情報コミュニケーション学実験 II
「コンピュータネットワーク」レポート

立命館大学情報理工学部

氏名:ZHENG YUHUI
学籍番号:2600120572-1

2014 年 10 月 29 日

Contents

1 実験目的

近年、インターネットの発展及びアプリケーションの多様化に伴う、同一データを複数のユーザに伝達する機会が増えている。送信ノードから同一内容のパケットを複数の受信ノードに送信する方法として、ユニキャスト通信を用いる方法と、マルチキャスト通信を用いる方法が考えられる。本実験は、複数のノードに対してユニキャスト通信及びマルチキャスト通信を利用してデータを送受信するプログラムを実装し、両者の特性を測定し、比較することを目的としている。この実験を通じて、JAVA 言語によるユニキャスト通信及びマルチキャスト通信のソケットプログラミングの実装法及び特性の差異と有効性の確認について習います。

2 実験内容

2.1 ユニキャスト通信の送信処理

2.1.1 概要

ユニキャスト通信によるデータ送信プログラム“UnicastSender.java”を利用して、読み込んだバイナリファイルをクライアントへ送信する、また、クライアントから返却される Echo を確認する。マルチスレッドも利用する方式をプログラムし、複数のクライアントにファイルを転送する機能もある。
一方、処理時間計測プログラムを利用して、送信時間を計測する。
ここ利用するポート番号は 55555 で、パケット長は 1792 バイトです。

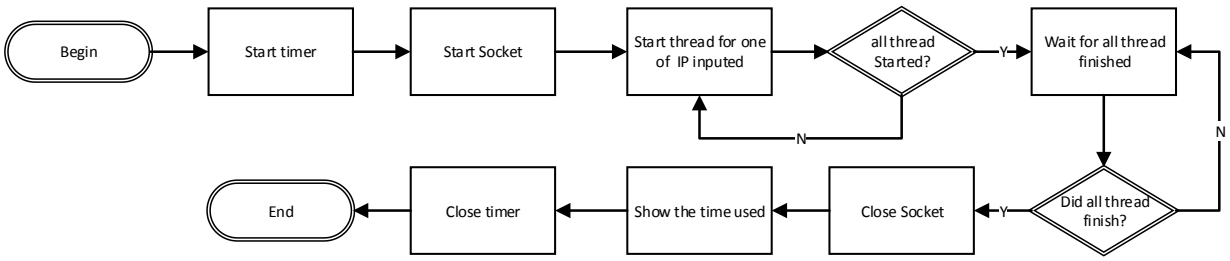
2.1.2 外部仕様

1. 各システムのコマンドラインインタフェース（例えば WINDOWS の cmd、LINUX の bash）を起動する
2. 作業ディレクトリをコンパイルされた UnicastSender.class のディレクトリに変換する。
3. Java UnicastSender filename IP0 IP1 IP2……を入力する。Filename は今転送したいファイルの名で、IP0,IP1,IP2……は目標の IP アドレスです。
4. 一部転送成功の時は「Send done for ある IP」という提示が出る。
5. 全部転送終了した時、総使用時も提示する。
6. 終了する時は Ctrl+C を入力する。

注意：
一回運行は一つのファイルしか転送できません。
入力される IP 数はゼロになった時は Error Messages を返事される。
運行システムに JRE（Java Running Environment）は必要です。

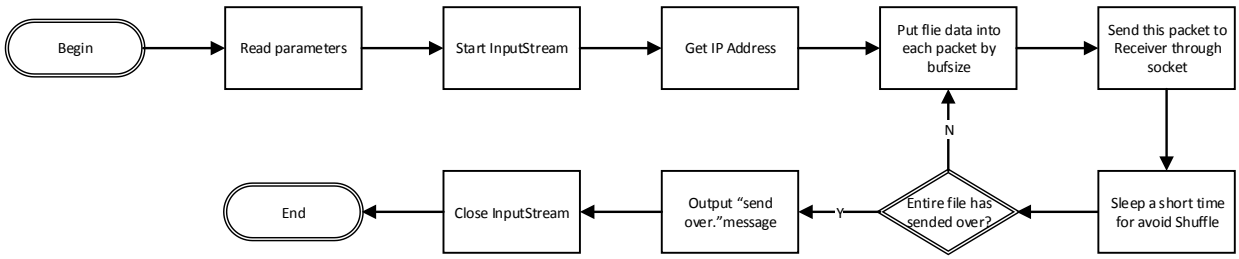
2.1.3 内部仕様

1. 主クラス UnicastSender：
   - グローバル静的変数：
     - UNI_PORT: 今回使うポート番号
     - BUFSIZE: 毎転送パケットが使うバッファーの大きさ
   - インタフェースメソッド main(String[] args)：
     - 変数：
       * start end：処理時間のタイマー
       * sendSocket：UDP ソケット
       * thread[] th: スレッドアレイ
     - フローチャート



