**Killian Ronan – Student Number : 18328687**

**Telecommunications Assignment 1 : Command & Control**

**My Approach:**

The approach I took to begin my assignment was to dive into the sample code provided from previous projects. It was a tough process to begin with as reading other people’s code can be harder to interpret than reading your own. I decided to break each class down and researched what each path in every method would accomplish in order to get greater understanding of the flow of the application. As I inspected each of the member variable it became clearer what was actually happening. I then chose my starting point to be the `Sample Code with byte arrays' that was given to us. I chose this as I thought it was the clearest implementation to visualise at least for me.

From this the two main classes I used were the extended Node classes: Client and Server, I made changes where necessary.

**Classes:**

Worker.java:

The source port for worker [0] is 50000. This increments based on the index of the worker i.e worker at index [2] = port 50000 + 2, worker at index [3] = port 50000 + 3.

The Brokers port number is included also; 49998.

The constructor initializes the terminal, destination address, broker socket and starts the listener.

Functions:

onReceipt(): This function works similarly to the Broker onReceipt function and will be discussed later.

sendMessage(): This function takes 2 parameters.

int index,int type

The index is the index of the worker send the message. It is sent in the header of the packer. The data is packetized and sent to the Broker based on the type of message passed in the parameter. This can either be a result or a request for work. I will talk of this function later.

Main(): The worker list is created here with the use of the imported class java.util.Scanner. Each worker enters their name and is assigned a port number and initialized. The sendMessage() function is called with the parameters x and the worker index. This sends the volunteer for work request.

Broker.java:

The source port for Broker is 49998. The C&C port number and the Worker[0] port number are included which are 49999 and 50000 respectively.

The constructor works the same as the Worker class constructor.

onReceipt():

This function accepts a datagram packet as a parameter. There are 3 main variables declared:

String content;

byte[] data;

byte[] buffer;

The packet is depacketized based on what TYPE\_POS it has assigned through the use of a switch statement. The data variable holds the data of the packet and it is copied into the buffer variable and then converted to a string. An acknowledhement packet is sent back indicating the packet was sent successfully. This response is also sent to the C&C to alert it that workers are ready. This works different for acknowledgements received which calls the notify() function and prints a ‘received ack’ message. I will talk of the other routes in the Design and Implementation section.

sendMessage(): This works similarly to the Worker sendMessage function. However it uses a for loop to loop through all of the workers and send the message to each available one. It then makes the workers who were sent the work description unavailable by setting the value at their index in the availableWorkers array to 0.

Main(): The terminal is created and passed when the Broker class is initialized. The start function is then called which means the Broker will wait for contact of a packet.

This is put in an endless loop as we want the Broker to constantly look for packers of data to forward.

C&C.java:

The source port for C&C is 49999. The Brokers port number is included also; 49998.

The constructor works the same as the Worker class constructor.

onReceipt(): This works with a switch statement similar to the two above but accepts only acknowledements. Once it receieves an acknowledgement that workers are ready, the sendMessage() function is called which allows the C&C to send a work description through the terminal input line. This only occurs however when the index passed is not equal to -1. This is to make sure when the C&C receives acknowledgements that his work instructions has been delivered, that it will not be prompted to input another until a worker requests one. This will be discussed in further detail later.

sendMessage(): This works the same as above, packetizes the inputted work description and sends it to the Broker to be forwarded to the available workers.

Main(): Here the C&C class is initialized with a terminal, same as in the Broker class.

Common to all above:

The host name is “localhost”, which means ‘this computer’. We use localhost because the protocol operates across only the computer using it.

static final String ***DEFAULT\_DST\_NODE*** = "localhost"; // Name of the host for the server

static final int ***HEADER\_LENGTH*** = 3; // Fixed length of the header

static final int ***TYPE\_POS*** = 0; // Position of the type within the header

static final int ***LENGTH\_POS*** = 1;

static final int ***INDEX*** = 2;

static final byte ***TYPE\_UNKNOWN*** = 0;

static final byte ***TYPE\_STRING*** = 1; // Indicating a string payload

static final byte ***TYPE\_ACK*** = 2; // Indicating an acknowledgement

static final byte ***TYPE\_WORK*** = 3; // Indicating an acknowledgement

static final byte ***TYPE\_RESULT*** = 4; // Indicating a string payload

static final int ***ACKCODE\_POS*** = 1; // Position of the acknowledgement type in the header

static final byte ***ACK\_ALLOK*** = 10; // Inidcating that everything is ok

Terminal.java:

This is a class which I used from the sample code with byte array. It is a simple terminal to use.

