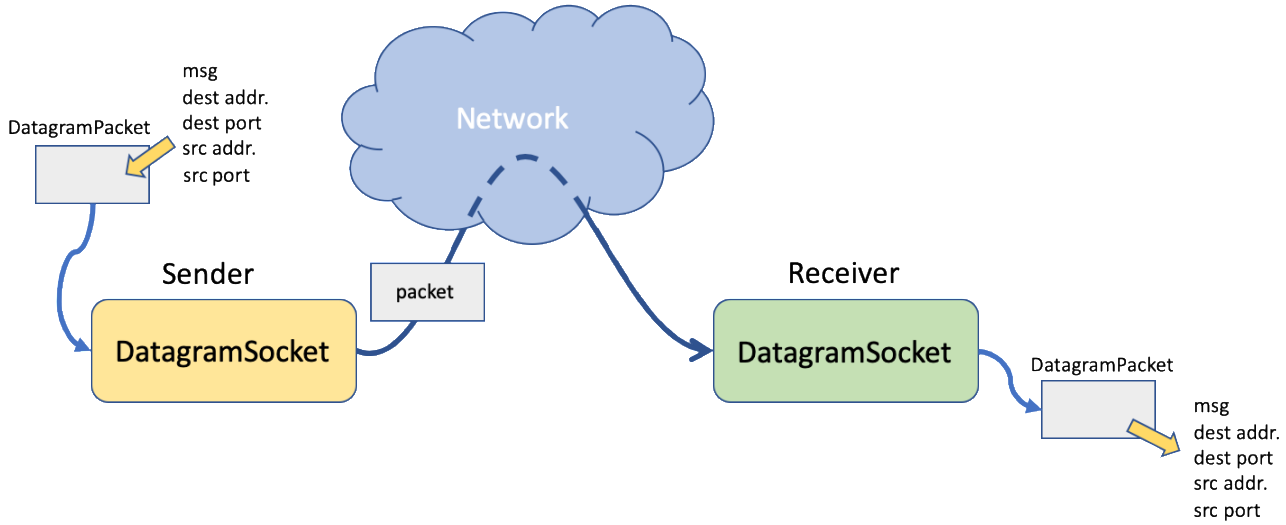
COMP3015
Data Communication and Networking
Socket Programing – User Datagram Protocol (UDP)

Introduction

UDP is a connectionless protocol. It does not have any mechanisms for controlling data transmission and handling the errors. But it has better performance for sending short messages (compare with Transmission Control Protocol (TCP)) because it uses a simple transmission model.



Sender and Receiver

The sender sends datagram packets through the socket. The receiver receives datagram packets through the sockets. Actually, a single **DatagramSocket** can be used for both sending and receiving packets.

DatagramSocket

A **DatagramSocket** can be used for both sending and receiving datagram packets.

DatagramPacket

In the sender, a **DatagramPacket** stores the information of a packet being sent. In the receiver, a **DatagramPacket** stores the information of a packet recently received.

Sending Messages

Before sending messages, we need to create a **DatagramSocket**. That socket binds a free port (a port is not used by another socket).

```
DatagramSocket socket = new DatagramSocket(12345);
```

Next, we need a **DatagramPacket** to store the message, destination address, and destination port number. The source address and port number will be assigned automatically.

Note that the maximum size of the message is *1024* bytes. **If the message size is larger than the maximum size, the message will be truncated.**

```
byte[] msg = "Hello World".getBytes();
InetAddress dest = InetAddress.getByName(destIP);
int port = 45678;
DatagramPacket packet = new DatagramPacket(msg, msg.length, dest, port);
```

After the packet is ready, we then use the **send()** method to send the packet.

```
socket.send(packet);
```

The following is an example sender program:

```
public class UDPSender {

  DatagramSocket socket;

  public UDPSender() throws SocketException {
     socket = new DatagramSocket(12345);
  }

  public void sendMsg(String str, String destIP, int port) throws IOException {
     InetAddress destination = InetAddress.getByName(destIP);

     DatagramPacket packet =
        new DatagramPacket(str.getBytes(), str.length(), destination, port);

     socket.send(packet);
  }
```

```
public void end() {
     socket.close();
     System.out.println("bye-bye");
  }

  public static void main(String[] args) throws IOException {
     Scanner scanner = new Scanner(System.in);
     UDPSender sender = new UDPSender();

     System.out.println("Input something:");
     while(true) {
        String str = scanner.nextLine();
        if (str.trim().length() == 0) break;

        sender.sendMsg(str, "127.0.0.1", 45678);
     }
     scanner.close();
     sender.end();
  }
}
```

