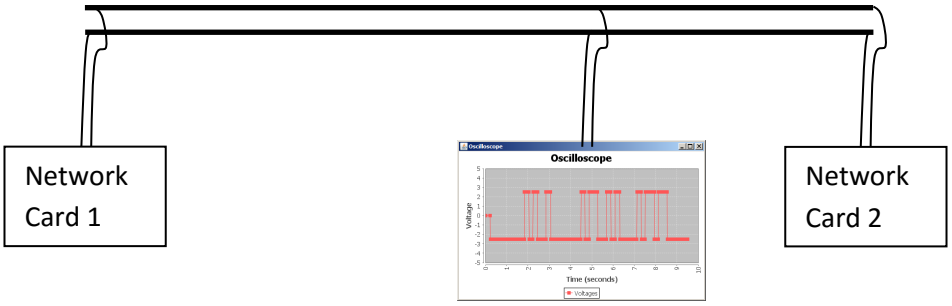
COMP201P Written Coursework: Using a Stop & Wait ARQ protocol to send data frames reliably across a noisy wire.

The use of twisted wire pairs is a dominant communication media used between network cards in a wired local area network such as wired Ethernet. In this written coursework we will simulate a twisted pair of wires connecting two network cards together and also an oscilloscope to monitor voltage levels on the wire over time as data frames are sent.

On the Moodle site you will find a zip archive named COMP201P_Coursework.zip. This contains only the Java source files and libraries for a 'framework' for modelling this network link (thus you will need to incorporate these into your favourite IDE environment to work on this project). This framework using JFreeChart and so the appropriate jar library files (contained in the lib directory) need to be linked to the framework for it to work.

The following classes are provided:

Main.java This contains the main method which creates a shared twisted wire pair and then creates 2 network cards which are connected to the same shared wire. An oscilloscope is also created and joined to the wire so that voltages on the wire can be monitored over time. (See schematic below for the overall setup.)



MyTwistedWirePair.java This is currently a 'stub class' that models a wire that allows multiple devices to 'tap' into it (i.e. connect to it). It implements the **TwistedWirePair interface** which essentially lets a device set a voltage on the wire or get the overall voltage on the wire. The general model is that different devices are connected at different locations along the wire – and the wire will take the total voltage set by all devices currently to the wire. (So if 3 network cards are connected with voltages +2.5v, +2.5v and +2.5v then the overall total voltage on the wire will be (2.5 + 2.5 + 2.5) = 7.5 volts. Also this total voltage will be propagated instantaneously across the entire wire (so no propagation delay). This is clearly just an approximate model of the behaviour of a real wire.

DataFrame.java This is a class which encapsulates the data within a data frame. It simply encapsulates a 'payload' for the data frame consisting of a byte array. (So it currently does not handle such things as source/destination addresses or error correction codes.) ***You need to extend this DataFrame class to include required header information.***

