CS2013 - Telecommunications II Assignment I

Anton Yamkovoy.

November 3rd 2018

1 Publish - Subscribe implementation.

1.1 Requirements for the Publish-Subscribe implementation:

- Subscribers have the ability to subscribe and unsubscribe to and from multiple topics.
- Subscribers accept a subscribe / unsubscribe confirmation from a broker.
- A subscriber should have messages received via a broker from a publisher printed out in order of sequence number.
- A Publisher should have the ability to send out publish requests to all the subscribers via the broker who are subscribed at the current time to the topic referenced in the publish request.
- The broker should be able to accept different types of requests from publishers and subscribers, and should schedule and send back acknowledgements and negative-acknowledgements to the senders and should transfer successful messages to the recipient.
- A Go-Back-N system should be implemented between the publisher and broker.
- The implementation should allow, multiple subscribers, multiple publishers, and the creation of multiple separate networks being controlled by separate brokers.

1.2 Additional features implemented:

- Custom terminal jframe implementation of Subscriber terminal, containing a feed for all received messages, input fields, activated by button to limit errors in inputting commands. A way to turn off the receipt of acknowledgements to simulate a real system and testing.
- A custom publisher implementation with input text fields and button as an input method, and feed for all sent publish requests and acks received, and a input field for variable timeout times.
- A subscriber should have messages received via a broker from a publisher printed out in order of sequence number.
- A custom broker terminal with a transcript and button to turn acks on/off
- All the terminal implementations also have windows for lists for subscribers/brokers to make debugging easier, these fields are set by the user when deciding the number of publishers/subscribers in a system and appointing the network broker.
- Multiple instances of terminals can be changed easily using command line arguements.

2 Subscriber Analysis.

2.1 Subscriber components:

- Subscriber class:
  The subscriber class was the implementation of all the functional methods needed to interact with the broker and publishers, It extends the Node class, and uses the SubTerminal as one of its main instance variables.

- SubTerminal class:
  The SubTerminal is a jframe implementation of a terminal, it has awt/swing components and a print method and action listeners for button and textfield inputs.

- SubscriberTopicsList:
  The SubscriberTopicsList class is used by the broker to hold and ArrayList of topics, and implements a find method.

