YI XIANG TAN
18315028

CS2031 Telecommunication II
Assignment 1 Command & Control
© Trinity College Dublin

Source from **https://knowyourmeme.com** and edited.

YI XIANG TAN
18315028

Index

YI XIANG TAN
18315028

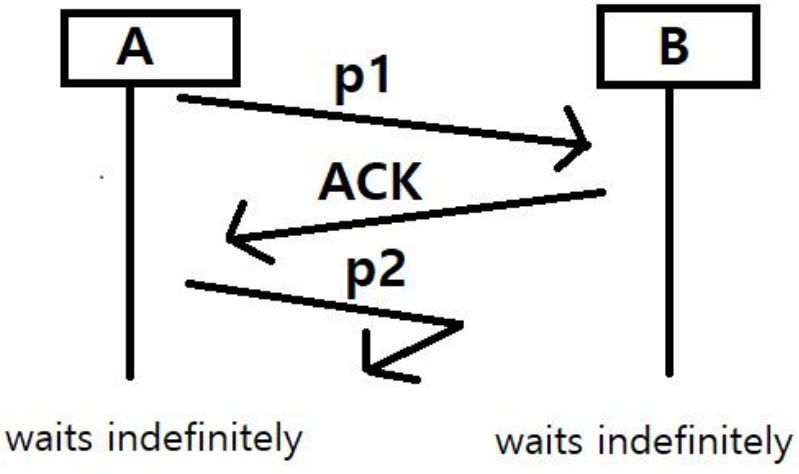
Introduction

Problem Statement

The problem description for this assignment outlined a network consisting of one or more CAndC, one broker, and one or more workers. It is defined that the implementation should allow the transmission of data between the workers and broker, and the broker and the CAndCs. I will be attempting to implement a Stop-N-Wait ARQ.

Stop-N-Wait ARQ

A sends a packet to B and waits for an ACK.
B sends an ACK to A, A receives the ACK and gets notified.
If the packet sent by A is lost, B still waits for packets indefinitely.
A also waits for an ACK indefinitely.

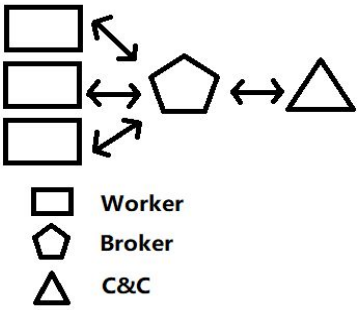


YI XIANG TAN
18315028

Overall Design

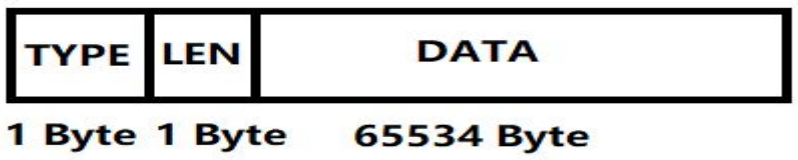
Structure

First, I shall explain my implementation of the assignment.
It is clear that the communication between any two nodes is exclusive, e.g if A could communicate with B, and B could communicate with C, it does not necessarily mean that A could communicate with C.
Instead, ideally for our application A would forward the data to B, and B would transfer the data to C.
The following is a simplified illustration of the network structure.



Framing

All packets transmitted follows a format.
1 byte indicating packet type, 1 byte indicating the length of data, and the data itself.
This allows the handling of data easier.
The following is an illustration of the packet(s).



YI XIANG TAN
18315028

Implementation

Nodes

I would call both parties that were involved in a communication nodes.

In my implementation, there are three node classes. The worker, the broker, and the CAndC.
I will also be implementing an API class for easy initialization of the nodes.

I would explain my implementation of each class, which includes the worker, the broker, and the CAndC.
I would start by stating their function, a very basic description of the flow of communication, supported by a pseudo-code of the class, and stating the functions that were defined in the class.

YI XIANG TAN
18315028

Worker

Volunteer for work, work, return the result of the work.

My implementation of the worker class would volunteer when first initialized by sending a VOLUNTEER packet to the broker.

When forwarded a job description from the broker, the worker does the work and returns a reply to the broker. The worker is then prompted to volunteer again to work. The worker could choose to not volunteer.
The following is a pseudo-code for the worker class.

```
newWorker.volunteer();
  while(true){
    if(packetReceived){
      switch(packetType){
      case String:
      terminal.println(data);
      break;

      case ACK:
      notify();
      break;

      case WORK:
        switch(workType){
        case PRINT:
        for(int i=0;i<ntimes;i++)
        terminal.println(workDescription);
        break;

        case  REVERSE:
        reverseString(workDescription);
        break;

        default: invalidJob();
        }
      newWorker.sendMessage();
      break;

      default: terminal.printlnUnexpected(packet.toString());
      }
    }
  }
```

The worker class consists of the following methods.

-onReceipt(DatagramPacket):
Called when receiving a packet(s).
A switch statement on the type of datagram (packet[0]) is contained in the method, the following are the actions for such cases.
TYPE_STRING: Prints out the string in the terminal.
TYPE_ACK: Notify the thread.
TYPE_WORK: Does the work following the work described in the datagram packet.

-sendMessage():
Invoke the method volunteer() when anything is entered.

-volunteer():
Volunteer to work by sending a TYPE_VOLUNTEER packet with no data to the broker.

-invalidJob():
Send a reply of TYPE_REPLY with no data to the CAndC, forwarded by the broker, indicating invalid work description.

-replyFinished(String message):
Send a reply of TYPE_REPLY with message as the data to the broker. The String is the result of the work

assignment, which would either be an indication that the work has been completed or the result of the work.

YI XIANG TAN
18315028

Broker

Forwards packets from and to the workers and CAndC.