2. クラス thread(extends Thread):

- プライベート変数:
  - fileIn:原入力ストリーム
  - fileBin: バッファー入力ストリーム
  - socket: 今使う UDP ソケット
  - packet:UDP 転送パケット
  - ip: 転送目標 IP アドレス
  - port: 転送目標ポート番号
  - buf:入力バッファー
- コンストラクタ関数:
  - パラメーター
    * socket
    * IPstring
    * port
    * bufsize
    * filename
  - 運行メソッド:
    フローチャート:



2.1.4 実行例

```
➜ multithread git:(master) ✗ > java UnicastSender cat.jpg 172.30.140.20
Send done for 172.30.140.20
time: 7.566sec
```

```
➜ multithread git:(master) ✗ > java UnicastSender cat.jpg 172.30.140.20 172
.30.140.20
Send done for 172.30.140.20
Send done for 172.30.140.20
time: 7.585sec
```

2.1.5 ソースコード

```
/*
 *
 * java UnicastSender filename IP0 IP1 IP2 .......
 *
 */

import java.io.*;
import java.net.*;

public class UnicastSender {
    static final int UNI_PORT = 55555;
    static final int BUFSIZE = 1792;

    public static void main(String[] args) {
        long start, end;                          //timer
        DatagramSocket sendSocket = null;         //socket
        thread[] th = new thread[args.length];    //thread array
        try {
            start = System.currentTimeMillis(); //timer begin
            sendSocket = new DatagramSocket();
                int ipnumber = 1;
            //start Threads
            while (ipnumber < args.length) {
                th[ipnumber] = new thread(sendSocket, args[ipnumber],
                UNI_PORT, BUFSIZE, args[0]);
                th[ipnumber].start();
                ipnumber++;
            }
            //wait for thread stop
            try {
                ipnumber = 1;
                while (ipnumber < args.length) {
                    th[ipnumber].join();
                    ipnumber++;
                }
            } catch (InterruptedException ee) {};

            end = System.currentTimeMillis(); //timer end
            //print time
            System.out.println("time: " + ((end - start) / 1000.0) + "sec");
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

}

class thread extends Thread {
    DatagramSocket socket = null;
    DatagramPacket packet = null;             //send packet
    FileInputStream fileIn = null;            //file input
    BufferedInputStream fileBin = null;       //bin file input
    String IPString;                          //String of IP
    InetAddress ip = null;                    //IP
    int port;
```

```
    int bufsize;
    String file;                                    //file name
    thread(DatagramSocket socket, String IPString,
           int port, int bufsize, String file) {
        this.socket = socket;
        this.IPString = IPString;
        this.port = port;
        this.bufsize = bufsize;
        this.file = file;
    }
    public void run() {
       try{
         fileIn = new FileInputStream(file);
         fileBin = new BufferedInputStream(fileIn);
         ip = InetAddress.getByName(IPString);
         byte[] buf = new byte[bufsize];
         int i;
         //send file
         while ((i = fileBin.read(buf, 0, bufsize)) == bufsize) {
             packet = new DatagramPacket(buf, buf.length, ip, port);
             socket.send(packet);
             Thread.sleep(5);
         }
         packet = new DatagramPacket(buf, 0, i, ip, port);
         socket.send(packet);
         Thread.sleep(5);
         System.out.println("Send done for "+IPString);
         fileBin.close();
       } catch (Exception e) {
           e.printStackTrace();
       }
    }
}
```

2.2 ユニキャスト通信の受信処理

2.2.1 概要

ユニキャスト通信によるデータ受信プログラム“UnicastReceiver.java”を利用して、サーバ側から転送したバイナリファイルを受信して、コマンド引数で指定したファイルに書き込む。
また、サーバへ Echo を転送する。
ここ利用するポート番号は 55555 で、パケット長は 1792 バイトです。

