



Assignment #1: Publish-Subscribe

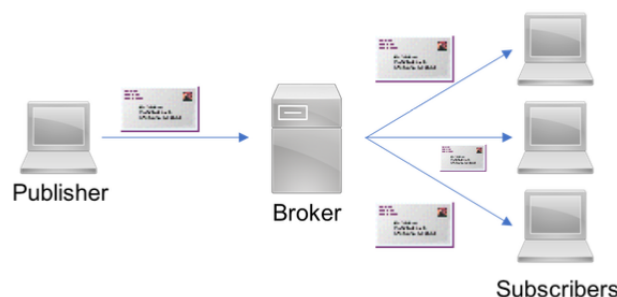
Ruth Brennan
17329846
31st December 2018

Contents

INTRODUCTION	1
DESIGN	2
I. <i>FRAME LAYOUT</i>	2
II. <i>PUBLISHER AND BROKER</i>	4
III. <i>BROKER AND SUBSCRIBER</i>	5
IMPLEMENTATION	5
I. <i>PUBLISHER</i>	5
II. <i>BROKER</i>	6
III. <i>SUBSCRIBER</i>	6
SUMMARY	6
REFLECTION	6

Introduction

The outline for this assignment described a publish-subscribe protocol. In which messages published by a publisher and are forwarded onto subscribers by a broker, if the subscriber has subscribed to the topic of the message.



^from assignment description^

The assignment also had a feature list which detailed


- Subscribers who would
 - Accept a topic as input
 - Send a subscription message to a broker
 - And print published messages that have been forwarded by a broker in order of sequence number
 - Send an unsubscribe message to a broker
- Publishers who would
 - Accept a topic and a message as input
 - Transmit a packet with the topic and message to the broker
 - Give messages sequence numbers
- Brokers who would
 - Receive subscribe/unsubscribe messages
 - Maintain lists of subscribers to various topics
 - Forward incoming messages to the subscribers with matching topics
- Acknowledgments
 - Brokers and Subscribers may implement acknowledgements
 - Publishers may wait for an acknowledgement from a broker before proceeding to accept input of another topic and message.

Design

The design is primarily based on the criteria for the assignment following all the specifications detailed above.

I. Packet Layouts

Transmitting Data

Control	Length	Payload
<1 byte>	<2 bytes>	
		

Control

Code Bits (6,7) – These bits are used to indicate the role of the sender of the frame. The codes work as follows: 01 from broker (acknowledgement), 10 from subscriber (subscription), 11 from publisher (data message). 00 would be treated as an error.

Sequence Bits (3,4,5) – These three bits are used to indicate the sequence number of the message being transmitted.

Acknowledgement Sequence Bits (0,1,2) – These bits are used to piggyback acknowledgements. The frame sent carries the sequence number for the next frame the broker should expect to receive.

Length

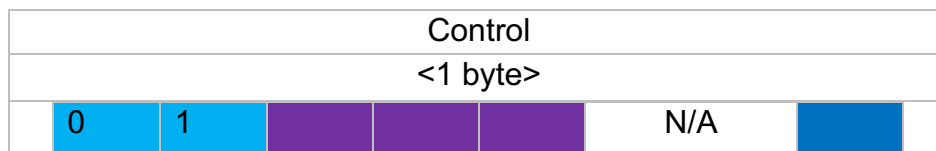
The 2 length bytes are used to transmit the size of the payload. Both the length and the Control bytes are the header of the frame.

Payload

This is where the data that is being transmitted is.

This was my original design choice for a packet. The others are variations of this and hence have some gaps that I would have preferred to avoid.

Acknowledgments



Control

Code Bits (6,7) – These bits are again used to indicate the role of the sender of the frame. The codes work as follows: 01 from broker, 10 from subscriber, 11 from publisher. 00 would be treated as an error.

Sequence Bits (3,4,5) – These three bits are used to indicate the sequence number of the message being acknowledged.

N/A (1,2) – Indicates the bits that are not used in this frame.

Subscribe/Unsubscribe Bit (0) – This bit is used to indicate whether a subscribe or unsubscribe message is being acknowledged. If it is set it indicates a subscription message and if not it indicates an unsubscribe message. If this bit is being used then the sequence bits are not and vice versa.

Subscribe/Unsubscribe

Control		Length		Payload	
<1 byte>		<2 bytes>			
1	0	N/A			

Control

Code Bits (6,7) – These bits are again used to indicate the role of the sender of the frame. The codes work as follows: 01 from broker, 10 from subscriber, 11 from publisher. 00 would be treated as an error.

