Hands on M2M Communications

Chapter - 01

About

I am writing this tutorial exclusively for Microsoft Student Partner Community and the hackers out there who have a craze on building things from scratch. This Chapter just deals with the basic messaging patterns (pub-sub) and their implementation. We extensively concentrate on M2M (Machine-to-Machine) communications. The information about the hardware used in this tutorial is given accordingly when they are used.

If you find any errors or doubts within this tutorial, then you can reach out to me at
Shashank.J@studentpartner.com

TOC

What are we planning to do ?

In the tutorial we will be walking through *publisher-subscriber* messaging pattern.

Publisher-Subscriber is one of the most basic and important type of M2M messaging pattern. The *Publisher* publishes multiple streams of data under different topics concurrently. *Subscribers* who subscribe to the topics get the messages that were published by the publisher for the topic. A single subscriber can subscribe to multiple topics. We will see this in practice, lets get started.

I am planning to use **ZeroMQ** (socket library) in this tutorial. You are free to use any library of your interest but the concept and messaging/communication patterns remains the same. So let's get started in the next section.

Let's Get Started

Installing the zeromq library.

In this tutorial I will be using zeromq version 3.2.5 (you are free to use any 3.x.x version) and note that I will be using linux based distros in all my tutorials. If you are a windows user you can install **Ubuntu - Windows**

Subsystem for Linux (WSL) from Windows Store.

Step 1: Get the source code from the git. Open your terminal and enter these commands.

```
$ sudo apt-get update
```

```
$ sudo apt-get install -y git build-essential libtoolpkg-config autotools-dev autoconf automake cmake uuid-dev libpcre3-dev libsodium-dev valgrind
```

```
$ wget https://github.com/zeromq/zeromq3-x/releases/download/v3.2.5/zeromq-3.2.5.zip
```

```
$ unzip zeromq-3.2.5.zip
```

```
$ cd zeromq-3.2.5/
```

```
$ ./autogen.sh
```

```
$ ./configure
```

```
$ make check
```

```
$ sudo make install
```

```
$ sudo ldconfig
```

Step 2: Get the header file (zhelpers.h). Just copy the entire code and save it in a file named `zhelpers.h`.

Publisher

Before we write the publisher, it is very much necessary to understand the basic building blocks of the messaging library (*zeromq*) that we are using.

context : A thread safe instance used to initialize and create ZMQ sockets.

> Coding the Publisher:

First lets import the necessary header file (*zhelpers.h*). This header file contains code which simplifies our imports and automatically handles the way we send messages across different machines.

```
/* importing the necessary header file - zhelpers.h */
#import "zhelpers.h"
```

The code for publisher program is given below.

> *publisher.c*

```
#include "zhelpers.h"

int main (int argc, char *argv[])
{
    // creating a zmq context
        void *context = zmq_ctx_new ();

    // creating a publisher socket. ZMQ_PUB - created a publisher socket.
        void *publisher = zmq_socket (context, ZMQ_PUB);

    // binding as it is the update server
        int rc = zmq_bind (publisher, "tcp://*:8080");
        assert (rc == 0);

    // initialize the random function
        srandom ((unsigned int)time(NULL));

    printf("Publisher started publishing the updates ...\n");

        // start publishing the updates
        while (1)
        {

                /* publish weather updates */
                char bangaloreUpdate[20];
                int temperature, sunlight, windSpeed;
                temperature = randof (1000);
                sunlight = randof (300);
                windSpeed = randof (20);

                sprintf (bangaloreUpdate, "BANGALORE - %d %d %d",
temperature,
                sunlight, windSpeed);
```

```
                s_send (publisher, bangaloreUpdate);

        // sleep for one second
        s_sleep (1000);
        }

        // close what's opened
        zmq_close (publisher);
        zmq_ctx_destroy (context);

        return 0;
}
```

> Program Description

In the first few lines of code, we setup the ZMQ context and create a publisher socket. Then, we bind the publisher to the port 8080 (you are free to use any ports within range). Most of the time we bind the entities which remain static in the network, hence publishers which remain static will do the binding unlike subscribers (dynamic entities of the network) who join and leave the network at any time.

If binding the port returns value 0, then everything went smooth as expected. We check that using `assert(rc == 0)` statement.

In furthur lines of code we initialize a random number seed and using `srandom` function. Next, populate the updates and store it in bangaloreUpdate character array. `sprintf` function is being used to format our string.

Using the `s_send()` function we send the messages to all subscribers who have subscribed to this topic (How to subscribe for topics? is covered under *Coding the subscriber* part of the chapter).

Subscriber

> Coding the Subscriber:

First lets import the necessary header file (*zhelpers.h*). This header file contains code which simplifies our imports and automatically handles the way we send messages across different machines.

The code for subsciber program is given below.

> *subscriber.c*

```
#include "zhelpers.h"

int main (int argc, char *argv[])
{
        void *context = zmq_ctx_new ();
        void *subscriber = zmq_socket (context, ZMQ_SUB);

        // connect to the publisher
        int rc = zmq_connect (subscriber, "tcp://localhost:8080");
        assert (rc == 0);
```

```
        char *bangaloreUpdate = "BANGALORE";

        /* set the socket options saying that we are
       subscribing to certain topicssubscribe the
       updates from bangalore */
        rc = zmq_setsockopt (subscriber, ZMQ_SUBSCRIBE, bangaloreUpdate,
                        strlen (bangaloreUpdate));
        assert (rc == 0);

        // start recieving the updates
        while (1)
        {
                char *recvUpdate = s_recv (subscriber);
                printf ("%s\n", recvUpdate);
        }

     // close what's opened
        zmq_close (subscriber);
        zmq_ctx_destroy (context);

        return 0;
}
```

> Program Description:

`void *context = zmq_ctx_new ()` - creates a new ZMQ context.
`void *subscriber = zmq_socket (context, ZMQ_SUB)` - create a new subscriber socket.
We use ZMQ_SUB along with the context to create a subscriber socket.

Using the `zmq_connect` function lets connect our subscriber to the publisher which is running at localhost on port 8080.

Subscribing to topics:
The `zmq_setsockopt` function adds a filter to your subscriber socket so that you only receive messages starting with the string topic. You can add multiple filters by calling it more than once(which we will see in next chapter). Providing empty filter as `zmq_setsockopt(subscriber, "", bangaloreUpdate, strlen(bangaloreUpdate))` makes the subscriber to recieve all the messages sent by different publishers under different topics and also this way of subscription removes any existing filter so your subscriber gets all messages sent by the publisher.

Next we infinitely loop to get the messages from the publisher. `s_recv()` function gives the message that was recieved from the publisher.

Lets run the code

Open a terminal and navigate to the directory where the supporting program files are present.

```
.
├── publisher.c
```

```
├── subscriber.c
└── zhelpers.h
```

compile the publisher program

```
$ gcc publicher.c -lzmq -o publisher
```

compile the subscriber program

```
$ gcc subscriber.c -lzmq subscriber
```

start the publisher

```
$ ./publisher
```

connect the subscriber (in another terminal)

```
$ ./subscriber
```

(1) You can disconnect and reconnect the subscriber multiple times.
(2) Can connect multiple subscibers to a single publisher.
(3) A single publisher is capable of connecting ~ 10,000 concurrent subscribers (may vary depending on your system NIC, processor and memory).
(4) Start publisher or subscriber in any order.

Conclusion

In the next chapter we will see how we can send messages from **NodeMCU** (publisher) to our zeromq client program (subscriber) over a wireless local area network.