EECS 6381: Distributed Systems Principles, Spring 2018

Instructor: Aniruddha Gokhale (a.gokhale@vanderbilt.edu)

Handed out: 02/01/2018; Due in 3 weeks, midnight 11:59 pm (team-based submission)

Publish-Subscribe using ZMQ and Mininet

In this assignment we will build upon the PUB/SUB model supported by ZMQ. One of downsides of this 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). So we lose some degree of decoupling with such an approach.

A more desirable solution is where publishers and subscribers remain anonymous. Moreover, it is also possible that there could be more than one publisher publishing similar information in which case there is a possibility that subscribers will receive the same information multiple times (one each from publishers publishing the same information). Some pub/sub solutions like OMG's Data Distribution Service support the idea of OWNERSHIP and OWNERSHIP_STRENGTH. So whichever publisher's OWNERSHIP STRENGTH is the highest wins among all of them. If that publisher dies, the next one in line becomes the leader and its publications make their way to subscribers, and so on.

Maintaining HISTORY is also an important artifact of pub/sub systems. This enables late joining subscribers to acquire the last N number of samples because the publisher is required to maintain a sliding window of N samples.

In this assignment, we will build a small layer of pub/sub middleware on top of existing ZMQ capabilities to support anonymity, OWNERSHIP_STRENGTH and HISTORY (parametrized by N, where N is the amount of past history a publisher publishes). Feel free to use any other capability of ZMQ that you feel appropriate. But ZMQ operations should be hidden from your publisher and subscriber. They should use your APIs, which then translate into ZMQ calls under the hood.

API

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

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

Publisher uses this to register itself with the broker.

```
register_pub (topic = <some string>,
              ownership_strength = <some default, say 0 meaning system gives default>,
              history = <some default, say 0 meaning no history>
```

Subscriber uses this to register itself with the broker.

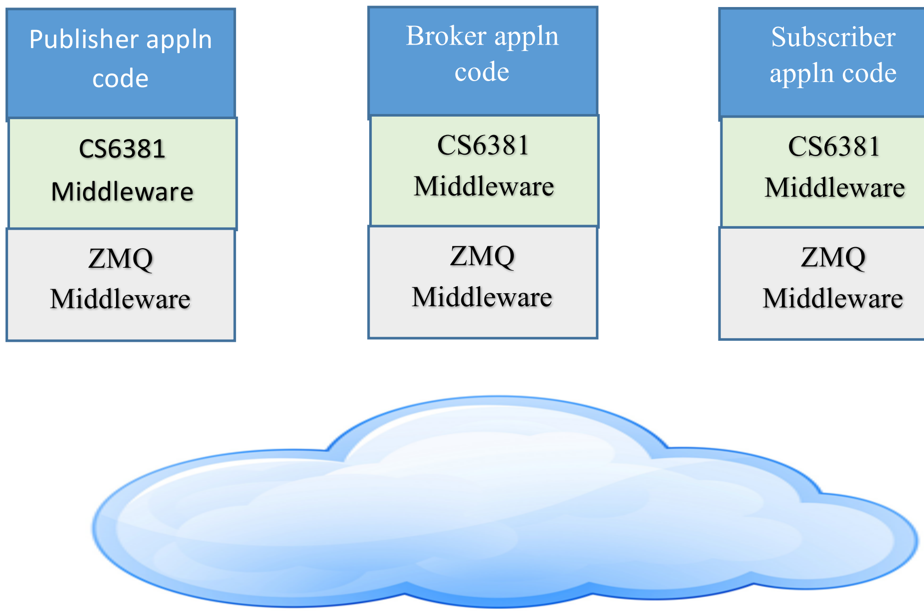
```
register_sub (topic = <some string>,
              history = <some default, say 0 meaning no history>
```

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

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



Demonstration:

- Should work with all these properties including failing publishers
- 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 (I might supply some topologies)
- Subscribers may join any time and leave any time
- Do end-to-end measurements (time between publication and receipt of info; since the clock is the same on all emulated hosts, we do not have the issue of clocks drifting apart from each other).

What to submit?

Source code, unit tests, results; README file indicating how to compile and run the code.

Use of github and Evaluation:

Since we will be peer grading, a numbered team will grade the team number one higher than theirs modulo class size (so last team grades team 1). 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.

At the end of this exercise we want to collect all of this material and brainstorm on how we can make this into an “autograded” assignment.

I suggest all teams working in github. Please share your CS6381 specific contributions with me. My handle is asgokhale