Receiving Messages

To receive messages, we also need a **DatagramSocket**. That socket should bind to a free port.

```
DatagramSocket socket = new DatagramSocket(45678);
```

Next, we need to prepare a **DatagramPacket**. It will be used for storing the received packet.

```
DatagramPacket p = new DatagramPacket(new byte[1024], 1024);
```

Then, we use the **receive()** method to receive a packet. The **receive()** method blocks the program and waits for a packet. Once, a packet delivers. The data of the packet will be stored in the **DatagramPacket** we prepared previously.

```
socket.receive(p);
```

We can retrieve the details of the packet, such as its data, data size, source address, and source port number.

```
byte[] data = packet.getData();
String str = new String(data, 0, packet.getLength());
int size = packet.getLength();
String srcAddr = packet.getAddress().toString();
int srcPort = packet.getPort();
```

The following is an example receiver program:

```
public class UDPReceiver {
  DatagramSocket socket;

  public UDPReceiver() throws SocketException {
     socket = new DatagramSocket(45678);
  }

  public void receive() throws IOException {
     DatagramPacket packet = new DatagramPacket(new byte[1024], 1024);

     socket.receive(packet);

     byte[] data = packet.getData();
     String str = new String(data, 0, packet.getLength());
     int size = packet.getLength();
     String srcAddr = packet.getAddress().toString();
     int srcPort = packet.getPort();

     System.out.println("Received data:\t" + str);
     System.out.println("data size:\t" + size);
     System.out.println("sent by:\t" + srcAddr);
     System.out.println("via port:\t" + srcPort);
  }
```

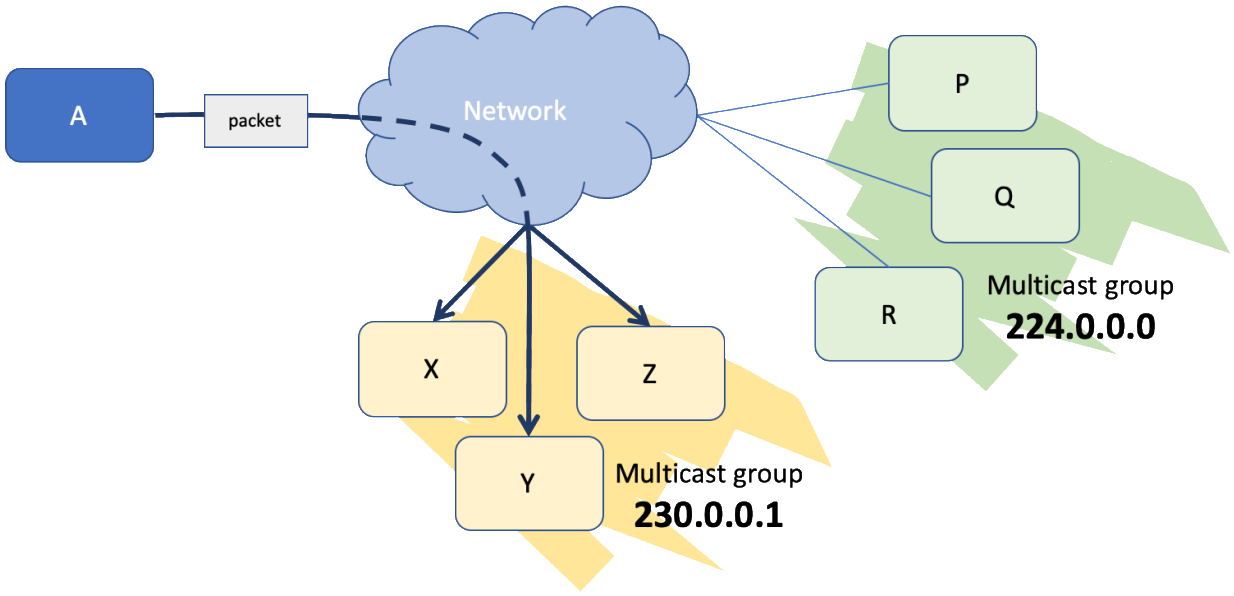
```
  public static void main(String[] args) throws IOException {
     UDPReceiver receiver = new UDPReceiver();

     while (true) {
        System.out.println("\nWaiting for data...");
        receiver.receive();
     }
  }
}
```

Unicast

In the previous example sender and receiver programs, the communication between the sender and the receiver is unicast. Unicast refers to one-to-one transmission – a packet is sent from a single sender to a single receiver.

