Trinity College
The University of Dublin

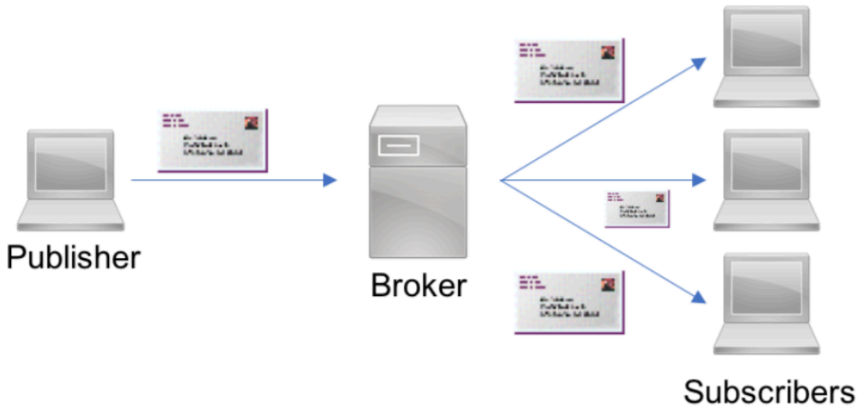
Telecommunications Assignment

Assignment 1 - Publish-Subscribe

Senán d'Art

1. Introduction

The problem description for this assignment was to create a Publish-Subscribe system. This is a protocol involving 3 types of entities: publishers, brokers and subscribers. The task of a publisher is to send messages to a broker, the broker must then forward this message to the appropriate subscriber/s. The subscribers must be able to subscribe to various topics, this should be done through communication with a broker.



There were further optional extensions to include some method of flow control and the capability of having multiple Publishers and Brokers.

2. Theory of Topic

2.1 *Sequence Numbers*

Sequence numbers are an indication of the order in which packets have been sent. They allow for packets to be realigned into the correct if they arrive at different times. They also allow for the implementation of flow control mechanisms such as Selective Repeat.

2.2 Timeouts

When a packet has not been responded to in a certain amount of time (either ACK or NAK), it is presumed lost and is resent. This is done on a packet-by-packet basis. See Figure 1 for an example.

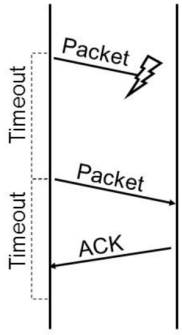


Figure 1. An example of packet timeout. The packet is lost in transmission and when no response is receiving in the timeout window, the packet is presumed lost and resent.

2.3 Flow control

Flow control refers to any method of limiting the amount of data that is sent in order to avoid overflowing the receiver. Flow control with Automatic Repeat Request (ARQ) describes methods used to handle and recover from lost packets.

2.4 *Selective Repeat ARQ*

Selective repeat is a flow control and error recovery mechanism. It allows for efficient bandwidth usage and the resending of lost packets. The diagram below (**Figure 2**) shows the flow of packets in a system using Selective Repeat ARQ.

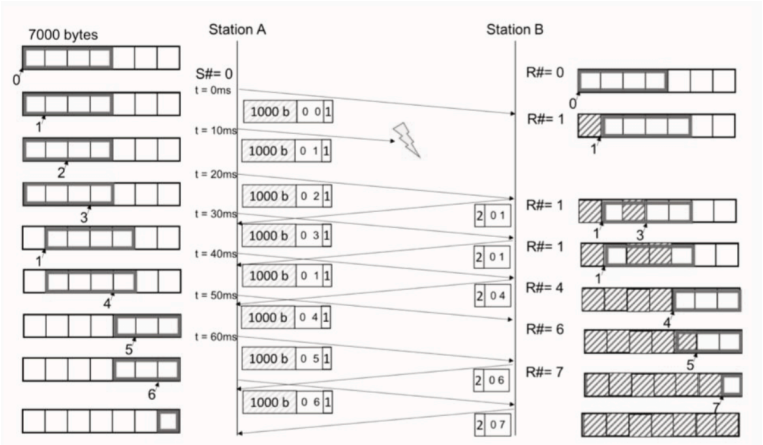


Figure 2. A diagram of selective repeat in operation. Note what occurs when a packet goes missing. Src : PPP-State Diagrams - S. Weber

2.5 *Header*
A header is a piece of data that is attached to the start of a packet. The header contains various pieces of metadata such as the intended recipient of the packet, the length of the packet, etc.

2.6 *Publisher*
The publisher had the task of accepting a topic and message as an input. This was then transmitted to a broker.

2.7 Broker
The broker was tasked with receiving transmissions from a publisher and forwarding these to the appropriate subscribers.

2.8 *Subscriber*
The subscriber was required to have several abilities: the ability to subscribe to and unsubscribe from topics as well as the ability to receive messages from the broker.

3. Implementation

3.1 *User Interface*

The user interface was implemented as a simple Command Line Interface (CLI). The interface accepts an input of either a Topic and Filename in the case of the Publisher, or a command eg subscribe, etc. in the case of the Subscriber.

