# Telecoms II - Assignment I Report

## Publish - Subscribe

Samuel Petit

petits@tcd.ie - 17333946

## Contents

# 1. Meeting the requirements - the core of the program

## a. Sending & receiving data

The way I send data from one place to another is using Datagram Packets, this enables sending packets based on data specific to that packet (i.e. any packet gets bytes to send and a SocketAddress to send to). The other advantage to using these is that it enables connectionless sending of data - no connection needs to be established before sending data (now a quick note on this, this might not be such a good thing in a lot of situations but for the purpose of this assignment it makes my job easier without so many compromises. When a client or publisher decides to connect I can just send a packet to my broker which handles connections, the broker can just remove that connection when an end of communication is requested. The one downside to this is when a packet is sent to the brokers address but it is not connected, handling that “broker not available” status is made more difficult, the way I counter this is by sending a “disconnection” message, the broker will then remove that client from its list of connections).

The way data is actually received and sent is using Datagram Sockets which sends Datagram Packets (which we just talked about earlier). This makes setting up connections very simple as the way both of these objects are initialised is using the following piece of code:

```
DatagramSocket s = new DatagramSocket(SOURCE_PORT);
DatagramPacket p = new DatagramPacket(data, data_length,
DESTINATION_ADDRESS);
```

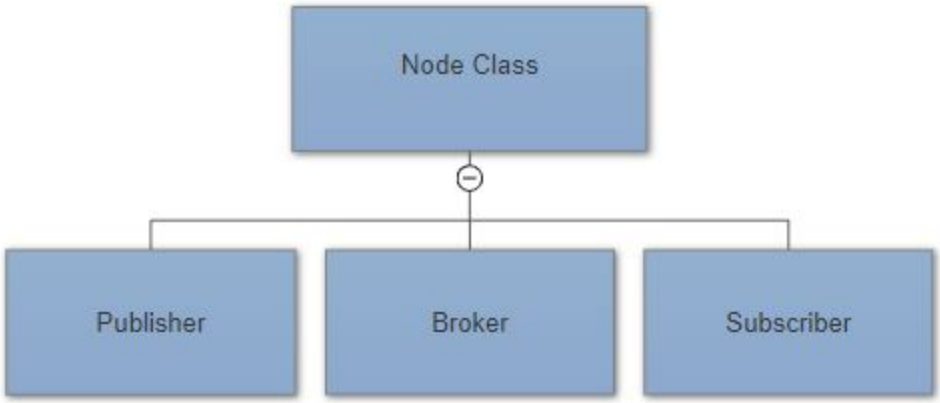
The communication protocol used in this case is UDP which does not include all of the methods of TCP that ensures data is properly received and enables me to try to implement those myself.

## b. Program structure

I’ve started my program using the files provided for the assignment except I didn’t keep all of them : in the end I kept the Node class which I slightly modified and then made my own classes. My program has 4 classes in total, which might seem small but I think it makes my program very simple to understand. I very much could have used subclasses to automate things such as packet creation, JSON creation & parsing (I will get to that later) amongst other actions.

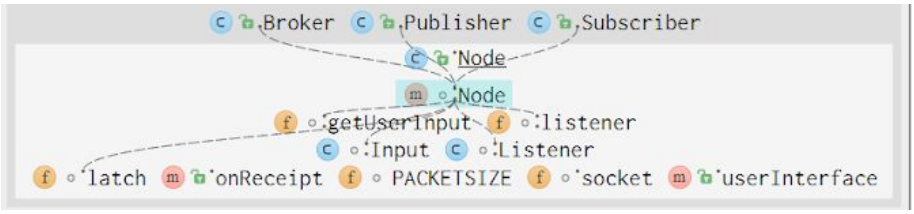
I then have : a Broker class, a Client (subscriber) class, a Publisher class and a Node class. The first 3 of these extend the last one : the Node class. It is useful for multiple things :

- The Node class extends the Thread class which makes it possible to run multiple methods concurrently
- The Node class implements the method which receives packets & executes the “onReceipt” method upon receipt. The method needs to be implemented by any of the 3 classes that extend the Node class
- Similarly, all 3 classes (broker, subscriber, publisher) may execute different “actions” (subscribing, publishing etc..), these all happen through the command line -- through the constant listening of user input. After every single user input in the command line, the method “userInterface” gets executed. Once again any class extending Node must override this method too. The implementation of that method will then decide based on user input what the program should do.