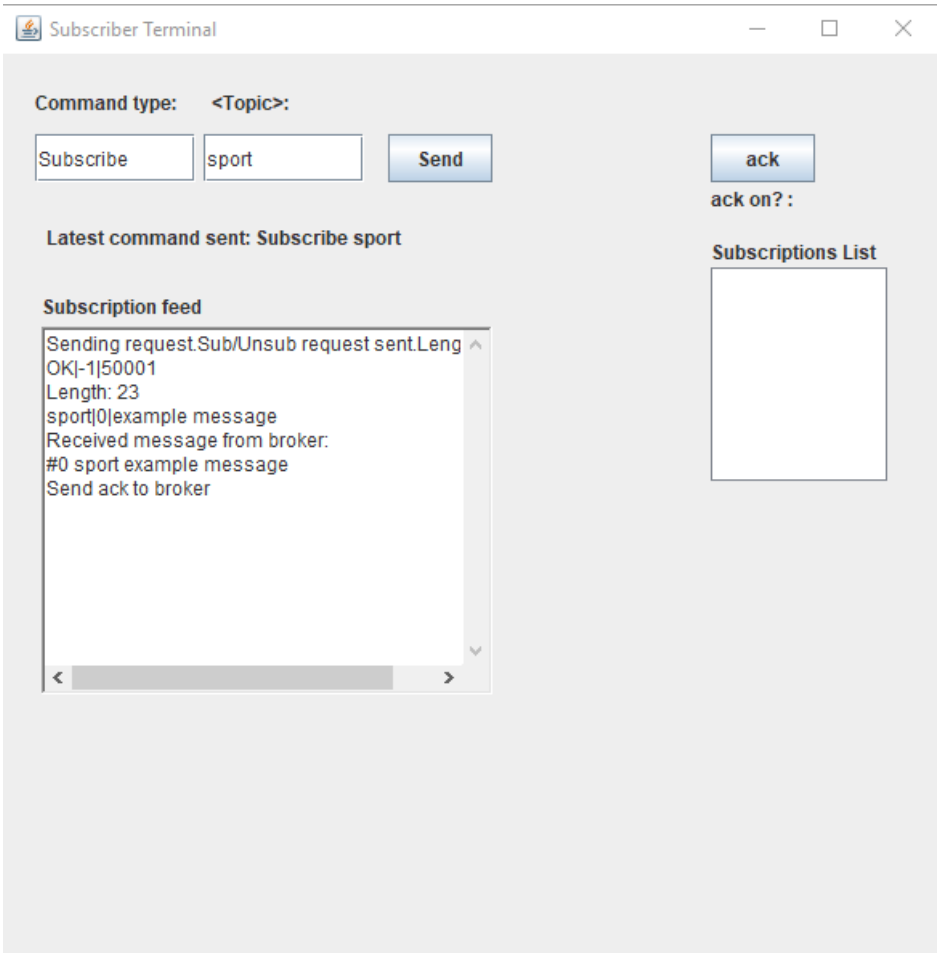

Figure 1: Subscriber implementation using SubTerminal.

Figure 1 shows the Subscriber terminal in action, as seen on the image, are the input fields "command type:" and topic together with the send button, the subscription feed, and the acknowledgement turn on/off button

2.2 Subscriber operation explained:

The subscriber is a child class of Node, which contains sendAck() and sendNack() methods, together with the methods that control the thread operation. Subscriber has fields such as:

- private Subscriber owner;
- constants for different default src and dst ports.
- ArrayList of strings : messagesReceived;
- ArrayList of strings : topics;
- int subscriberPort, delay, timeoutTime

The Subscriber implements three key methods to function properly

- synchronized onReceipt(Datagram packet)
  This method is a synchronized method which is only called on receipt of a datagram packet from the publisher via the broker, in all cases when a packet is received the thread is notified using this.notify(), the method checks whether the message received is a ack, "OK", or if it's a message from the broker, in the second case, the message is split by delimiter "—" and printed to the sub feed, with info such as topic, sequence number, message, and the full message is added to the messagesReceived arrayList.

- subscribe()
  The subscribe() method is called when the SubTerminal catches the event of the sendButton being pressed, this triggers the method to be called, it creates a new datagram packet, checks the command types where Subscribe : s and Unsubsribe : u, this char type field is placed in the packet, together with a string for the topic, the topic from the textField input is added to the topics list, and the datagram packet is sent to the default dst address which is the subscribers linked broker. The subscribe method sends update messages and confirmations into the terminal feed.

- start()
  When launched the program creates a new subscriber instance and launches it's start() method, it has an infinite loop that places the receipt thread on hold with the this.wait() method.

2.3 Subscriber Additional Information/ Edge Cases

- Wrong Subscriber input:
  If the user doesn't input a command which is equivalent to "Subscribe" or "Unsubscribe", the subscriber will send their message to the broker nonetheless, but assuming all acknowledgement settings are turned on the subscriber should receive, a "NOK" message telling them that their message was invalid, I will write later about a possible solution to this error checking scenario.

- Subscriber creation:
  In my implementation it's possible to create as many subscribers as you wish (where available ports on localhost are the limiting factor), the subscriber has a field for it's own port when dealing with acks from the publisher, and the port is set using command line arguments before launch of each subscriber.

- Standard Terminal Problems:
  I ran into a problem when starting off the project, using the default terminal provided, the problem was that I could not input data into the terminal, while also receiving a message or ack from the broker, this was caused by the thread being constantly in the process of reading in input, It required a manual enter key press to print-out one pending message in the pending messages list being formed, this was solved by using a textField input and a button for the event that launches the subscribe() method.

