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

CSU33031 Computer Networks Assignment #1: Protocols

Name: Satwik Chandra
Student ID: 19321427
Date: October 30, 2021

Table of Contents

Introduction

Making a publish-/subscribe protocol using the socket programming functionality in Python and utilizing custom header names to identify unique entities in the network and allow communication between the server and the clients.

Theory of Topic

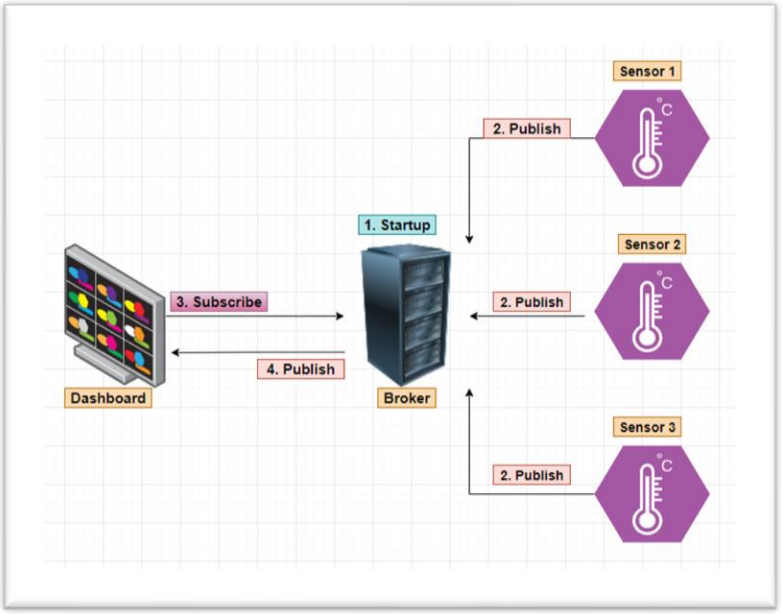
The system consists of 3 main components. Namely The Broker, the Dashboard, and the Sensors. The Broker is a server to which the Dashboard and the sensors connect to and send information. The dashboard subscribes to the Broker and sends an “ask request ”to notify the Broker to send the information to the Dashboard so that it can be displayed on the screen. Before the Dashboard requests the Broker, the Sensors connect to the server sending a small packet of string which contains the temperature at the time the sensor measured it. The Broker stores this information in a variable which it then sends to the Dashboard when requested. All the communication is done with strings with a special Header concatenated at the front which tells the entity utilizing the string, which entity is the source of the string message and how to process the string.

```
CLIENT_MESSAGE_HEADER = "Client”    #sent by the sensors(Client)

SERVER_MESSAGE_HEADER = "Server"    #sent by the Broker(Server)

DISCONNECT_MESSAGE = "DISCONNECT"   #sent by all entities

ASK_REQUEST = "SEND TEMPS"          #sent by the Dashboard(Client)
```

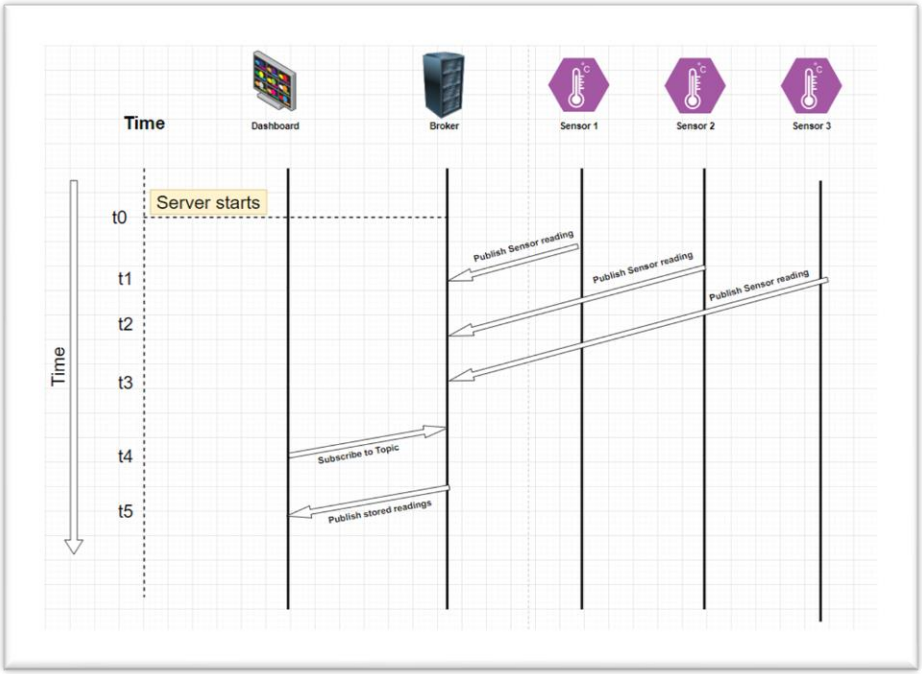
List of different possible headers which determine how a packet will be processed and stored.