Multicast

A sender is able to send a packet to multiple receivers using multicast. In Java, no change is required in the code of the sender program except the destination address. A multicast address is used instead of the IP address of the destination.



Multicast address range:

224.0.0.0 - 239.255.255.255

In the receiver program, we use a **MulticastSocket** instead of **DatagramSocket**. And, the **joinGroup()** method will be used for joining a multicast group.

```
System.setProperty("java.net.preferIPv4Stack", "true");

MulticastSocket socket = new MulticastSocket(port);

socket.joinGroup(InetAddress.getByName("224.0.0.0"));
```

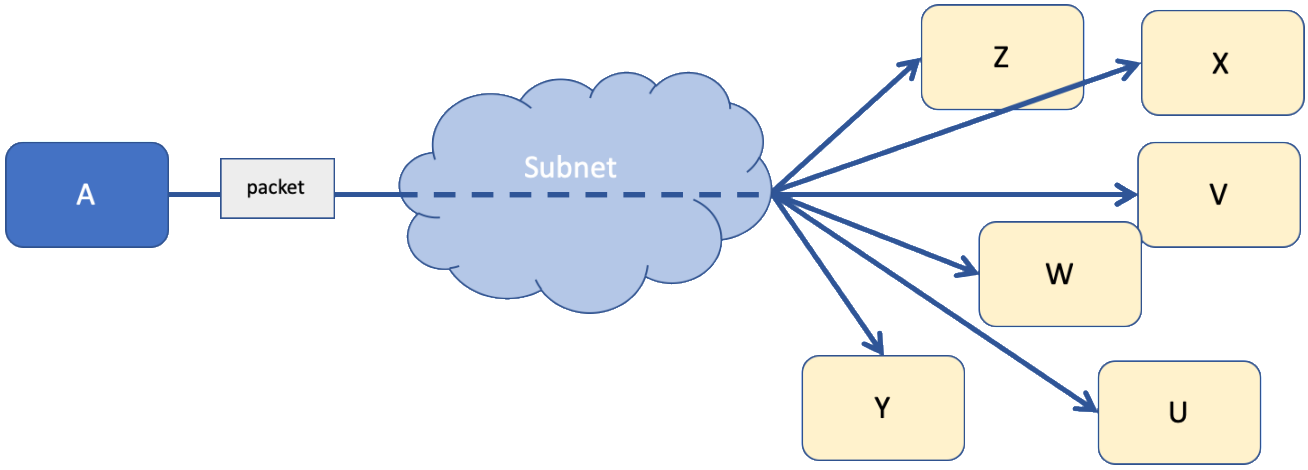
If a sender sends a packet and its destination address is same as the group address (multicast address), the receiver will receive that packet.

MulticastSocket can join multiple groups. To leave a group, we use the **leaveGroup()** method.

```
socket.leaveGroup(InetAddress.getByName("230.0.0.0"));
```

Broadcast

The sender has another way to send a packet to multiple receivers – all receivers in the same subnet through broadcast. We use the unicast version of the sender program and receiver program for sending and receiving broadcast messages respectively. We only need to change the destination addresses in the sender program to a broadcast address.



Note that broadcast messages will not be transmitted by routers. Therefore, we can send broadcast messages to the receivers that are in the same subnet.

```
byte[] msg = "Hello World".getBytes();
InetAddress dest = InetAddress.getByName("255.255.255.255");
int port = 45678;
DatagramPacket packet = new DatagramPacket(msg, msg.length, dest, port);
socket.send(packet);
```

Broadcast address = the last address of the subnet, such as:

158.182.8.255

192.168.1.255

255.255.255.255

...

Exercise

Task 1

Write a Java program – **UDPTimerDisplayer** with the following requirements:

1. Create an Eclipse project named **UDPTimerDisplayer**.

2. Create a new class named **UDPTimerDisplayer** in the project. It has a **MulticastSocket** that joins multicast group 224.0.0.0 and 230.0.0. That socket listens the port number 34343.