3.2 *Communication*

3.2.1 Packet Structure

The data within the packet is structured as follows:
1 byte : Packet Type, ranging from 0-11
1 byte : Sequence Number, ranging from 0-10 during normal operation but is -1 for Start Packet
0-1500 bytes : Data, usually a string that is being sent

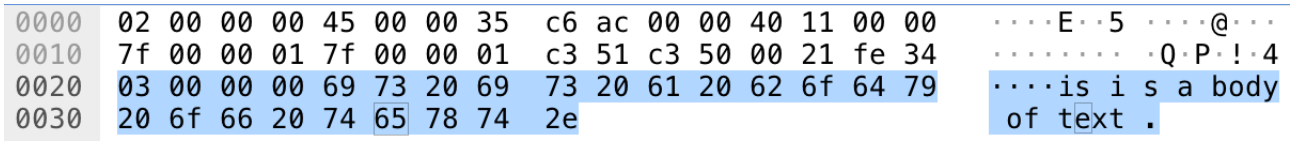


Figure 2. A captured data packet. The first byte of the highlighted section shows the packet type: 3 = Data. The second byte shows the sequence number: 0 = first packet in sequence. This is then followed by the data contents of the packet.

3.2.2 Packing Data Into Packets

If the file being transmitted is larger than the maximum size allowed by a single packet, the data is split into multiple packets. The maximum size of data contained in the packets is limited to 1500 bytes. The file is temporarily stored as a single string, this is then split into byte arrays of size 1500. Sequence Numbers and Packet Type are then added as individual bytes.

```
private byte[][] splitStr(String theString) {// split a string into a series of byte arrays
    byte[] strByteArr = theString.getBytes(Packet.DEF_ENCODING);// string -> byte[]
    int noOfStrings = strByteArr.length / Packet.MAX_LENGTH_BYTES;// number of byte[] required
    if (strByteArr.length % Packet.MAX_LENGTH_BYTES != 0)// check for a shorter byte[] on the end
        noOfStrings++;
    byte[][] resByte = new byte[noOfStrings][];// new byte array array
    int start = 0;
    // if the length of the array is less than the max, use it as the max
    int end = (strByteArr.length < Packet.MAX_LENGTH_BYTES) ? strByteArr.length : Packet.MAX_LENGTH_BYTES;
    for (int i = 0; i < resByte.length; i++) {// for every byte array
        resByte[i] = Arrays.copyOfRange(strByteArr, start, end);// copy section of byte[]
        start += Packet.MAX_LENGTH_BYTES;
        end = ((end + Packet.MAX_LENGTH_BYTES) > strByteArr.length) ? strByteArr.length
                : (end + Packet.MAX_LENGTH_BYTES);
    }
    return resByte;
}
```

Figure 3. The function used to split a string into multiple byte arrays. It first determines the number of arrays required. Then it converts the input string to a byte array. Section of this array are then copied to other arrays which are then saved as a result.

3.3.3 Communication Protocol

When a node wants to start communicating, the following sequence is followed:
1) Transmitting node sends START request to the target node.
2) The receiving node responds with a START_ACK packet in order to confirm it is ready to receive.
The transmitting node is now free to send its payload:
In the case a data transmission this occurs as follows:
3) Transmitting node sends all packets in window.
4) Receiving node sends ACK for packets received and NAK for packets missing.
5) Repeat 3 & 4 as required.
6) Transmitting node sends END packet with sequence number of last packet.
7) Receiving node sends END_ACK if sequence numbers line up correctly and transmission is complete. Otherwise receiving node sends NAK for each missing packet and return to step 3.
8) Transmission complete.
In the case of subscription or un-subscription:
3) Transmitting node sends SUB or USUB packet.
4) Receiving node responds with SUB_ACK.
5) Transmission complete.

3.2.4 Timeouts

Timeouts are handled by Frame objects. These contain a packet and use the Java.util.Timer class to automatically resend packets when their timer expires. These timers are cancelled if a response is received.

3.2.5 Selective Repeat ARQ

The implementation of Selective Repeat ARQ relied on the use of an array of Frame objects. The range of the active frames in the window was limited to 5 slots ie. there were a maximum of 5 active frames at a time. The index of the item in the array was the same as the sequence number of the packet. When an ACK was received for a packet: its timeout was cancelled, the range of active frames was incremented and another frame was sent. (See **Figure 2.** for example)

4. Summary

My implementation includes:
A publisher that communicates with a broker.
The ability to create multiple publishers.
A broker that accepts communication and retransmits data to subscribers.
The ability to create multiple brokers.
Several subscribers that can subscribe to and unsubscribe from various topics, and can also receive messages relating to those topics from the broker.
Selective Repeat ARQ with automatic timeouts and lost packet recovery.
Extremely large transmission sizes as data is split into shorter sections which can then be transmitted and recombined at the receiver side.

5. Reflection

Overall I am happy with my implementation. This was my first time using multiple execution threads and I believe it went quite well. Some of the code is not as efficient as it could be, for example the implementation of Selective Repeat is a bit bloated and inefficient, the implementation of creating byte arrays from data is also slightly cumbersome, if I were rewriting this I would use a different method than manually modifying byte arrays.
Excluding the report I spent roughly 30 hours on this project.