Node.java:

This was also a class from the same sample code. Each of the C&C, Broker and Worker classes extend this class.

**Design and Implementation:**

Upon startup of the C&C, I create an instance of the C&C class which opens its terminal. It then awaits to receive an acknowledgement from the Broker that workers are available to be sent a work description. In its constructor the ‘listener.go()’ method is called which alerts that the C&C socket has been initialized and meaning the port is ready to receive data. The port number chosen to represent the C&C class was 49999. This listener class loops constantly attempting to receive a packet and notify users with acknowledgements, through the ‘onReceipt()’ function. The C&C would only ever be contacting the Broker and so I did not include the workers port number as a final member variable, only the C&C and Broker. The same applied for the worker class. As the broker is the medium between the two there is only a need to include the 2 connected ports of each.

The broker on startup calls the start() method which allows it to await workers to come online and submit their availability for work. The constructor for this class was similar to the C&C, however the destination port was not set as it will be set based on which direction messages are being forwarded i.e:

C&C->Broker->Workers - localhost**:***workerport* ***or*** Workers->Broker->C&C - localhost**:***C&Cport.*

The broker however only sends an acknowledgement to the C&C, which triggers the ‘OnReceipt()’ function in the C&C. With the TYPE\_POS indicating the type of message being an acknowledgement because of the TYPE\_ACK being set, the code follows this path.

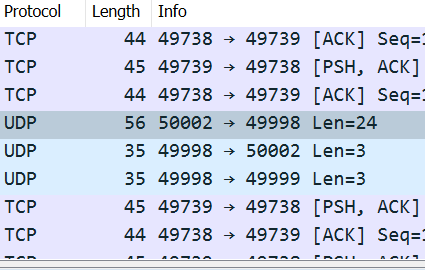
Once the Worker class is run, each worker accepts a name as input, through the use of the Eclipse console. With each inputted name a new worker is created, incrementing the source port number after each instantiation so no two workers have the same port number. I decided to add the source port number to the names of each worker so as to get a clearer view of each route created.

1. **Workers that accept a name as input, send a message that they are volunteering for work to a broker and print work assignments that have been forwarded by a broker.**

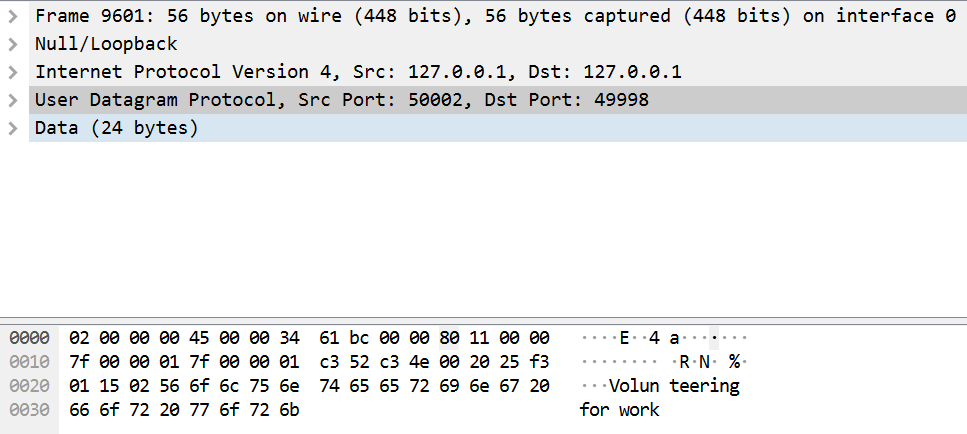
In order for the workers to accept a name as input, my first plan of action was to import the Scanner class and use the eclipse console to input the name of each worker. Then I would initialize the worker class passing the inputted name as a parameter for the terminal name. I decided to add the source port number to each of the workers so as to get a clearer view of each worker individually.

