授業やゼミ向けの画面配信システム TreeVNCの機能拡張

Improvements of Screen Sharing System TreeVNC

平成25年度 卒業論文

琉球大学　工学部　情報工学科

115747H 大城 美和

指導教員 河野 真治

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究背景と目的

第2章　TreeVNC について

2.1　RFB プロトコル

RFB(remote frame buffer) プロトコル [1] とは、自身の画面を送信し、ネットワーク越しに他者の画面に表示するプロトコルである。ユーザが居る側を RFB クライアント側と呼び、Framebuffer への更新が行われる側は RFB サーバと呼ぶ。Framebuffer とは、メモリ上に置かれた画像データのことである。RFB プロトコルでは、最初にプロトコルバージョンの確認や認証が行われる。その後、クライアントに向けて Framebuffer の大きさやデスクトップに付けられた名前などが含まれている初期メッセージが送信される。RFB サーバ側は Framebuffer の更新が行われるたびに、RFB クライアントに対して Framebuffer の変更部分だけを送信する。更に RFB クライアントの FramebufferUpdateRequest が来るとそれに答え返信する。RFB プロトコルは、描画データに使われるエンコードが多数用意されており、また独自のエンコードを実装することもできるプロトコルである。

2.2　TightVNC

TightVNC(Tight Virtual Network Computing)[2] は Java を用いて作成された RFB プロトコルのクライアントである。本研究で作成した TreeVNC は TightVNC を元に作成されている。

2.3　多人数で VNC を使用するときの問題点

多人数で従来の VNC を使用する際、1 つのコンピュータに多人数が同時につながり、処理が集中してしまい、性能が大幅に落ちてしまうという問題が生じる。

ゼミ等の画面配信者が頻繁に切り替わる場合、配信者が替わる度にアプリケーションを終了し、接続をし直さないといけないという問題がある。

2.4　TreeVNC の構造

多人数で VNC を用いるために、クライアントの接続がサーバに一極集中してしまう問題を解決する。そのために、 TreeVNC はサーバへ接続しに来たクライアントをバイナリツリー状に接続する (図 2.1)。バイナリツリーなら、各 node に最大 2 台分のクライアントしか接続されない。N 台のクライアントが接続しに来た場合、画面配信の画像データをコピーする回数は、従来の VNC ではサーバ側で N 回する必要があるが、TreeVNC では各 node が 2 回ずつコピーするだけで済む。TreeVNC は、root への負荷を各 node に分散することにより、処理性能が向上している。

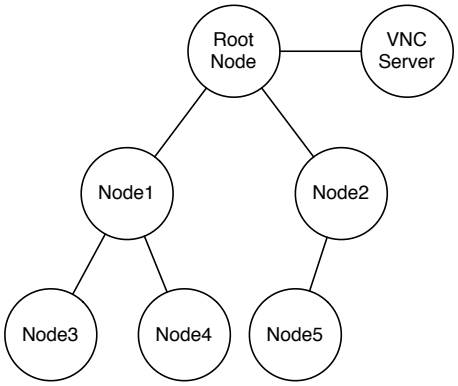

図 2.1: 構成される木構造

2.5　Multicast や Broadcast を用いた VNC

Multicast とは、同一ネットワーク内でマルチキャストアドレスを持っている端末に対してデータを送信することである。Broadcast とは、同一ネットワーク上の全ての端末に対してデータを送信することである。どちらの通信方法も、root からのデータ送信は 1 回でよく、1 度送信するとデータの複製はルータが行う。

VNC を Multicast や Broadcast の通信方法を用いて実装すると、画像データの送信が 1 度で済み、負荷分散のために木構造を形成する必要もなくなる。

しかし、これらの通信方法でのパケットの扱いには

- 送信可能なパケットのブロックサイズが 64000byte までであると決まっている
- パケットが途中で消失してしまっても特定することができない

といった性質がある。

VNC でこれらの通信方法を用いて実装する場合、パケットの扱いの性質を乗り越えなければならない。

送信可能なパケットのサイズが決まっているので、画面データは 64000byte 以下に分割し送信しなければならない。しかし、現在の TreeVNC で用いている方法では、データ分割の処理には時間がかかってしまう。

パケットの消失を検知するためには、各パケットに対してシリアル番号を振り分ける。パケットを受信した node 側で、シリアル番号が連番で届いているのかどうかを調べれば検知することが可能である。もし届いていなかった場合は、root に対して再送要求を送信すれば良い。しかし、Multicast や Broadcast 通信ではパケットロス率が高かった。

これらの通信方法を用いての VNC の実装にはもう少し工夫が必要である。

2.6　node 間で行われるメッセージ通信

RFB プロトコルで提供されているメッセージに加え、 TreeVNC 独自のメッセージを使用している。TreeVNC で使用されるメッセージの一覧を表 2.1 に示す。

通信経路	message	説明
send direct message (child to root)	FIND_ROOT	子 node 接続時に root を探す。
	WHERE_TO_CONNECT	どの node に接続すれば良いかを聞く。
	LOST_PARENT	親 node の接続が切れた時に root に知らせる。
send direct message (root to child)	FIND_ROOT_REPLY	FIND_ROOT への返信。
	CONNECT_TO	node と node の接続をする。
	CONNECT_TO_AS_LEADER	左子 node として、node と node の接続をする。
message down tree (root to child)	FRAMEBUFFER_UPDATE	画面の差分の画像データ。EncodingType を持っている。
	CHECK_DELAY	通信の遅延を測定する message。
message up tree (child to root)	CHECK_DELAY_REPLY	CHECK_DELAY への返信。
	SERVER_CHANGE_REQUEST	画面切り替えのリクエスト。
send message (root to VNCServer)	FRAMEBUFFER_UPDATE_REPLY	FRAMEBUFFER_UPDATE のリクエスト。
	SET_PIXEL_FORMAT	pixel 値の設定。
	SET_ENCODINGS	pixel データの encodeType の設定。
	KEY_EVENT	キーボードからのイベント。
	POINTER_EVENT	ポインタからのイベント。
	CLIENT_CUT_TEXT	テキストのカットバッファを持った際の message。
send message (VNCServer to root)	FRAMEBUFFER_UPDATE	大本の画面の差分の画像データ。EncodingType を持っている。
	SET_COLOR_MAP_ENTRIES	指定されている pixel 値にマップする RGB 値。
	BELL	ビープ音を鳴らす。
	SERVER_CUT_TEXT	サーバがテキストのカットバッファを持った際の message。