High level view of the program structure

Here is one last screenshot taken with the tool Structure101, enabling me to show in a graphical way the dependencies of my program I just explained above:

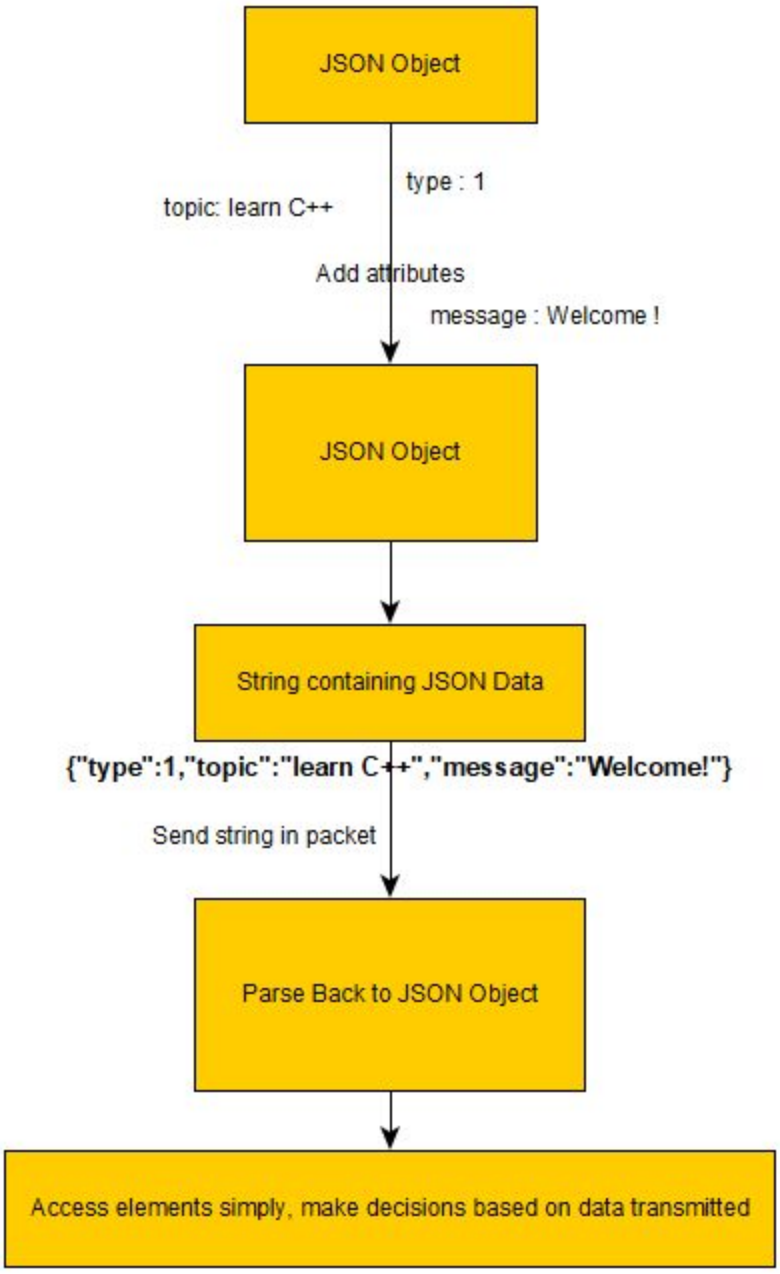


One quick observation to make here is that both the listener for packet receiving and user input are made through separate threads in order to make them run concurrently.

## c. Packet structure

The way I format packets in order for them to be understood in a simple way by its receivers is by using JSON. A JSON object is built with the relevant information I want to send. I then extract the JSON string from that object & put its bytes into the packet that is sent. This enables me not to have to deal with lower level data formatting & parsing the relevant data piece by piece upon receipt (in order for the receiver to know what to do after receiving this packet).

Here is how the process works :




One thing to note is that since java does not have Json objects / methods by default I had to add a (widely used) library in order to use those. The library does not simplify anything but manipulating JSON objects. The name of that library is gson.

## d. Constant user input - execute actions

One thing I mentioned earlier is that for all of my 3 classes, the program will

actively listen of user input, executing a function proper to that class which enables it to execute different actions Let's now talk about the specifics of the APIs for all 3 classes.