On initialization each worker then calls the ‘sendMessage()’ method passing their index and the value ‘0’. The ‘0’ indicated that it is a volunteer for work request and is needed to differentiate between that and a work results message. The message “Volunteering for work” is packaged into a DatagramPacket and sent along with the index of the worker to the Broker.

**Example packet:**

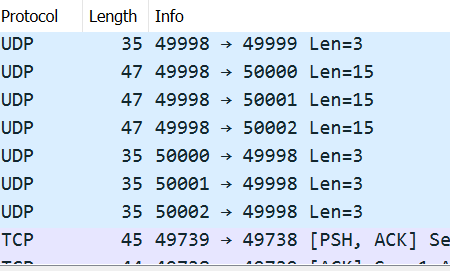


Above and below is an example packet of Length 24. The two following packets are the acknowledgements sent from the Broker to the worker indicating it has received the work request, the other is to tell the C&C that a worker is ready for work.

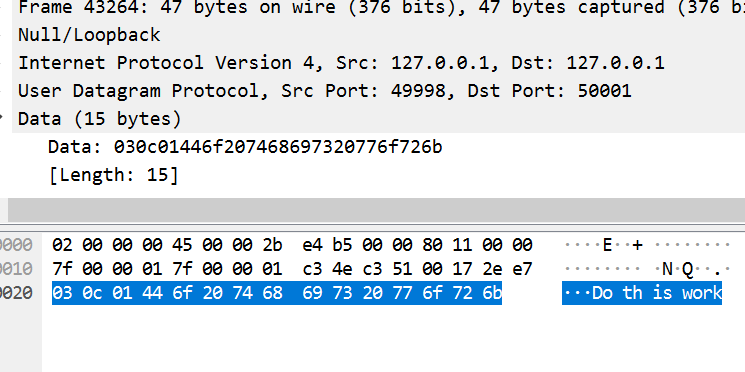


When the broker receives a work instruction from the C&C it forwards it to each of the available workers. This calls the ‘onReceipt()’ method of the workers and the code flows through the switch statement down the TYPE\_WORK path, which indicates it is a work description being sent. The packet is deconstructed, and the message is printed to the terminal of the worker.

In the screenshot below the first packet is an acknowledgement of the work description being sent from the C&C to the Broker. The following packets are the broker forwarding the message on to the 3 available workers.



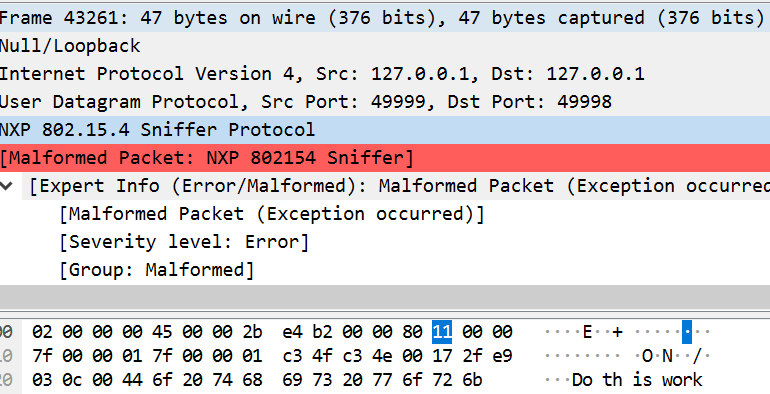
**Detail view of forwarded packet:**



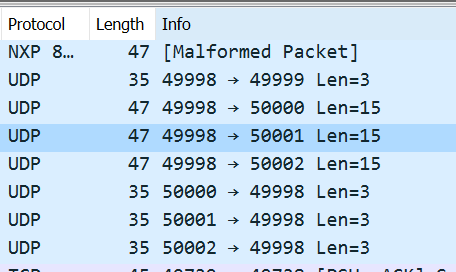
2. **C&C server that accepts a work description as input and transmits a message with the work description to a broker.**