表 2.1: 通信経路と message 一覧

図 2.2 は、TreeVNC で node が root に接続し、画像データを受信するまでのメッセージ通信の様子である。

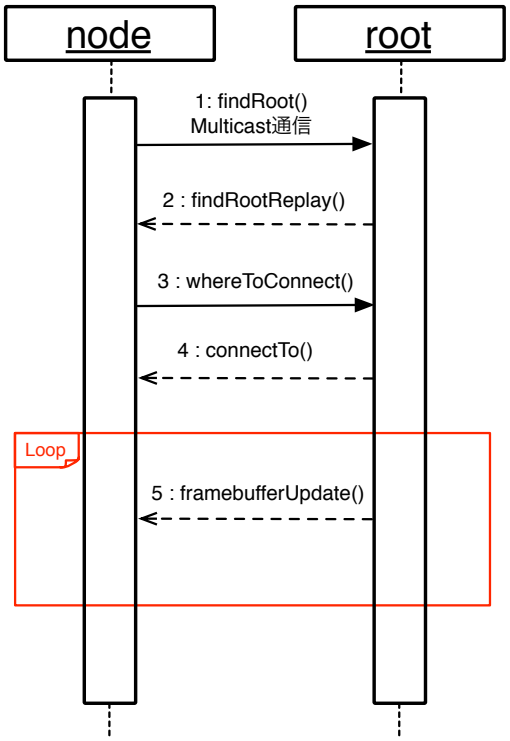

図 2.2: node 間で行われるメッセージ通信

- node は Multicast 通信で root に対して FIND_ROOT を送信する (1:findRoot())
- root が受信し、FIND_ROOT_REPLY を送信する (2:findRootReplay())
- node 側でどの root に接続するか選択するパネルが表示される
- node はパネルで root を選択し、接続先を要求する WHERE_TO_CONNECT を送信する (3:whereToConnect())
- 受信した root は node の接続先を CONNECT_TO で送信する (4:connectTo)
- node は root の指定した接続先に接続する
- root・node 間の接続が確立後、root から node に対して定期的に画像データ FRAME_BUFFER_UPDATE を送信する (4:framebufferUpdate())

2.7　配信画面切り替え

ゼミ等では参加者が順々に発表を行う度に、画面の切り替えが必要となる。ゼミを円滑に進めるために、この画面の切り替えをスムーズに行う必要がある。

プロジェクタを使用する場合、ケーブルの抜き差しを行わないとならない。その際に、ディスプレイの解像度の設定をし直す、接続不良が起こる等の煩わしい問題が生じることがある。

従来の VNC を使用する場合、画面の切り替えの度に一旦 VNC を終了し、発表者の VNCServer へと再接続を行う必要がある。

TreeVNC は、これらの配信者の切り替えの度に生じる問題を解決している。TreeVNC を立ち上げることで、ケーブルを使用せずに手元の PC に発表者の画面を表示させることができる。画面の切り替えは、ユーザが VNCSever への再接続を行うことなく、切り替えた発表者の画面を取得することができる。

TreeVNC の root は配信者の VNCServer と通信を行っており、VNCServer から画面データを受信し、そのデータを node へと送信している。画面切り替え用の share button が押されると、root は share button を押したクライアントの VNCServer と新しく通信をし直す。これにより、配信者切り替えのために VNC を終了し、再接続する必要がなくなる。

しかし、配信者が変わることで起きる画面サイズの違いや、マルチディスプレイ全体を表示してしまうことがあるので、それを解決する必要がある。

2.8 MulticastQueue

配信側の画面が更新されると、 VNCServer から画像データが framebufferUpdate として送られてくる。TreeVNC は、画像の更新を複数の node に同時に伝えるため、MulticastQueue という Queue に画像データを格納する。

各 node は MulticastQueue からデータを取得するスレッドを持つ。node それぞれがデータを取得するため、MulticastQueue は複数のスレッドから使用される。

MulticastQueue は、CountDownLatch を用いている。java.util.concurrent.CountDownLatch とは、java の並列用に用意された API で、他のスレッドで実行中の操作が完了するまで、複数のスレッドを待機させることのできるクラスである。スレッドを解放するカウントを設定することができ、カウントが 0 になるまで await メソッドでスレッドをブロックすることができる。

TreeVNC では、それぞれの画像データにカウントが追加され、カウントが 0 になると、その画像データは消される。親 node が接続されている子 node の数だけカウントを設定する。子 node が画像データを pull すると、そのカウントが減る。全ての子 node が画像データを pull するとカウントが 0 になり、画像データが消される。

node が MulticastQueue からデータを取得せずに、Queue にデータが溜まり続け、メモリのオーバーフローが起こる場合がある。この問題を避けるために、Timeout 用のスレッドを用意している。ある一定時間データが取得されなければ、Timeout を検知し、取得されなかったデータが全て poll されるという仕組みである。

第3章　TreeVNC のリファクタリング

3.1　動的な port 番号の指定

TreeVNC は複雑な分散アルゴリズムを用いたシステムであり、デバッグを行う環境を整える必要がある。

従来の TreeVNC は、メッセージ通信の際に固定 port 番号を複数利用していた。port 番号は一意なので、1 台で複数の TreeVNC を立ち上げることができなかった。

今回、動的に port 番号を割り当てることで、1 つの PC で複数の TreeVNC を起動することを可能にした。そして、最低限のソケットポートを開けることによって、メモリの使用量を抑えることにも繋がった。

以下に、動的に port 番号を割り当てているソースコード 3.1 を記述する。

ソースコード 3.1: 動的な port 番号の割り当て

```
1     public int selectPort(int p) {
2         int port = p;
3         while (true) {
4             try {
5                 servSock = new ServerSocket(port);
6                 acceptPort = port;
7                 myAddress = "127.0.0.1";
8                 nets.getNetworkInterfaces();
9                 break;
10            } catch (BindException e) {
11                port++;
12                continue;
13            } catch (SocketException e) {
14                e.printStackTrace();
15            } catch (IOException e) {
16                e.printStackTrace();
17            }
18        }
19        System.out.println("accept port = " + port);
20        return port;
21    }
```

selectPort メソッドは、TreeVNC を立ち上げた際に呼ばれる。ソースコード 3.1 の try 節で ServerSocket を生成する。ServerSocket 生成の際に port 番号を結びつける。この時、既に port 番号が使用されている場合、BindException error が起こる。その場合、catch 節に処理が移行する。catch 節では port をインクリメントし、continue でもう一度 ServerSocket を生成する try 節に戻る。成功するまで port はインクリメントされるので、ユニークな port 番号を使用することが可能となる。

以前は固定 port 番号を使用し message の通信を行っていたが、一意な port を割り当てられている node が通信を行うことによって、通信の際にどの port 番号が使用されているかを意識する必要がなくなった。

3.2　QUALITY モードと SPEED モード

高解像度のまま拡大・縮小の処理を行うと、PC のスペックによって描画処理に時間がかかってしまうことがある。

画像描画処理には、高画質優先の QUALITY モードと描画速度優先の SPEED モードがある。今まで TreeVNC は QUALITY モードで使用していた。

しかし、授業中に TreeVNC を使用する際、拡大・縮小をしてしまうと描画処理が重くなり遅延が生じていた。

今回、どちらのモードを使用するかをビューワから変更出来るようにした。これにより、描画処理の遅延を解決することができた。

3.3　ホスト切り替え時の挙動の修正

画面の切り替えを行う際、新しいホスト側の画面に生じたビデオフィードバックが他のユーザに配信されてしまう問題があった。

ホストの切り替えの際、新しいホスト側の viewer を閉じることでこの問題を解決した。

3.4　Tree の構成の変更

従来の TreeVNC は、クライアントの接続する木構造が単一であった。そのため、ネットワークインターフェースが違うクライアントが同じ木に混在している状況が生じた。速度の遅いクライアントが木に存在すると、そのクライアント以下の通信速度が遅くなってしまう。

