**Tutorial: Exploring UDP Multicast in Java and with WireShark**

In this tutorial you will develop a feed handler, or line handler as it is sometimes known, for a UDP multicast feed. This is **a miniature version** of the general situation for stock market data feeds such as the Hong Kong Stock Exchange Orion Market Data feed that is presented in the lecture on UDP during the module.

Specifically, as compared to an actual product, there are **no TCP recovery channels and no refresh lines involved in this tutorial**. There is only one multicast (MC) channel involved, which we will call line A. There is **no line B,** an entity that the systems designers would route in a diverse geographic way for this feed.

**The market data feed**

During the practical class a server program will broadcast messages continuously on Line A. Actually the same set of messages will be rebroadcast in a continuous loop during the class.

The following shows the first three messages as an example:

MSG=0001,TICKER=BT.L,PRICE=103.5,VOL=1000,TIME=02/02/2017 3:39:41 PM,

MSG=0002,TICKER=BT.L,PRICE=104.5,VOL=990,TIME=02/02/2017 3:39:41 PM,

MSG=0003,TICKER=FB.NYSE,PRICE=42.2,VOL=876,TIME=02/02/2017 3:39:41 PM,

Each message is sent in ONE UDP packet alone. Each message is a set of comma-separated values with each value consisting of a key value pair. These are:

MSG index/counter starting at 1 and increasing with each message sent.

TICKER a name for the stock

PRICE price paid in most recent transaction for that stock in UK pounds

VOL number of shares traded in the most recent transaction for that stock

TIME the date and time that the message was broadcast by the server

**Step 1: Connecting to Line A**

Line A is defined for today’s practical by the following UDP details

UDP Class D address 225.4.5.6 Port 6984

The relevant Java class to be used is MulticastSocket. You should read the documentation for this class at

<https://docs.oracle.com/javase/8/docs/api/java/net/MulticastSocket.html>

and use it throughout the tutorial.

Using the stubbed code which you have been given

1. create a socket that connects to Line A
2. then sits listening for messages
3. receives and prints messages

**Write notes comparing and contrasting** this to your work last week in which you created a TCP client. What is different?

Did you check whether any messages were dropped in transit from the server program? You can stop and restart your client at any point. The server runs continuously during the practical class. When you restart you will join the feed at some random point. Can you detect this?

Actually, the server is simply looping over the data file

stock\_prices.txt.

Note that the while the same price and volume data will be rebroadcast, the timestamps will change.

Note: we **do not have** a stock market provider driving this – the data is completely simulated.

**Step 2: Process the market data**

The method processMsg() is left as a stub for you to work on it. You should at the very least split each message from the market feed into its fields using string manipulation functions in Java and then print these fields.

You are encouraged to be as creative as you wish in implementing a solution in this section. There is no single correct answer for this part.

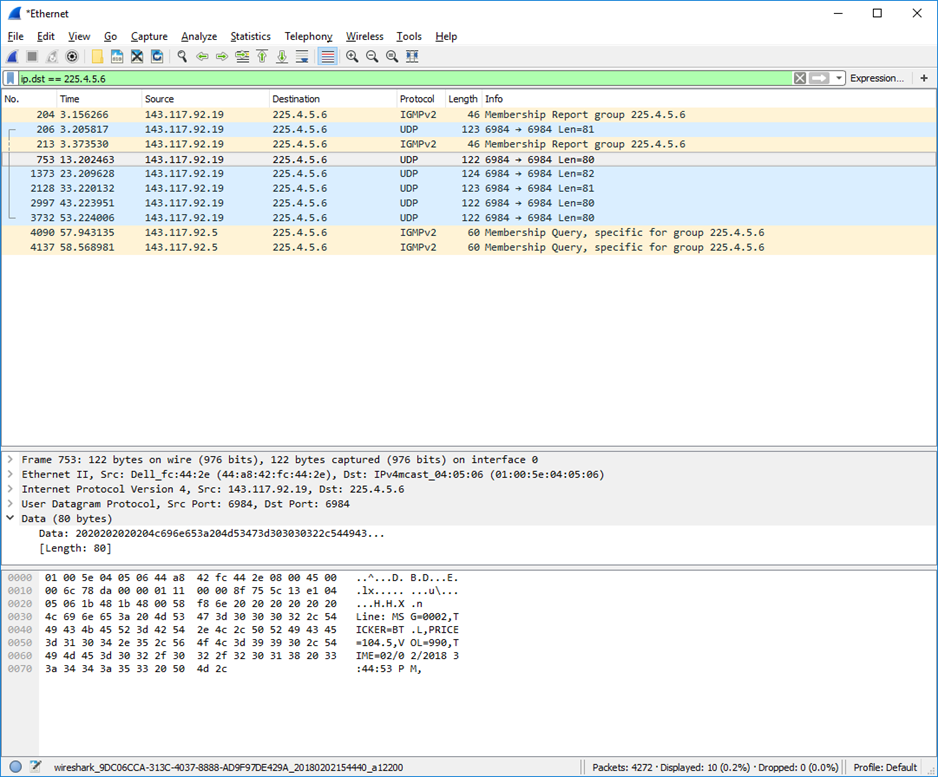


Figure 1 An expanded UDP MC packet capture on the NIC of the server where it was sent.

A more advanced solution would use perhaps HashMap or Vector classes to maintain separate tables for each stock, so that you can show a history of price changes for each stock.

This means that you would print the list of prices for each stock, when you receive an update. Although not necessary, if you have GUI skills, you could present a dynamic table of price and volume for each stock.

**Step 3: Monitoring with WireShark**

Once you are happy with the operation of your client you should run WireShark, as done in last week’s tutorial. Can you identify the packets associated with Line A? Remember that your computer is operating on **the client, or receiver** side of this dialogue.

Figure 1 shows the expanded details of a one such packet as it was captured at the **NIC of the server, or sender computer,** being broadcast onto Line A. The above capture received several packets. These were filtered on the destination address, which in this case is the MC group 225.4.5.6. This is achieved in the WireShark window using the string

ip.dst == 225.4.5.6 (note the spaces before and after the ==)

Expand the details of the complete packet at all layers of encapsulation. Can you relate the values of the fields to the packet headers for IP and UDP as discussed in the lectures for the module. You can also see the details of the Ethernet header.

**Note on Multicast Server code in Java**

The MC server code will be made available after the practical class. You should note that it needs to JOINGROUP on the appropriate NIC to the MC group in order to operate. In this case, we want to send packets onto the 143.117.x.x network in the CSB lab. If no JOINGROUP operation is performed, the default situation is that MC packets are sent only on the FIRST NIC that the OS knows about. This is likely to be something other than the CSB lab network.

**[ End Document ]**