



**INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH
TECHNOLOGY**

MULTI TASKING – SOCKETS IN DATA TRANSMISSION

C.Saranya*, L.Gomathi

* Research Scholar, Department of Computer Science, Muthayammal College of Arts & Science, India

* Associate Professor, Department of Computer Science, Muthayammal College of Arts & Science, India

ABSTRACT

This Paper a new socket class which supports both TCP and UDP communication. But it provides some advantages compared to other classes that you may find here or on some other Socket Programming papers. First of all, this class doesn't have any limitation like the need to provide a window handle to be used. This limitation is bad if all you want is a simple console application. So this library doesn't have such a limitation. It also provides threading support automatically for you, which handles the socket connection and disconnection to a peer. It also features some options not yet found in any socket classes that I have seen so far. It supports both client and server sockets. A server socket can be referred as to a socket that can accept many connections. And a client socket is a socket that is connected to server socket. You may still use this class to communicate between two applications without establishing a connection. In the latter case, you will want to create two UDP server sockets (one for each application). This class also helps reduce coding need to create chat-like applications and IPC (Inter-Process Communication) between two or more applications (processes). Reliable communication between two peers is also supported with TCP/IP with error handling. You may want to use the smart addressing operation to control the destination of the data being transmitted (UDP only). TCP operation of this class deals only with communication between two peers.

KEYWORDS: TCP, UDP, Server, Packets, Peer

INTRODUCTION

This is about a client/server multi-threaded socket class. The thread is optional since the developer is still responsible to decide if needs it. There are other Socket classes here and other places over the Internet but none of them can provide feedback (event detection) to your application like this one does. It provides you with the following events detection: connection established, connection dropped, connection failed and data reception (including 0 byte packet).

PROBLEM DESCRIPTION

This paper presents a new socket class which supports both TCP and UDP communication. But it provides some advantages compared to other classes that you may find here or on some other Socket Programming papers. First of all, this class doesn't have any limitation like the need to provide a window handle to be used. This limitation is bad if all you want is a simple console application. So this library doesn't have such a limitation. It also provides threading support automatically for you, which handles the socket connection and disconnection to a peer. It also features some options not yet found in any socket classes that I have seen so far. It supports both client and server sockets. A server socket can be referred as to a socket that can accept many connections. And a client socket is a socket that is connected to server socket. You may still use this class to communicate between two applications without establishing a connection. In the latter case, you will want to create two UDP server sockets (one for each application). This class also helps reduce coding need to create chat-like applications and IPC (Inter-Process Communication) between two or more applications (processes). Reliable communication between two peers is also supported with TCP/IP with error handling. You may want to use the smart addressing operation to control the destination of the data being transmitted (UDP only). TCP operation of this class deals only with communication between two peers.

EXISTING

This paper presents a new socket class which supports both TCP and UDP communication. But it provides some advantages compared to other classes that you may find here or on some other Socket Programming papers. First of all, this class doesn't have any limitation like the need to provide a window handle to be used. This limitation is bad if all you want is a simple console application. So this library doesn't have such a limitation. It also provides threading support automatically for you, which handles the socket connection and disconnection to a peer. It also features some options not yet found in any socket classes that I have seen so far. It supports both client and server sockets. A server socket can be referred as to a socket that can accept many connections. And a client socket is a socket that is connected to server socket. You may still use this class to communicate between two applications without establishing a connection.

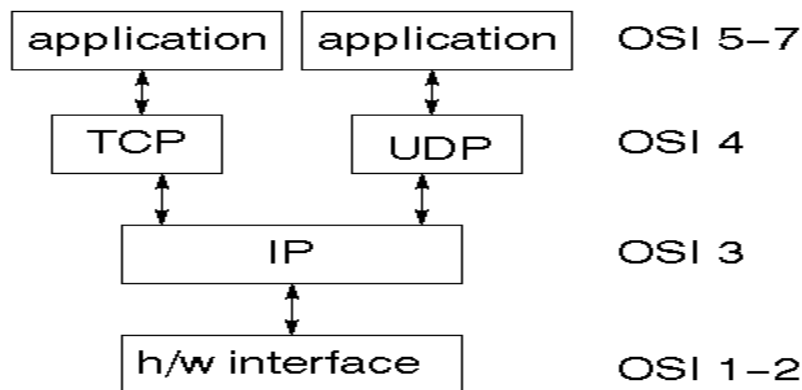
PROPOSED

In the latter case, you will want to create two UDP server sockets (one for each application). This class also helps reduce coding need to create chat-like applications and IPC (Inter-Process Communication) between two or more applications (processes). Reliable communication between two peers is also supported with TCP/IP with error handling. You may want to use the smart addressing operation to control the destination of the data being transmitted (UDP only). TCP operation of this class deals only with communication between two peers.

