Detailed Description of Classes in ARQ Simulator

Simulator

This is the main class of the simulator which controls and runs the whole simulator application. In the main function, a single instance of this class is created as soon as user runs simulator.

```
private:
      Node Source;
      Node Sink;
      double propogationDelay;
      double bitRate;
      double duration;

public:
      char settingsFile[256];
```

Source

The source node for the simulation.

Sink

The sink node for the simulation.

propogationDelay

The network propagation delay to use in the simulations.

bitRate

The transmission bit rate to use in the simulations.

duration

The duration of the simulation.

settingsFile[256]

Name of the settings file from which different simulation settings are loaded.

```
public:
      int readSettings ();
      int simulate();
      int verify();
```

readSettings

Read the simulation settings from the ***settingsFile*** and update the relevant fields.

verify

Verifies the simulation settings for correctness.

simulate

Iteratively executes the relevant methods of ***Source*** and ***Sink***, and exchange messages between them so that ARQ protocol functionality can be simulated.

Node

This class describes a source or sink node.

```
private:
      char type;
      static double dataPacketSize;
      static double dataOverhead;
      static double ackSize;
      static double processingDelay;
      static double packetRate;
      int sequenceNumber;
      PacketQueue PacketBuffer;

public:
      PacketQueue OutgoingPacketQueue;
      PacketQueue IncomingPacketQueue;
```

type

Indicates whether it is a source (type='1') or a sink (type='2') node.

dataPacketSize

Size of data packet in bits.

dataOverhead

Number of overhead bits in data packet.

ackSize

Size of ACK message.

processingDelay

Processing delay of a node. This value is same for both data and ACK packet.

packetRate

The rate at which the source node generates data packets. Packet generation is independent of ARQ functionality and the source should keep on generating packets as per this value.

sequenceNumber

The current sequence number of this node. In source, it represents the sequence number of last generated data packet. However, in sink its use is not defined and user can implement it as per his/her needs..

PacketBuffer

This buffer is used to store the generated packets. Whenever a node generates a packet (data or ACK), this packet should first be placed in this buffer. Later, after processing delay this packet should be moved to ***OutgoingPacketQueue*** for transmission.

OutgoingPacketQueue

Contains packets which are ready for transmission. A packet should be placed in this buffer only when all necessary actions are performed on this packet.

IncomingPacketQueue

The simulator places the packets received from other nodes for this node in this buffer. A node can start processing these packets.

```
public:
      char getType();
      static double getDataPacketSize();
      static double getDataOverhead();
      static double getACKSize();
      static double getProcessingDelay();
      static double getPacketRate();
      void setType(char typ);
      static void setDataPacketSize(double dPS);
      static void setDataOverhead(double dO);
      static void setACKSize(double aS);
      static void setProcessingDelay(double pD);
      static void setPacketRate(double pR);
      int generatePackets();
      int sendPackets();
```

getDataPacketSize, getDataOverhead, getACKSize, getProcessingDelay, getPacketRate

Get functions for different attributes/parameters.

setType, setDataPacketSize, setDataOverhead, setACKSize, setProcessingDelay, setPacketRate

Set functions for different attributes/parameters.

generatePackets

This method simulates the packet generating entity. The data packet generation scheme of source should be implemented in this method. Any generated packets should be placed in *PacketBuffer*.

sendPackets

This method is responsible to perform all ARQ operations before and after the transmission of a packet, as well as on the reception of a packet. Since, this method is same in both the source and the sink, the operations of source and sink should be differentiated.