N/A (1,2,3,4,5) – Indicates the bits that are not used in this frame.

Subscribe/Unsubscribe Bit (0) – This bit is used to indicate whether a subscribe or unsubscribe message is being transmitted. If it is set it indicates a subscription message and if not, it indicates an unsubscribe message.

Length

The 2 length bytes are used to transmit the size of the payload. Both the length and the Control bytes are the header of the frame.

Payload

This is where the data that is being transmitted is.

II. Publisher and Broker

All three of the Publisher, Broker and Subscriber classes act as nodes. This was so that they could make use of the onReciept function and the thread to which it belongs.

The main objectives of the publisher are to

1. Accept topic and message as input
2. Transmit the topic and message to broker

The main objectives of the broker are to

1. Forward incoming messages to subscribers with matching topics

This program takes a Stop and Wait approach to flow control. This means that once the publisher takes the topic and message as input, creates a packet and sends it to the broker it will not accept another packet until the previous one has been acknowledged. The Broker then deals with the message by comparing the topic to it's list of existing topics if the topic already exists the Broker attempts to forward the message to those subscribed to the topic. If the topic doesn't not already exist then the topic is added to the Broker's list of topics and the message's journey ends there, as they're no subscribers subscribed to the topic if it has not been seen before.

This was a design choice I made to preserve memory. I felt it would have been wasteful to store messages in hopes that their topic may be subscribed to at some point.

III. Broker and Subscriber

The main objectives of the broker are to

1. Receive Subscriptions
2. Maintain list of subscribers

The main objectives of the subscriber are to

1. Accept Topic as input
2. Send subscription message to broker
3. Print published messages from the broker

The subscription protocol again takes a stop and wait approach to flow control. The subscriber takes a topic as input, then asks whether the user would like to subscribe/unsubscribe. It then forms the subscribe/unsubscribe packet and sends this to the broker. The Subscriber will only proceed to other tasks once an acknowledgement has been received.

The subscriber will also show messages that have been received from the publisher in order on request. The broker forwards the messages as soon as it receives them from the publisher but the subscriber will not print them until requested.

Implementation

This section presents the implementation details of the individual publish-subscribe elements, a publisher, a broker and a subscriber.

I. Publisher

This class extends Node and has two main synchronized methods: onReciept and publish. Publish runs in a continuous loop to allow for the publishing of multiple topics and their subsequent messages.

For this program I make use of the tcd.IO library an all reading in and output are done via the terminal. So once the topic and message have been read in, the header has to be formatted, into the data transmitting layout from above, and it is sent via socket to a broker. The calling thread is then told to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify(). This happens in the onReciept method. This is how the stop and wait flow control method is implemented as the calling thread will not be notified until an acknowledgment packet is received from a broker.

A note on creating multiple publishers, with my approach creating multiple publishers is a rather straight-forward practice. However each publisher is 'related' to a broker so upon initialization of the publisher the address of this broker must be given.

II. Broker

This class again extends Node and has one main method; onReciept. This is where all the action happens. This method is called when a packet has been received and it then handles the packet as described earlier. If it is a subscription packet the Broker will add/remove the address of the subscriber that sent the packet to it's topics HashMap. If it is from the publisher, it will forward the packet based on it's topic to the subscribers of that topic. Using the InetSocketAddresses stored in the HashMap.

III. Subscriber

This class again extends Node and has two main synchronized methods; subscribe and onReciept. It works similarly to a publisher but with an entirely different functionality. Subscribe runs in a loop. This allows the user to complete a variety of tasks. At the beginning it presents the user with an option to either view messages or subscribe/unsubscribe to a topic. If the user chooses to subscribe they are asked to provide a topic and specific subscribe/unsubscribe. This is then formatted into a subscribe packet layout and sent via a socket to a broker.

However, if the user chooses to view messages then the calling thread is then told to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify() or the time is elapsed. This notification happens in the onReciept method. And this process is repeated until the time in one of it's invocations elapses.

Summary

This report details my attempt at a solution to a publish-subscribe protocol. While the implementations of the protocol are rather straight-forward, it demonstrates my understanding of sockets, DatagramPackets and the protocol. The description of the implementation in this document highlights the essential components of my solution on a topological level.

Reflection

I found this assignment of great value in understanding protocols and know that it will be very useful when studying concurrent programming in the future. Excluding the time it took to write this report I would say that the assignment took about 28 hours to complete.