Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU33031 Computer Networks
Assignment 1

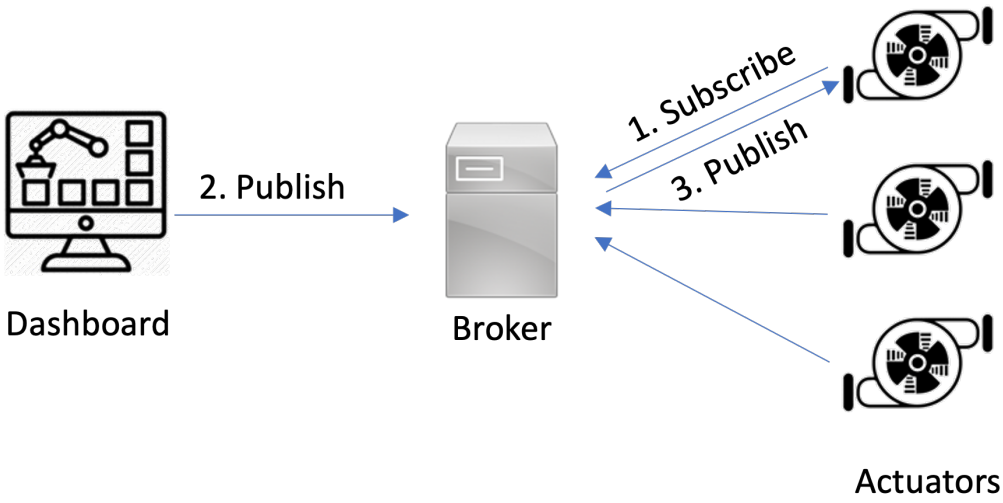
Liam Junkermann - 19300141

November 2, 2021

Contents

1 Introduction

The problem described involved creating a Publish-Subscribe system. In order to complete this, the necessary components needed to be built: the broker, the subscribers, and publishers. The publishers send data to the broker, which then processes and routes the traffic to the appropriate subscribers. The subscribers are allowed to subscribe to a variety of topics, transactions which are also handled by the broker. Through this assignment, the context of a system with various sensors and a dashboard as shown below (from assignment declaration).



2 Design and Implementation

A very basic design approach was used. Given that the scope of this project was fairly limited (the context of a few sensors and dashboards) the setup of a prospective system was straight forward with the three clients as described above being built: The broker, the publishers – in this case the Command and Control (C&C) module, and the subscribers – in this case the Client.

2.1 The Packet structure

Each packet sent is of the same structure and length. The packets each have a header length of 2 bytes which can be assigned to different values depending on the context. There are two types of packet: acknowledgement and data packets. The Data packets are typed by their sender, so any packets from the Broker, Client, or C&C carry their own value in the first header position. Data packets include data length in the second header byte, while acknowledgement packets have their acknowledgement or acceptance state stored in the second byte of the header.

2.2 The Communication Method

This entire system is written in Java and relies on the `java.net.*` packages. All data is transferred through `DatagramPackets` with connections to the sockets opened by the nodes. In order to complete packet transactions the source node connects to the target nodes open socket. These socket addresses are hard-coded for the Broker and C&C modules, the Client nodes are assigned sockets randomly due to the current topology (see Deployment Topology).

2.3 The Broker

The broker is the most complicated part of this service, as its purpose is to handle the traffic from all publishers and subscribers, as well as maintain the packet format. At its core the Broker is just another node, with a

Listener for packets, which are then parsed and responded to accordingly. The Broker receives packets from the clients (publishers and subscribers), then respond with an acknowledgement, and if the packet was received from a publisher, the relevant subscribers will then be notified. The Broker sends two types of packets:

Packet Receipt Acknowledgement packet The first packet the Broker sends is an acknowledgement type packet, declared in the first byte of the header. This packet is sent in response to a packet from either the C&C Module or the Client module to let them know that the packet has been received. In the case of a packet from the C&C module being made, this triggers the relevant data packets being sent to the subscriber list as well.

Broker Data Packet The other packet the broker sends is a Broker Data Packet, this is defined in the header, with the data length being described in the second header byte. Finally the string value of the C&C command is added to the data body of the packet, to be sent to the subscribed clients.

2.4 Command and Control

The Command and Control (C&C) module utilises the interactive `Terminal` prompt in order to allow the user to issue commands to the necessary clients list. The C&C module issues one type of packet:

Command and Control packet The C&C packet is sent to the broker with the data headers set to the control value for a C&C packet and the data length, in each of the header bytes respectively. The packet data is the command which will be relayed to the various listening subscribers.