My implementation of the broker class wait() when initialized, waiting to receive packets. When the broker received a packet(s), it would forward the packets to the workers or CAndC depending on the type of the datagram indicated by the leftmost byte in the packet.

The following is a pseudo-code for the broker class.

```
while(true) {
    if(packetReceived) {
        returnACK(packetAddress);
        switch(packetType) {
        case String:
            terminal.println(data);
            break;
        case VOLUNTEER:
            if(!addressList.contains(packetAddress))
                addressList.add(packetAddress);
            break;
        case WORK:
            if(addressList.size()>0) {
                int[]times = distribute(ntimes,addressList.size());
                sendPackets(times,addressList);
                removeAddress(times,addressList);
            }
            else
                replyNoWorker();
            break;
        case REPLY:
        case INVALID:
            sendPacket(packet,CAndC);
            break;
        default:
            terminal.printlnUnexpected(packet.toString());
        }
    }
}
```

The broker class consists of the following methods.

-returnACK(SocketAddress srcAddress):
Send a packet of TYPE_ACK to the srcAddress, indicating that the packet has been received.

-onReceipt(DatagramPacket packet):
Called when receiving a packet(s).
Invoke returnACK()
A switch statement on the type of datagram (packet[0]) is contained in the method, the following are the actions for such cases.
TYPE_STRING: Prints out the string in the terminal if the length is larger then 0.
TYPE_VOLUNTEER: Checks if the packet's address is in the ArrayList of InetSocketAddress, if not add it to the ArrayList.
TYPE_WORK: Checks if there is volunteered worker(s). If no, invoke replyNoWorker(CAndCAddress).
If there is volunteered worker(s), extract the number of times to complete from the work description in the packet, invoke distribute(times_to_complete, ArrayList.size()) to get the distributed times of work for each worker, and send a packet of TYPE_WORK to the workers after rebuilding the packet with the times of work for each worker.
Remove the address of workers that have been assigned work.

-replyNoWorker():
Send a packet of TYPE_REPLY with no data to CAndC, indicating that no worker(s) is available.

-start():
Printing out the port the broker is listening to to the terminal and invoking wait() to receive packet(s).

-distribute(int ntimes, int npeople):
Distribute ntimes work as evenly as possible to npeople, and return an array of integers containing the times to complete for each worker according to the index.

CAndC

Sends work description to the broker.

My implementation of CAndC starts by invoking sendMessage(). Once a work description has been entered, CAndC checks if the work description is in the correct format.
If it is not, CAndC is prompted to enter a new work description.
If it is, a packet of TYPE_WORK is sent to the worker, the data containing information about the type of work, the work details and the number of times to complete. The following is an illustration of the structure of data.

<Type>_<Description>:<Times>

int **String** **int**

E.g. data for task 0, description of HelloWorld, and 4 times would be:

0_HelloWorld:4

The following is a pseudo-code for the CAndC class.

```
while(true) {
   sendMessage();
   wait(packetReceived) {
      switch(packetType) {
      case String:
         terminal.println(data);
         break;
      case REPLY:
         if(data.length>0) {
            terminal.println(data);
         }
         else
            terminal.printNoWorker();
         break;
      case INVALID:
         terminal.printlnInvalid();
         break;
      case ACK:
         notify();
         break;
      default:terminal.printlnUnexpected();
      }
   }
}
```

The CAndC class consists of the following methods.

-onReceipt(DatagramPacket packet):
Called when receiving a packet(s).
A switch statement on the type of datagram (packet[0]) is contained in the method, the following are the actions for such cases.
TYPE_STRING: Prints out the data contained in the packet to the terminal.

TYPE_REPLY: Prints out the data in the packet to the terminal. If there's no data, prints out an error message for no available worker to the terminal.
TYPE_INVALID: Prints out an error message for invalid work type to the terminal.
TYPE_ACK: Notify the thread.

-sendMessage():
Send a packet of TYPE_WORK to the broker, and wait().

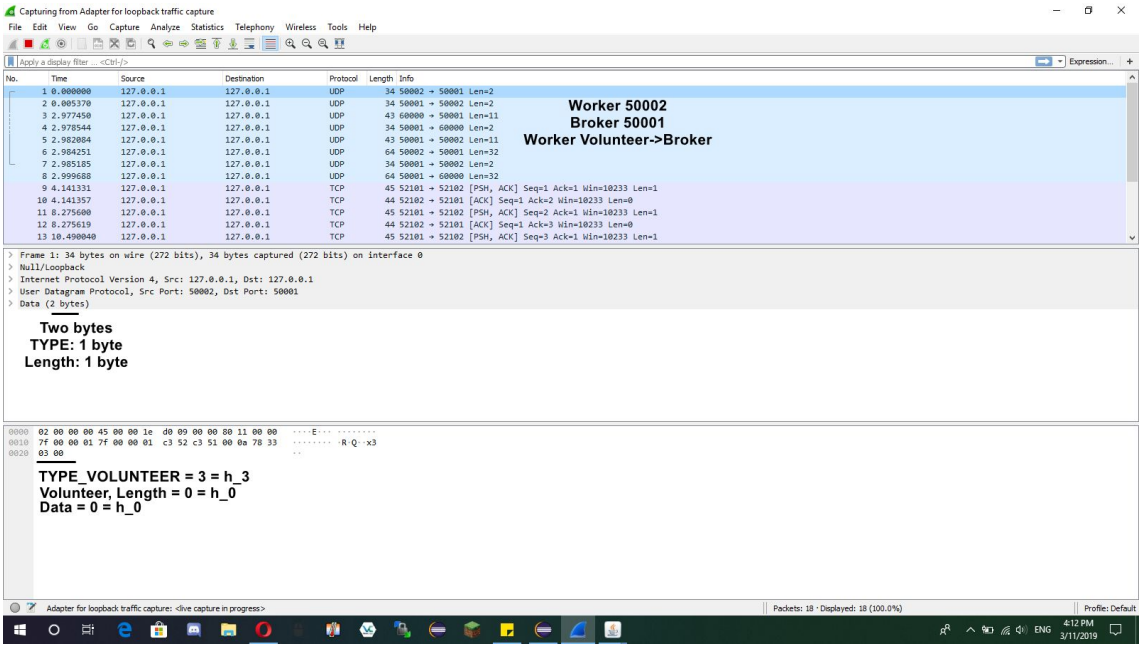
YI XIANG TAN
18315028

Packet Encoding

Following **Framing,** all packets follow one same format. A datagram packet is built as such.