この問題を解決するために、図 3.1 の様に、ネットワークインターフェース別に木構造を形成するように設計した。

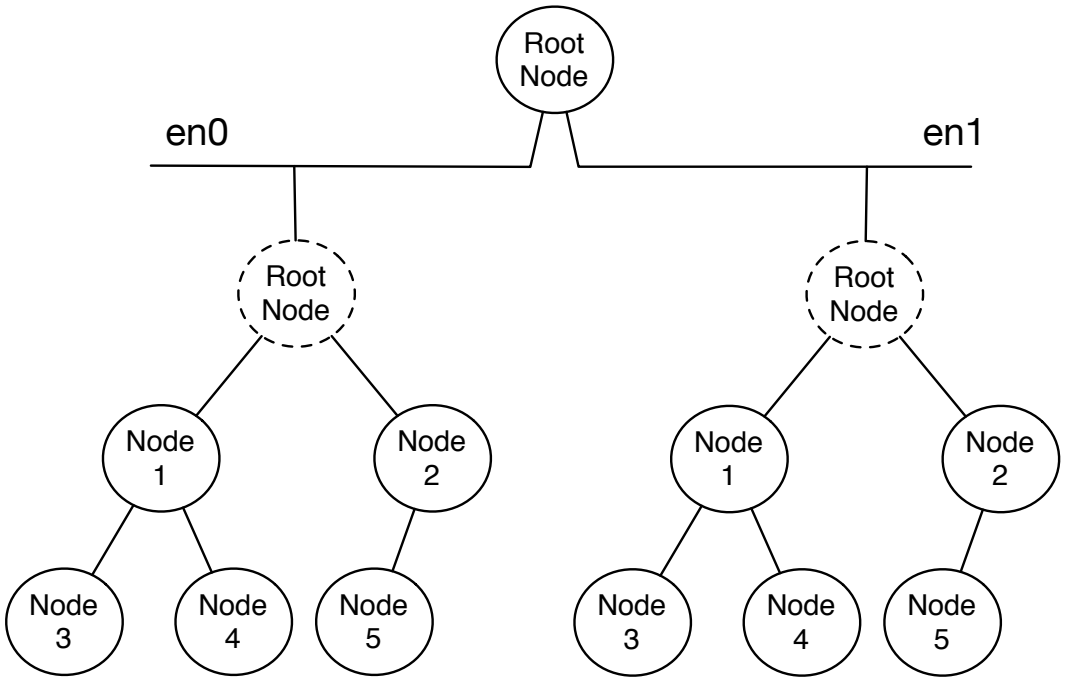

図 3.1: Multi Network Tree

TreeVNC は、root が TreeManager というオブジェクトを持っている。TreeManager は TreeVNC の接続部分を管理している。TreeManager では木構造を管理する nodeList が生成される。この nodeList を元に、新しい node の接続や、node の切断検出時の接続の切り替え等を行う。

今回、root の保持しているネットワークインタフェース毎に TreeManager を生成する様に変更した。ソースコード 3.2 に、nodeList を生成する部分を示す。

ソースコード 3.2: TreeManager の生成

```
1   public void getNetworkInterfaces() throws SocketException {
2     for (Enumeration<NetworkInterface> e = NetworkInterface.getNetworkInterfaces();e.
          hasMoreElements();) {
3       NetworkInterface ni = e.nextElement();
4       if (ni.isUp() && ni.supportsMulticast() && !ni.isLoopback()) {
5         for (InterfaceAddress ipaddress : ni.getInterfaceAddresses()) {
6           byte [] netmask = getNetMask(ipaddress);
7           String hostName = ipaddress.getAddress().getHostName();
8           TreeManagement treeManager = new TreeManagement(hostName, ConnectionParams.
                DEFAULT_VNC_ROOT, myRfb.getViewer().getShowTree());
9           treeManager.getList().getFirst().setPort(myRfb.getAcceptPort());
10          byte[] netaddr = ipaddress.getAddress().getAddress();
11          for(int i=0;i<netaddr.length;i++) {
12            netaddr[i] &= netmask[i];
13          }
14          treeManager.setNetMask(netmask,netaddr);
15          addNetworkInterface(ni, treeManager);
16          System.out.println("Interfaces␣:" + ni.getName());
17        }
18      }
19      }
20  }
```

- for 文を使用し root が所持しているネットワークインタフェースを取得する (3.2 中, 2 行目)
- その中から、起動しており Multicast に対応しており、また、ループバックインタフェースでないネットワークインタフェースを取得する (3.2 中, 4 行目)
- 取得してきたネットワークインタフェースの、ネットマスク、ホストネームを取得する (3.2 中, 6,7 行目)
- TreeManager を生成する (3.2 中, 8 行目)
- TreeManager にネットマスクとネットアドレスを追加する (3.2 中, 14 行目)
- HashMap である interfaces に ネットワークインタフェースと対応する TreeManager を追加する (3.2 中, 15 行目)

新しい node が接続してきた際、interfaces から node のネットワークインタフェースと一致する TreeManager を取得する。その TreeManager に、node 接続の処理を任せる。

こうすることによって、TreeVNC を複数のネットワークインターフェース別に木構造を構成することができる。

3.5　切断時の検知方法の変更

接続していたクライアントとの接続が切れた際の検知方法を変更した。

root は nodeList という TreeVNC のネットワークトポロジーを管理するためのリストを持っている。root は TreeVNC の接続処理の全てを担っている。node の接続が切れた場合、代わりとなる node の接続が必要となるため root に知らせなければならない。

変更前は、lostParent という検知方法を採用していた。この方法は、親となる node の接続が切れた場合、子となる node から root に対して lostParent message を送信する。これにより root は lostParent を検知し、代替 node の接続を行う。

以下に、lostParent の検知・再接続方法を記述する。

- 親 node の接続が切れる
- 切れた親 node に接続していた子 node が root に LOST_PARENT message を送信する
- root が nodeList の更新を行う
- 切れた親 node の代わりに、nodeList の最後尾 node を配置する
- 親 node を失った子 node は、新しい親 node に接続する

この方法では、子のいない末端の node の接続が切れた際に root にメッセージが送信されない。root は切断を検知できないと、nodeList の更新を行うことができない。nodeList が正しく更新されない場合、図 3.2 のように、新しい node を既に切断されている node に接続しようと試み、失敗してしまう。