ANALYSIS OF NETWORK CLIENT SERVER

- TCP/IP stack

The TCP/IP stack is shorter than the OSI one:



TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.

- IP datagram's

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through an Internet. It is also responsible for breaking up large datagram into smaller ones for transmission and reassembling them at the other end.

- UDP

UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model - see later.

- TCP

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

- Internet addresses

In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32 bit integer which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

- Network address

Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16 bit network addressing. Class C uses 24 bit network addressing and class D uses all 32.

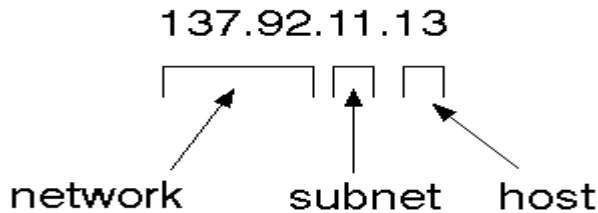
- Subnet address

Internally, the UNIX network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

- Host address

8 bits are finally used for host addresses within our subnet. This places a limit of 256 machines that can be on the subnet.

Total address



The 32 bit address is usually written as 4 integers separated by dots.

- Port addresses

A service exists on a host, and is identified by its port. This is a 16 bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

MODULE DESIGN SPECIFICATION

part 1 - Create a server socket that listen for a client to connect

part 2 - send / receive data from client to server

part 3 - Read unknown size of data from client

- Create a server socket that listens for a client to connect

Associates a local address with a socket This routine is used on an unconnected datagram or stream socket, before subsequent connects or listens. When a socket is created with socket, it exists in a name space (address family), but it has no name assigned. bind establishes the local association (host address/port number) of the socket by assigning a local name to an unnamed socket. In the Internet address family, a name consists of several components. For SOCK_DGRAM and SOCK_STREAM, the name consists of three parts: a host address, the protocol number (set implicitly to UDP or TCP, respectively), and a port number which identifies the application. If an application does not care what address is assigned to it, it may specify an Internet address equal to INADDR_ANY, a port equal to 0, or both. If the Internet address is equal to INADDR_ANY, any appropriate network interface will be used; this simplifies application programming in the presence of multi- homed hosts. If the port is specified as 0, the Windows Sockets implementation will assign a unique port to the application with a value between 1024 and 5000. The application may use getsockname after bind to learn the address that has been assigned to it, but note that getsockname will not necessarily fill in the Internet address until the socket is connected, since several Internet addresses may be valid if the host is multi-homed. If no error occurs, bind returns 0. Otherwise, it returns SOCKET_ERROR, and a specific error code may be retrieved by calling WSAGetLastError.

Establishes a socket to listen to a incoming connection To accept connections, a socket is first created with socket, a backlog for incoming connections is specified with listen, and then the connections are accepted with accept. listen applies only to sockets that support connections, i.e. those of type SOCK_STREAM. The socket s is put into "passive" mode where incoming connections are acknowledged and queued pending acceptance by the process. This function is typically used by servers that could have more than one connection request at a time: if a connection request arrives with the queue full, the client will receive an error with an indication of WSAECONNREFUSED. listen attempts to continue to function rationally when there are no available descriptors. It will accept connections

until the queue is emptied. If descriptors become available, a later call to listen or accept will re-fill the queue to the current or most recent "backlog", if possible, and resume listening for incoming connections.

This routine extracts the first connection on the queue of pending connections on *s*, creates a new socket with the same properties as *s* and returns a handle to the new socket. If no pending connections are present on the queue, and the socket is not marked as non-blocking, accept blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, accept returns an error as described below. The accepted socket may not be used to accept more connections. The original socket remains open. The argument *addr* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the *addr* parameter is determined by the address family in which the communication is occurring. The *addrlen* is a value-result parameter; it should initially contain the amount of space pointed to by *addr*; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types such as SOCK_STREAM. If *addr* and/or *addrlen* are equal to NULL, then no information about the remote address of the accepted socket is returned. Making client connection with server In order to create a socket that connects to another socket uses most of the functions from the previous code with the exception of a struct called HOSTENT

- HOSTENT:

This struct is used to tell the socket to which computer and port to connect to. These struct can appear as LPHOSTENT, but it actually means that they are pointer to HOSTENT.