3 Publisher Analysis

3.1 Publisher components:

- PublisherTerminal class:
  THe publisherTerminal object is a terminal jframe implementation very similar to the subscriber implementation in terms of how it was made, It contains three text input fields, "Command", "Topic" and "Message", where command in our case is only valid for Publish, the topic and message fields are of finite length, limited by the packet size. The terminal also implements an ack on /off button which is used in the testing of the Go-back-N solution between the broker and publisher

- Publisher class:
  The publisher class, implements the methods needed for the functioning of the class, these will be explained in detail in the explanation section.

- Message class:
  This class is utilized, by the publisher to implement the go-back-N flow control, it represents messages sent out successfully by the publisher, it's instance variables are:
  public Timestamp timeSentAt;

  public DatagramPacket packet;
  public int recieverAddress;
  public boolean gotAck;
  public boolean readyToResend;
  public int sequenceNumber;
  public boolean gotNack;

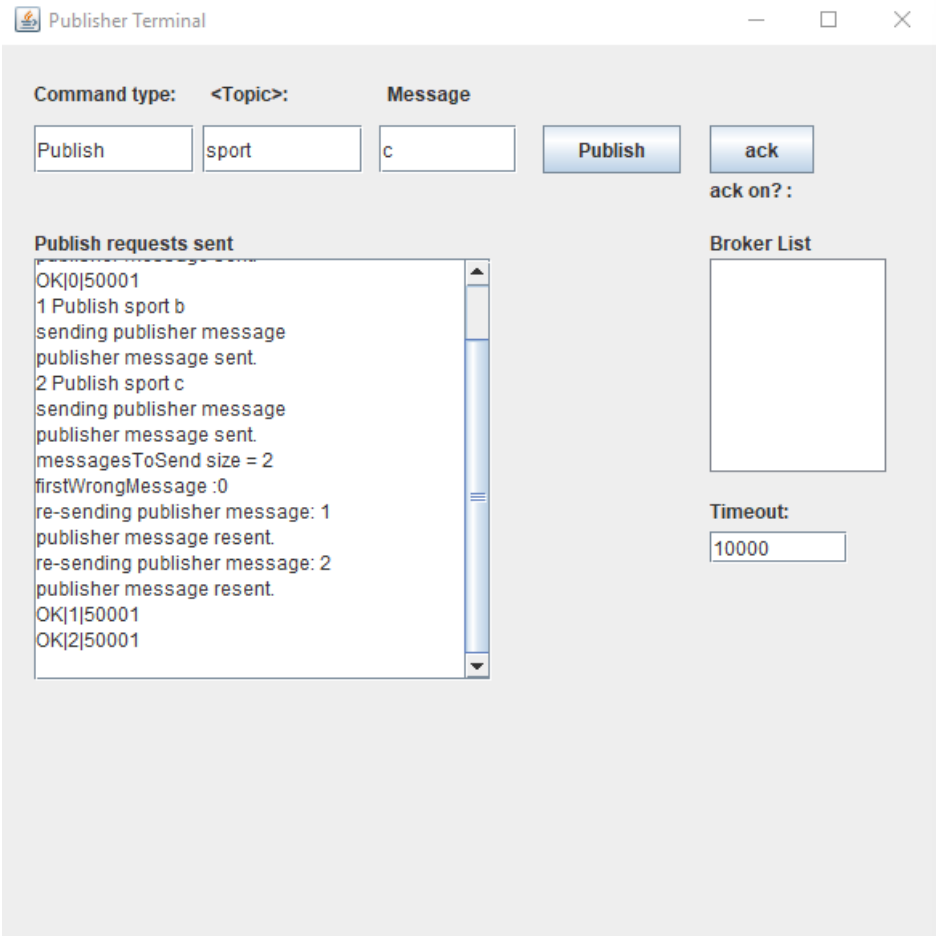

Figure 2: Publisher implementation using PublisherTerminal

Figure 2 shows a successful publish message sent for message with sequence number 0, it receives an OK—0—50001 acknowledgement message where OK is the acknowledgement, 0 is sequence number of the message being acknowledged, and 50001 is the port of the broker, However the next 2 messages with sequence numbers 1, 2, are at first sent but haven't received an acknowledgement, as the timeout runs out, the publisher sends out these messages again, this happens in a cycle until the publisher receives the acknowledgements for them as seen in the bottom of figure 2.