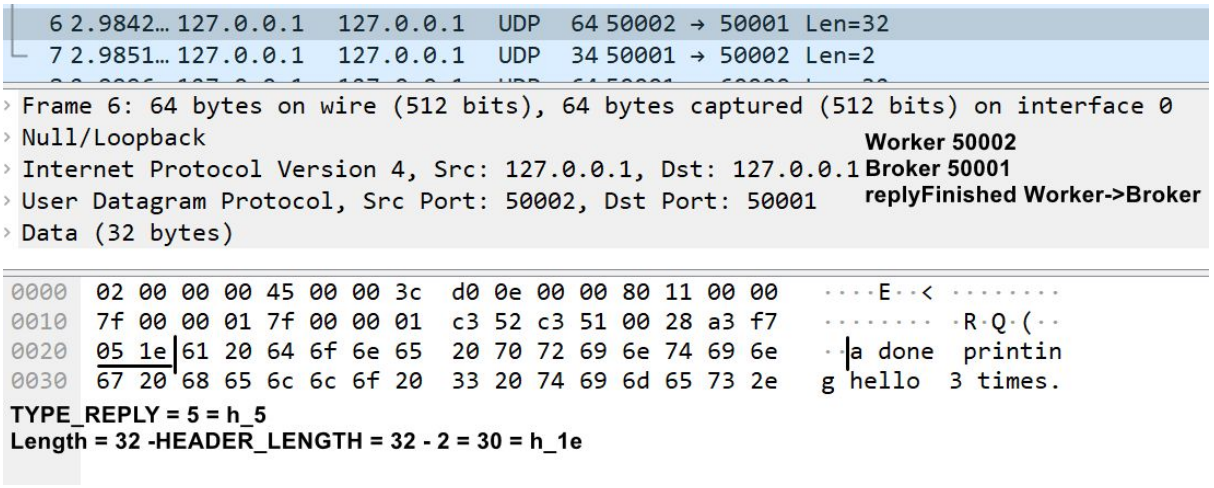
1. Determine packet type.
2. Get the data.
3. Convert the data to a byte array.
4. Get the data byte array length.
5. Create a byte array with the length of HEADER_LENGTH (2 bytes) + data length.
6. Fit the leftmost byte with the packet type defined as constant.
7. Fit the second leftmost byte with the length of the data.
8. Fit the data byte array from the index[HEADER_LENGTH].

Let's look at an example, the picture below is the packet sent when a worker volunteers.



YI XIANG TAN
18315028

Let’s look at another example, in which a worker finished work and send a reply to the broker.



The data in the packet would be “a done printing hello 3 times.”

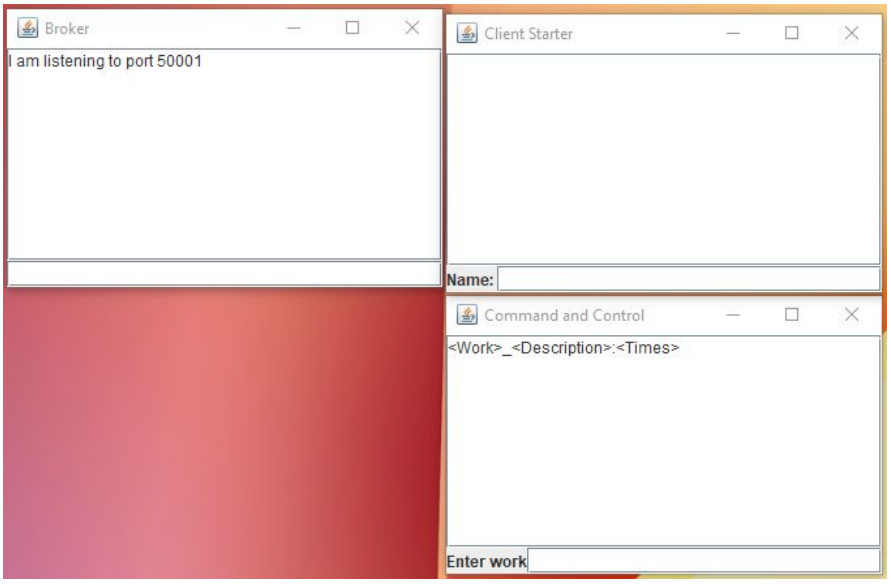
The data was a string and turned into an array of bytes, packed into a packet and sent over.
When received, the bytes of data were extracted and turned into a string again.

