Background

In this assignment, you will implement an extension of UDP, namely Guaranteed UDP (GUDP). GUDP transfers UDP datagrams with guarantees on delivery and ordering. This is an extension of regular UDP, which does not guarantee delivery or ordering datagrams.

To guarantee delivery and ordering, GUDP adds the following mechanisms to UDP:

- Sliding window flow control
- Detection and ARQ-based retransmission of lost or reordered packets

Note that GUDP is not TCP! Several factors distinguish GUDP from TCP:

- GUDP transfers datagrams, while TCP transfers byte streams.
- GUDP provides a connection-less communication service, while TCP is connection-oriented.
    - TCP requires that the client application establishes a connection, and the server accepts the connection, before communication. GUDP does not have the notion of connections.
- GUDP is unidirectional, while a TCP connection is bidirectional.

GUDP Header

GUDP consists of an additional protocol header that comes directly after the UDP header in an IP packet. The GUDP header has the following format:

Bit 0-15	**Bit 16-31**
Version	Type
Sequence number	

The fields in the GUDP header have the following meaning:

- Version: version of the GUDP protocol. The current version is version 1.
- Type: GUDP packet type (DATA, BSN, and ACK; see below)
- Datagram sequence number

All header fields are in *network byte order*.

Protocol Description

Since GUDP provides a unidirectional communication service, there is no data flow in the opposite direction onto which control information can be "piggybacked". Instead, GUDP uses separate control packets (DATA, BSN, ACK) to guarantee delivery.

GUDP provides guaranteed communication from a *sender UDP endpoint* to a *receiver UDP endpoint*. A UDP endpoint (sender or receiver) is uniquely defined by an IP address and a UDP port number.

DATA Packets

DATA packets (type field = 1) carry application data; the sequence number increases with one for each DATA packet sent from the sender to the receiver. So, unlike TCP, GUDP numbers packets, not bytes.

BSN Packets

BSN (Base Sequence Number) packets (type field = 2) are used to control sequence numbers. Like TCP, GUDP initial sequence numbers are randomised, and the BSN packet is used by the sender to tell the receiver the Base Sequence Number – the starting point of the sequence numbers for DATA packets. The sequence number for DATA packets starts with one plus the BSN and consecutively increases by one for each new DATA packet sent.

ACK packets

ACK packets (type field = 3) acknowledge the reception of DATA packets. The sequence number field in an ACK packet contains the sequence number of the last DATA packet received by the receiver UDP endpoint, plus one. In other words, the sequence number field is the sequence number of the DATA packet the receiver expects to receive next.

ACK packets are also used by the receiver UDP endpoint to acknowledge the reception of BSN packets. So when a receiver gets a BSN packet, the receiver responds with an ACK packet that contains the sequence number from the BSN packet plus one (since that is the sequence number of the DATA packet that the receiver is now expecting).

Timers

If an ACK is not received within a certain time after that the DATA packet was sent, the DATA packet is considered to be lost and should be retransmitted.

The sender can attempt to retransmit a packet up to a certain number of times. After that, communication has failed, and the application should be informed.

The same applies to BSN packets – if no ACK is received within a certain time, communication has failed.

Windows

GUDP maintains a window size, which is the maximum number of outstanding DATA packets that have not been acknowledged. So when the sender has reached the window size, it cannot send any more packets to the receiver. Instead, it has to wait for an ACK packet or a time-out.

GUDP Socket Application Programming Interface (API)

Applications use GUDP through the GUDP Sockets API. A GUDP socket is represented in Java by the GUDPSocket class. It has one constructor:

```
public GUDPSocket(DatagramSocket socket);
```

The constructor takes an existing DatagramSocket (UDP socket) into a GUDP socket.

The class has the following methods:

```
public void send(DatagramPacket packet) throws IOException;
public void receive(DatagramPacket packet) throws IOException;
public void finish() throws IOException;
public void close() throws IOException;
```

The send() method takes a DatagramPacket and encapsulates it in a GUDP packet before sending it to its destination endpoint. The receive method receives a packet from a remote endpoint. The received packet is decapsulated and placed in the DatagramPacket before passing it as a parameter to receive().

The finish() method is called by a GUDP sender to indicate that it has sent the last datagram in a communication. If the finish method returns normally, it means that all previous datagrams have been delivered successfully. If the delivery fails, the finish() method throws an IOException (see below).

The close() method deletes the socket and frees up any resources used by the socket.

Error Handling

Should an error occur during communication so that a retransmission timer expires, it is indicated by an IOException being thrown by send(), receive(), finish(), or close(), with an appropriate error message.

Threads and Buffers

In this assignment, we implement GUDP as a Java class, which means that GUDP will be part of the application program. In other words, a GUDP application will use the GUDP Java class and call its methods to send and receive data.

GUDP and the application must be able to execute independently, and in parallel. Therefore, GUDP, as well as the application, are implemented as Java *threads* – lightweight processes that can execute simultaneously. In fact, you probably want to implement GUDP as two threads; a sender thread and a receiver thread.

This also means that GUDP needs to use buffers to hold application datagrams. For instance, when the application wants to send a datagram, it calls the GUDP socket's send() method, which places the datagram in a buffer for outgoing datagrams, and tells the GUDP sender thread that there is data in the buffer. The GUDP sender then processes the datagram: assigns a sequence number to the datagram, adds a GUDP header, transmits the packet (if the transmit window is open), etc.

In the same way, there needs to be receive buffers where the GUDP receive thread places received datagrams (GUDP DATA). The application fetches datagrams from the buffer by calling the GUDP socket's receive method. If there are no datagrams in the receive buffer, the application will be suspended (blocked) until a datagram arrives.

Multiplexing

Unlike TCP sockets, a UDP socket allows the application to communicate with several endpoints using the same socket. For example, a sender application can send a datagram to one endpoint, and then another datagram to a different endpoint, on the same socket.

We want to keep the same communication model for GUDP so that communication with several endpoints can be "multiplexed" on a single GUDP socket. However, GUDP still needs to maintain a sliding window for each endpoint, which means that most likely the GUDP socket needs to have multiple sender and receiver buffers.

Java source code

You can find the tarball of the source code template (GUDP.tgz Download GUDP.tgz) in the lab module in Canvas. It contains three files as follows:

- GUDPPacket.java is the class for GUDP protocol declarations with associated methods to access the GUDP packet header and payload.
- GUDPSocketAPI.java is the interface, providing the well-defined API that you must use for your implementation
- GUDPSocket.java is the class for GUDP library that you will implement

Your main task is to implement GUDP in the GUDPSocket.java file to handle send and receive with sliding window flow control and automatic repeat request (ARQ).