Network structure and Flow Diagram

Topology



Flow Diagram



Components

Sensors

The Sensors act as publishers which send the temperature reading to the server. The sensors connect to the server through sockets and send a packet of string containing the temperature reading at the time it was recorded.

Broker

The Broker acts like a server hence is the first component of the system to start up. The Broker establishes connection with the Dashboard and the Sensors and allows data storage and data transfer capabilities. The broker stores the readings got from the Sensors and publishes them to the Dashboard whenever the Dashboard subscribes to the Broker.

Dashboard

The Dashboard subscribes to the Broker and receives the information published by the Broker. After getting the data, the Dashboard disconnects from the Broker and displays the readings that we got from the sensors in a user-friendly format.

Implementation of Components

Sensors

```
# Import socket module
import socket
from random import randint
MESSAGE_HEADER = "Client"
DISCONNECT_MESSAGE = "DISCONNECT"

# Create a socket object
s = socket.socket()
temprature = randint(25,60)
temprature_msg = f"Sensor 1 Reading: {temprature}"
temprature_msg = MESSAGE_HEADER + temprature_msg


# Define the port on which you want to connect
port = 12345

# connect to the server on local computer
s.connect(('127.0.0.1', port))

# receive data from the server and decoding to get the string.
print (s.recv(1024).decode())
# close the connection
s.send(temprature_msg.encode())
s.close()
```

Broker

```
import socket
import pickle

s = socket.socket()

print ("Socket successfully created")
messages = []
DISCONNECT_MESSAGE = "DISCONNECT"
TERMINATE_MESSAGE = "TERMINATE"
ASK_REQUEST = "SEND TEMPS"
CLIENT_MESSAGE_HEADER = "Client"
SERVER_MESSAGE_HEADER = "Server"
# reserve a port on your computer in our
PORT_CLIENTS = 12345

 s.bind(('', PORT_CLIENTS))
print ("socket binded to %s" %(PORT_CLIENTS))
# put the socket into listening mode
s.listen(4)
print ("Server socket is listening")


# a forever loop until we interrupt it or
# an error occurs
while True:

# Establish connection with client.
  c, addr = s.accept()
  print ('Got connection from', addr )

  # send a thank you message to the client. encoding to send byte type.
  c.send('Thank you for connecting'.encode())
  print ()

  msg = c.recv(1024).decode()
  if msg == DISCONNECT_MESSAGE:
      c.close()
  elif msg == ASK_REQUEST:
      print("Sending info to the dashboard")
      pickled_SERVER_Header = pickle.dumps(SERVER_MESSAGE_HEADER)
      pickled_messages = pickle.dumps(messages)
      pickled_messages = pickled_SERVER_Header + pickled_messages
      c.send(pickled_messages)
  elif msg[0:6] == CLIENT_MESSAGE_HEADER:
      messages.append(msg[6:])
      print("Message recieved from Sensor \n")
```