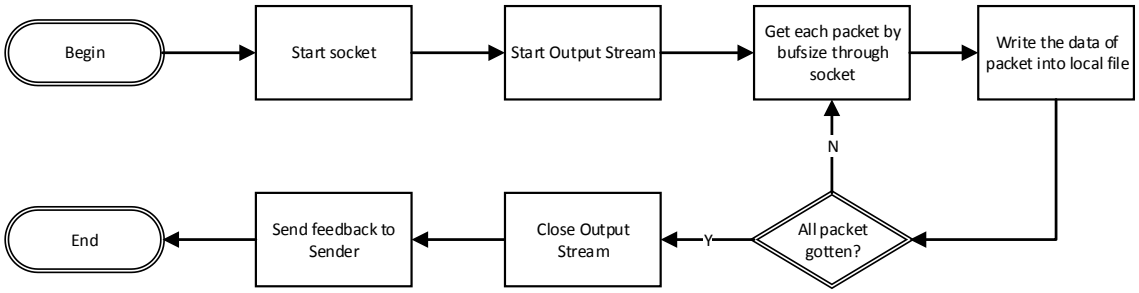
2.2.2 外部仕様

1. 各システムのコマンドラインインタフェース(例えば WINDOWS の cmd、LINUX の bash)を起動する
2. 作業ディレクトリをコンパイルされた UnicastReceiver.class のディレクトリに変換する。
3. Java UnicastReceiver filename を入力する。Filename は今ゲットした UnicastSender が転送するファイルのローカルファイル名です。
4. 起動成功する時は、「Server start」というメシージュを提示する。
5. 受信成功する時は、「Get done」というメシージュを提示する。
6. 終了する時は Ctrl+C を入力する。

注意:
一旦運行始め、Ctrl+C までずっと受信受けます。ご不用の時止めることをご注意ください。
運行システムに JRE(Java Running Environment)は必要です。

2.2.3 内部仕様

1. 主クラス UnicastReceiver:
   - グローバル静的変数:
     - UNI_PORT: 今回使うポート番号
     - BUFSIZE: 毎転送パケットが使うバッファーの大きさ
   - インタフェースメソッド main(String[] args):
     - 変数:
       * out:原出力ストリーム
       * bout: バッファー出力ストリーム
       * serverSocket: 今使う UDP ソケット
       * getPacket:UDP 受信用パケット
       * sendPacket:UDP 送信用パケット(feedback)
       * ip: 転送目標 IP アドレス
       * port: 転送目標ポート番号
       * buf:出力バッファー
     - フローチャート



2.2.4 実行例

```
→ c0 git:(master) ✗ > java UnicastReceiver cat0.jpg
Server start.
Get done.
```