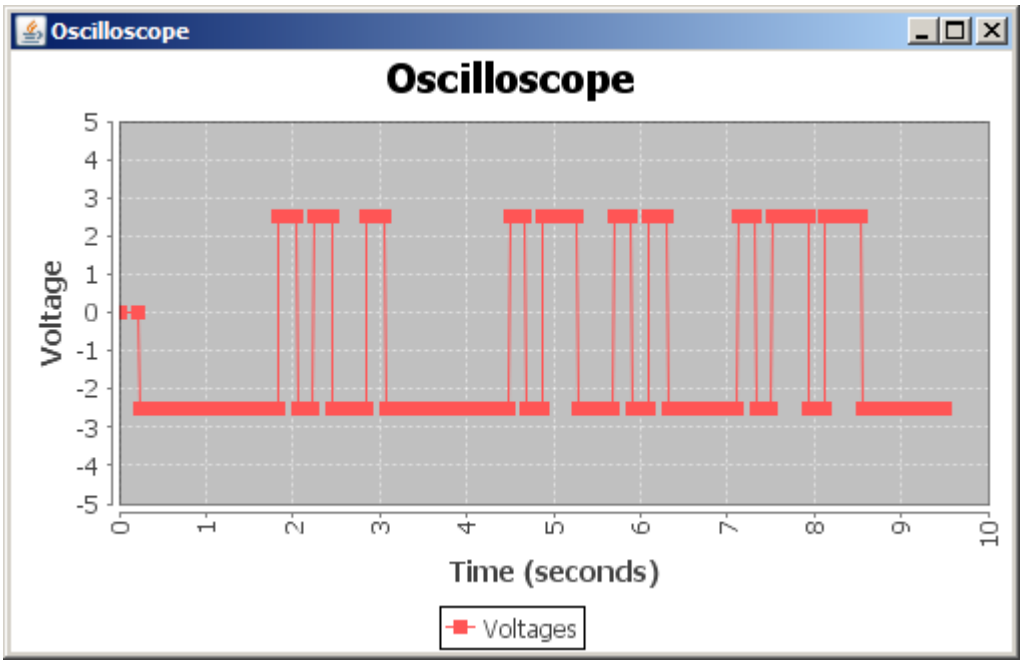
NetworkCard.java The constructor of a NetworkCard is given a unique device name to identify the device, and it is also given a reference to the shared TwistedWirePair object which it will use to transmit data. There is a `send` method which is used to send a data frame across the wire – it will just add the data frame to an output queue and will only block if that output queue is at capacity. There is also a `receive` method which will return the next data frame contained in the input queue – it will block if the input queue is currently empty. The NetworkCard contains a TXThread that transmit data frames from the output queue onto the wire using an asynchronous byte protocol, with data frame delimited by sentinels and byte stuffing being employed. The NetworkCard also contains an RXThread which receives bytes from the wire (using the same asynchronous protocol) and constructs data frames from these bytes which it then adds to the input queue.
You need to extend this NetworkCard class to include required behaviour.

The output given below shows that data frames can be successfully transmitted by this current system. **So what do you need to do?** Well, another thread exists called ThermalNoise that can add noise to the wire. The level of noise is currently set to 0.0 volts and data can be transmitted successfully. But if you set the noise level to 3.5 volts then bit errors occur and data frames often get corrupted (see the second output below). Your key task is to add a Stop & Wait ARQ protocol to the network card so that it can do reliable transmission of network frames over the noisy wire. This requires a number of extensions:

1. Currently both network cards (1 and 2) receive data frames being transmitted across the wire. So Network Card 1 actually receives bytes that it is actually sending across the wire! (And it will be adding data frames to its input queue, even though it is sending these!) For the ARQ protocol to successfully be used across this shared wire, the data frames need to have some form of destination address added to the header information. Thus the DataFrame class needs to be extended to include this destination address information which the receiver would use to determine if it is the destination of the particular data frames. It should not process the data frame if it is not the destination.
2. The ARQ protocol also requires a checksum to be added to the system. **You should implement the Internet Checksum in your own Java code by doing 1s-complement addition of 16-bit values followed by taking the 1s-complement of the total sum (i.e. without using a standard Java library function for such functionality)**. This should calculate an appropriate checksum for both the header information and the payload data of the data frame (in a similar manner to IP). The receiver should then use this checksum to determine if the frame has been corrupted during transit.
3. The ARQ protocol also requires a few other additional header elements that should also be added to the current DataFrame class, with these additional header bytes also transmitted in the data frame bytes.
4. The TXThread and RXThread should then be extended to include the ARQ protocol including the required acknowledgment frames (ACKs) and timeouts. **Note that you should keep both existing TX and RX threads in the network card class – these threads will need to interact to accomplish the ARQ protocol to produce reliable transmission.**

An overall approach might be to:

1) Look through the current code to ensure that you know how it works. Look at the output oscilloscope with the transmit/receive code to see how it is implementing the asynchronous transmission of data bytes with frames delimited with sentinel values.



2) Design new header elements for the DataFrame class and add appropriate functionality to allow these elements to be set (and tested). The byte array being transmitted should include these additional header bytes as well as the payload.
3) See if you can get Network Card 2 to only receive the transmitted data frames by employing a destination address.
4) See if you can get the receiver to detect when the data frame has been corrupted.
5) Then try to implement the full ARQ Stop & Wait protocol using destination addresses and the Internet checksum to determine when a data frame has been corrupted.