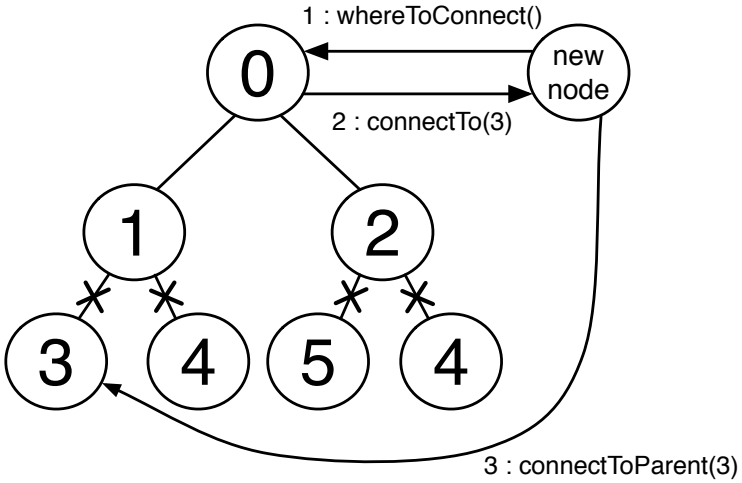

図 3.2: lostParent

末端 node の切断が検知できない問題を解決するために、lostChild という検知方法に変更した。

TreeVNC は、画像データ (framebufferUpdate) が MulticastQueue という Queue に蓄積される。node はこの Queue から画像データを取得し、描画している。lostChild の検出方法は、この MulticastQueue を使用している。ある一定時間、MulticastQueue から画像データが取得されない場合、その node との接続が切れたと判断することができる。

以下に、lostChild の検知・再接続方法を記述する。

- 子 node の切断を検知した node が root へ LOST_CHILD message を送信する (図 3.3 中, 1:)
- LOST_CHILD message を受け取った root は nodeList の更新を行う (図 3.3 中, 2:)
- 切断した node を nodeList から消し、nodeList の最後尾の node に切断した node number を割り当てる
- root は最後尾の node に、切断した子 node が接続していた親 node に接続する様に CONNECT_TO message を送信する (図 3.3 中, 3:)
- 最後尾の node が子 node を失った親 node へ接続しに行く (図 3.3 中, 4:)

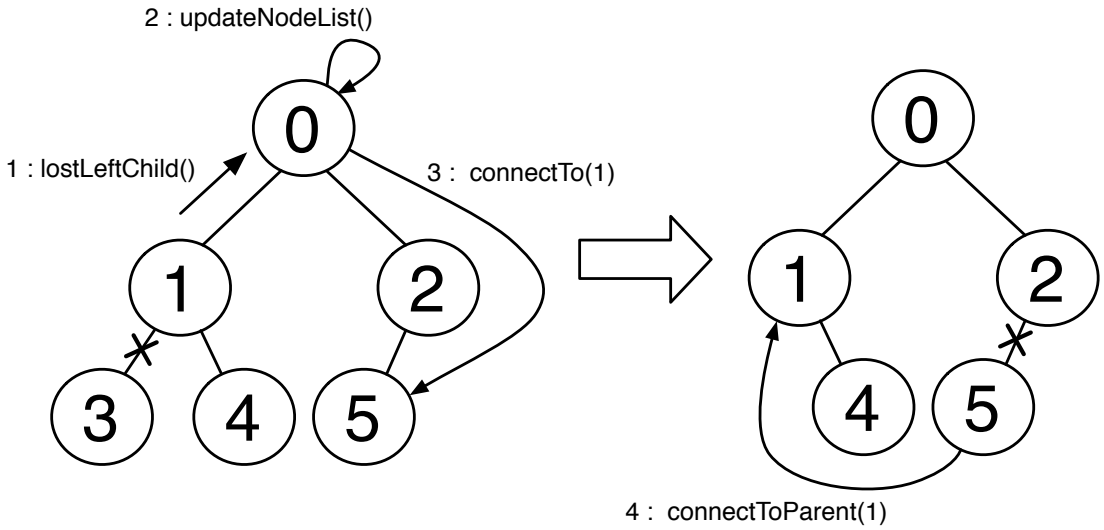

図 3.3: lostChild を検知・再接続

lostChild を検知することより、切断されてしまった全ての node を検知することができるので、nodeList の更新が正しく行われる。

新しい node からの接続要求 WHERE_TO_CONNECT message に対して、適切な node への接続を提供することができる (図 3.4 中, 1,2:)。

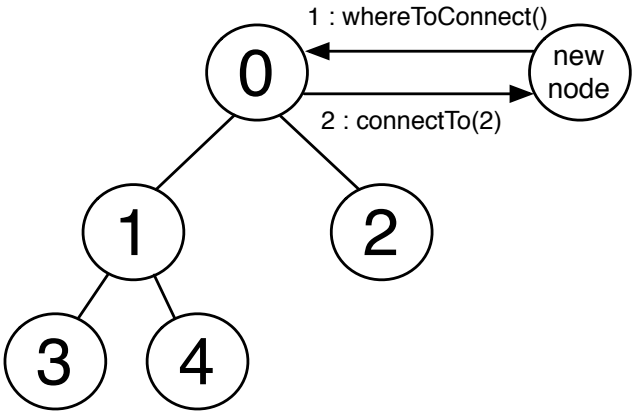

図 3.4: 新 node の接続

第4章　TreeVNC の新機能

4.1　表示画面サイズ調整機能

TreeVNC は、配信側の解像度を配信するので画質が荒くなることはない。しかし、配信側とクライアントで画面サイズに差がある場合、画面に入らない、或いは表示画面が小さすぎる等の問題が生じる。

今までは、ユーザが viewer に用意されている拡大・縮小ボタンを使用し調整していた。

今回、ビューワに HD ボタンと fit screen ボタンを追加した。HD ボタンを押すと、画面サイズが 1920x1080 サイズに拡大・縮小される。fit screen ボタンを押すと、クライアントの画面サイズに合わせてフルサイズで拡大・縮小される。

更に、root として起動し viewer も表示される -d オプションを使用した場合は、表示される画面が常にフルサイズに調整されるよう実装した。

4.2　配信画面サイズ指定機能

配信する画面サイズを指定できるオプションを追加した。TreeVNC 起動時にオプション (--fixSize) を追加することによって、指定した幅・高さの画面サイズのみを配信することができる。起動方法をソースコード 4.1 に記述する。

ソースコード 4.1: オプション–fixSize

```
1  java -jar TreeVNC.java -d --fixSize 1920 1080
```

VNCServer からは、配信する側の画面全体のデータが送信される。root は指定したサイズ領域のデータのみを表示するため、領域内の更新のみを node に送信し、領域内のみを描画している。そして、VNCServer へ更新データを要求する際は、領域内のみの画像データを要求する。これにより、node に指定された領域以外は表示されない。

4.3　マルチディスプレイ対応

画面配信側がマルチディスプレイの場合でも、VNCServer からは全画面データが送信されるので、配信側の保持している画面全てが共有される。しかし、プレゼンテーションを行う際、複数枚の画面表示が要らない場合がある。

そこで、一画面のみをフィルタリングし表示するためのオプション機能 (--filterSingleDisplay) を追加した。オプションを追加した起動方法をソースコード 4.2 に記述する。