When an acknowledgement is sent to the C&C, it triggers the ‘OnReceipt()’ function. I used this function to call the ‘sendMessage()’ method which allows for the input of a work description to be sent through the C&C terminal. This input box however only appears when the broker sends the acknowledgement that workers are ready for work. The datagram packet is created with TYPE\_POS being set to TYPE\_WORK indicating it is a work instruction being sent. The input is read from the terminal and added to the packet also. The destination port is set to the Broker and the package is sent to be forwarded.

My implementation was the same to all my other uses of sendMessage(). However I encountered a problem which I could not find the answer to, in the form of a Malformed packet. Whenever the packet from the C&C to the Broker containing the work description was sent, it would appear malformed in WireShark. I was unable to solve this problem. The forwarding from the Broker to the workers works as if the packet is fine however.



**Malformed initial packet + Forwarded packets:**

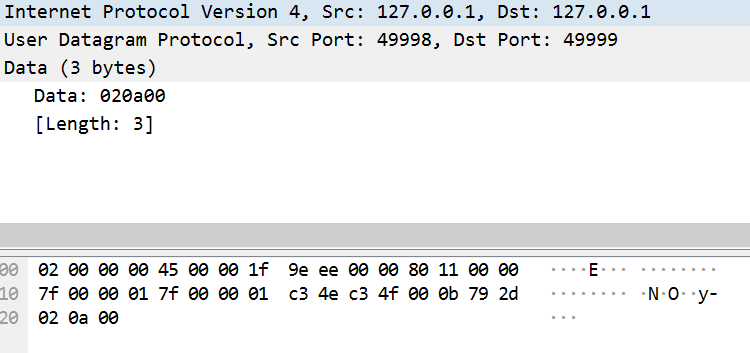


3. **A broker that receives messages from workers, maintains lists of available workers and forwards incoming work descriptions to the workers.**

As I specified before, on startup the broker calls the start() function. It waits until another thread invokes the ‘notify()’ method which which wakes the thread from waiting. This means my broker class awaits contact from the workers requesting work. When it does receive a package, the ‘onReceipt()’ method is called, and with TYPE\_POS in the packet being of TYPE\_STRING it sends the packages through the switch statement down TYPE\_STRING route. As each worker calls this method in the same manner, it is here that I set the ports of the available workers array with each call.

After sending the acknowledgement response to each worker, informing them that the broker has received their request for work, I set the INDEX part of the header and resend the acknowledgement to the C&C this time. I do this to inform the C&C that workers are now ready for work. Setting the INDEX part of the header specifies for the C&C whether this is an acknowledgement of workers ready or their inputted work instruction was successfully sent.

**Workers ready acknowledgement:**

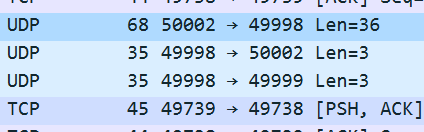


Once the C&C application sends the work description to the Broker, again the ‘onReceipt()’ method is invoked. The TYPE\_POS in the header this time specifies it is of TYPE\_WORK meaning that it is a work description being sent and so it follows the relevant path through the switch statement. An acknowledgement is sent back to the C&C letting it know it has received the packet. The broker then calls the ‘sendMessage()’ function passing it the content of the packet in String form. It is then repackaged again as the code enters a for loop which loops through all worker ports and and sends the package to each available one, before setting them to unvavailable once they have been forwarded the work instruction. In section 1 there is a snippet of the forwarded packets.