YI XIANG TAN
18315028

Illustration of Communication

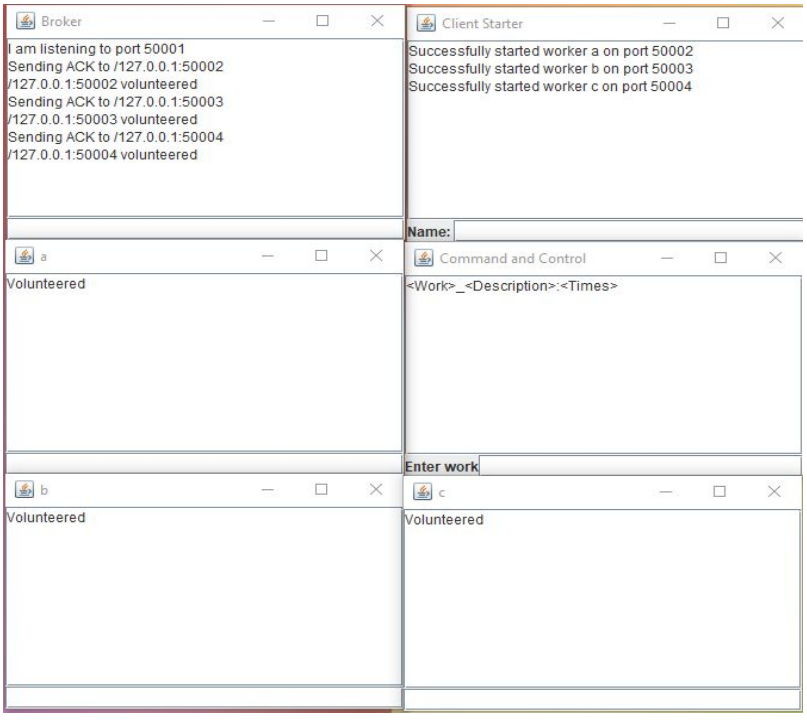
Normal situation

1. First, we run the API.

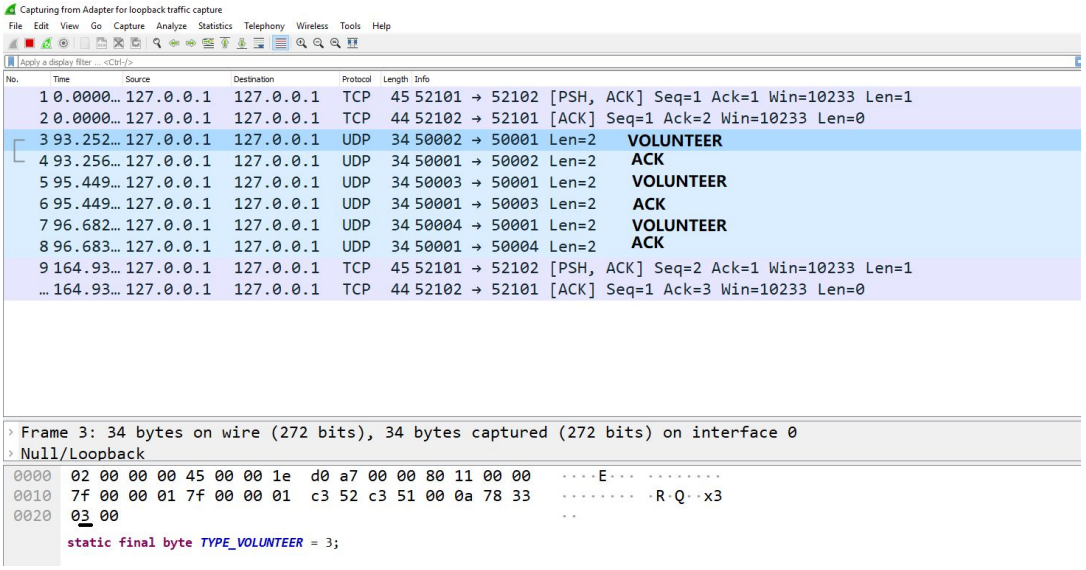


The broker is initialized and listening to port 50001.
The CAndC is initialized and listening to port 60000.

2. We initialize 3 workers, a, b, c which listens to port 50002, 50003, 50004 respectively.



We see that all three workers have sent a packet of TYPE_VOLUNTEER to the broker, and have received ACK from the broker for the packet.



3. Now we assign a new work from the C&C.
   The work: Print “hello” 11 times is represented as such.

The broker received the packet, determine it is a packet of TYPE_WORK,

did some calculations and send work assignment(s) to the workers.

```
   9 164.93… 127.0.0.1   127.0.0.1   TCP   45 52101 → 52102 [PSH, ACK] Seq=2 Ack=1 Win=10233 Len=1
   … 164.93… 127.0.0.1   127.0.0.1   TCP   44 52102 → 52101 [ACK] Seq=1 Ack=3 Win=10233 Len=0
   … 274.58… 127.0.0.1   127.0.0.1   TCP   45 52101 → 52102 [PSH, ACK] Seq=3 Ack=1 Win=10233 Len=1
   … 274.58… 127.0.0.1   127.0.0.1   TCP   44 52102 → 52101 [ACK] Seq=1 Ack=4 Win=10233 Len=0
   … 552.03… 127.0.0.1   127.0.0.1   UDP   44 60000 → 50001 Len=12   CAndC to Broker
   … 552.03… 127.0.0.1   127.0.0.1   UDP   34 50001 → 60000 Len=2    ACK
   … 552.03… 127.0.0.1   127.0.0.1   UDP   43 50001 → 50002 Len=11   Broker to Worker
   … 552.03… 127.0.0.1   127.0.0.1   UDP   43 50001 → 50003 Len=11   Broker to Worker
   … 552.03… 127.0.0.1   127.0.0.1   UDP   43 50001 → 50004 Len=11   Broker to Worker
   … 552.04… 127.0.0.1   127.0.0.1   UDP   64 50003 → 50001 Len=32
   … 552.04… 127.0.0.1   127.0.0.1   UDP   64 50004 → 50001 Len=32
   … 552.04… 127.0.0.1   127.0.0.1   UDP   34 50001 → 50003 Len=2
   … 552.04… 127.0.0.1   127.0.0.1   UDP   64 50002 → 50001 Len=32
   … 552.05… 127.0.0.1   127.0.0.1   UDP   64 50001 → 60000 Len=32
   … 552.05… 127.0.0.1   127.0.0.1   UDP   34 50001 → 50004 Len=2
```

```
› Frame 13: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface 0
› Null/Loopback
0000  02 00 00 00 45 00 00 28  d0 b1 00 00 80 11 00 00   ····E··(  ········
0010  7f 00 00 01 7f 00 00 01  ea 60 c3 51 00 14 aa 6a   ········  ·`·Q···j
0020  04 0a 30 5f 68 65 6c 6c  6f 3a 31 31               ··0_hell o:11
```

data(job description)