ソースコード 4.2: オプション–filterSingleDisplay

```
1  java -jar TreeVNC.java -d --filterSingleDisplay
```

root は全画面データから一画面のみをフィルタリングする必要がある。シングルディスプレイサイズは、個々のクライアントでしか取得できない。配信側は画面の切り替えを行う際に、シングルディスプレイサイズを取得する。そして、root へ送信する画面切り替えの要求メッセージ SERVER_CHANGE_REQUEST にシングルディスプレイサイズを付加する。

root はメッセージを受け取り initData を変更する。本来 initData は、RFB プロトコルで行われる通信中に VNCServer から受信する ServerInit message から生成される。マルチディスプレイの場合、ServerInit message は全画面サイズ様に生成されている。なので、そのままの initData を使用すると複数画面全体を描画してしまう。それを避けるため、initData をシングルディスプレイサイズ用の originalInitData に生成し直す。図 4.1 の様に、root は接続されている node へ originalInitData を送信する。

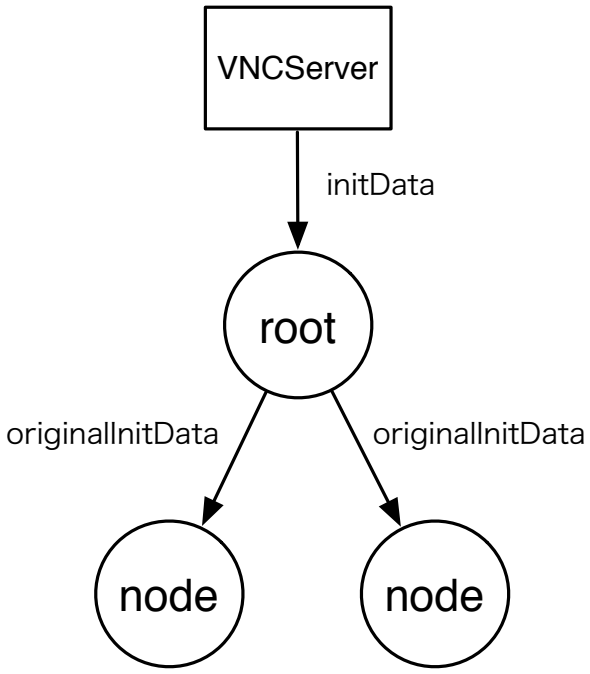

図 4.1: シングルディスプレイサイズ用の initData

さらに VNCServer から送信されてきた全画面データをそのまま node に流すのではなく、シングルディスプレイサイズの領域の更新部分のみを root 側でフィルタリングし流す。これにより、一画面のみの表示が可能となる。

4.4　Retina のマルチディスプレイ対応

Retina ディスプレイ等の高解像度ディスプレイには、より画素密度を高く表示する HiDPI (High-dot per inch) モードがある。HiDPI モードの場合、PC で設定する解像度に対して縦横 2 倍の画像データを表示している。TreeVNC でシングルディスプレイサイズを取得すると PC で設定する解像度のサイズになるが、VNCServer から送信される画像データサイズは解像度の 2 倍のサイズになる。

マルチディスプレイ対応のためには、シングルディスプレイサイズを VNCServer から送信される画像データサイズに合わせなければならない。そこで、HiDPI モードであるかどうかを検知する必要がある。

VNCServer は、接続されているディスプレイサイズを合わせて画像データを送信してくる。HiDPI モードであるかどうかを検知には、1 枚目以降のディスプレイサイズが必要となる。

以下に、HiDPI モードの取得方法を記述する。

- VNCServer から送信される width は図 4.2 の VNCServer Single width である
- VncServer Single width から、図 4.2 の 2nd Single width を引く
- 余りのサイズが取得してきた図 4.2 Single width の 2 倍であれば、HiDPI モードである

HiDPI モードの場合、originaiInitData を取得してきたシングルディスプレイの 2 倍サイズで生成する。この方法を用いて、HiDPI モードでもマルチディスプレイ対応ができた。

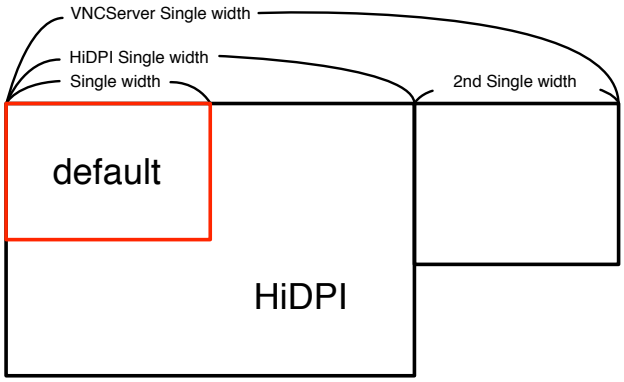

図 4.2: Single Display Width

4.5　遠隔地からの接続

遠隔地からでもゼミや授業に参加できるよう、異なるネットワークインタフェースから TreeVNC への接続を可能にした。

遠隔地からの接続を実現した TreeVNC を図 4.3 に示す。図 4.3 では、ネットワーク A で立ち上げた TreeVNC に対し、遠隔地のネットワーク B, C, D から接続している状態である。図の様に、各ネットワーク毎に TreeManager を持つ root node が存在する。TreeManager を持つ node は、そのネットワーク上での接続の木構造を管理する root となる。遠隔地ネットワークから直接 TreeVNC に接続した node は root となる。