4. **Workers may choose to reply to the broker with results from work they carried out or withdraw their availability.**

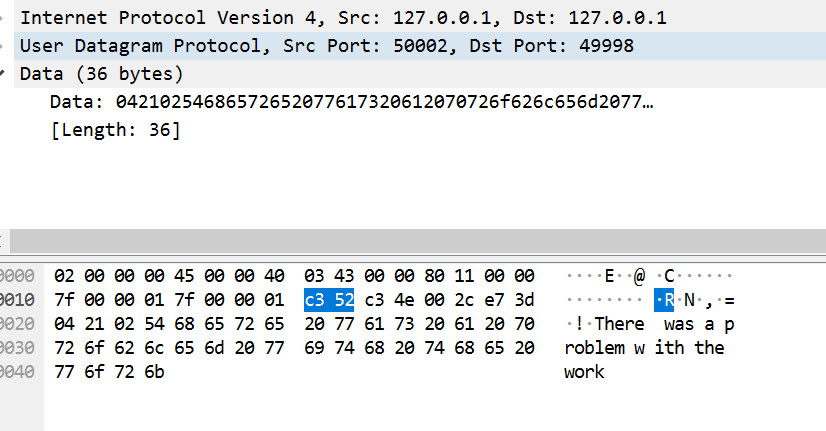
In order for the workers to be able to reply with results from their work, I had to adjust my ‘sendMessage()’ function and add a switch statement for differentiating between sending a result or a volunteer request for work. I used TYPE\_RESULT to specify that a result was being sent. In order to implement this I called the ‘sendMessage()’ function from my ‘onReceipt()’ function, so that whenever a work instruction is received, the worker can then reply to it. I gave the ability for the worker to enter ‘quit’ and withdraw from work. I did this using a simple String compare.

**Example packet:**



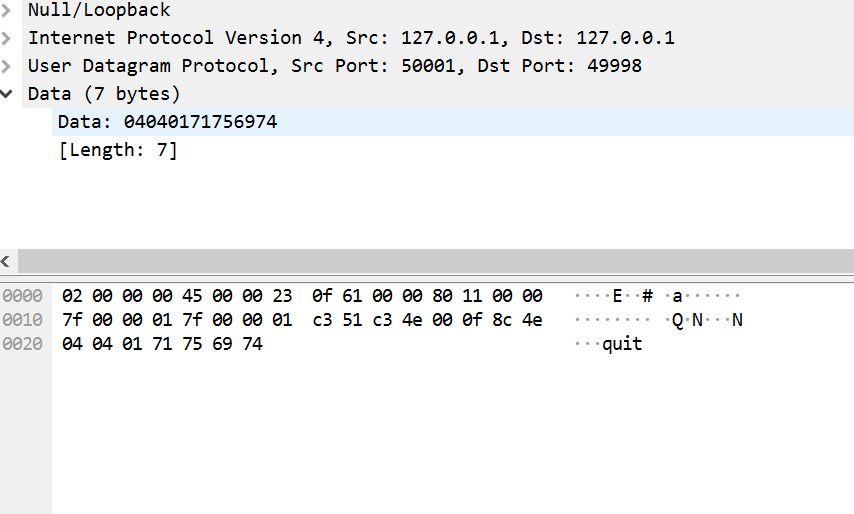
Above shows the packet being sent from the worker to broker of length 36, containing the result message. The first acknowledgement is an acknowledgement that the broker has received the result, the second is an acknowledgement to inform the C&C a worker is ready.

Below is a detail view of the result packet.

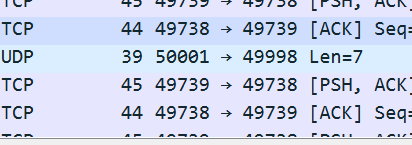


The broker ‘onReceipt()’ method would then retrieve the sent package, in the TYPE\_RESULT path of its switch statement. I then re-add the worker to the list of available workers, send an acknowledgement to the worker that the result has been successfully received.

