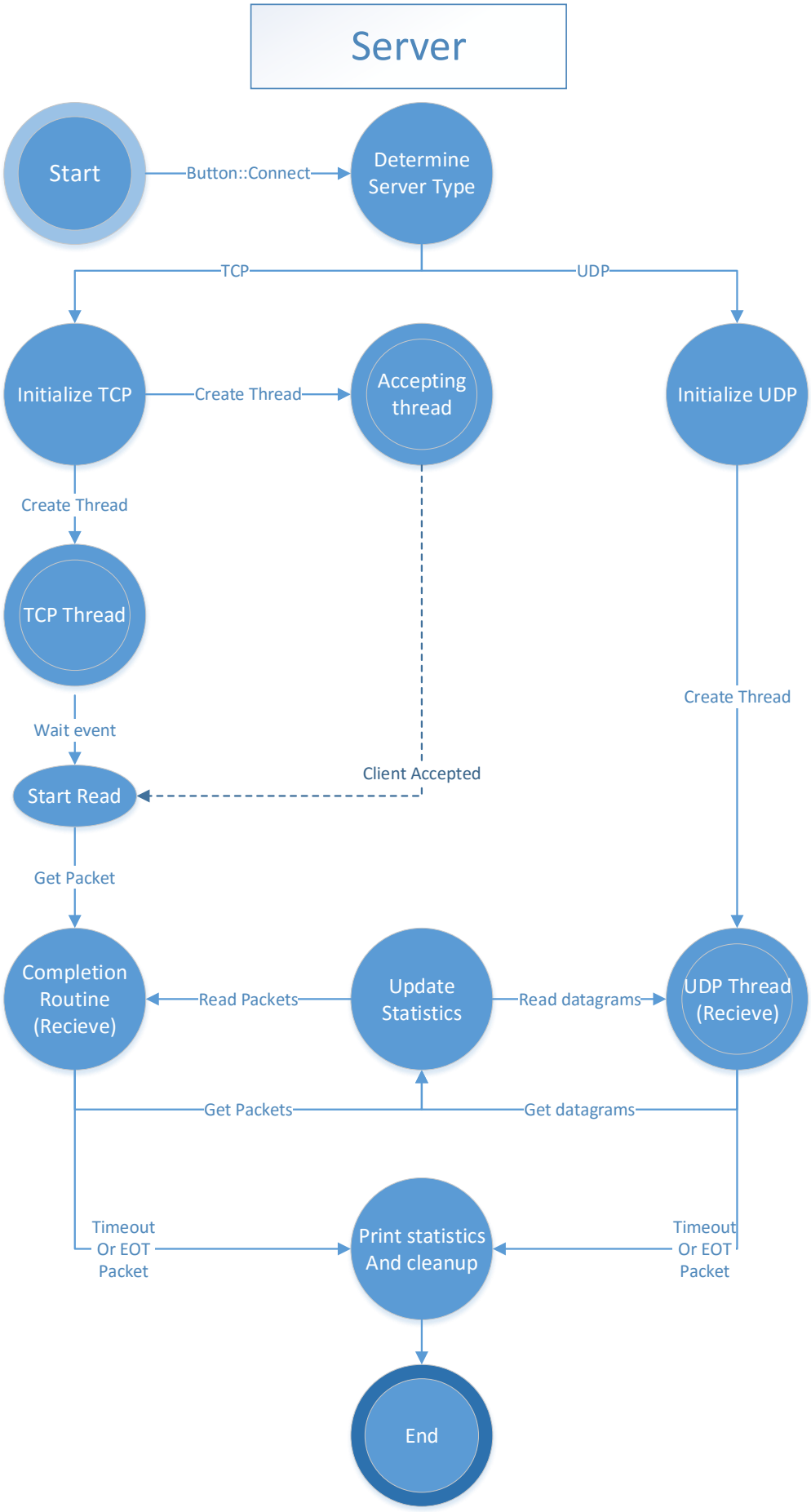
Server
Start
Button::Connect
Determine Server Type
TCP
UDP
Initialize TCP
Create Thread
Accepting thread
Initialize UDP
Create Thread
TCP Thread
Create Thread
Wait event
Client Accepted
Start Read
Get Packet
Completion Routine (Recieve)
Read Packets
Update Statistics
Read datagrams
UDP Thread (Recieve)
Get Packets
Get datagrams
Timeout Or EOT Packet
Print statistics And cleanup
Timeout Or EOT Packet
End