3.2 Publisher operation process:

The publisher is also a extension of the Node, class, which contains some of the thread methods, and the acknowledgement methods, It contains the following key instance variables:

- dest, src ports and localhost
  PublisherTerminal terminal;
  InetSocketAddress dstAddress;
  int sequenceNumber;
  ArrayList¡DatagramPacket¿ messages;
  int timeoutInterval = 10000;
  Timer t;
  ArrayList¡Message¿ window;

The publisher implements the following methods:

- synchronized void onReceipt()
  The onReceipt() method is for receiving packets from the broker, which in this case can only contain acknowledgements/negative acknowledgements, the acknowledgements consist of a type "OK"/"NOK", they have a sender port and a sequence number, all this info is gathered using a split by the delimiter "—", If a ack is received, the message in question is removed from the pending messages list, called ArrayList¡DatagramPacket¿ messages, but in the case when a negative acknowledgement is received, the messages in question gotNack field is set in the message object, this has no practical use yet, as there is no error checking system, but is a proof of concept for a possible error checking model for the publisher. Also the this.notify() method is called to awaken the operational publisher thread.

- synchronized void start()
  The start method in publisher is similar to the subscriber implementation, however it starts the timer, used in the go-back-N implementation, this timer is acted on in the onTimer() method.

- void publish()
  The publish method creates a datagramPacket, containing the information from the textFields, and is called on the pressing of the sendButton in the publisherTerminal. It also updates and increments the sequence number of the message being sent. It adds this newly pending message to the window arrayList of messages. This new message has a time-stamp to compare later if it is due to be sent again if it's still in the window, meaning no acknowledgement has been received.

- void publishResend()
  The publish resend method is similar to the publish() method, but it updates the timestamp, It is called in onTimer() on messages which are still pending and need to be resent.

- synchronized void onTimer()
  The onTimer() method is a periodically called method governed by a constant repeat time, It goes through all the pending messages located as a collection in the messages arrayList, and compares their launch time, with the expected time of receiving an acknowledgement using the timeout constant, if they have successfully received the OK as needed, they are removed from the pending messages list, but in the case where nothing has changed they are updated by publishResend() and continue in this cycle until they receive an ack relating to their specific sequence number. Only in this case are they removed and marked as successful messages.

- void removeMessageFromWindow()
  This is a helper method for onTimer() it removes specific messages in the case they have received their acknowledgements as needed, explained in the onTimer() description

- void nackUpdate()
  As mentioned in onReceipt() the nackUpdate() method sets a certain messages gotNack field to true.

3.3 Publisher Additional info / Edge Cases

- Nack vs Ack
  In our case in this project, the NACKs don't have a very good purpose, in terms of error correction as they don't trigger any methods for correcting anything, however they do tell me when the packet is incorrect.

- Go-Back-N
  In a traditional go-back-N system, you would need to have a window limit on how many messages, can be sent and repeated, for example the go-back-N system would cut out for messages more than 10 away, but in my case I don't have full frames, so It doesn't make much sense to enforce a limit, however it could be implemented easily as a proof of concept by limiting the size of the window arrayList to n, where n is a constant given at the launch of the program.

4 Broker Analysis

4.1 Broker components:

- BrokerTerminal class
  The broker terminal is similar to the subscriber and publisher terminal, although it doesn't require any input methods, as the broker is mostly self contained, it has a boolean for acknowledgements (button), and has a transcript to print out all logs of exchanges of packets

- Broker class
  The broker class, implements the functionality through the onRepeipt() and start() methods.

4.2 Broker operation process:

The broker implements the following methods, mentioned above:

- synchronized void onReceipt() The brokers onReceipt() method is it's main operational method, as the broker mostly receives packets and sends out acknowledgements based on the contents of the packet.
  The first check that is made is if the packet is an acknowledgement, in this case the current broker thread is notified by this.notify() and we leave the method. In the case where the first 2 bytes of the packet equal "s", or "u", implying it's a sub/unsub request, A new subscriberTopicsList is created for a new subscriber, their sub/ unsub request, is processes by adding / removing the desired topic from their personal subscription list. In the case of a successful removal / addition, the sender subscriber is sent an acknowledgement.