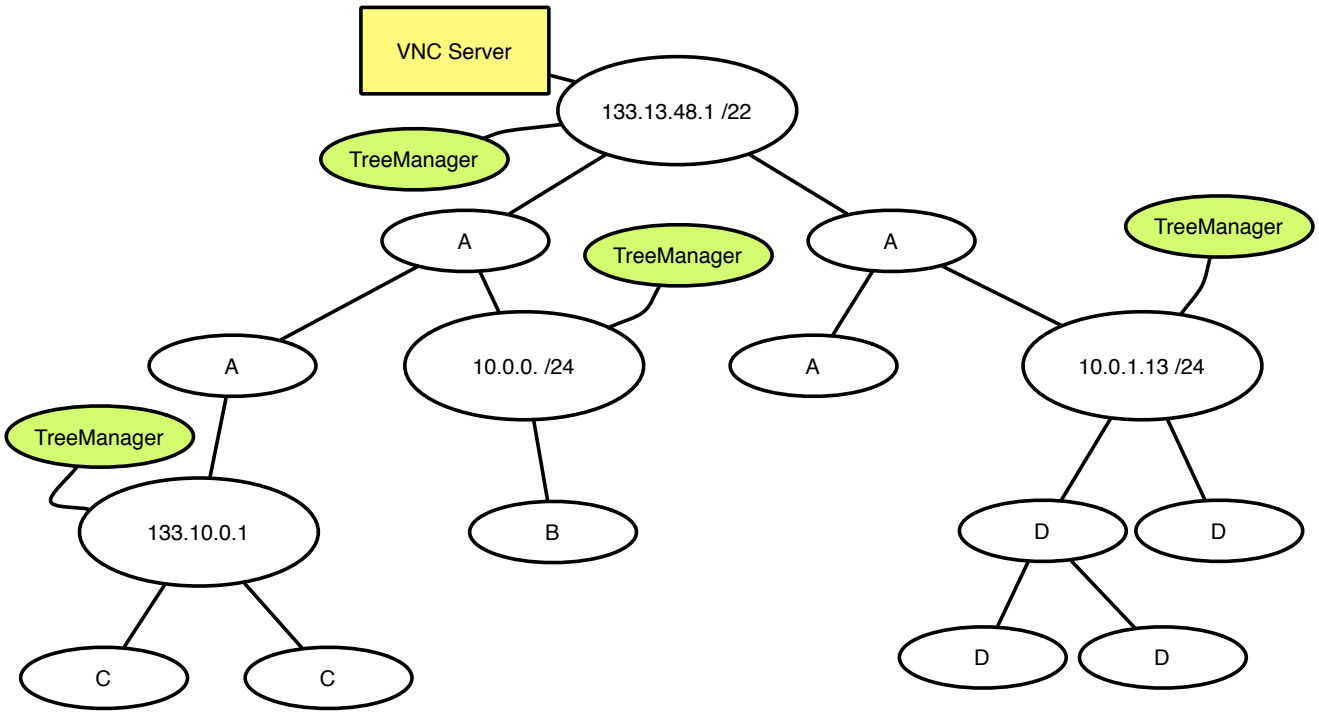

図 4.3: Remote Network Tree

以下に、遠隔地からの接続の手順を記述する。

- 遠隔地 node から接続したい root に対して接続を要求する WHERE_TO_CONNECT message を送信する (図 4.4 中, 1:)
- root は、遠隔地 node に対して接続先を含む CONNECT_TO message を送信する (図 4.4 中, 2:)
- 遠隔地 node は指定された接続先に対して接続しに行く (図 4.4 中, 3:)

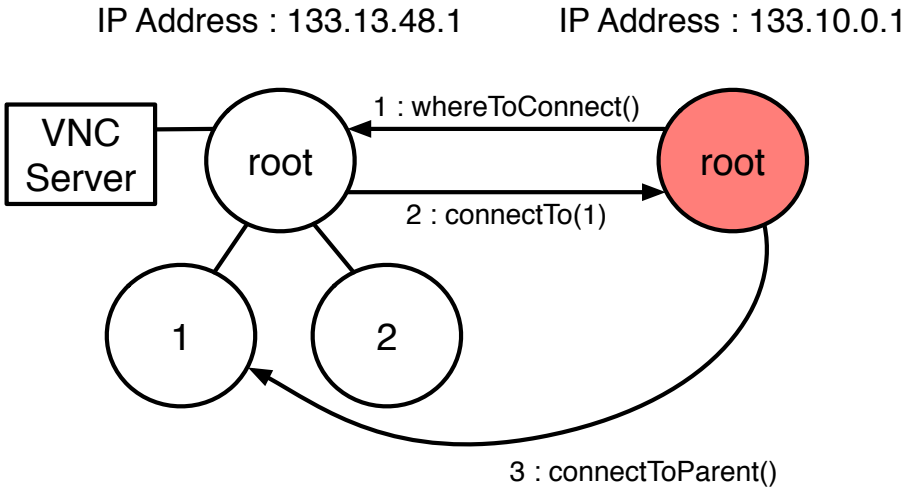

図 4.4: 遠隔地 node からの接続

ネットワーク毎に TreeVNC の木構造は管理される。図 4.5 の TreeVNC の木構造はバイナリツリーを形成している。しかし、遠隔地 node が接続している node には合計 3 つ node が接続している。遠隔地 node は、root の管理する nodeList に追加されない。これは、node の接続・切断・切り替えに遠隔地 node を関与させないためである。

ネットワーク毎に root が存在する。root は同じネットワーク上の新しい node からの接続を受け付ける (図 4.5 中, 1,2,1',2':)。

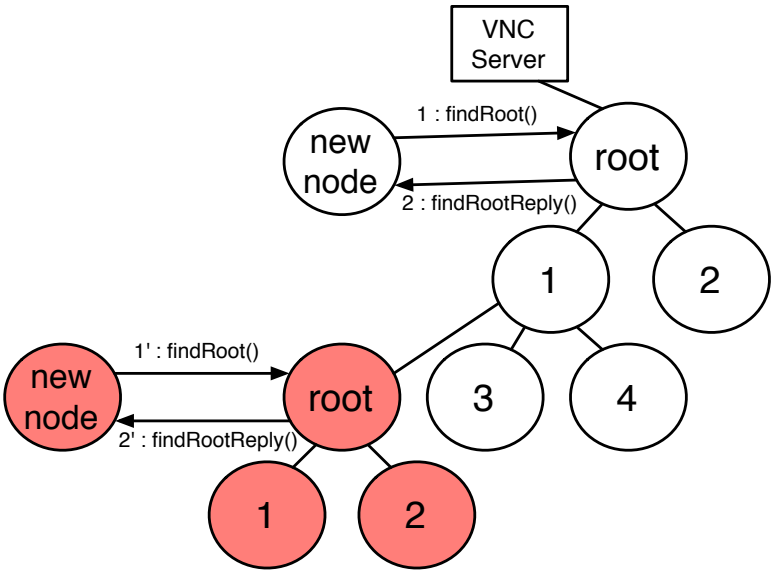

図 4.5: Remote Network Tree

遠隔地 node からでも、画面を配信できるようにする。遠隔地 node から画面配信を行う場合の画面の切り替えを図 4.6、図 4.7 に示す。以下に、画面切り替えの説明を記述する。

- 遠隔地 node が親 node へと SERVER_CHANGE_REQUEST を送信する (図 4.6 中, 1:)
- SERVER_CHANGE_REQUEST には、id が付いており、遠隔地 node からであれば -1 が付加されている
- SERVER_CHANGE_REQUEST は root へと送信される (図 4.6 中, 2:)
- root は SERVER_CHANGE_REQUEST の id を確認し、id = -1 の場合、遠隔地 root に対して WHERE_TO_CONNECT を送信する (図 4.6 中, 3:)
- 遠隔地 root は WHERE_TO_CONNECT の返信として、接続先を含む CONNECT_TO を送信する (図 4.7 中, 4:)
- root は指定された接続先へと接続しに行く (図 4.7 中, 5:)