- Client key function

Most of the functions that have been used for the client to connect to the server are the same as the server with the exception of a few. I will just go through the different functions that have been used for the client. The contents of this structure correspond to the hostname name. The pointer which is returned points to a structure which is allocated by the Windows Sockets implementation. The application must never attempt to modify this structure or to free any of its components. Furthermore, only one copy of this structure is allocated per thread, and so the application should copy any information which it needs before issuing any other Windows Sockets API calls. A gethostbyname implementation must not resolve IP address strings passed to it. Such a request should be treated exactly as if an unknown host name were passed. An application with an IP address string to resolve should use *inet_addr* to convert the string to an IP address, then *gethostbyaddr* to obtain the hostent structure.

- Part 2 - Send / receive

Up to this point we have managed to connect with our client to the server. Clearly this is not going to be enough in a real-life application. In this section we are going to look into more details how to use the send/rcv functions in order to get some communication going between the two applications. Factually this is not going to be difficult because most of the hard work has been done setting up the server and the client app. before going into the code we are going to look into more details the two functions send(SOCKET *s*, const char FAR * *buf*, int *len*, int *flags*) send is used on connected datagram or stream sockets and is used to write outgoing data on a socket. For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets, which is given by the *iMaxUdpDg* element in the WSADATA structure returned by WSASStartup. If the data is too long to pass atomically through the underlying protocol the error WSAEMSGSIZE is returned, and no data is transmitted.

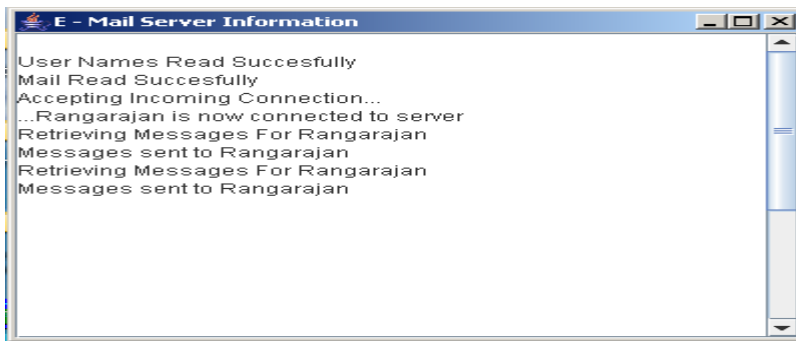
For sockets of type SOCK_STREAM, as much information as is currently available up to the size of the buffer supplied is returned. If the socket has been configured for in-line reception of out-of-band data (socket option SO_OOBINLINE) and out-of-band data is unread, only out-of-band data will be returned. The application may use the ioctlsocket SIOCATMARK to determine whether any more out-of-band data remains to be read.

- Read unknown size of data from client

Us mentioned earlier in part 2, we are now going to expand on the way that we receive data. The problem we had before is that if we did not know the size of data that we were expecting, then we would end up with problems. In order to fix this here we create a new function that receive a pointer to the client socket, and then read a char at the time, placing each char into a vector until we find the '\n' character that signifies the end of the message. This solution is clearly not a robust or industrial way the read data from one socket to another, because but its a way to start reading unknown length strings. the function will be called after the accept method

Starting Server:

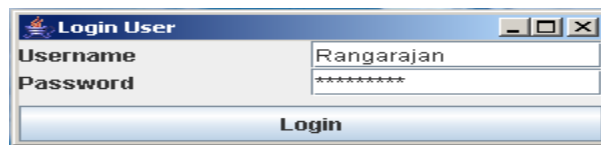
I update data transaction automatically.



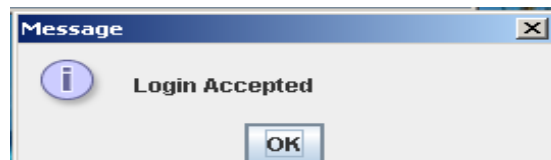
While we are starting Clients the main Login menu display is shown below.

