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 (given in main.cpp), a single instance of this class is created to perform the simulation operations.

```
private:
	Node Source;
	Node Sink;
	double propogationDelay;
	double processingDelay;
	double duration;
	int simulationSetSize;
	double* bitRate;
	double* dataPacketSize;
	double* dataOverhead;
	double* ackSize;

public:
	static char settingsFile[256];
	static char arqMethod;
```

Source

The source node for the simulation.

Sink

The sink node for the simulation.

propogationDelay

The network propagation delay to use in the simulations (in nano seconds).

processingDelay

The node processing delay to use in the simulations (in nano seconds).

duration

The duration of the simulation (in nano seconds).

simulationSetSize

The total size of the simulation set. The simulator provides the option of simulating more than one value of different simulation parameters at a time and this member represents the overall count of simulation runs in such a situation. For example, if two different values of bit rate, three different values of Packet Size and one value each for other parameters is provided in the settings file then simulationSetSize would be six.

bitRate

The transmission bit rate to use in the simulations (number of bits per nano second). Multiple values of bit rate can be provided in the settings file.

dataPacketSize

Size of data packet in bits. Multiple values of packet size can be provided in the settings file.

dataOverhead

Number of overhead bits in data packet. Multiple values of overhead can be provided in the settings file.

ackSize

Size of ACK message in bits. Multiple values of ACK size can be provided in the settings file.

settingsFile[256]

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

arqMethod

The ARQ methods specified in the settings file. This variable is used in a bitwise format to represent the protocols-to-simulate. The right most bit indicates "Stop and Wait", the bit to its left indicates "Go Back N" and the bit to its left indicates "Selective Repeat". For example, if arqMethod has value "6", then "Go Back N" and "Selective Repeat" are to be simulated.

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

readSettings

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

run

Simulates the whole bunch of simulations. This function first decides that which ARQ methods are asked to execute (in the settings file). For each specified method, it executes the whole simulation set by iterating each value of bit rate, packet size, overhead size and ACK size respectively.

simulate

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

verify

Verifies the simulation settings for correctness before starting the simulation.

Node

This class describes a node, source or sink.

```
private:
      char type;
      static double propagationDelay;
      static double processingDelay;
      static double duration;
      static double packetRate;
      static double bitRate;
```

```
      static double dataPacketSize;
      static double dataOverhead;
      static double ackSize;
      static char arqMethod;
      int sequenceNumber;
      PacketQueue PacketBuffer;

public:
      PacketQueue OutgoingPacketQueue;
      PacketQueue IncomingPacketQueue;
```

type

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

propogationDelay

The network propagation delay to use in the simulations (in nano seconds).

processingDelay

The node processing delay to use in the simulations (in nano seconds).

duration

The duration of the simulation (in nano seconds).

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. Since this value is not used anywhere else in the simulator, a user can provide this value as either: the number of packets generated per nano second, or the delay in nano seconds between two packets, and can accordingly use this value in his/her implementation.

bitRate

The transmission bit rate to use in the current simulation run (number of bits per nano second). Before starting a simulation run, the simulator passes one of the bit rate values to the Node.

dataPacketSize

Size of data packet in bits to use in the current simulation run. Before starting a simulation run, the simulator passes one of the packet size values to the Node.

dataOverhead

Number of overhead bits in data packet. Before starting a simulation run, the simulator passes one of the overhead values to the Node.

ackSize

Size of ACK message. Before starting a simulation run, the simulator passes one of the ack size values to the Node.

arqMethod

The ARQ methods being used in the current simulation run. This variable is used in a bitwise format to represent the protocol being used. However, at a time only one bit is on i.e. its value can be 1, 2 or 4 only.

sequenceNumber

The current sequence number of this node. This variable has no specific definition and can be used as per user requirement/logic.

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 and necessary ARQ actions, 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 getPropagationDelay();
      static double getProcessingDelay();
      static double getDuration();
      static double getPacketRate();
      static double getBitRate();
      static double getDataPacketSize();
      static double getDataOverhead();
      static double getACKSize();
      static char getARQMethod();
      void setType(char typ);
      static void setPropagationDelay(double pD);
      static void setProcessingDelay(double pD);
      static void setDuration(double dur);
      static void setPacketRate(double pR);
      static void setBitRate(double bR);
      static void setDataPacketSize(double dPS);
      static void setDataOverhead(double dO);
      static void setACKSize(double aS);
      static void setARQMethod(char aM);
      int generatePackets();
      int sendPackets();
```

Get/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 accordingly.

Packet

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

```
private:
      char type;
      char source;
      char destination;
      char transmissionStatus;
      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 (type='1') or sink (type='2').

destination

The node for which this packet is destined, either source (type='1') or sink (type='2').

transmissionStatus

Indicates whether this packet is being transmitted (in transmission). The user should set this value as soon a packet is moved from the PacketBuffer to OutgoingPacketQueue. Once the transmission is completed, the simulator automatically resets this value before putting the packet in the destination node's IncomingPacketQueue.

sequenceNumber

The sequence number of this packet. This variable has no specific definition and can be used as per user requirement/logic.

size

The size of this packet in bits.

overhead

The overhead of this packet. In case of ACK and NACK, it should be set to zero (the default value).

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();
      char getTransmissionStatus();
      int getSequenceNumber();
```

```
      double getSize();
      double getOverhead();
      double getCreationTime();
      double getReadyToSendTime();
      void setType(char typ);
      void setSource(char src);
      void setDestination(char dst);
      void setTransmissionStatus(char trSt);
      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 features necessary for different types of packet buffers used in this simulator.

```
private:
      int length;
      Packet** Packets;
```

length

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

Packets

The array of stored packets.

```
public:
      int getLength();
      double getCreationTime(int index);
      double getReadyToSendTime(int index);
      double getSize(int index);
      void setTransmissionStatus(int index, char trSt);
      Packet* dequeuePacket(int index);
      void enqueuePacket(Packet* pkt);
```

Get functions for different attributes/parameters.

enqueuePacket

Adds a packet to this queue/buffer.

dequeuePacket

Removes the specified packet from this queue.

Time

This is a utility class to keep the 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.

```
ARQ Methods c1 c2 c3
Bit Rate n11 n12 n13 ……
Data Packet Size n21 n22 n23 ……
Data Packet Overhead n31 n32 n33 ……
ACK Size n41 n42 n43 ……
Simulation Duration n5
Packet Rate n6
```

c1

Can be 0 or 1. Indicates whether Selective Repeat method should be simulated or not.

c2

Can be 0 or 1. Indicates whether Go Back N method should be simulated or not.

c3

Can be 0 or 1. Indicates whether Stop and Wait method should be simulated or not. At least one of the c1, c2 or c3 should be 1.

n11, n12, n13, n6

A non zero, positive whole or decimal number.

n21, n22, n23, n31, n32, n33, n41, n42, n43, n5

A non zero, positive whole number.