### e. Subscriber API

The subscriber API enables the following actions :

- Subscription
- Un-subscription
- End Communication

The way these work are pretty similar, for the first two, the user will then be asked to input a topic of his choice, that client will then start or stop receiving messages for that topic. This is done by creating a packet that is sent to the broker containing : the type of message (an Integer value that represents either subscribe or unsubscribe), and the topic. This data is put into JSON format and then sent to the broker that changes its list of subscribers accordingly and then sends an acknowledgement packet back upon the receipt of that packet.
The 3rd type is slightly different : the end of communication will simply send a packet to the broker notifying it of the end of the communication between those instances. The broker will once again update its lists accordingly so that no unnecessary packets are sent to this client while it is disconnected (remember that the way we are transferring data is via a connectionless method so this situation would be very likely to happen).

### f. Publisher API

The publisher API enables the following actions :

- Select a topic
- Publish
- End Communication

The first two of these options are linked together : a publisher needs to have set a topic before being able to publish. As a result, if a publisher tries to publish a message before he has selected a topic, he will first be asked to select that topic. That option remains available if at any point the publisher wishes to publish for another topic. Then, in order to send a message for that topic he will then simply be asked for the message to publish, the packet will then be composed of : information about the index for the message (used for go back N), the type of communication (publish), the topic and the message.
The end of communication works the exact same way as it does for subscribers.

## g. Broker - the communication manager (& its API)

The brokers API is slightly more different in comparison to the publishers and subscribers as in it is not necessary for the program to function. It provides with additional functionality which might make testing the program easier.
Here is the list of those actions :

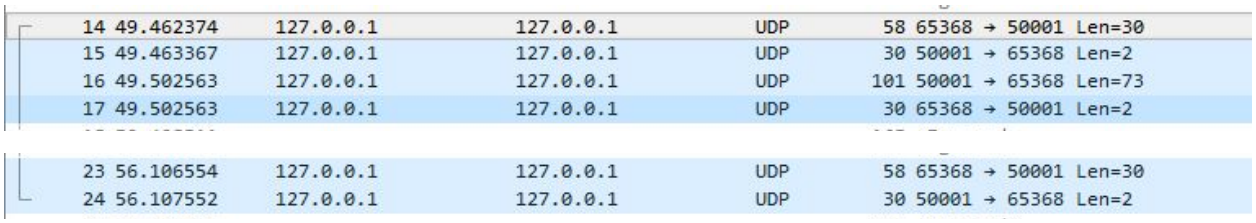
- Clear the subscribers list
- Clear the publishers list
- Clear the messages list

The are fairly straightforward so I won't spend too much time explaining those, what I will mention however is that the last option : "clear the messages list" is useful when testing go-back-N.

## h. Keeping messages in sequences

The way my programs keeps track of the order in which messages were sent is pretty simple : these are kept in an arraylist (resizable array), as a result, the order in which the messages were received is preserved.

## i. Packet Capturing, Flow overview




Flow between a broker and a subscriber, in order, we can see : subscription, message forwarding, unsubscription

We can observe multiple things here : first of we can see a subscriber sending a subscription message and receiving and acknowledgement back from the broker :

```
> Frame 14: 58 bytes on wire (464 bits), 58 bytes captured (464 bits)
  Raw packet data
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v User Datagram Protocol, Src Port: 65368, Dst Port: 50001
    Source Port: 65368
    Destination Port: 50001
    Length: 38
    Checksum: 0x05e4 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 2]
v Data (30 bytes)
    Data: 7b2274797065223a312c22746f706963223a226c6561726e...
    [Length: 30]
```

```
0000  45 00 00 3a 4a 4b 00 00  80 11 00 00 7f 00 00 01   E··:JK·· ····????
0010  7f 00 00 01 ff 58 c3 51  00 26 05 e4 7b 22 74 79   ·····X·Q ·&··{"ty
0020  70 65 22 3a 31 2c 22 74  6f 70 69 63 22 3a 22 6c   pe":1,"t opic":"l
0030  65 61 72 6e 20 43 2b 2b  22 7d                     earn C++ "}
```

*Subscribe message : type 1*

```
> Frame 15: 30 bytes on wire (240 bits), 30 bytes captured (240 bits)
  Raw packet data
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v User Datagram Protocol, Src Port: 50001, Dst Port: 65368
    Source Port: 50001
    Destination Port: 65368
    Length: 10
    Checksum: 0xcfc1 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 2]
v Data (2 bytes)
    Data: 6f6b
    [Length: 2]
```

```
0000  45 00 00 1e 4a 4c 00 00  80 11 00 00 7f 00 00 01   E···JL·· ····????
0010  7f 00 00 01 c3 51 ff 58  00 0a cf c1 6f 6b         ·····Q·X ····ok
```