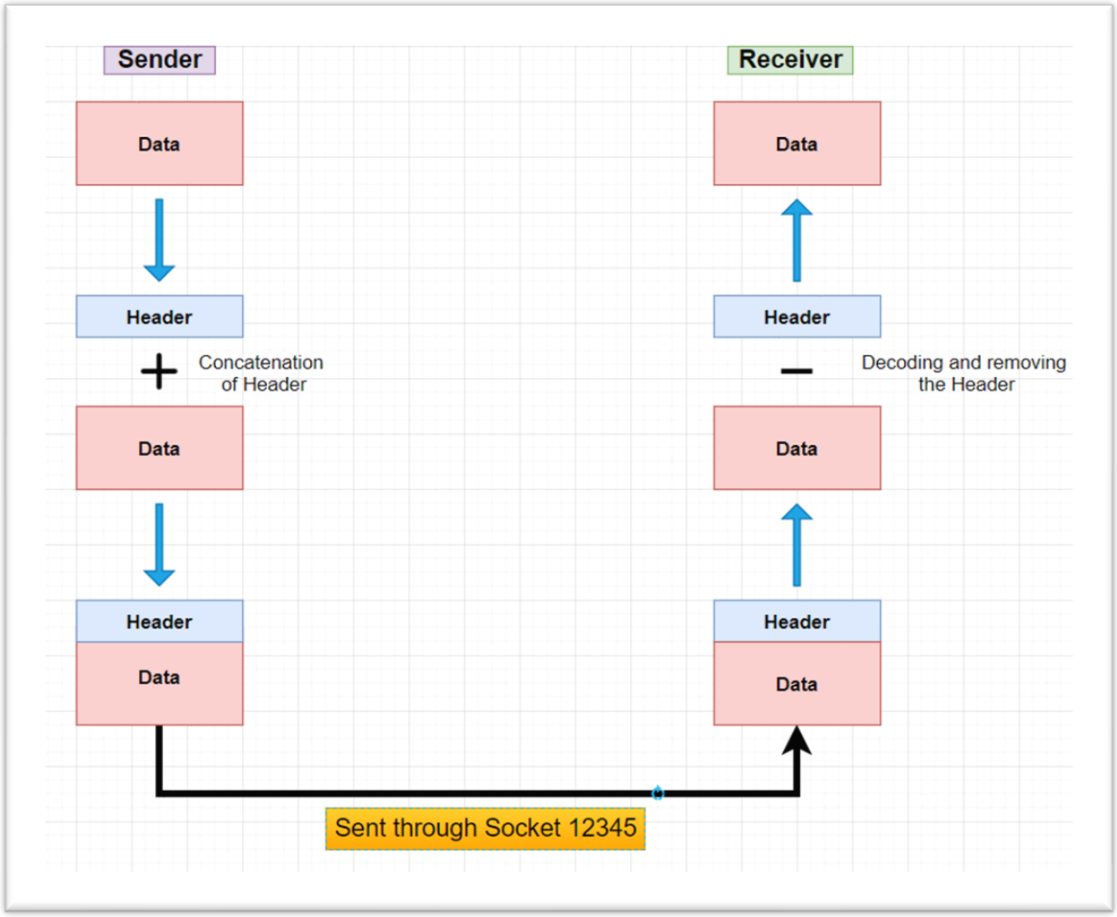
Dashboard

```
# Import socket module
import socket
import pickle
DISCONNECT_MESSAGE = "DISCONNECT"
ASK_REQUEST = "SEND TEMPS"
SERVER_MESSAGE_HEADER = "Server"
# Create a socket object
s = socket.socket()

# Define the port on which you want to connect
port = 12345

# connect to the server on local computer
s.connect(('127.0.0.1', port))

# receive data from the server and decoding to get the string.
print (s.recv(1024).decode())
# close the connection
s.send(ASK_REQUEST.encode())

pickled_message = s.recv(1024)
pickled_SERVER_Header = pickle.dumps(SERVER_MESSAGE_HEADER)

if pickle.loads(pickled_message[0:len(pickled_SERVER_Header)]) == SERVER_MESSAGE_HEADER:
    pickled_message = pickled_message[len(pickled_SERVER_Header):]
    messages = pickle.loads(pickled_message)

for msg in messages:
    print(msg + "\n")

s.close()
```