2.2.5 ソースコード

```
/*
 *
 * java UnicastReceiver filename
 *
 */

import java.io.*;
import java.net.*;

public class UnicastReceiver {
    static final int UNI_PORT = 55555;
    static final int BUFSIZE = 1792;

    public static void main(String[] args) {
        FileOutputStream out = null;          //file
        BufferedOutputStream bout = null;     //bin file
```

```
        DatagramSocket serverSocket = null;     //socket
        DatagramPacket getPacket = null;     //get packet
        DatagramPacket sendPacket = null;     //send packet
        InetAddress ip = null;              //ip

        try {
            System.out.println("Server start.");
            int port = UNI_PORT;
            serverSocket = new DatagramSocket(port);
            //output stream begin
            out = new FileOutputStream(args[0]);
            bout = new BufferedOutputStream(out);
            byte[] buf = new byte[BUFSIZE];
            int i = 0;
            //get file
            while (true) {
                getPacket = new DatagramPacket(buf, 0, buf.length);
                serverSocket.receive(getPacket);
                i = getPacket.getLength();
                bout.write(buf, 0, i);
                if (i < BUFSIZE) {
                    break;
                }
            }
            System.out.println("Get done.");
            //output stream end
            bout.close();
            //feedback
            SocketAddress sendAddress = getPacket.getSocketAddress();
            String feedback = "received successfully!";
            byte[] backBuf = feedback.getBytes();
            sendPacket = new DatagramPacket(backBuf, backBuf.length, sendAddress);
            serverSocket.send(sendPacket);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

2.3 マルチキャスト通信の送信処理

2.3.1 概要

マルチキャスト通信によるデータ送信プログラム“MulticastServer.java”を利用して、読み込んだバイナリファイルを指定された同じグループの複数クライアントへ送信して, 一回複数のファイルの転送もできます。不要の時“QUIT”を入力し、終了できる。

一方、処理時間計測プログラムを利用して、送信時間を計測する。

ここ利用するポート番号は 55555 で、パケット長は 1792 バイトです。

2.3.2 外部仕様

1. 各システムのコマンドラインインタフェース(例えば WINDOWS の cmd、LINUX の bash)を起動する
2. 作業ディレクトリをコンパイルされた MulticastServer.class のディレクトリに変換する。
3. Java MulticastServer multiIP を入力する。multiIP は目標のグループ IP アドレスです。
4. 画面に“send>”という提示があります。その提示の後に転送したいファールの名を入力する。

5. 転送成功の時は「Send done.」という提示と総使用時が出る。

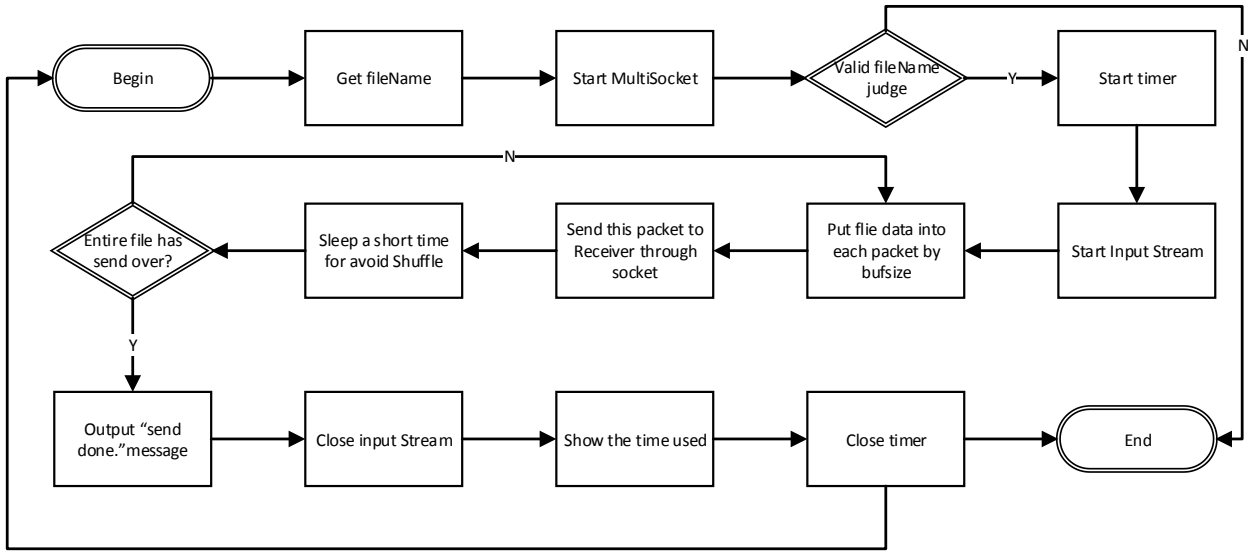
6. “send>”又出る時、ほかのファールの名を入力する可能です。

7. 全部転送終了した時は、“quit”を入力し、Ctrl+C でプロセスを終了させる。

注意：
入力されるファールの名は無効の時はエラーを出ます。基本の文字列不法の判断がありますが、転送したいファイルはディレクトリに存在しない時、そのプロセスを正しく運行しません。
入力されるグループ IP はクラス D のアドレス（244.0.0.0 239.225.225.225）を指す。但し、224.0.0.0 224.0.0.255 他の特定のアドレスは永続的アドレスとして予約されているため、その範囲のアドレスの使用は避ける必要がある。
運行システムに JRE（Java Running Environment）は必要です。

2.3.3 内部仕様

1. 主クラス MulticastServer：
   - グローバル静的変数：
     - UNI_PORT: 今回使うポート番号
     - BUFSIZE: 毎転送パケットが使うバッファーの大きさ
   - インタフェースメソッド main(String[] args)：
     - 変数：
       * start end：処理時間のタイマー
       * in：原入力ストリーム
       * bin: バッファー入力ストリーム
       * socket：マルチキャスト UDP ソケット
       * sendPacket:UDP 送信用パケット
       * ipadr: 転送目標 IP アドレス
       * port: 転送目標ポート番号
       * KeyinCheck: 入力ファール名判断用
       * TTL：タイムアウト時間
       * buf,wbuf: 入力バッファー
     - フローチャート



2. 2. クラス KeyinCheck:
   - プライベート変数：

  - QUIT
  - RET
- コンストラクタ関数: プライベート変数の初期化
- メソッド quitCheck:QUIT 命令の判断

2.3.4 実行例

2.3.5 ソースコード

```
/*
 *
 *  java MulticastServer multiIP
 *  send> filename
 *
 */

import java.io.*;
import java.net.*;

public class MulticastServer {
    static final int MULTI_PORT = 50000;
    static final int BUFSIZE = 1792;

    public static void main(String[] args) {
        long start, end;                          //timer
        FileInputStream in = null;
        BufferedInputStream bin = null;
        MulticastSocket sock = null;
        DatagramPacket sendPacket = null;
        InetAddress ipadr = null;
        KeyinCheck key = null;
        byte TTL = (byte) 1;
        byte[] buf = new byte[BUFSIZE];
        byte[] wbuf = new byte[BUFSIZE];          //buf for filename words
        int len;
        int k;

        if (args.length != 1) {
            throw new IllegalArgumentException(
                    "usage: >java multicastServer <IP_Adress>");
        }
        try {
            ipadr = InetAddress.getByName(args[0]);
            while (true) {
                System.out.print("send>");
                k = System.in.read(wbuf);
                len = wbuf.length;
```