Acknowledgement from the broker for the subscription

We can then see that messages get forwarded to that subscriber from the broker followed by another acknowledgement but this time from the subscriber to the broker.

```
> Frame 16: 101 bytes on wire (808 bits), 101 bytes captured (808 bits)
  Raw packet data
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v User Datagram Protocol, Src Port: 50001, Dst Port: 65368
    Source Port: 50001
    Destination Port: 65368
    Length: 81
    Checksum: 0x94b7 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 2]
v Data (73 bytes)
    Data: 6c6561726e20432b2b203a2057656c636f6d652065766572...
    [Length: 73]
```

```
0000  45 00 00 65 4a 4d 00 00  80 11 00 00 7f 00 00 01   E··eJM·· ········
0010  7f 00 00 01 c3 51 ff 58  00 51 94 b7 6c 65 61 72   ·····Q·X ·Q··lear
0020  6e 20 43 2b 2b 20 3a 20  57 65 6c 63 6f 6d 65 20   n C++ :  Welcome
0030  65 76 65 72 79 6f 6e 65  20 3a 29 0a 0d 6c 65 61   everyone  :)··lea
0040  72 6e 20 43 2b 2b 20 3a  20 54 68 69 73 20 69 73   rn C++ :  This is
0050  20 6f 75 72 20 66 69 72  73 74 20 74 75 74 6f 72    our fir st tutor
0060  69 61 6c 0a 0d                                    ial··
```

Message sending from the broker to the subscriber

We can then finally see that this same subscriber sends an unsubscription request and gets an acknowledgement back :

```
> Frame 23: 58 bytes on wire (464 bits), 58 bytes captured (464 bits)
  Raw packet data
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v User Datagram Protocol, Src Port: 65368, Dst Port: 50001
    Source Port: 65368
    Destination Port: 50001
    Length: 38
    Checksum: 0x04e4 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 2]
v Data (30 bytes)
    Data: 7b2274797065223a322c22746f706963223a226c6561726e...
    [Length: 30]
```

```
0000  45 00 00 3a 4a 4f 00 00  80 11 00 00 7f 00 00 01   E··:JO·· ········
0010  7f 00 00 01 ff 58 c3 51  00 26 04 e4 7b 22 74 79   ·····X·Q ·&··{"ty
0020  70 65 22 3a 32 2c 22 74  6f 70 69 63 22 3a 22 6c   pe":2,"t opic":"l
0030  65 61 72 6e 20 43 2b 2b  22 7d                    earn C++ "}
```