Communication and Packet Description

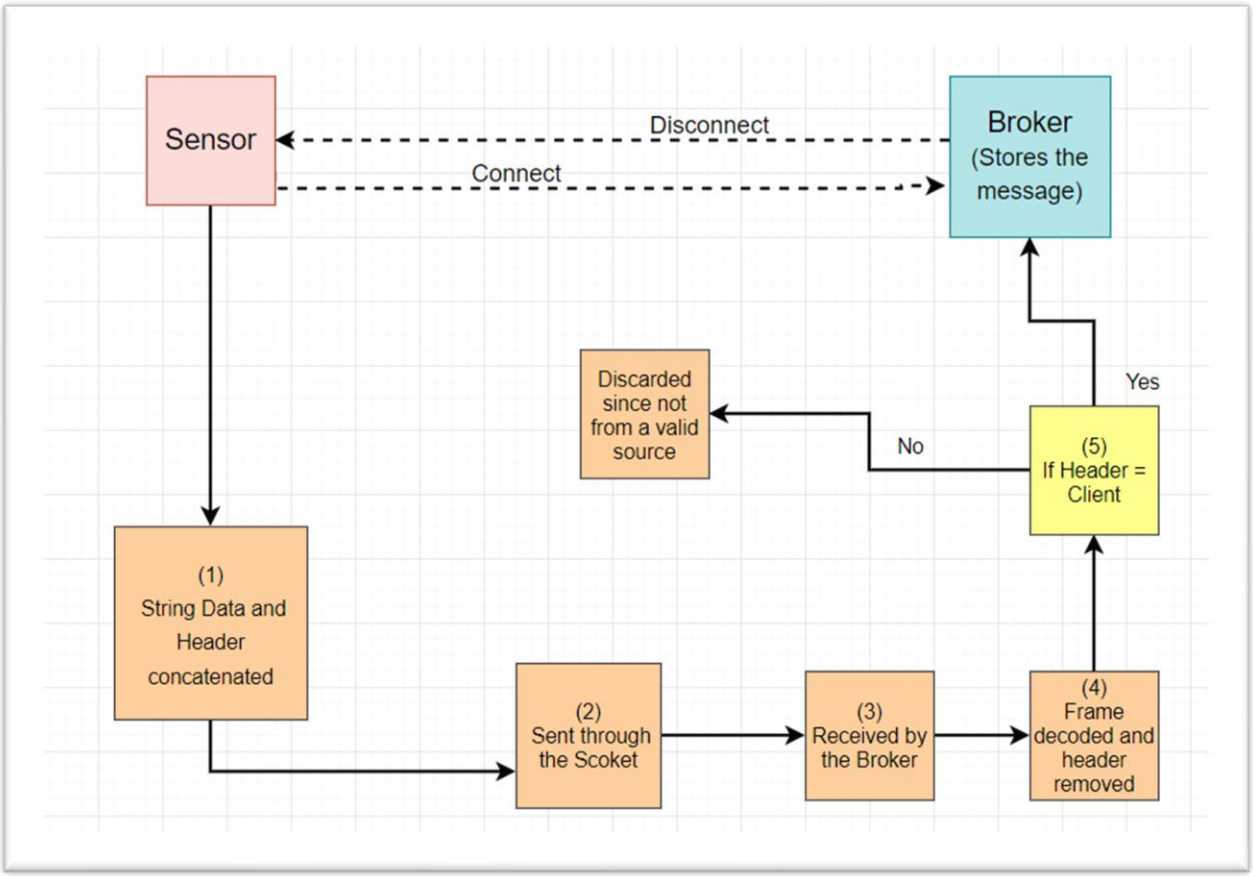
Outline

All the communication is done through sockets. Hence socket 12345 is reserved for this system. The data about to be sent is first concatenated with a Header which defines who is the sender of the data. The sender first attaches the header to the message about to be sent.