The worker receives the work.

```
a
Volunteered
Received work: 0
Description: hello
Times: 4
1. Printing: hello
2. Printing: hello
3. Printing: hello
4. Printing: hello
Sending packet...
```

4. The workers worked, and replyFinished(reply);

```
… 552.04… 127.0.0.1   127.0.0.1   UDP   64 50003 → 50001 Len=32    Reply
… 552.04… 127.0.0.1   127.0.0.1   UDP   64 50004 → 50001 Len=32    Reply
… 552.04… 127.0.0.1   127.0.0.1   UDP   34 50001 → 50003 Len=2     ACK
… 552.04… 127.0.0.1   127.0.0.1   UDP   64 50002 → 50001 Len=32    Reply
… 552.05… 127.0.0.1   127.0.0.1   UDP   64 50001 → 60000 Len=32
… 552.05… 127.0.0.1   127.0.0.1   UDP   34 50001 → 50004 Len=2     ACK
… 552.05… 127.0.0.1   127.0.0.1   UDP   64 50001 → 60000 Len=32
… 552.05… 127.0.0.1   127.0.0.1   UDP   34 50001 → 50002 Len=2     ACK
… 552.05… 127.0.0.1   127.0.0.1   UDP   64 50001 → 60000 Len=32
```

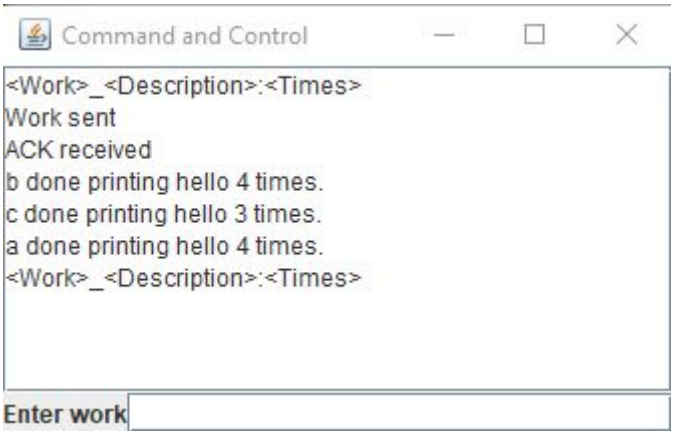
The broker receives the replies, and send back ACKs

```
Sending ACK to /127.0.0.1:50003
Forwarding reply from /127.0.0.1:50003 to C&C.
Sending ACK to /127.0.0.1:50004
Forwarding reply from /127.0.0.1:50004 to C&C.
Sending ACK to /127.0.0.1:50002
Forwarding reply from /127.0.0.1:50002 to C&C.
```

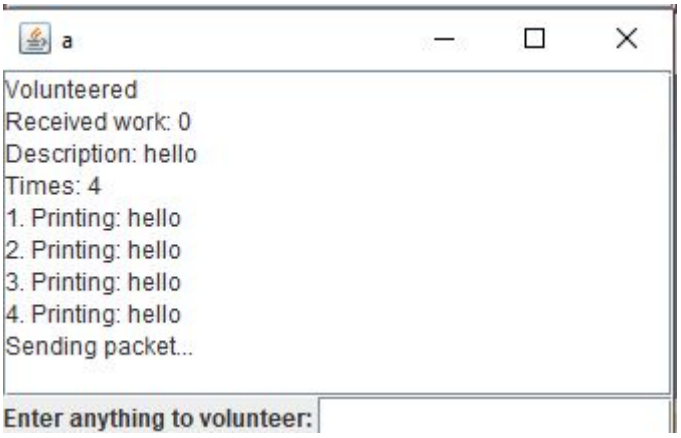
And proceeds to forward the replies to the CAndC.

… 552.04…	127.0.0.1	127.0.0.1	UDP	64	50003 → 50001 Len=32	
… 552.04…	127.0.0.1	127.0.0.1	UDP	64	50004 → 50001 Len=32	
… 552.04…	127.0.0.1	127.0.0.1	UDP	34	50001 → 50003 Len=2	
… 552.04…	127.0.0.1	127.0.0.1	UDP	64	50002 → 50001 Len=32	
… 552.05…	127.0.0.1	127.0.0.1	UDP	64	50001 → 60000 Len=32	**Forward reply**
… 552.05…	127.0.0.1	127.0.0.1	UDP	34	50001 → 50004 Len=2	
… 552.05…	127.0.0.1	127.0.0.1	UDP	64	50001 → 60000 Len=32	**Forward reply**
… 552.05…	127.0.0.1	127.0.0.1	UDP	34	50001 → 50002 Len=2	
… 552.05…	127.0.0.1	127.0.0.1	UDP	64	50001 → 60000 Len=32	**Forward reply**