You should only modify the "DataFrame.java" and "NetworkCard.java" classes and you should NOT change any of the other files in the framework (except for the Main.java class that you may wish to change for testing purposes, but keep the same interfaces to the network cards). You will only finally submit the DataFrame.java and NetworkCard.java files; these should work within the original framework as given to you.

Remember it is a Java concurrency exercise so try to think about those multiple threads within the system and what is required in terms of concurrency to make everything work safely.

Please use the discussion board to ask question about what is required and discuss any problems – since it's good to talk about things.

Example of output from the current system with noise set to 0.0 volts

```
 *** SENDING DATA FRAME: Hello World


 *** SENDING DATA FRAME: Earth calling Mars


 *** SENDING DATA FRAME: Hello Mars

NetCard2 RECEIVED BYTE = 48
NetCard1 RECEIVED BYTE = 48
NetCard2 RECEIVED BYTE = 65
NetCard1 RECEIVED BYTE = 65
NetCard2 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 6f
NetCard1 RECEIVED BYTE = 6f
NetCard1 RECEIVED BYTE = 20
NetCard2 RECEIVED BYTE = 20
NetCard2 RECEIVED BYTE = 57
NetCard1 RECEIVED BYTE = 57
NetCard1 RECEIVED BYTE = 6f
NetCard2 RECEIVED BYTE = 6f
NetCard2 RECEIVED BYTE = 72
NetCard1 RECEIVED BYTE = 72
NetCard2 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 64
NetCard2 RECEIVED BYTE = 64
NetCard1 RECEIVED BYTE = 7e
NetCard2 RECEIVED BYTE = 7e

 *** RECEIVED: Hello World

NetCard1 RECEIVED BYTE = 45
NetCard2 RECEIVED BYTE = 45
NetCard2 RECEIVED BYTE = 61
NetCard1 RECEIVED BYTE = 61
NetCard1 RECEIVED BYTE = 72
```

```
NetCard2 RECEIVED BYTE = 72
NetCard1 RECEIVED BYTE = 74
NetCard2 RECEIVED BYTE = 74
NetCard2 RECEIVED BYTE = 68
NetCard1 RECEIVED BYTE = 68
NetCard1 RECEIVED BYTE = 20
NetCard2 RECEIVED BYTE = 20
NetCard2 RECEIVED BYTE = 63
NetCard1 RECEIVED BYTE = 63
NetCard1 RECEIVED BYTE = 61
NetCard2 RECEIVED BYTE = 61
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 69
NetCard1 RECEIVED BYTE = 69
NetCard1 RECEIVED BYTE = 6e
NetCard2 RECEIVED BYTE = 6e
NetCard2 RECEIVED BYTE = 67
NetCard1 RECEIVED BYTE = 67
NetCard1 RECEIVED BYTE = 20
NetCard2 RECEIVED BYTE = 20
NetCard1 RECEIVED BYTE = 4d
NetCard2 RECEIVED BYTE = 4d
NetCard1 RECEIVED BYTE = 61
NetCard2 RECEIVED BYTE = 61
NetCard1 RECEIVED BYTE = 72
NetCard2 RECEIVED BYTE = 72
NetCard1 RECEIVED BYTE = 73
NetCard2 RECEIVED BYTE = 73
NetCard2 RECEIVED BYTE = 7e
NetCard1 RECEIVED BYTE = 7e

 *** RECEIVED: Earth calling Mars

NetCard2 RECEIVED BYTE = 48
NetCard1 RECEIVED BYTE = 48
NetCard1 RECEIVED BYTE = 65
NetCard2 RECEIVED BYTE = 65
NetCard2 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 6c
```

```
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 6f
NetCard2 RECEIVED BYTE = 6f
NetCard2 RECEIVED BYTE = 20
NetCard1 RECEIVED BYTE = 20
NetCard2 RECEIVED BYTE = 4d
NetCard1 RECEIVED BYTE = 4d
NetCard2 RECEIVED BYTE = 61
NetCard1 RECEIVED BYTE = 61
NetCard2 RECEIVED BYTE = 72
NetCard1 RECEIVED BYTE = 72
NetCard2 RECEIVED BYTE = 73
NetCard1 RECEIVED BYTE = 73
NetCard1 RECEIVED BYTE = 7e
NetCard2 RECEIVED BYTE = 7e

 *** RECEIVED: Hello Mars
```