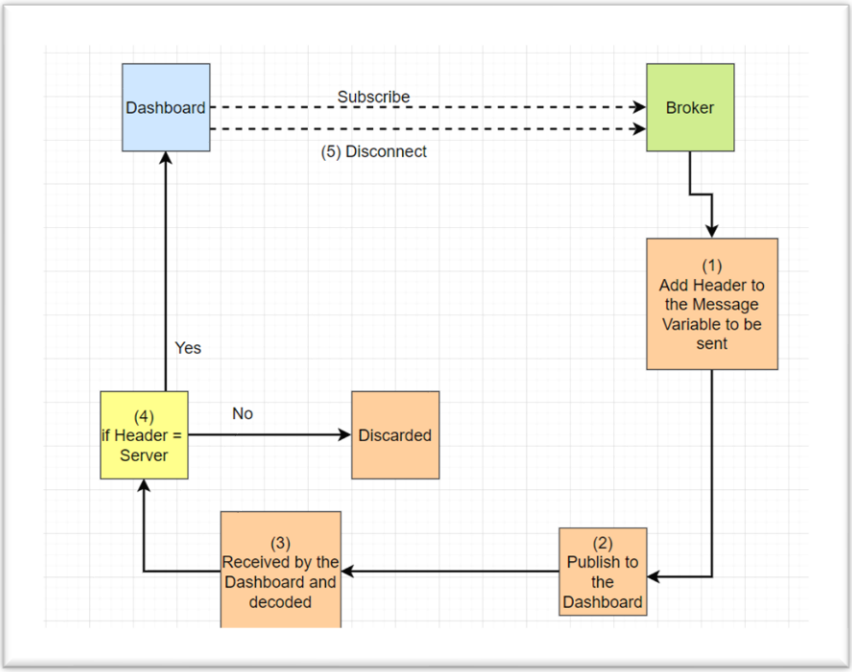


Then the resulting frame is sent in the form of a byte string to the receiver. Once the frame reaches the receiver, the receiver decodes the message, removes the header, reads the header to determine who is the sender and then processes the data according to the use case relating to the sender.

Sensors to Broker



Broker to Dashboard



Packet Encoding

Our packet consists of the Header and the Data that we are sending. The data which in our case is a string object is concatenated with a Header identifying the source entity of the string. After concatenation the frame is converted into a byte format and sent through the socket to the receiver. For instance, in the communication between Sensors and the Broker, The data is a string containing the temperature at a given time. The header in this case is Client, since the entity sending the information is a client entity. When the header is concatenated with eh data, it is then converted to a byte format and sent to the Broker. Once it reaches the Broker, the Broker decodes the message back into a string format, splits the header and the data and reads the contents of the header. Since in this case the Header is “Client” the data is stored. Secondly, in the communication between the Dashboard and the Broker, the Broker “pickles” the list that contains all the messages received from the Sensors, in the front the Header “Server” is concatenated indicating the sender entity. And then the message is sent through the socket to the Dashboard. Once it reaches the Dashboard, it is then converted to string format, The header is split from the data, then the dashboard reads the contents of the header to identify that the data is indeed sent from the broker, which then after it unpickles the list that was sent. After unpickling the data is displayed on the command line in a user-friendly format.

Summary

In conclusion, the entire system consists of 5 entities: 4 Clients (3 Sensors and 1 Dashboard), and 1 Broker(Server). The communication is done through Packets where the packets consist of a header and the data that we need to send; where the header defines where the data is coming from, hence it identifies the sender and can be used to process the data differently according to the use case pertaining to every sender.

Reflection

The main drawback of the system is that there would be a need for a dictionary that contains all the unique header names for all the entities in the system. For a small system like this it can easily be managed by identifying the clients and the server but for a large system, associating different headers for every entity can be cumbersome. Having unique headers for each entity does

ensure that the data sent is not leaked to another entity and the data received is used for what it was intended for. But again, for a large system, issue might arise where the dictionary of headers itself becomes too large to be stored locally on the server which can add to the cost of running of the system.