2.5 The Client

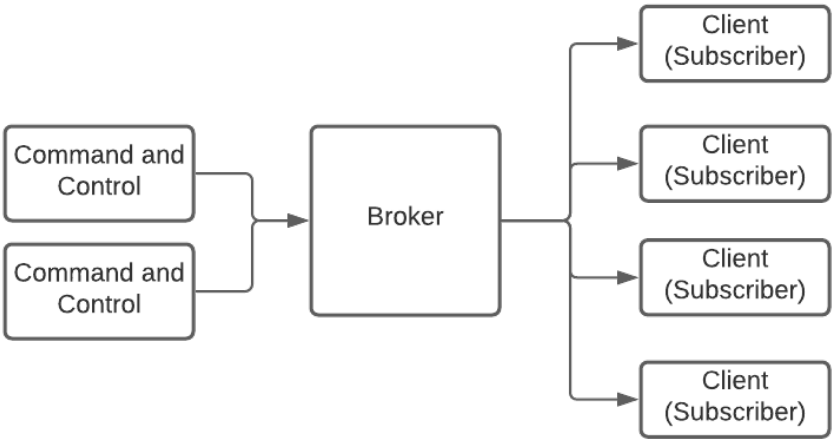
The Client's sole purpose is to subscribe to the brokers subscription streams, and in a real world application, would use the subscription data to trigger events either with an actuator of some kind. The Client still sends two types of packets:

Connection Packet The client – when launched – immediately begins to try to make a connection with the broker in order to subscribe to a stream. The connection packet attempts to send a packet to the broker to register for the subscription list on the broker side, this then expects an acknowledgment packet response which is confirmed to the actuator through the `Terminal`.

Command response packet When the client receives a command from the Broker it must acknowledge whether it can accept the job or not. This packet is typed as a `Client` packet, with the response data length flag set in the second header position.

3 Deployment Topology

A very basic deployment approach was used for the duration of this project. Each of the components described above (The Broker, Command and Control, and The Client) were deployed in their own docker containers all on their own Docker managed virtual subnet. There was a fourth container which hosted Wireshark to allow the capture of `.PCAP` files, due to some of the limitations of virtual networks within MacOS. Each of the containers was named accordingly to allow the hostnames to be hard-coded into the necessary class files (The Broker as `broker`, Command and Control as `CAndC`, and The Client as `client`). Currently, there are multiple clients running on the same `client` container. The Client class generates 3 clients for testing purposes which all utilise the same container resources.



4 Summary and Reflection

Building this Publish-Subscribe (pubsub) system has been a great learning experience and has allowed me to utilise skills and technologies I have had experience with outside of college to properly work together and produce an interesting final product. Next I will discuss the strengths and weaknesses of the outcome, as well as reflect on my learnings.

4.1 The systems strengths and weaknesses

The current implementation of the pubsub is very limited. While there was a lot of learning as a result of building this and it works as intended, there are many more features that could have been added to improve or extend the current implementation. Pubsub systems are used regularly in IoT systems with lots of sensors and actuators which work together to control or enhance environments, for example in smart homes. In general, pubsub systems are great when reliability is not crucial to the rest of the system, socket based applications tend to have some packet loss and latency and a broker might not even know it is missing packets from publishers as there is no other mode of communication between the publishers, brokers, and subscribers. That being said, pubsub systems – like this one and more developed implementations like MQTT – generally have lower overheads than a `TCP` based method with more handshakes and assurances that connections have been made, this makes them quicker in transferring data as well, something that is important when generally small data packets are sent to populate a sensor output, for example.

4.2 Learnings from Assignment 1

Having done some work with Docker and Docker clusters in the past I had a basic knowledge of how networking socket applications worked, this made it easier for me to understand how to work with the pubsub socket system. There were two main struggles when completing this project:

MacOS and Docker Networks Because of the way recent releases of MacOS and it's security has been built each application has it's own security container which has limited access to the rest of the machine, this includes Docker. In practice this means that any networks that Docker creates are actually just virtual networks within a virtual machine container managed by MacOS. This meant that Wireshark running on the host machine did not have access to the virtual network created to host the collection of Docker containers which deployed the pubsub system. Unfortunately, it took me far too long to figure out that this was the issue, but once I isolated this problem I was able to fashion a solution using X11 and Wireshark running from a container within the Docker network, allowing me to create the necessary `.PCAP` files.

Broker not receiving packets For the longest time, I was having an issue with packets being sent (and picked up by wireshark), but not making it to the Broker – and as a result not coming back to the C&C module that sent the first packet. After much debugging I found that I did not correctly implement the `Node` module provided, and had missed two lines:

```
1                   listener.setDaemon(true);
2                   listener.start();
```

These two lines were vital to actually listening for packets and once this issue was resolved my implementation worked great.

This assignment was very interesting and allowed me to put into context – and have a much better understanding of – some other projects and tools I've used for personal projects.