  In the case where the first two bytes equal "p" meaning it's a publisher request to publish, The publishers message is added to a queue, and an acknowledgement is sent.

  In the case where the packet is invalid, a NACK is sent to the address from which the packet was received.

- synchronized void start() This method functions while the queue containing message that need to be redistributed isn't empty, all these messages are split by delimiter "—" and redistributed, by doing a comparison of the packet topic to the subscriberTopicsList objects of all the current subscribers that are using this broker the processes requests are removed from the queue. After completing this action, the thread goes into sleep, waiting to be notified by another onReceipt() call.

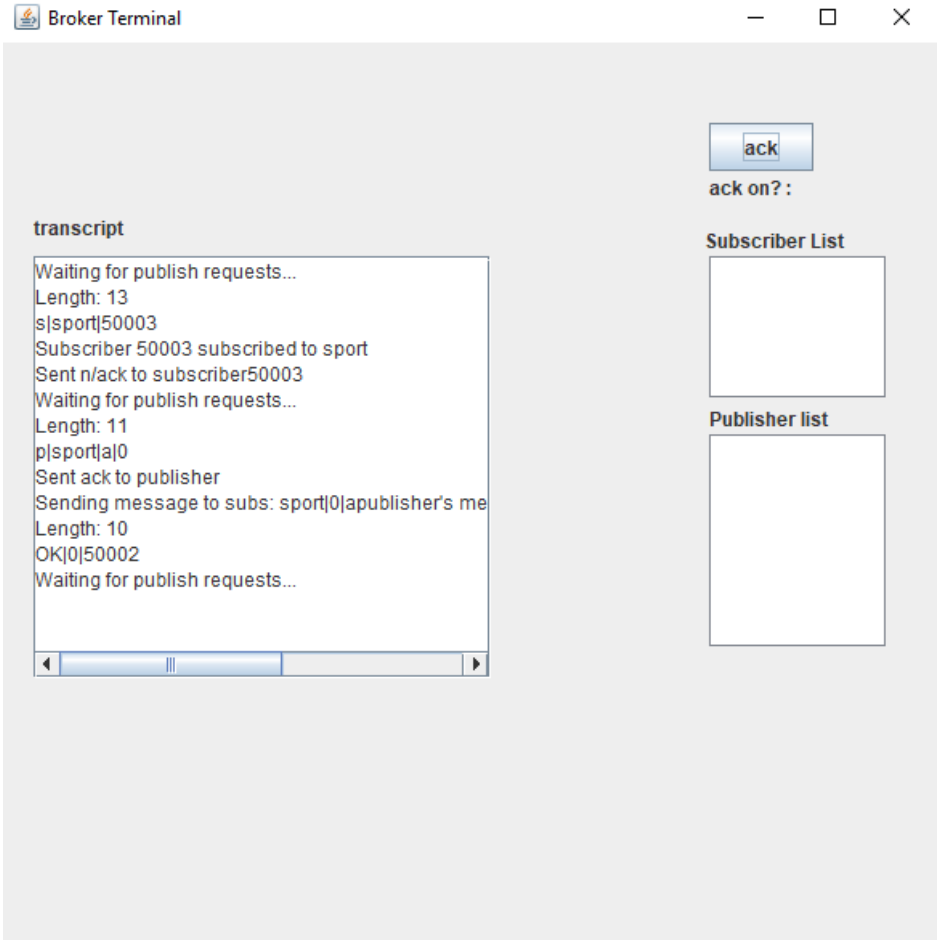

Figure 3: An example of a subscription request followed by the transfer of the accepted publishers message to the subscriber

5 Packet Types and Structure

5.1 Request type packets

- Subscribe/Unsubscribe request packet
  (s—topic) or (u—topic) with "—" delimiter
- Publish request
  (p—topic—message—seqenceNumber) with "—" delimiter

5.2 ACK/NACK type packets

- Subscribe/Unsubscribe/Publish ack packet
  ("OK"—-1—senderPort) with "—" delimiter
- Publish ack packet
  ("OK"—sequenceNumber—senderPort) with "—" delimiter
- Publish Subscribe/Unsubscribe/Publish nack packet
  ("NOK"—-1—senderPort) with "—" delimiter

The packets are constructed mostly using the textField input interface, or constructed by the sendAck/Nack() method in the case of the acknowledgement packets.

