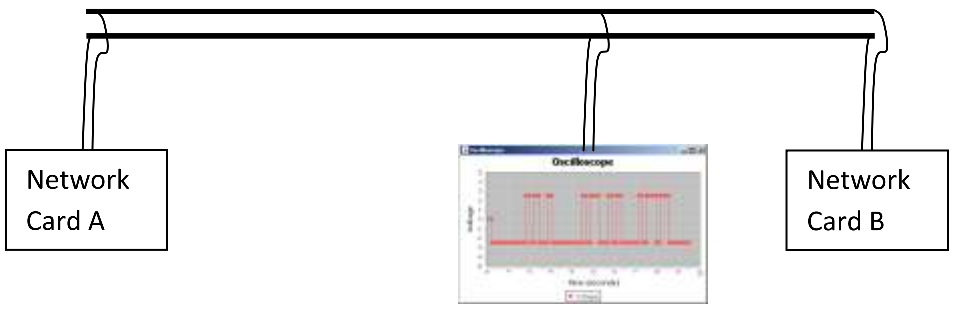
COMP2007 Coursework 1: Sending a data frame across a wire

Twisted pairs of wires provides the dominant way of connecting network cards together to form a computer network. In this first coursework we will model a twisted pair of wires connecting two network cards together and an oscilloscope.

On the Moodle site you will find a zip archive named COMP2007_CW1.zip. This contains only the Java source files and libraries for a 'framework' for modelling this computer 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. Network Card A is then told to sent a data frame (containing "Hello World") and a data frame listener is registered with Network Card B so that it prints out the data frame contents when it receives it. Your key task is to implement the sending and receiving of the data frame bytes by setting voltages over time across the wire. You should refer to the lecture notes for different ways of sending a data frame as voltages across a wire – you are free to use whatever protocol you wish to transfer the bytes across the wire (clearly you can only change the voltage over the wire to accomplish this). A rough schematic of this set-up is given below.



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.

You need to implement the correct behaviour for this MyWirePair class.

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.)

MyFrameListener.java This is a class which implements the FrameListener interface and simply prints out “RECEIVED DATA FRAME:” followed by the contents of the data frame received. It is used when setting up Network Card B so that it will print out the contents of any data frames received.

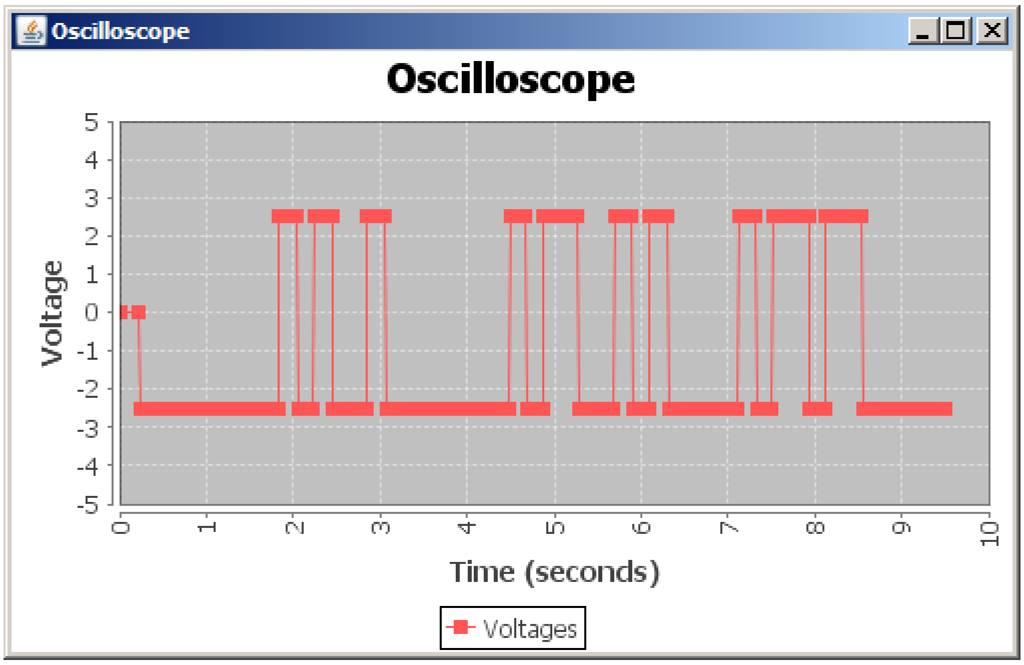
NetworkCard.java This is also currently a ‘stub class’ that currently does not have any real behaviour. The constructor of a NetworkCard is given a unique device name to identify the device, and it is also given a reference to the shared TwistedWirePair object which it will use to transmit data. It is also given a reference to a DataFrameListener which it will use to tell the computer about any data frames it receives (if the listener is set). There is a `send` method which is used to send a data frame across the wire – it should send all the data across the wire before returning from this method. When the network card is started as a thread and should start listening to the network for any data frames received (assuming the data frame listener is not null). The protocol used to send/receive the data frame over the network link (wire) is left entirely flexible – you can experiment with different protocols to see how effective they are. Some general voltages are set as default at the top of the network card (+2.5 volts for high voltage signal and -2.5 volts for low voltage signal). Also a ‘pulse period’ of 200 milliseconds is set at the top of the network card class – you should be able to send data signals at this rate.

You need to implement the correct behaviour for this NetworkCard class.

An overall approach might be to:

1) Implement the wire so that voltages can be set on the wire and these voltages can be detected using the oscilloscope.

2) Implement the `send()` method functionality so that data frames can be sent across the wire. An example output of the oscilloscope when sending a data frame may be this (although it should be pointed out that you may use a different protocol from this):



3) Implement the receive functionality for the network card so that it can receive and interpret data frames which are transmitted. It should use the registered data frame listener when it has received a data frame (which then prints out the data frame received to the console).

Example output is given on the following page.

You should only implement the "MyTwistedPair.java" and "NetworkCard.java" classes and you should NOT change any of the other files in the framework.

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 (but try not to simply put up the solution on the discussion board / Facebook !!!)

Note that the ***deadline*** is Sun. 17th Nov. 2013 at 11:55pm. Ok – good luck!

Example output for the system:

Note that the “SENDING DATA FRAME” and “RECEIVED DATA FRAME” lines are printed by the framework and should be present in your output. The other lines in this output are really just shown for demonstration purposes and are being printed by the network card – your output may be very different from this depending on what you print out and the link layer protocol / physical layer protocols you use. So this output is for demonstration purposes and your output may be very different (apart from the “SENDING DATA FRAME” and “RECEIVED DATA FRAME” lines which should be present).

```
WAITING ...
SENDING DATA FRAME: Hello World
WAITING TO RECEIVE DATA FRAMES
RECEIVED BYTE = 48
WAITING ...
RECEIVED BYTE = 65
WAITING ...
RECEIVED BYTE = 6c
WAITING ...
RECEIVED BYTE = 6c
WAITING ...
RECEIVED BYTE = 6f
WAITING ...
RECEIVED BYTE = 20
WAITING ...
RECEIVED BYTE = 57
WAITING ...
RECEIVED BYTE = 6f
WAITING ...
RECEIVED BYTE = 72
WAITING ...
RECEIVED BYTE = 6c
WAITING ...
RECEIVED BYTE = 64
WAITING ...
RECEIVED BYTE = 7e
RECEIVED DATA FRAME: "Hello World"
WAITING ...
```