5. Finally, the CAndC receives the replies.



6. The worker is then prompted to volunteer. If the worker volunteers, we're back to step 1.

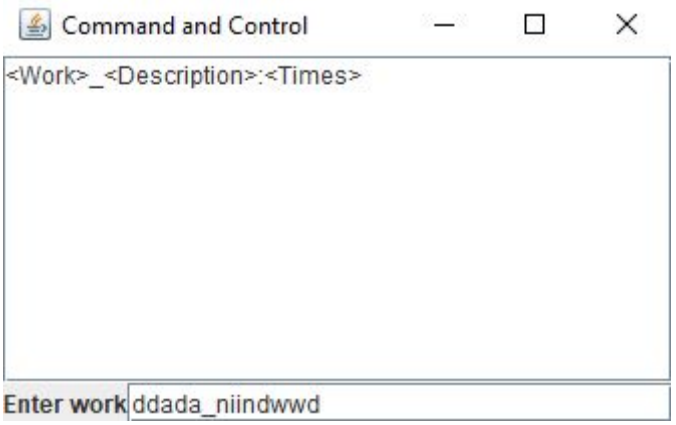


YI XIANG TAN
18315028

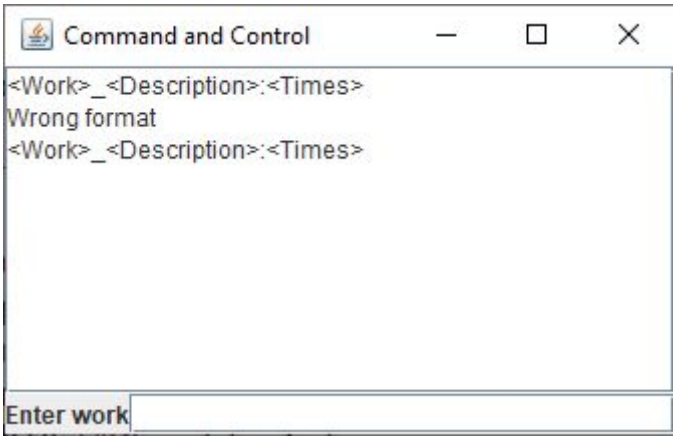
Invalid Work Assignment

Wrong format

In the case of a work assignment with the wrong format, there's a built-in check.



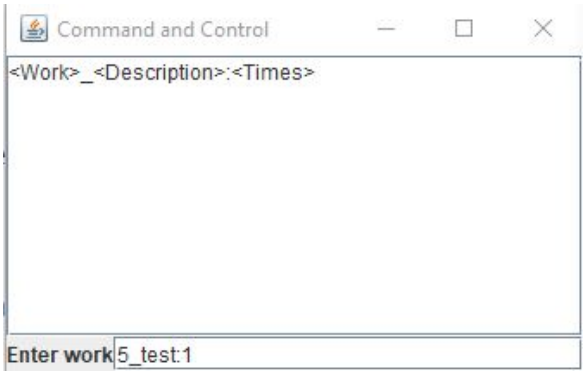
The data was not correctly encoded.
An error message would be printed out into the terminal, and the packet was not sent.



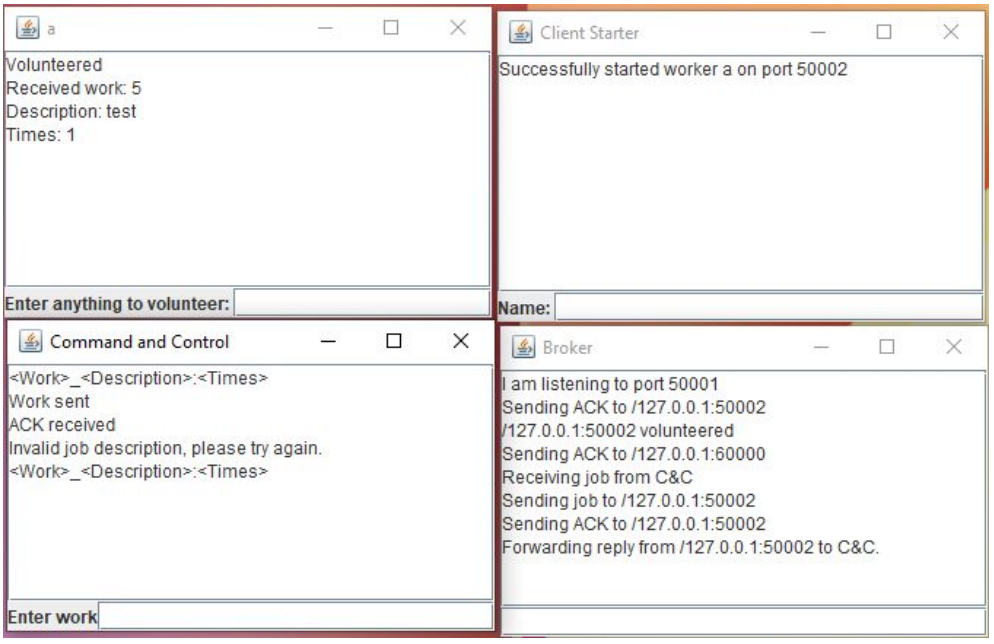
YI XIANG TAN
18315028

Undefined Work Type

In this case, the data was properly encoded.



The packet was sent to the broker and forwarded to the worker.