3. When the socket receives a packet, the content of the packet should be displayed in the console immediately.

4. Use the test program **UDPTimer** to test your **UDPTimerDisplayer**. The download URL of **UDPTimer** is http://www.comp.hkbu.edu.hk/~mandel/comp3015/lab2/UDPTimer.class

5. The source of **UDPTimer** is available at http://www.comp.hkbu.edu.hk/~mandel/comp3015/lab2/UDPTimer.java

6. To run **UDPTimer**, you need to open a command prompt (cmd.exe, for Windows) or terminal (for macOS). Then, run the following command:

   java UDPTimer *23232* *34343*

   where *23232* is the source port and *34343* is the destination port.

7. Zip your Eclipse project folder and submit it to Moodle.

Task 2

Write a Java program – **Checker** with the following requirements:

1. Create an Eclipse project named **Checker**.

2. Create a new class named **Checker** in the project. It has a **DatagramSocket** that binds to the port 5555.

3. When the program has just been executed, the **Checker** (instance) broadcasts the message, "Is anyone here?", through the socket.

4. If the socket receives the message "Is anyone here?", the **Checker** replies to the sender with the message "I am here!".

5. If the socket receives the message "I am here!", the **Checker** prints the IP address of the sender to the console.

6. To test your program, you are suggested to use two computers in *RRS638* or *FSC801*. Run your program in two computers and check the results showing in the console.

7. Note that broadcast packets will not be redirected by routers and routing switches (network switch with routing function, a.k.a. layer 3 switch). Therefore, you have to ensure that the two computers are in the same subnet.

Task 3

Write a Java program – **FileTransferrer** with the following requirement:

1. Create an Eclipse project named **FileTransferrer**.

2. Create a new class named **FileTransferrer** in the project.

3. A **FileTransferrer** can be a sender or receiver. If a file and destination IP address are provided to the **FileTransferrer**, that **FileTransferrer** will be a sender. Otherwise, the **FileTransferrer** will be a receiver.

4. The sender sends the file name, file size and the content of a file to the receiver using UDP. The receiver saves the received content to a file in the local drive.

5. To test your program, you are suggested to use two computers in *RRS638* or *FSC801*. Copy the class file of your program to the lab computers. And execute them using the following commands respectively. *Note that the receiver must be executed first*.

   - Execute the receiver:

     java FileTransferrer

   - Execute the sender:

     java FileTransferrer d:\something.mp3 158.182.9.128

     Where *d:\something.mp3* should be replaced with the existing file in your computer. You are suggested to use a **file larger than 1K bytes** to test your program.
     158.182.9.128 should be replaced with the real IP address of the computer running as the receiver.

6. Check whether two files are exactly the same or not. If the source file is an mp3 or mp4 file, you may play the copy to verify the content.

Appendix

Checking IP Address and Network Address

If you want to know what IP address is used by your computer currently, you can use the following command in the command prompt (Windows) or terminal (macOS):

Windows: ***ipconfig***

Sample output:

```
Ethernet adapter Ethernet 3:

   Connection-specific DNS Suffix  . : comp.hkbu.edu.hk
   Link-local IPv6 Address . . . . . : fe90::74ba:7d29:dbaf:94df%12
   IPv4 Address. . . . . . . . . . . : 158.182.9.128
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 158.182.9.2
```

*macOS****: ifconfig en0***

Sample output:

```
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
  ether 38:f9:d3:96:e8:57
  inet6 fe80::c1b:ef2d:9aa1:d2bd%en0 prefixlen 64 secured scopeid 0xa
  inet 192.168.1.16 netmask 0xffffff00 broadcast 192.168.1.255
  nd6 options=201<PERFORMNUD,DAD>
  media: autoselect
  status: active
```

In the sample outputs, the IP addresses and subnet masks (netmasks) are highlighted in yellow and cyan respectively. The subnet marks of both sample outputs are the same, but they are represented in two different formats – 4 octets (**255.255.255.0**) and hexadecimal (**0x ff ff ff 00**).

255.255.255.0 ⇔ 0x ff ff ff 00

In the first output, the IP address is **158.182.9.128** and the subnet mask is **255.255.255.0**. Because the first three octets of the subnet mask are *255*, it means that the first three octets of the IP address are used for representing the network address. Therefore, the network address of that computer is **158.182.9.0**.

In the second output, the IP address is **192.168.1.16** and the netmask is **0 x ff ff ff 0** (255.255.255.0). Same as the first output, the first octets of the IP address are used for representing the network address. The network address of that computer is **192.168.1.0**.

Due to the network addresses of these two computers are different (**158.182.9.0** and **192.168.1.0**), **they are NOT in the same subnet**. Therefore, the broadcast packets cannot be transmitted from one to another.

END