```
                DatagramPacket packet = new DatagramPacket(
                        wbuf, len, ipadr, MULTI_PORT);
                int port = MULTI_PORT;
                sock = new MulticastSocket();
                sock.setTimeToLive(TTL);
                sock.joinGroup(ipadr);
                sock.send(packet);
                //CIL read filename
                int i = 0;
                while (wbuf[i] != 0x0d && wbuf[i] != '\n') {
                    buf[i] = wbuf[i];
                    i++;
                    if (i >= len) {
                        break;
                    }
                }
                String fileName = new String(buf, 0, i);
                key = new KeyinCheck();
                if (key.quitCheck(fileName) == 1) {
                    sock.leaveGroup(ipadr);
                    break;
                }
                //input stream begin
                start = System.currentTimeMillis(); //timer begin
                in = new FileInputStream(fileName);
                bin = new BufferedInputStream(in);
                buf = new byte[BUFSIZE];
                i = 0;
                //send file
                while ((i = bin.read(buf, 0, BUFSIZE)) == BUFSIZE) {
                    sendPacket = new DatagramPacket(buf, buf.length, ipadr, port);
                    sock.send(sendPacket);
                    Thread.sleep(5);
                }
                sendPacket = new DatagramPacket(buf, 0, i, ipadr, port);
                sock.send(sendPacket);
                Thread.sleep(5);
                System.out.println("send done.");
                //input stream end
                bin.close();
                end = System.currentTimeMillis(); //timer end
                //print time
                System.out.println("time: " + ((end - start) / 1000.0) + "sec");

            }
            sock.close();
            while (true) {
            }
        } catch (java.net.SocketException e) {
            System.err.println(e);
        } catch (java.io.IOException e) {
            System.err.println(e);
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

```
class KeyinCheck {
    private int QUIT;
    private int RET;
    KeyinCheck() {
        QUIT = 1;
        RET = 0;
    }
    // quit check
    public int quitCheck(String keyinstr) {
        if (keyinstr.equals("quit") || keyinstr.equals("QUIT")) {
            System.out.println("Quit␣␣([Ctrl]+[C])");
            return QUIT;
        }
        return RET;
    }
}
```

2.4 マルチキャスト通信の受信処理

2.4.1 概要

マルチキャスト通信によるデータ受信プログラム“MulticastClient.java”を利用して、サーバ側から転送したバイナリファイルを受信して、当ディレクトリにサーバの転送されたファイルと同じ名のローカルファイルに書き込む。
また、サーバを終了する時、このプロセスも終了する。
ここ利用するポート番号は 55555 で、パケット長は 1792 バイトです。

2.4.2 外部仕様

1. 各システムのコマンドラインインタフェース(例えば WINDOWS の cmd、LINUX の bash)を起動する。
2. 作業ディレクトリをコンパイルされた MulticastClient.class のディレクトリに変換する。
3. Java MulticastClient multiIP を入力する。multiIP は参加したいグループの IP アドレスです。
4. ファイルをゲットしている時は、「Getting: (file name)」というメシージュを提示する。
5. 受信成功する時は、「Get done」というメシージュを提示する。
6. 対応の MulticastServer が終了する時「..END...」を提示して、プロセスは自動終了する。

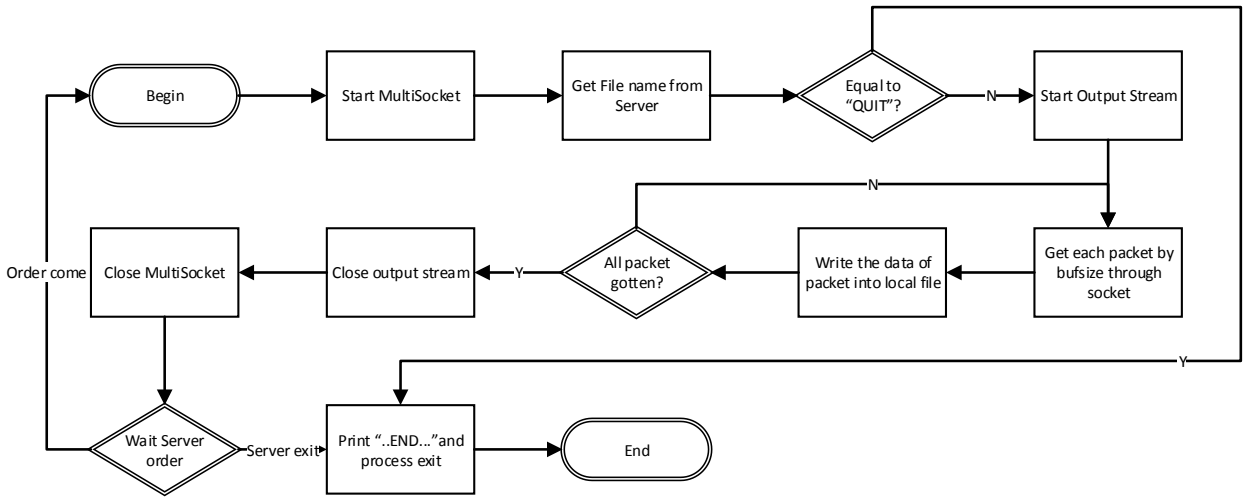
注意:
一旦運行始め、Ctrl+C までずっと受信受けます。ご不用の時止めることをご注意ください。
運行システムに JRE(Java Running Environment)は必要です。