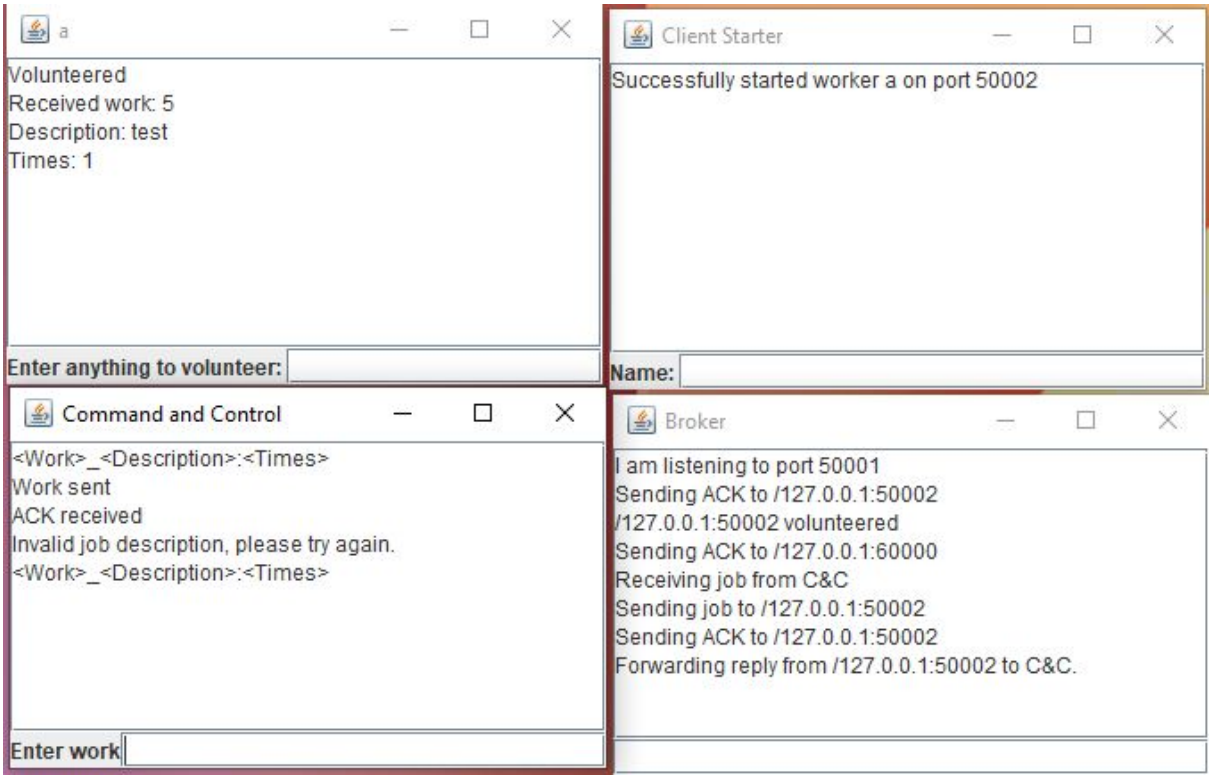
However, workType 5 is undefined, so the worker invoked invalidJob();

```
… 55.159… 127.0.0.1   127.0.0.1   UDP   34 50002 → 50001 Len=2
… 55.160… 127.0.0.1   127.0.0.1   UDP   34 50001 → 50002 Len=2
… 55.171… 127.0.0.1   127.0.0.1   UDP   34 50001 → 60000 Len=2
… 67.920… 127.0.0.1   127.0.0.1   TCP   45 52101 → 52102 [PSH, ACK] Seq=3 Ack=1 Win
… 67.920… 127.0.0.1   127.0.0.1   TCP   44 52102 → 52101 [ACK] Seq=1 Ack=4 Win=1023
```

```
› Frame 10: 34 bytes on wire (272 bits), 34 bytes captured (272 bits) on interface
› Null/Loopback
0000  02 00 00 00 45 00 00 1e  d1 69 00 00 80 11 00 00   ····E···  ·i····
0010  7f 00 00 01 7f 00 00 01  c3 52 c3 51 00 0a 75 33   ········  ·R·Q··u3
0020  06 00                                             ··
```

```
static final byte TYPE_INVALID = 6;
```

The broker forwarded the packet to the CAndC.

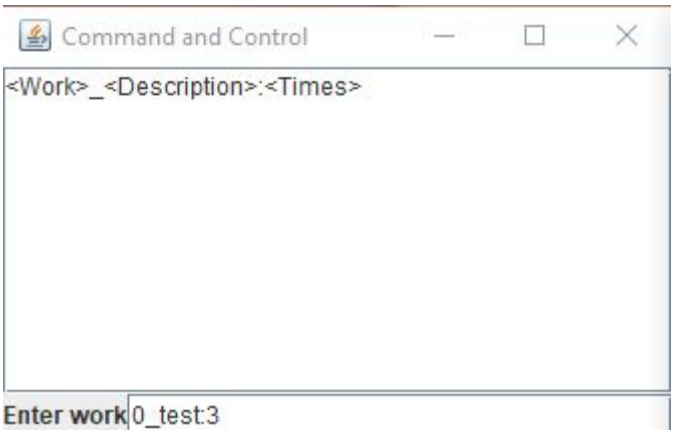


An error message was printed out.