**Example of worker quitting:**

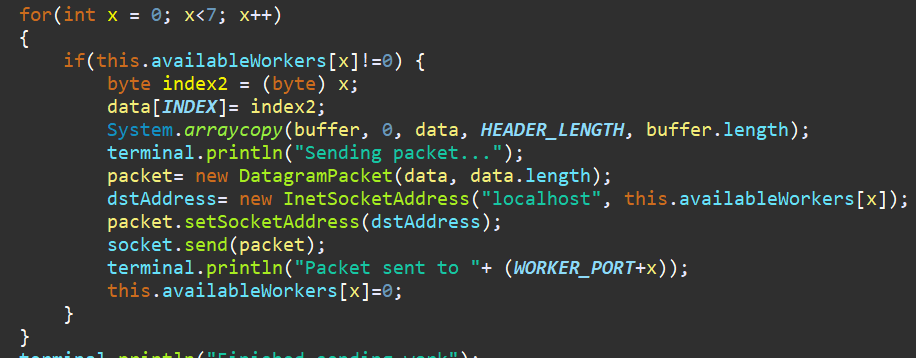
****

As can be seen below there is no acknowledgements sent to the C&C that a worker is ready as the worker has withdrawn by inputting ‘quit’ as a result.

****

5. **Work descriptions from a C&C application may request the same work description to be sent to one, a number of workers or all workers.**

In my implementation, I did not allow the C&C to specify the number of workers a description would be sent to. Each description is sent to all available workers at that time. If there is one available, it will only send it to one. This is implemented in the for loop within the Broker class.



6. **The broker and the workers may implement acknowledgements and the C&C may wait for acknowledgements from a broker before proceeding to accept input of another work description.**

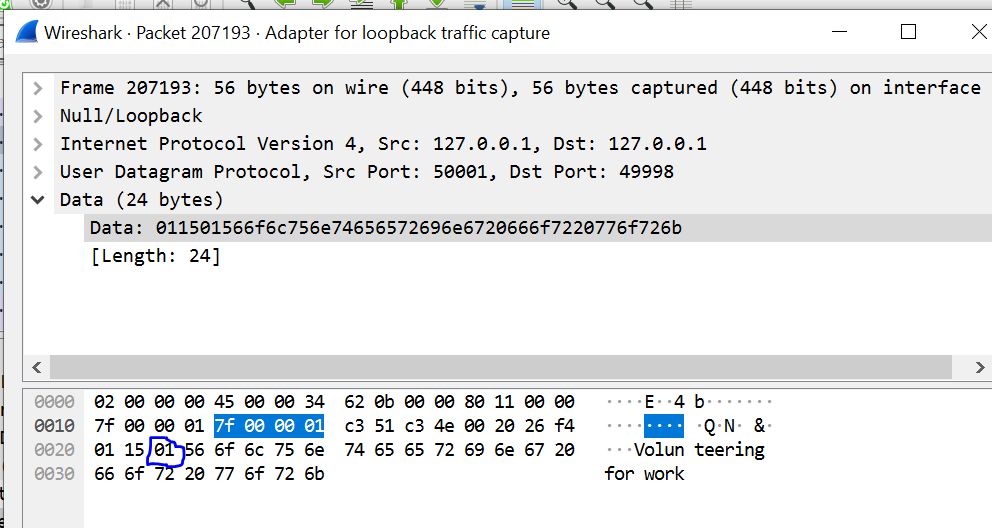
An acknowledgement is sent to the C&C with an index parameter whenever a worker requests work or submits their results without withdrawing. This triggers the ‘OnReceipt()’ function where it checks the passed index to see if it should allow input of a work description. This means it will not allow input of a work description unless there are workers ready. Acknowledgements are sent when the Broker receives work requests from the Worker, when the Broker receives work descriptions from the C&C and when results are sent by the Worker.