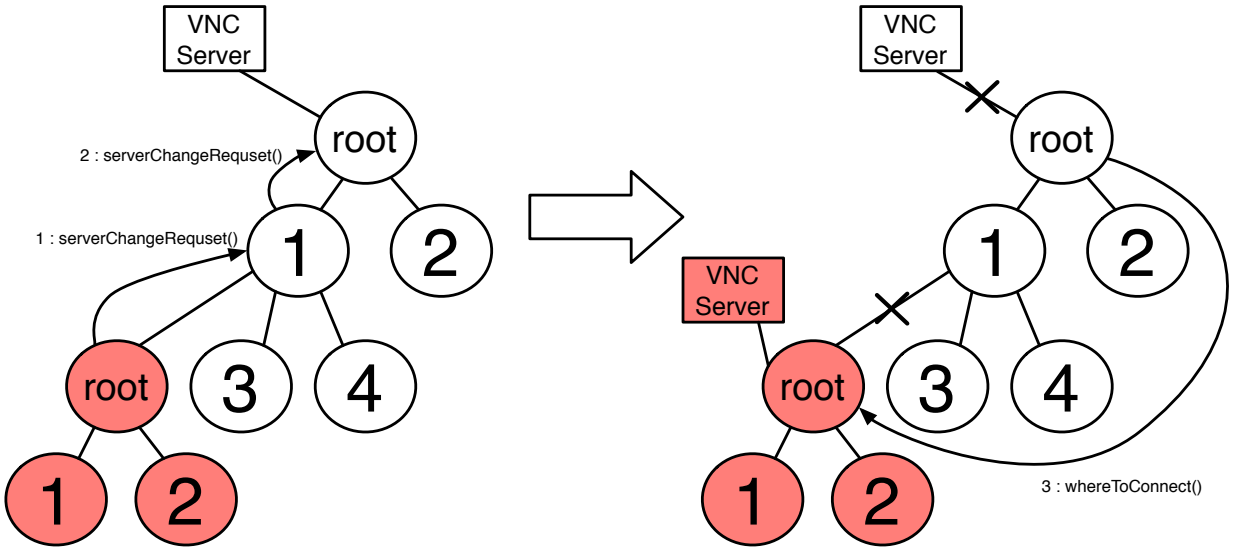

図 4.6: Remote Network Tree

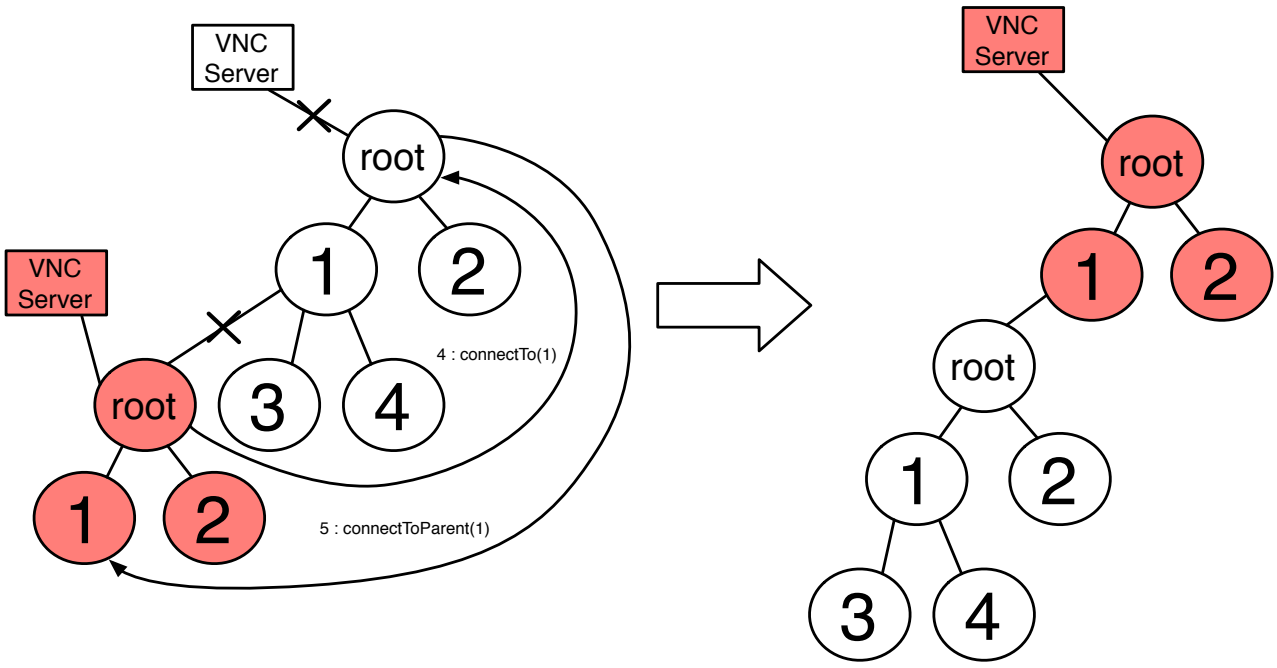

図 4.7: Remote Network Tree

第5章　TreeVNC の評価

5.1　木の深さによるメッセージ伝達の遅延

VNCServer から受信する画像データ、TreeVNC で扱われるメッセージ通信は構成された木を伝って伝達される。接続する人数が増える毎に木の段数は増えていく。そこで root から木の末端の node まで、メッセージが遅延することなく伝達できているかを検証する実験を行った。

5.2　実験環境

授業を受講している学生が TreeVNC を使用した状態で実験を行った。TreeVNC には最大で 34 名が接続していた。

5.3　メッセージを使用した実測

TreeVNC を伝搬するメッセージに、CHECK_DELAY・CHECK_DELAY_REPLY を追加した。CHECK_DELAY は root から node の末端まで伝達するメッセージ (図 5.1, 左)、CHECK_DELAY_REPLY は各 node から root まで伝達するメッセージ (図 5.1, 右) である。

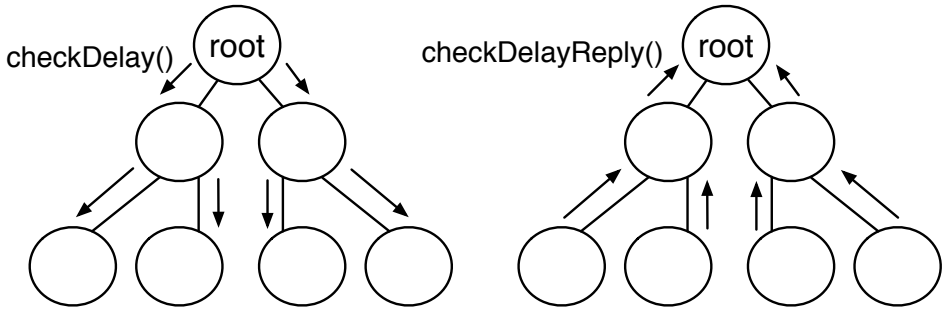

図 5.1: CHECH_DELAY, CHECK_DELAY_REPLY

CHECK_DELAY message は、送信時刻を付けて送信する。root から CHECK_DELAY 送信し、末端 node まで各 node を伝いながら伝達して行く。

CHECK_DELAY_REPLY には、CHECK_DELAY から受け取った送信時刻をそのまま付つけて送信する。CHECK_DELAY を受け取った各 node は、CHECK_DELAY_REPLY を接続している親 node に送信する。

CHECK_DELAY_REPLY を受け取った root は、メッセージの伝達にどれだけの時間がかかったかを計算する。

計算方法を以下のソースコード 5.1 に記述する。各 node にデータを下ろす際も、root にデータが上る際も、木を伝い受け渡されている。なので、データが root から末端 node に伝わる時間は、CHECK_DELAY を送信した時間と、CHECK_DELAY_REPLY を受信した時間の半分であるといえる。

