Technical Design Document

Basic ideas:

1. Initial socket first. If *sendto()* function is to be used, no bind() is needed. If *recvfrom* is to be used, *bind()* is a must.
2. Client initializes a connection, starts to send data and wait for ACK after each package is sent. If timeout, re-transmit.
3. Server keeps receiving data. If no data is received, return error. If a package is received, send ACK immediately or delay a while and send ACK (random delay/drop version).
4. Client receives ACK, starts to transmit the next package.
5. To simulate randomly package drop, just set a wrong ACK content at a random time.

To create a socket, as if both the server and client are using the same port, there is a clash at the port. In order to solve this, two socket creation function is implemented. One is for server, the other is for client. Use one socket/port to send and receive normal data. The other socket/port is used to send and receive ACK only.

The function *recvfrom* will block the channel. Thus, it is needed to set the connection to non-block model and function *setsocketopt* is being used.

For the client, the only condition for re-transmit package is that after timeout no ACK is received. This can happen because either some package sent is lost or ACK is lost. (Figure 1 and figure 2)

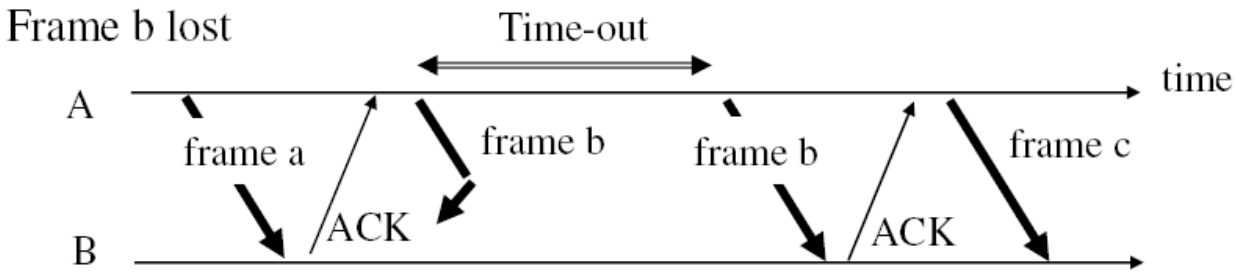

Figure 1

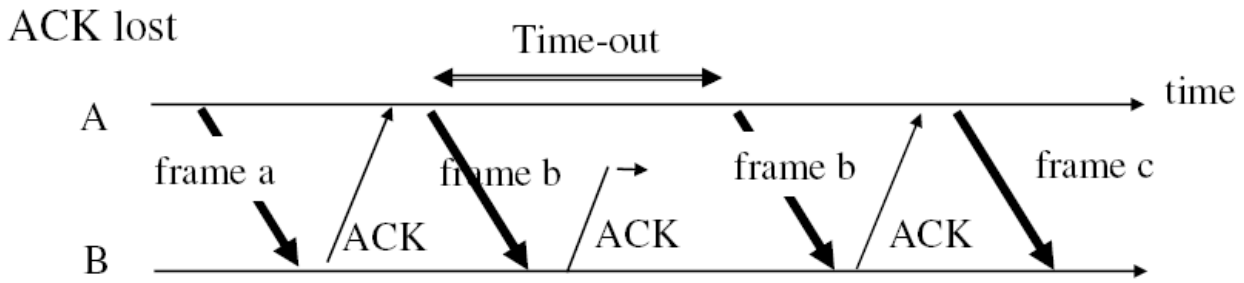

Figure 2

By using Stop-and-Wait ARQ, reliable transmission is made.

Each data package is defined into three parts:

1. The first two bytes contains the length of the whole message. As every is character string, the first bytes is represented by some ASCII code's corresponding integer value, which is exactly the multiple of 200 while the second byte is represented also by some ASCII code's corresponding integer value, which is exactly the arithmetical compliment. By calculating (1^{st} byte x 200 + 2^{nd} byte) can we get the length of the original message. The reason why a 33 is added is to make the integer value be larger than 33, which can avoid those undefined area in ASCII table.
2. The third byte of a package is exactly the package's order, which is represented by the same method as above.
3. The ACK is only one byte, which has a single character value represents the received package number.
4. As the input and output buffer is just a one dimensional string, in order to put a long string (greater than 1KB) which will be divided to several parts of 1KB each, the buffer needs to be cut into 32 pieces. For this reason, function *strcpy* cannot be used. The only way to do so is to copy the string byte by byte. This data will occupy the position from 3 to 1026.
5. The last byte is an ending symbol.

Case that may cause re-trasnmission:

1. Client sent data, server did not receive. Server did nothing, client sent the package again after time out.
2. Client sent data, server received and sent ACK. ACK lost. Client sent the package again after timeout.

3. Client sent data, server received and sent ACK. ACK delayed. Client sent the package again after timeout but received the previous ACK later. Client sent the next package after received the ACK.
4. Client sent data, server received and sent ACK. For some reason the client sent the same package twice, then, the server received sent the ACK with the duplicated packages' number back. And the server may receive the next package (due to delay) or the client sent the same package again.

For those who will implement or edit the code:

1. The timeout time is defined in the RUDP.h. Anyone who may implement this program can change this value. Or even change the way to do it, such as dynamic timeout setting.
2. If the pre-defined port number is being occupied, change the value in RUDP.h. By default, *SOCK1* will use port number defined as *PORT* while *SOCK2* uses port number *PORT+1*.
3. This program is for local host communication only. If it is used for non-local host, change the pre-defined IP in RUDP.h.
4. The number of attempts of re-transmitting packages is set to 5, also defined as *LIMIT* in RUDP.h. Change the value if needed.
5. For demo usage, functions of delaying a random time and randomly dropping a package are implemented, which has a switch as the second argument of function *RUDP_RreceiveFrom*. If the argument is set to 0, that means random delay/drop function is disabled where they are enabled if the argument is set to 1.
6. Connection speed testing program is not included in the sample program. To use it, just use *RUDP_speed* to get an integer value, which represents the average delay (in micro second) of five transmissions with 1KB data in each package.
7. If the program is used for multi-connections, further methods is needed to make the socket can deal with multiple connection requests.

Notice: If the program cannot be compiled by Visual Studio, etc., go to the Project Properties, Configuration Properties, Linker, Input, Additional Dependencies, add "ws2_32.lib".