2.4.3 内部仕様

1. 主クラス MulticastClient:
    - グローバル静的変数:
        - UNI_PORT: 今回使うポート番号
        - BUFSIZE: 毎転送パケットが使うバッファーの大きさ
    - インタフェースメソッド main(String[] args):
        - 変数:
            * out:原出力ストリーム
            * bout: バッファー出力ストリーム
            * serverSocket: マルチキャスト UDP ソケット
            * getPacket:UDP 受信用パケット
            * ipadr: 転送目標 IP アドレス

* port: 転送目標ポート番号
* buf,wbuf: 出力バッファー

– フローチャート



2.4.4 実行例

```
→ c0 git:(master) ✗ > java MulticastClient 224.13.89.1
Getting:cat.jpg
Get done.
Getting:data.jpeg
Get done.

..END...
```

2.4.5 ソースコード

```
/*
 *
 * java MulticastClient multiIP
 *
 */

import java.io.*;
import java.net.*;

public class MulticastClient {
    static final int MULTI_PORT = 50000;
    static final int BUFSIZE = 1792;

    public static void main(String[] args) {
        int len;
        int i;
        byte[] buf = new byte[BUFSIZE];
        byte[] wbuf = new byte[BUFSIZE];      //buf for filename words
        MulticastSocket sock = null;
        InetAddress ipadr;
        FileOutputStream out = null;          //file
        BufferedOutputStream bout = null;     //bin file
```

```
DatagramPacket getPacket = null;     //get packet
if (args.length != 1) {
    throw new IllegalArgumentException(
            "usage: >java multicastClient <IP_Address>");
}
try {

    sock = new MulticastSocket(MULTI_PORT);
    ipadr = InetAddress.getByName(args[0]);
    sock.joinGroup(ipadr);
    while (true) {

        DatagramPacket packet = new DatagramPacket(buf, buf.length);
        sock.receive(packet);
        //trim and get filename
        String recvdata = new String(packet.getData());
        recvdata = recvdata.trim();
        wbuf = recvdata.getBytes();
        i = 0;
        len = wbuf.length;
        buf = new byte[BUFSIZE];
        while (wbuf[i] != 0x0d && wbuf[i] != '\n') {
            buf[i] = wbuf[i];
            i++;
            if (i >= len) {
                break;
            }
        }
        recvdata = new String(buf, 0, i);
        if ("QUIT".toString().equalsIgnoreCase(recvdata)) {
            break;
        }
        System.out.println("Getting:" + recvdata);
        //output stream begin
        out = new FileOutputStream(recvdata);
        bout = new BufferedOutputStream(out);
        buf = new byte[BUFSIZE];
        i = 0;                          //length of packet
        //get file
        while (true) {
            getPacket = new DatagramPacket(buf, 0, buf.length);
            sock.receive(getPacket);
            //System.out.println(new String(getPacket.getData()));
            i = getPacket.getLength();
            //System.out.println(i);
            bout.write(buf, 0, i);
            if (i < BUFSIZE) {
                break;
            }
        }
        //output stream end
        bout.close();
        System.out.println("Get done.");
    }
    sock.leaveGroup(ipadr);
    sock.close();
} catch (java.io.IOException e) {
    System.err.println(e);
```