ソースコード 5.1: 遅延時間の計算方法

```
1  Long delay = System.currentTimeMillis() - time;
2  double halfDelay = (double) delay / 2;
```

5.4　depth 毎の遅延結果

バイナリツリーで木を構成した場合、node 数が 34 台だと深さが 5 となる。各木構造の階層毎に、メッセージの伝搬にかかった時間を測定した。

図 5.2 は遅延の分布を示したヒストグラムである。X 軸はメッセージ伝達にかかった秒数 (ms)、Y 軸は CHECK_DELAY_REPLY を送信した node の割合を表している。

ほとんどのメッセージの伝達は 0.0 〜 4.0 ミリ秒内に収まっている。木の段数毎に、メッセージ伝達速度の差が生じている。深い段数の node ほど、メッセージ伝達速度が落ちている。

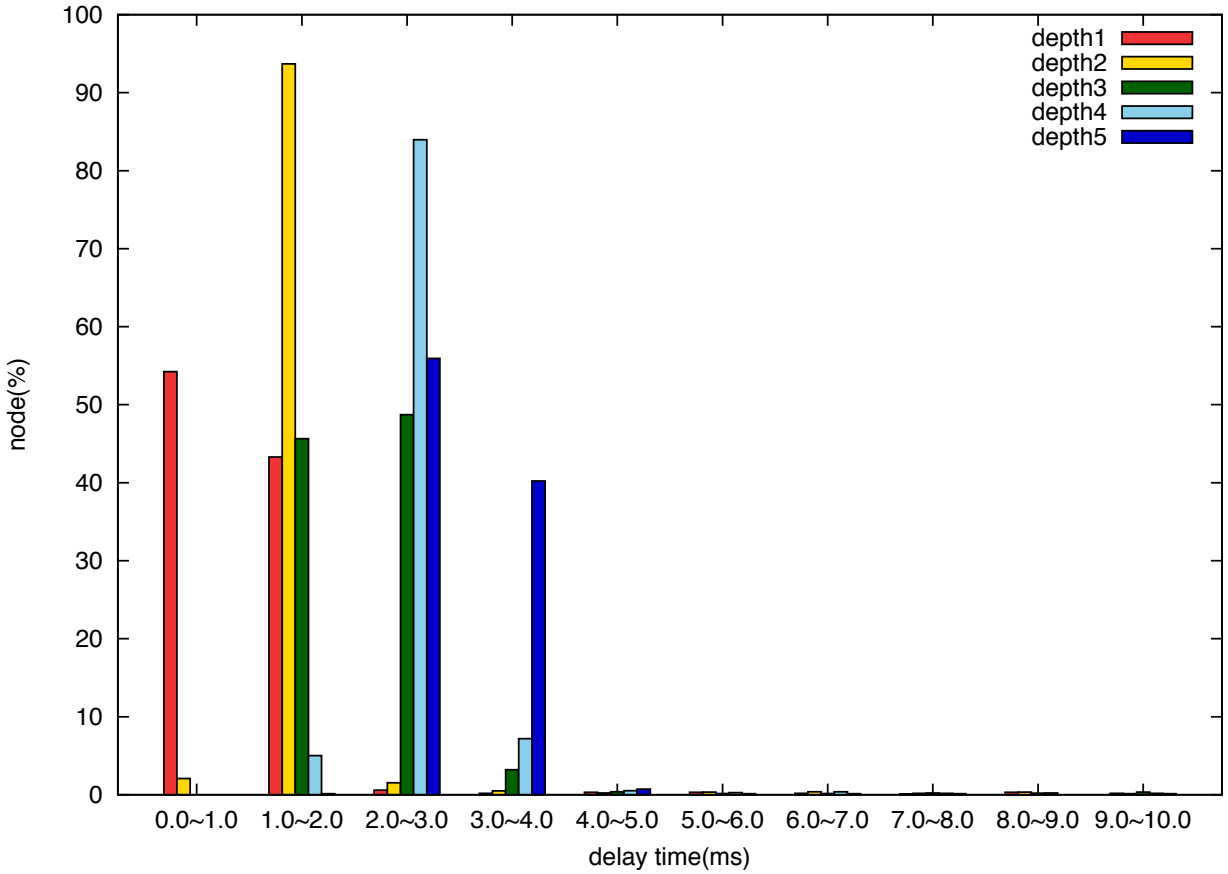
100
90
80
70
60
50
40
30
20
10
0
node(%)
depth1
depth2
depth3
depth4
depth5
0.0~1.0
1.0~2.0
2.0~3.0
3.0~4.0
4.0~5.0
5.0~6.0
6.0~7.0
7.0~8.0
8.0~9.0
9.0~10.0
delay time(ms)

図 5.2: step1

第6章　まとめ

6.1　改良点のまとめ

今回の TreeVNC のリファクタリング・新機能実装を終えて、改良した点を以下に箇条書きにまとめる。

- 1 台の PC で TreeVNC を複数台立ち上げることが可能になり、デバッグしやすくなった
- 共有された画面の描画モードの切り替え (QUALITY モード・SPEED モード) が可能になった
- 画面を切り替え時のビデオフィードバックを失くした
- 複数のネットワークインタフェースが混在することで起こる遅延を解決した
- 切断された node の検知を正しく行えるようになった
- 画面表示サイズをボタンで簡単に切り替えることが可能になった
- 配信画面サイズを指定することが可能になった
- シングルディスプレイのみの配信を可能にし、マルチディスプレイに対応できるようになった
- 遠隔地からの TreeVNC への参加ができるようになった

6.2　今後の課題

音声機能の追加

遠隔地からの接続が可能になったことにより、画面の配信は行えるようになった。しかし、遠隔地からだと画面の表示だけでなく、配信者の音声も取得したい。画像データと同様に、音声データをメッセージとして送信することは可能である。

記録機能の追加

TreeVNC 使用後、配信されていた画面データをまとめて保存したい。

質問・意見の共有

授業で TreeVNC を使用する際に、受講者から教授に対して質問できるようにしたい。授業中の書記等も全体で共有できるようにしたい。

MindMap での書記

会議等で TreeVNC を使用する際、配信者とは別に、新たに MindMap の画面を用意したい。書記の人の MindMap をリアルタイムで共有したい。あとから MindMap のデータを共有したい。

第7章　謝辞

参考文献

[1] Tristan Richardson. The rfb protocol. `http://www.realvnc.com/docs/rfbproto.pdf`.

[2] TightVNC Software. `http://www.tightvnc.com`.

[3] 谷成雄, 河野真治. 授業やゼミ向けの画面共有システム treevnc の設計と実装 a screen sharing system using tree structure for seminar and classwork. 琉球大学工学部情報工学科 平成 25 年度 学位論文 (修士), 2008.

[4] 谷成雄, 大城信康, 河野真治. Vnc を用いた授業用画面共有システムの設計・開発. 情報処理学会, may 2012.

[5] 谷成雄, 大城信康, 河野真治. Java による vnc を用いた授業用画面共有システムの設計と開発. 日本ソフトウェア科学会, sep 2011.