**Packet Content:**

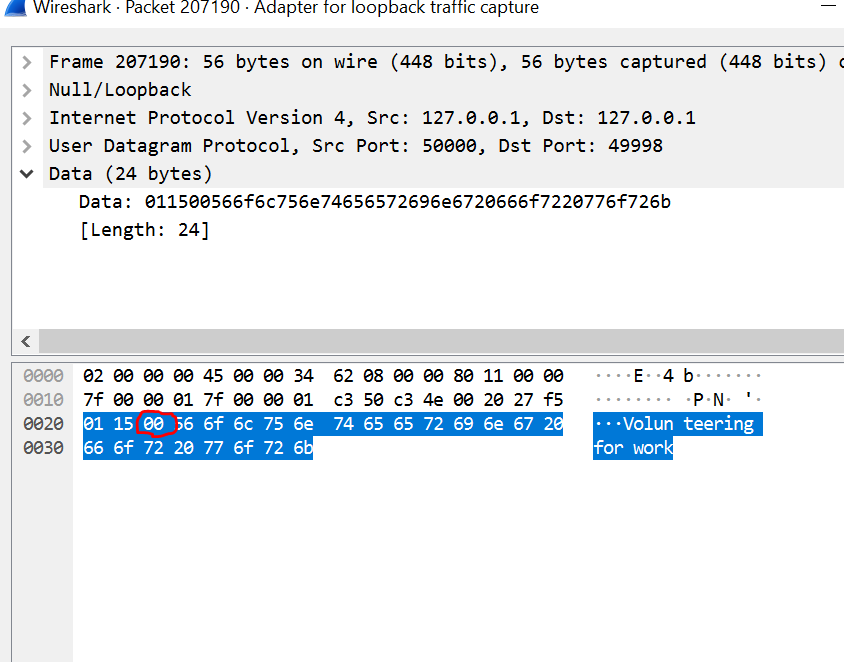
The packet sent contains a header of length 3. The TYPE\_POS indicating the type of message being sent was set to 0. The LENGTH\_POS indicating the length of the packet being sent was set to 1. The INDEX which was used to send the index of the worker in its array was set to 2. This index was also used in the C&C class to differentiate between an acknowledgement that work description had been successfully sent or whether workers were requesting work.

The INDEX part of the header was used mainly however for the Broker to keep track of the available workers. When the workers submitted their work requests, their port number would be assigned to the index in the availableWorkers array. Each time a work request is being forwarded, using a for loop, the broker loops through this array and forwards the message on to any workers whose port numbers are set in the array. If they are not available for work the value at their index is set to 0 and they are never sent the packet. I am sure there are many other ways this could have been achieved but the way I approached it worked best for me. The disadvantage of my method was that I had to end up declaring a global variable in my broker, otherwise I would have had to pass the array of available workers between classes, which could have gotten messy. A possible adaptation could have been to create an alternate class for the broker to use to manage the workers.

As can be seen in the below snippets, the index is passed at the beginning(Header) of the packet.



TYPE\_POS is of TYPE\_STRING here as its value is set to 1. The index of the worker at port 50001 is 1.



The header length of 3 can be seen clearly with the 3 dots before “Volunteering for Work”.

TYPE\_POS is of TYPE\_STRING here as its value is set to 1. The index of the worker at port 50000 is 0.

TYPE\_POS at position 0, LENGTH\_POS at position 1 and INDEX at position 2.

**Conclusion:**

This was a completely new experience for me which was difficult at first but I eventually grew to enjoy doing this project. With the application of Wireshark the picture became a lot clearer to me. It was my first experience with concurrent programming but I feel like it was an interesting introduction. I feel maybe working in groups for some of this project would make it easier for students to manage but overall enjoyed it immensely.

**What went well:**

I felt working problems out not just alone but with the help of my peers worked very well. Sometimes a little push is all you need to get you back going again. That came to me sometimes with the help of my peers explaining some of the more complicated sample code provided. Using Wireshark definitely had a large positive impact on my project as a whole also.

**What I could have done better:**

For me the beginning could have gone a lot better. It took me a long time to get my head around this assignment because I find reading someone else’s code is a tough way to find your own way of doing things. I would recommend for future students to use Wireshark as much and as quickly as possible in order to track the packets and see exactly what is going right or wrong. Problems arose when I tried to allow the C&C provide a number of workers for who the work description was sent to. I think with a little more time I could have found a solution.

**Estimation of time spent:**

38 hours roughly.