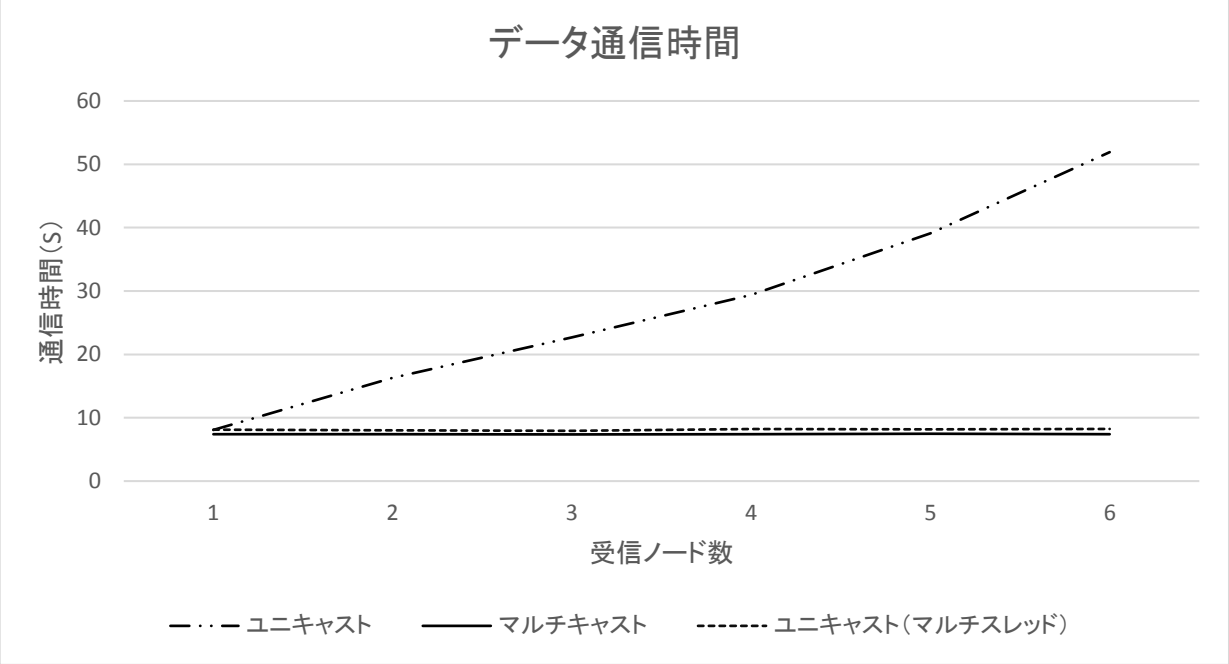
```
            }
            System.out.println("\r\n..END...");
        }
}
```

3 結果

3.1 データ通信時間

ここ転送するファイルの大きさは 20840448bits で、利用するパケット長1792 byte です
結果は下図通りです。

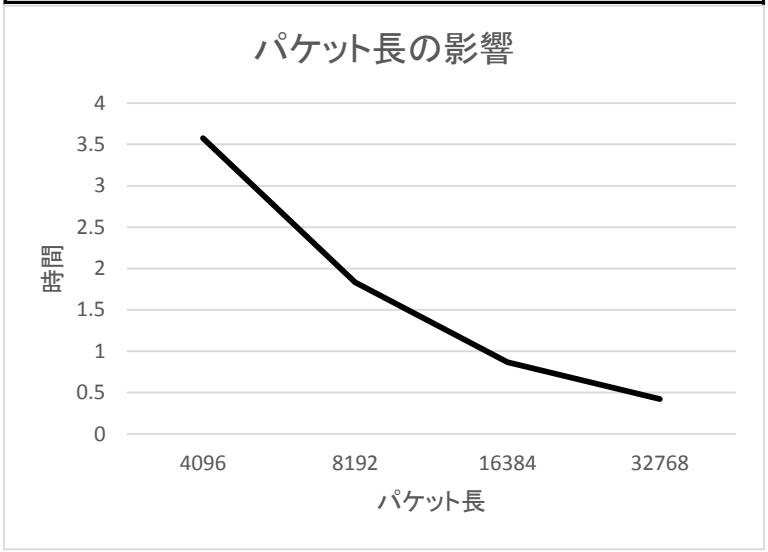
受信ノード数	1	2	3	4	5	6
ユニキャスト	8.072	16.301	22.678	29.391	39.089	51.919
マルチキャスト	7.4	7.388	7.364	7.413	7.473	7.376
ユニキャスト（マルチスレッド）	8.131	8.002	7.933	8.231	8.176	8.233