FUNCTIONAL DESCRIPTION

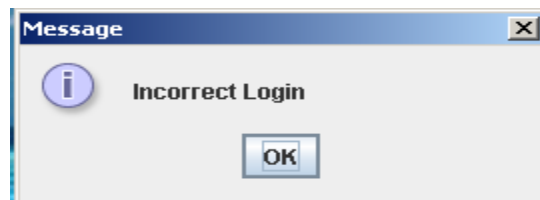
Login User form Getting input for email ID and password from user



if it is valid ID then Displays



if the user is not valid then the form will be



- Transmitting Message and Files between Client and Server.
- The data will be valid until the Server is Valid.
- Everything defined as Object oriented.
- Server has been developed using Server Socket programming in Java.

- Client has been developed using Socket Programming.
- Transmitting data between Client and Server has been developed IP address.
- Usage of Sql Server will not be needed for sending Messages and attachments.

OPERATIONS INVOLVED IN EMAIL SERVER

- Create new instance of server storage vectors
- Create new instance of server GUI to display messages
- Starts the server
- Set up socket to accept connections
- Accept new client
- Update server window details
- Run indefinitely
- Starts up the server in the correct manner
- Extract information for server from files
- Read object from file and add to server vector
- Gets a vector of emails from persistent storage
- Update server message window
- Send all server mail to a file



CONCLUSION

In this paper, we study TCP and UDP for the downlink of a single cell that can maximize the asymptotic decay rate of the queue-overflow probability as the overflow threshold approaches infinity. Specifically, we focus on the class of “UDP,” which pick the user for server at each time that has the largest product of the transmission rate multiplied by the backlog raised to the power. We show that when approaches infinity, the asymptotically achieve the largest decay rate of the queue-overflow probability. A key step in proving this result is to use a function to derive a simple lower bound for the minimum cost to overflow under the tcp - server. This technique, which is of independent interest, circumvents solving the difficult multidimensional calculus- of-variations problem typical in this type of problem. Finally, using the insight from this result, we design hybrid scheduling algorithms that are both close to optimal in terms of the asymptotic decay rate of the overflow probability and empirically shown to maintain small queue-over flow probabilities over Queue-length ranges of practical interest. For future work, we plan to extend the results to more general network and channel models.

REFERENCES

- [1] M. I. Andreica, A. Dragus,, A.-D. Sambotin, and N. T, apus,, “Towards a (Multi-) User-Centric Stack of (Multi-) Point-to-(Multi-) Point Communication Services”, in Proc. 5th Worksh. Enhanced Web Serv. Technol. WEWST, Ayia Napa, Cyprus, 2010, pp. 36–45.
- [2] Valentin Stanciu, Mugurel Ionut, Andreica, “Design and development of a UDP-Based Connection-Oriented Multi-Stream One-to-Many Communication Protocol” 2012.
- [3] Paul Stalvig, “Introduction to the Stream Control Transmission Protocol (SCTP)” Oct2007
- [4] R. Stewart, RFC 4960 “Stream Control Transmission Protocol” September 2007
- [5] David A. Hayes, Jason But, Grenville Armitage ”Issues with Network Address Translation for SCTP”
- [6] Shwartz and A.Weiss, Large Deviations for Performance Analysis: Queues, Communications, and Computing. London, U.K.: Chapman & Hall, 1995.
- [7] Dembo and O. Zeitouni, *Large Deviations Techniques and Applications*, 2nd ed. New York: Springer-Verlag, 1998.
- [8] I. Elwalid and D. Mitra, “Effective bandwidth of general Markovian traffic sources and admission control of high speed networks,” *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 329–343, Jun. 1993.

AUTHOR BIBLIOGRAPHY

 <p>89882</p>	<p>C.Saranya Received <i>B.sc Computer Science degree</i> from Trinity College for Women, <i>Affiliated to Periyar University</i>, Namakkal 2012. M.Sc computer Science at Trinity College for Women, <i>Affiliated to Periyar University</i>, Namakkal , 2014 & Pursuing my M.Phil (Full time) at Muthayammal College of Arts & Science , <i>Affiliated to Periyar University</i>, Namakkal, India.</p>
	<p>L.Gomathi Currently doing Ph.D. She received her BCA degree from Bharathidasan University, Chitode 2002 and MCA degree from Bharathidasan University, Trichirapalli 2005. She has completed her M.Phil at Periyar University, Salem, 2007. She is having 9 years of experience in collegiate teaching and she is the Associate Professor, Department of BCA in Muthayammal College of Arts and Science, Rasipuram affiliated by Periyar University, Namakkal, India.</p>