Global defined structs

```
typedef struct _SOCKET_INFORMATION
{
	OVERLAPPED Overlapped;
	SOCKET Socket;
	CHAR Buffer[DATA_BUFSIZE];
	WSABUF DataBuf;
	DWORD BytesSEND;
	DWORD BytesRECV;
} SOCKET_INFORMATION, *LPSOCKET_INFORMATION;

typedef struct _TRANSMISSION_INFORMATION
{
	DWORD PacketSize;
	DWORD PacketsExpected;
	DWORD PacketsRECV;
	SYSTEMTIME StartTimeStamp;
	SYSTEMTIME EndTimeStamp;
	LPSTR ProtocolType;
} TRANSMISSION_INFORMATION, *LPTRANSMISSION_INFORMATION;
```

Server Pseudocode

Determine Server Type

Waits for windows messages received from **WNDPROC**
	Check the message and see if TCP radio button is selected
		If it is
			Close winsock session if its already opened
			transition to **Initialize TCP**
		Else
			Close winsock session if its already opened
			Transition to **Initialize UDP**

Initialize TCP

Start a WINSOCK session
Create a TCP socket for receiving packet streams
Initialize the server address structure
Bind the server address structure to the accepting socket
Tell the accepting socket to only listens for 1 connection (Peer to peer)

Create a thread for accepting connections so the main thread doesn't hang
Transition to **Accepting Thread**
Create a thread to handle TCP communications
Transition to **TCP Thread**

Initialize UDP

Start a WINSOCK session
Create a UDP socket for receiving datagrams
Initialize the server address structure
Bind the server address structure to the server socket
Create a thread to handle UDP communications
Transition to **UDP Thread**

Accepting Thread

Create a dummy WSA Event object
Calls accept on the listening socket
When accept returns, assign the return value to a new socket, called accepted socket
Set / signal the dummy event object, which is being blocked on the initialize **TCP Thread**
End thread

TCP Thread

Has a global Boolean variable to indicate EOT
Open a file for writing incoming packet contents
Forever loop 1:
    Forever loop 2:
        Call WSAWaitForMultipleEvents on a dummy object, which is triggered by either an IO event on the socket, or the dummy WSA Event being triggered on the **Accepting Thread.**
        When the event is triggered, check if it is the event being triggered after accepting a connection
            If it is, break out of loop 2:
        Check if the EOT Boolean is set to TRUE
            Increment packets received if there are any remaining bytes in the last packet
            Transition to **Print statistics and Cleanup**
    Reset the dummy event, so it can be listened again

Call WSARecv on the socket to start listening for packet streams.
When a IO event has been triggered and a packet stream is read
Transition to **Completion Routine (Recieve)**

Completion Routine (Recieve)

Called when an IO has been triggered on WSARecv from the **TCP thread**
If error code is not zero:
Exit thread
If the size of the packet has not been set yet, indicating the first packet just arrived
Start system timer for start of time
Extract the control message, and store it into TRANSMISSION_INFORMATION struct
If no bytes is received and the packet content is all null characters
End system timer
Set EOT to TRUE
Close socket and exit thread
Transition to **Update Statistics**
Call WSARecv on the socket
When a IO event has been triggered and a packet stream is read
Transition to **Completion Routine (Recieve)**

UDP Thread (Recieve)

Open a file for writing incoming datagram contents
Create a dummy timer event
Calls recvfrom on the socket, which queries for an initial message
When message is received, check the content and see if it contains the control message format
The control message format will be in "packet size.send times", extract it into the TRANSMISSION_INFORMATION structure
Start system timer for start of time
Create the timer thread for timeouts when no packets are arriving anymore
Forever loop:
Post an asynchrounous recvfrom request
Transition to **Update Statistics**
Set / signal the timer thread that an IO event has occurred
If last datagram is received, which contains a datagram filled with null characters. Or number of bytes received is 0

Break out of the loop
Get system end time
Transition to **Print statistics and cleanup** state

Update Statistics

Called by either **UDP Thread** or **Completion Routine (Recieve)** when something is received from the socket
If the receive call failed
Transition to **UDP Thread** or **Completion Routine (Recieve)**
Check if we have enough bytes to fill up a packet by and if bytes received is greater than a packet length
Increment number of total packets received
Write the received buffer message to a file
Transition to **UDP Thread** or **Completion Routine (Recieve)**

Print statistics and cleanup

Open file to append statistics info
Write the TRANMISSION_INFORMATION statistics into the file
Set EOT Boolean to False
Close the accepted socket
Close WSA session
Close file
Transition to **END**