Packet

This class represents a raw packet with only necessary components and operations to implement ARQ functionality.

```
private:
      char type;
      char source;
      char destination;
      int sequenceNumber;
      double size;
      double overhead;
      double creationTime;
      double readyToSendTime;
```

type

The type of packet: Data (type='1'), ACK (type='2'), or NACK (type='3').

source

The node which has generated this packet, either source or sink.

destination

The node for which this packet is destined, either source or sink.

sequenceNumber

The sequence number of this packet. In case of ACK, it represents the sequence number of data packet for which this ACK is generated, while in case of NACK it represents the sequence number of missing data packet.

size

The size of this packet.

overhead

The overhead of this packet. In case of ACK and NACK, this is always zero.

creationTime

The time when this packet was created.

readyToSendTime

The time when this packet was ready for transmission i.e. placed in the ***OutgoingPacketQueue***.

```
public:
      char getType();
      char getSource();
      char getDestination();
      int getSequenceNumber();
      double getSize();
      double getOverhead();
      double getCreationTime();
      double getReadyToSendTime();
      void setType(char typ);
      void setSource(char src);
      void setDestination(char dst);
      void setSequenceNumber(int sn);
      void setSize(double sz);
      void setOverhead(double oh);
      void setCreationTime(double ct);
      void setReadyToSendTime(double rtst);
```

Different get/set functions for accessing fields of the packet.

PacketQueue

This class provides the necessary features for different types of packet buffers used in this simulator.

```
private:
      int length;
      double leastTime;
      int leastIndex;
      double leastSize;
      Packet** Packets;
```

length

Length of this queue/buffer i.e. number of packets in this buffer.

leastTime

The least value of ***creationTime*** or ***readyToSendTime*** stored in the buffer i.e. the ***creationTime*** or ***readyToSendTime*** of the earliest most packet placed in this buffer. In case of ***PacketBuffer***, this is the ***creationTime*** while in case of ***IncomingPacketQueue*** or ***OutgoingPacketQueue*** it is the ***readyToSendTime*** of the packet.

leastIndex

The index of the packet in the buffer with ***leastTime***.

leastSize

The size of the packet in the buffer with ***leastTime***.

Packets

The array of stored packets.

```
public:
      int getLength();
      double getLeastTime();
      int getLeastIndex();
      double getLeastSize();
      Packet* dequeuePacket(int index, int qType);
      void enqueuePacket(Packet* pkt, int qType);
```

getLength, getLeastTime, getLeastIndex, getLeastSize

Get functions for different attributes/parameters.

enqueuePacket

Adds a packet to this queue/buffer. This method automatically identifies the packet with least time and adjust the relevant attributes. The ***qType*** parameter indicates that whether it is a ***PacketBuffer***, or an ***IncomingPacketQueue***/***OutgoingPacketQueue*** so that the least packet should be selected based on ***creationTime*** or ***readyToSendTime***.

dequeuePacket

Removes the specified packet from this queue. Like ***enqueuePacket***, this method also adjust the least packet related attributes after removing the packet based on the value of ***qType*** parameter.

Time

This is a utility class to keep track of simulation time so that different events can be performed on specified time. A user should neither create an instance of this class nor should use/call any methods or members of this class except the ***getTime***.

```
private:
      static double nanoseconds;
```

nanoseconds

The number of nanoseconds passed since the start of the simulation.

```
public:
      static double start();
      static double getTime();
      static double increase();
```

start

This method is called in response to “run” command to mark the beginning of the simulation.

getTime

Returns the current time i.e. the nanoseconds passed since the beginning of the simulation.

increase

This method marks the end of one simulation tick i.e. one nanosecond or in other words increases the time.

Commands supported at the simulator command line

This section describes the syntax of different commands that can be used at simulator's command line to perform different operations.

```
load  settings_file
```

This command loads the simulation settings described in the file name specified.

```
unload
```

This command removes the current simulation settings. User can load new simulation settings afterwards.

```
exit
```

Exits from the simulator.

```
run
```

Starts ARQ simulation on the active simulation settings.

Simulation Settings File

A simulation settings file should contain all the following variables. Furthermore, the values of all these variables should be greater than zero. The variable names should have the same spellings and the same case (Title Case). However, the sequence of the variables and the white space (except new line) in the file does not effect.

```
Propagation Delay n1
Processing Delay n2
Bit Rate n3
Data Packet Size n4
Data Packet Overhead n5
ACK Size n6
Simulation Duration n7
Packet Rate n8
```