# **EECS 6381: Distributed Systems Principles**

**Handed out during the live session of week 2; due 24 hours before the live session of week 5**

**(Team-based effort and single submission per team)**

**Single Broker-Based Publish-Subscribe Using ZMQ and Mininet**

In this assignment we will build upon the PUB/SUB model supported by the ZeroMQ (ZMQ) middleware. One of the downsides of ZMQ’s approach is that there is no anonymity between publishers and subscribers. A subscriber needs to know where the publisher is (i.e., subscriber must explicitly connect to a publisher using its IP address and port). So, we lose some degree of decoupling with such an approach (recall the time, space, and synchronization decoupling that we studied). A more desirable solution is where application logic of the publishers and subscribers remains anonymous to each other; naturally something else will still need to maintain the association. This entity is the pub-sub middleware, which is the focus of this assignment.

In this assignment, we will build a small additional layer of pub/sub middleware on top of existing ZMQ capabilities to support anonymity. See the diagram below of the expected abstraction.

Application Logic

CS6381 API

Underlying ZMQ

Feel free to use the underlying capabilities of ZMQ that you feel appropriate. For instance, the REQ-REP and PUB-SUB are the most likely features of ZMQ you will need. No matter what, the ZMQ operations should be hidden from your publisher and subscriber application logic. The application code should use the APIs of your middleware, which then translate into ZMQ calls under the hood.

We will maintain a broker that we assume is going to be well known to the publishers and subscribers. It is this broker that then does the matchmaking. Once the match is made, you have two choices as to how data should be disseminated from publishers to subscribers. Write your code in such a way that the approach is globally configurable across the system to use one of the two approaches when the system is deployed.

1. Have the publisher’s middleware layer directly send the data to the subscribers who are interested in the topic being published by this publisher. This means that, as the number of subscribers for this topic increases, the publisher has to send data to each of the subscribers.

OR

1. Have the publisher’s middleware always send the information to the broker, which then sends it to the subscribers for this topic. Such an approach saves the amount of data sent by the publisher but clearly can make the broker a bottleneck as the number of publishers and subscribers in the system increases.

Ensure that your code is designed with extensibility in mind because subsequent assignments are going to enhance this basic design. **We will use Python3 for our assignments.**

After writing the code for the assignment, you should be able to conduct a variety of end-to-end performance measurement experiments to get a sense of the impact on amount of data sent, latency of dissemination, pinpointing the source of bottlenecks, etc. Automation in this testing process is preferred so that you do not have to manually set up each experiment.

**API Specification**

One might consider an API that your middleware provides as described below.

First, both publishers and subscribers will require an API to talk to the broker, for which we might use something like this:

**Publisher uses this to register itself with the broker for a given topic it publishes.**

register\_pub (topic = <some string>, <some identification of the publisher>)

**Subscriber uses this to register itself with the broker for the topic it is interested in.**

register\_sub (topic = <some string>, <some identification of the subscriber>)

Since a publisher application may be publishing different kinds of topics, it will use the following API to publish its publication:

publish (topic = <string>, value = <val>)

Subscriber application can have a notify method, which is a push-based method as follows:

notify (topic = <string>, value = <val>)

Publisher appln code

CS6381 Middleware

ZMQ Middleware

Broker appln code

CS6381 Middleware

ZMQ Middleware

Subscriber appln code

CS6381 Middleware

ZMQ Middleware



**Use of Github and Grading Criteria:**

We will be peer grading the assignments. Each team will be assigned a team number. A numbered team will grade the team number one higher than theirs modulo number of teams (so last team grades team 1). **Thus, T1 grades T2; T2 grades T3; T3 grades T4; T4 grades T5; T5 grades T6; and T6 grades T1.** Proper unit tests must be created to test various features. The unit tests should be supplied by a team to their peer grading team. The peer grading team must evaluate whether the supplied unit tests will test all (maximal) number of features.

I suggest all teams use github for their code base. Please share your CS 6381-specific contributions with me in addition to your grading team. **Your instructor will provide his/her Github handle during a live session.**

**Assignment-Specific Demonstration:**

* Should work with all the properties mentioned in the assignment.
* Should work with multiple publishers publishing multiple different publications and multiple subscribers, all of them distributed over different hosts over different kinds of network topologies that we can create in Mininet (instructor may provide their topologies).
* Subscribers may join any time and leave any time.
* Do end-to-end measurements (time between publication and receipt of information; since the clock is the same on all emulated hosts, we do not have the issue of clocks drifting apart from each other). Plot graphs and have them saved in the repo that you maintain.

**Rubrics: Total 100 Points**

Please refer to the generic programming assignment grading guidelines for the grading rubrics. Thus, ensure that your assignment fulfills all the requirements as specified and meets the rubrics requirements including the README.

**Feedback to Instructor**

The “grading team” should send a detailed report to the instructor on how the submitted assignment met the grading requirements and grade received by the submitting team along the specified rubrics they used. They should also get a sense of the effort put in by each team member.