3.2 パケット長の影響

ここ転送するファイルの大きさは 237568bits です。
結果は下図通りです。

パケット長	4096	8192	16384	32768
時間	3.576	1.832	0.867	0.422



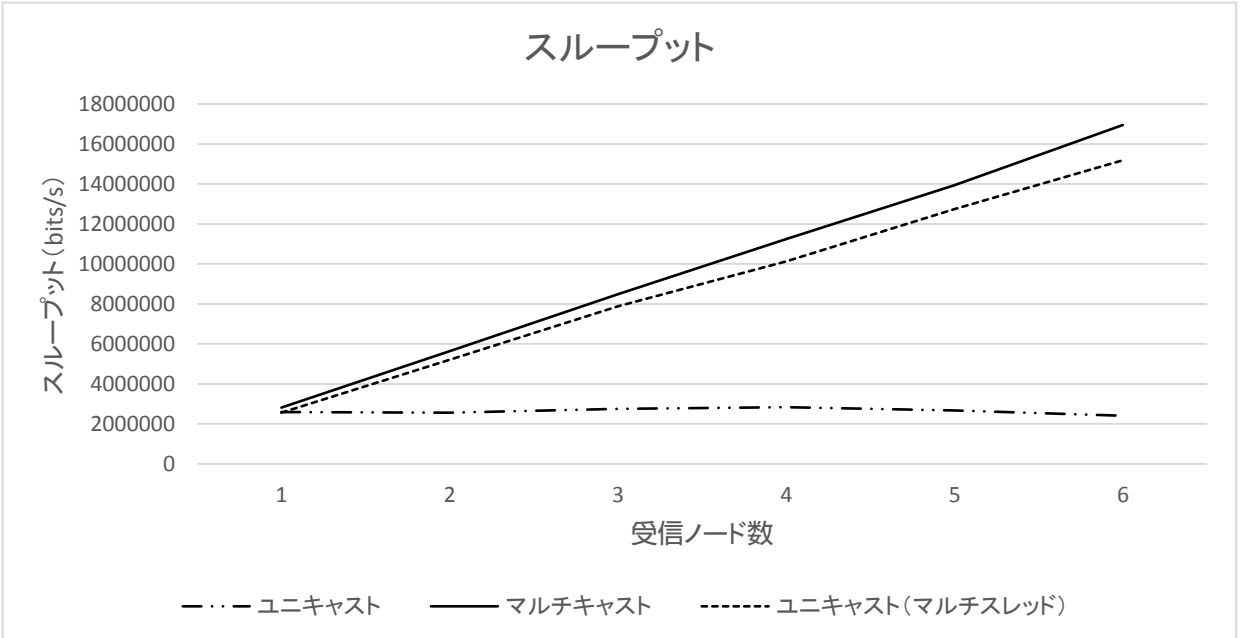
3.3 パケットロス特性

インターバルパケット発送方法を利用するので、実験の中にパケットロスという状況は発生することがありません。

3.4 スループット

ここ転送するファイルの大きさは 20840448bits で、利用するパケット長1792 byte です
結果は下図通りです。

受信ノード数	1	2	3	4	5	6
ユニキャスト	2594677	2556953	2756916	2836303	2665769	2408418.65
マルチキャスト	2816277	5641702	8490134	11245352	13943830	16952642.1
ユニキャスト（マルチスレッド）	2563085	5208810	7881173	10127784	12744892	15187985.9



4 考察

ユニキャスト通信は単一のアドレスを指定して特定の単一の相手に対してデータを送信する通信方式で、マルチキャスト通信は、特定のグループを指定複数の相手に対してデータを送信する通信方式です。

この二つの通信方式の目指す使用状況は違うから、具体的にデータ通信時間とスループットなどに差異があります。

ユニキャスト通信は単一相手に対してデータを送信する時、効率が良いですが、複数の相手に対してデータを送信する時は、時間のかかるのがリニア成長しています。一方、マルチキャスト通信のほう、複数の相手に対してデータを送信する時はいくら相手でも時間のかかるのがほぼ同じです。

他に、Java のマルチスレッドを利用して、ユニキャスト通信を最適化すると、マルチキャスト通信のように近づいたが、まったくマルチキャスト通信を利用する場合のほうが効率が良いです。

でも、マルチキャスト通信は、特定のグループのノードへ送信するしかありませんから、分散 IP への通信の場合は、マルチスレッドを利用するユニキャスト通信のほうが自由度が高いので適当です。

ですから、単一の相手に対してデータを送信する場合は、ユニキャスト通信を利用する。特定のグループを指定複数の相手に対してデータを送信する場合は、マルチキャスト通信を利用する。分散ノードへの通信する場合は、マルチスレッドを利用するユニキャスト通信のほうがよい。

5 感想

実験の原理は簡単ですが、実際に実装すればいろいろな間違いが出来ました。例えば、文字列の合法性の判断メソッドか、出力ストリームがクローズしましたかどうかなど、エラーが発生することもあります。やはり基礎のコーディング能力に油断大敵です。

JAVA 言語には、UDP で通信するために様々のクラスとメソッドもちゃんとパッケージしたから、便利に利用することができます。でも、その原理に深く理解するために、そのパッケージしたクラスを利用しないで、基本的な機能から、UDP のユニキャスト通信とマルチキャスト通信を実装できれば、十分理解しましたと言えます。

私にとっては、一番難しいことは、このレポートを作ることです。日本に住む 2 か月ぶりですから、日本人らしいにレポートをスムーズで作ることまたできません。大変時間をかかります。でも、作ったら、大変勉強になりました。これからもより良いレポートができるように一生懸命頑張ります。

6 外部リンク

1. Github project,
   site：https://github.com/univoid/Ritsumei_comm_expt,
   ZHENG YUHUI

2. Sharelatex project,
   site：https://www.sharelatex.com/project/5450846c0cd7930776998a19,
   ZHENG YUHUI