Example of output from the current system with noise set to 3.5 volts

```
 *** SENDING DATA FRAME: Hello World


 *** SENDING DATA FRAME: Earth calling Mars


 *** SENDING DATA FRAME: Hello Mars

NetCard2 RECEIVED BYTE = 40
NetCard1 RECEIVED BYTE = 40
NetCard2 RECEIVED BYTE = 48
NetCard1 RECEIVED BYTE = 48
NetCard1 RECEIVED BYTE = 65
NetCard2 RECEIVED BYTE = 65
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 6f
```

```
NetCard1 RECEIVED BYTE = 6f
NetCard1 RECEIVED BYTE = 0
NetCard2 RECEIVED BYTE = 20
NetCard1 RECEIVED BYTE = 57
NetCard2 RECEIVED BYTE = 57
NetCard1 RECEIVED BYTE = 6f
NetCard2 RECEIVED BYTE = 6f
NetCard1 RECEIVED BYTE = 72
NetCard2 RECEIVED BYTE = 72
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 64
NetCard2 RECEIVED BYTE = 64
NetCard1 RECEIVED BYTE = 7e
NetCard2 RECEIVED BYTE = 7e

 *** RECEIVED: @Hello World

NetCard2 RECEIVED BYTE = 45
NetCard1 RECEIVED BYTE = 45
NetCard1 RECEIVED BYTE = 61
NetCard2 RECEIVED BYTE = 61
NetCard1 RECEIVED BYTE = 72
NetCard2 RECEIVED BYTE = 72
NetCard1 RECEIVED BYTE = 74
NetCard2 RECEIVED BYTE = 74
NetCard1 RECEIVED BYTE = 68
NetCard2 RECEIVED BYTE = 68
NetCard1 RECEIVED BYTE = 20
NetCard2 RECEIVED BYTE = 20
NetCard1 RECEIVED BYTE = 63
NetCard2 RECEIVED BYTE = 63
NetCard1 RECEIVED BYTE = 61
NetCard2 RECEIVED BYTE = 61
NetCard2 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 69
NetCard1 RECEIVED BYTE = 69
NetCard2 RECEIVED BYTE = 6e
NetCard1 RECEIVED BYTE = 6e
```

```
NetCard2 RECEIVED BYTE = 47
NetCard1 RECEIVED BYTE = 67
NetCard2 RECEIVED BYTE = 20
NetCard1 RECEIVED BYTE = 20
NetCard2 RECEIVED BYTE = 4d
NetCard1 RECEIVED BYTE = 4d
NetCard2 RECEIVED BYTE = 61
NetCard1 RECEIVED BYTE = 61
NetCard1 RECEIVED BYTE = 72
NetCard2 RECEIVED BYTE = 62
NetCard1 RECEIVED BYTE = 73
NetCard2 RECEIVED BYTE = 73
NetCard2 RECEIVED BYTE = 7e

 *** RECEIVED: Earth callinG Mabs

NetCard1 RECEIVED BYTE = 7e
NetCard2 RECEIVED BYTE = 48
NetCard1 RECEIVED BYTE = 48
NetCard1 RECEIVED BYTE = 65
NetCard2 RECEIVED BYTE = 65
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 4c
NetCard1 RECEIVED BYTE = 6c
NetCard2 RECEIVED BYTE = 6c
NetCard1 RECEIVED BYTE = 6f
NetCard2 RECEIVED BYTE = 6f
NetCard2 RECEIVED BYTE = 20
NetCard1 RECEIVED BYTE = 20
NetCard2 RECEIVED BYTE = 4d
NetCard1 RECEIVED BYTE = 4d
NetCard2 RECEIVED BYTE = 61
NetCard1 RECEIVED BYTE = 61
NetCard2 RECEIVED BYTE = 72
NetCard1 RECEIVED BYTE = 72
NetCard2 RECEIVED BYTE = 73
NetCard1 RECEIVED BYTE = 73
NetCard1 RECEIVED BYTE = 7c
NetCard2 RECEIVED BYTE = 7e

 *** RECEIVED: HeLlo Mars
```