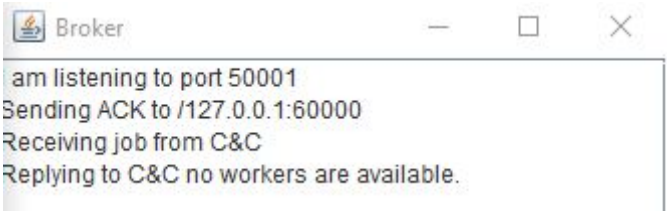
YI XIANG TAN
18315028

No Worker

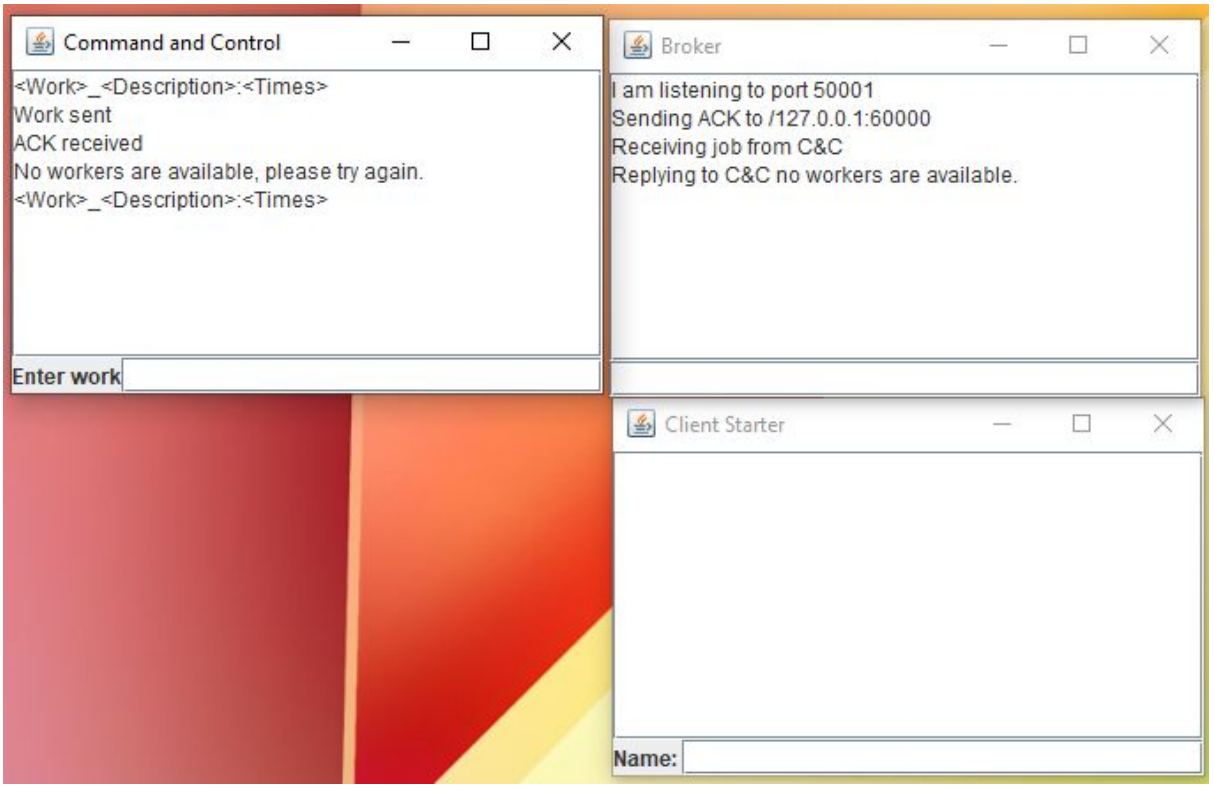
The CAndC send out a valid work assignment.



The broker receives it, and invoke replyNoWorker();



CAndC receives the packet.



Discussion

Strengths

1. Fairly easy to implement.
2. Allow easy addition of workers' methods.
3. Allow the addition of CAndC with minimal tweaks.
4. Allow a huge range of workers as long as the ports are free and the broker has enough memory.
5. Relatively small header size.
6. Isolated connections between nodes increase privacy.
7. Some amount of error checking on the server-side.

Weaknesses

1. Small packet size, up to 253 bytes of data per packet.
2. The header was not so helpful.
3. Data in plain text, causing packets captures easy to exploit.
4. Packet loss causes most communication to stop.
5. The CAndC might not be able to print out messages correctly with high latency packets.
6. A worker has to do at least one work (could be changed easily).
7. No implementation that actually uses the name of the worker.

Challenges & Ways to Overcome

1. Threads

   I have started two threads for the CAndC, one to sendMessage(), and one to receipt packets. However, the terminal.readline() will stop the terminal from printing out correctly, causing a delay of sorts.

   I have searched for a solution on the web, and have discussed the issue with multiple colleagues and has yet to find the exact cause of the issue. I ended up setting a timed loop to call the sendMessage() method to allow the packets to be printed out correctly.
   I have set the timer to 1 second, which should be enough for most tasks to complete.

2. Starting to Code

   I would say that I have a sufficient amount of knowledge to at least implement a very basic structure of the assignment.
   However, when I started to code, I get lost pretty quickly. Bugs appear and reappear with me not knowing what fixed it and what broke it.
   One implementation usually depends on the other, and with wait and notify cycle in the fray, I feel like I could not complete the assignment.

I started to read the description of the assignment, taking note of what steps to implement first. I took the sample code, tore away anything I could that still allows the Client to send to the Server and worked from thereon.
I have issues with threads from the very start(1), and it has plagued me since the very beginning. I had to go out of my way to avoid the issues, and other than that I could do well.
As a result, I ran out of time to do the advanced implementation.

Possible Improvements

1. Implement Go-Back-N ARQ to cope with packet loss.
2. Longer header with more information.
3. Larger packet for more data transfer per transmission.
4. Implement a hash map to keep a list of workers.
5. Better wait() and notify() cycle.
6. Encrypt the data.
7. Allow the worker to start up without volunteering.
8. Allow the worker to quit after volunteering even without finishing work.

YI XIANG TAN
18315028

Reflection

The assignment has taught me much about how the network works and how to manage a project.
First look into the assignment, and it is intimidating. After I spend some time trying to think through what needs to be done, it because easier.

Understanding the assignment is not as troublesome as I might think. The tutorials do help a lot. I have watched some youtube videos on networking, and they do provide me with something to work with.

Breaking down the assignment into parts do help a lot. To understand the assignment, and to try and write it down before going all-in coding is definitely important.
I was surprised about the amount of code I have disposed of just because of me not understanding a concept very well.

I have spent an estimate of about 50 hours on the assignment and maybe more. Most of the time was wasted trying to deal with terminal blocking.