Unsubscription message, type 2

## 2. Additions - Going the extra mile

### a. Subscriber / Broker Acknowledgements

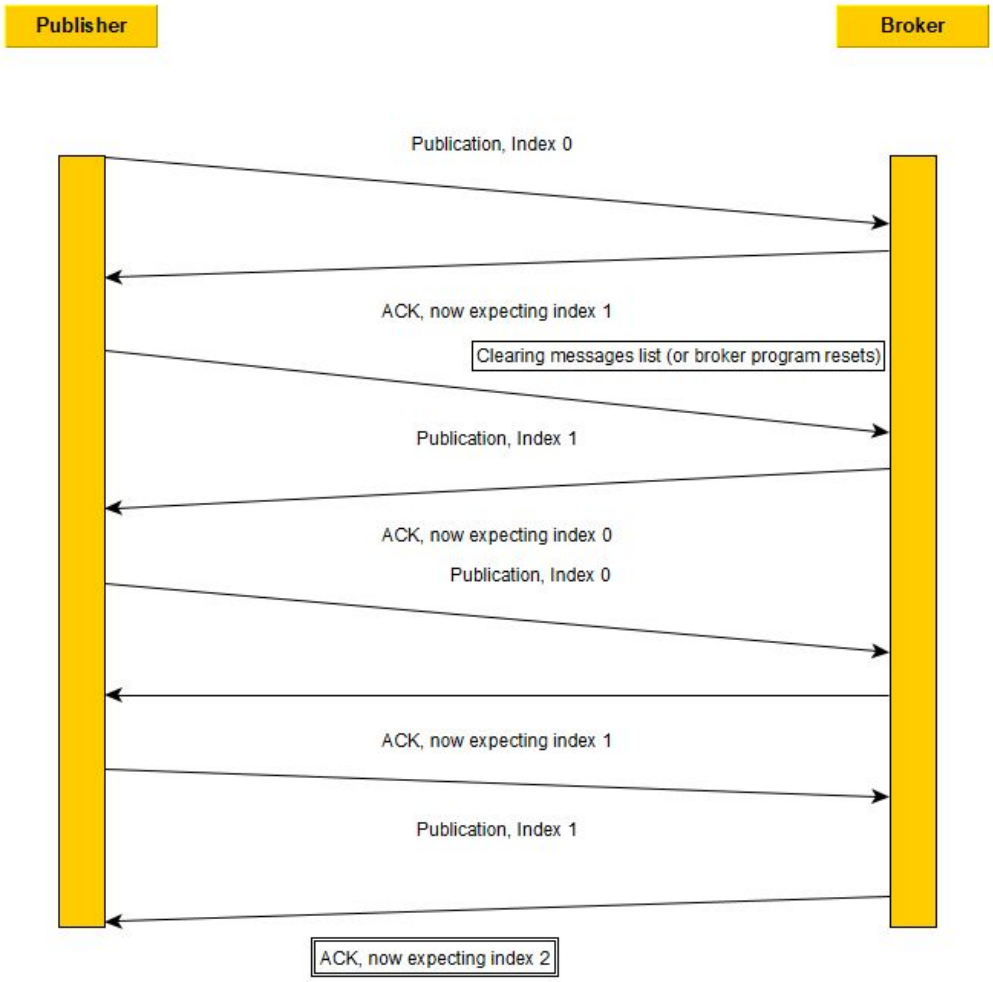
Let's start by saying that my program only partially implements ACKs. These packets will be sent after the receipt of packets, however nothing is currently done about this, ideally, after sending a packet, a timer would be started and if no acknowledgements have been received by a set time lapse then the packet would be sent again.

### b. Publisher / Broker acknowledgements

Once again, these are partially implemented. However, these have added functionality. For one, the publisher will wait to receive an acknowledgement packet from the broker before being able to send another message/packet. Where this fails is in the case that the ACK packet fails to be received and would require once again to implement a set amount of time to wait for an ack packet before sending the packet again. This would obviously be an issue in the long run and would have to be the first in line in the list of additions for this program.

### c. Go Back N - My Implementation

On top of classic acknowledgements, the communication between a publisher and the broker uses go-back-N. This is implemented by giving an index to each message and sending the index of the current message in the packet so that the broker can verify that it was indeed expecting that message. Here is a diagram of my implementation of go back N in my program:




One thing to note is that my implementation uses a window of size 1 which is a difference from the original go back N protocol which we will talk about in the next paragraph.

## d. Handling multiple clients & publishers

The way I handle multiple clients and publisher is fairly straightforward (although not the most elegant one) is to give the port 0 when creating sockets - by doing this java will allocate any available port to the socket. For the purpose of this program I believe doing this is totally acceptable.

## e. Things to improve on & potential additions

A bit on future improvements that this program would benefit.
First of all, finishing to implement the acknowledgement packets would be the

first in terms of priority in my opinion. Currently, an acknowledgement not properly received might make the communication between one publisher and one client completely crash. This would require to implement a system that expects an acknowledgement and, after a set time lapse, if no acknowledgement is received, send the packet again.
Another addition to this program could be the optimization of threads, as it is now, every single instance uses multiple threads : one to listen of incoming packets, another one listening for user input in the command line. There could be serious optimization done here.
Finally, I could optimize the use of ports : so far every single client uses a different port, I could make it so that multiple clients use the same port. One last nice addition would be to make it so that multiple brokers could be used for this program.

## f. Final thoughts

I thought that this assignment went well overall, the one criticism to myself is that I waited too long to start it even though there is a second one to do. It was very interesting to learn about concurrent programming which I had never done before. I also got to learn how to use java objects such as sockets, threads, runnables and such which I very much liked.
I spent a lot of time actually starting the program (due to my lack of knowledge of programming in this “area”) which could have gone a bit better if it were perhaps introduced a bit in lectures but it wasn’t hard to get going once I understood the basics (through personal research as well as lectures).
I spent roughly 25 hours on this assignment.