6 Go-Back-N implementation

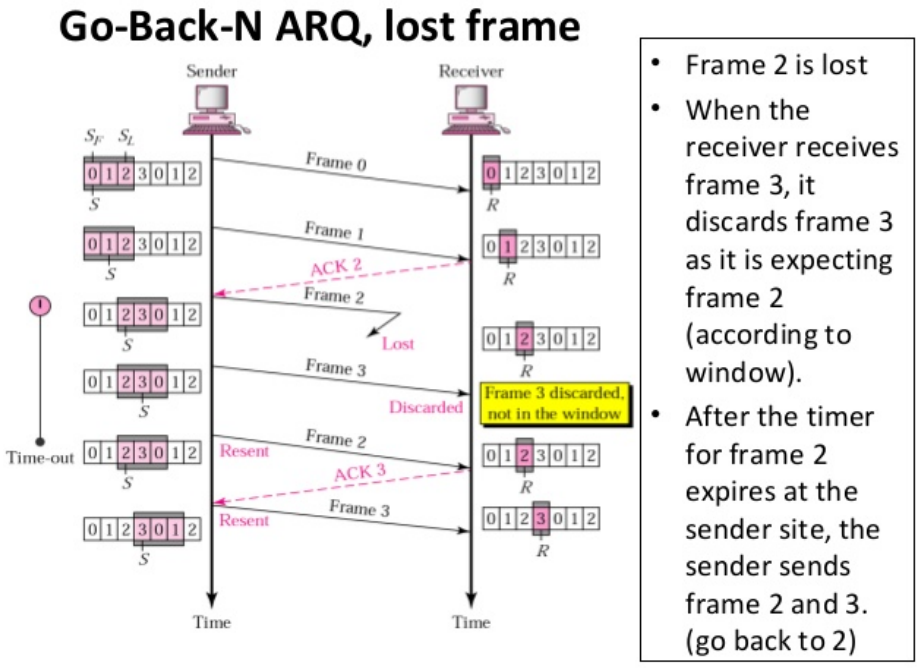

Figure 4: Go back N mechanism

- In my implementation the normal scenario of Go-Back-N happens, when as seen in figure 3, the ack button is set to true, in this scenario the packets come through by the artifical nature of the system without errors.

- However, when the ack buttons on the publisher and broker are turned to the false setting, if the publisher hasn't received an ack for the packets it has sent out within it's timout timer which starts with each packet sent, it re-sends the last packets in it's window size

- In this error scenario, the publisher will keep re-sending it's packets until it receives a positive ack from the broker that everything is ok.

- This mechanism is implemented in between the broker and publisher, the subscriber doesn't have this available in this version.

7 Assignment Reflection

7.1 Implementation Advantages

Extendable implementation in terms of amounts of broker systems, publishers and subscribers per broker.

Possibility to alter acknowledgement sending scenarios on the go

Go-Back-N protocol, makes sure no packets are lost

7.2 Implementation disadvantages

A existing protocol was not followed closely in the design of the data packets and the ack/nack packets, In this field there are many improvements to be made in terms of saved memory/space and efficiency

In this implementation it was too complex to implement a suitable multiple broker system, where publishers and subscribers chose which broker to connect to, and further options such as subscribers using multiple broker networks.

7.3 Design Considerations Time Spent

If I was starting again, I would think more at the start of the project about packet structure and following a set protocol, in terms of packets, this would standardize the project more,

Roughly 20-22 hours was spent writing the code, Due to the new nature of the all the libraries and concepts, and thinking of the implementations together with a few days on and off of writing this report.