有線上のPC画面配信システムTreeVNCの改良

伊波 立樹[1]　河野 真治[1]

概要：ゼミや授業等で、それぞれがPC端末を持っている場合では、PCの機能を活かしたコミュニケーションが可能である。教員が操作する画面をそのまま学生に配信したり, ゼミなどで、発表する学生の画面を切り替えたりすることを可能にしたい。TreeVNCは参加したクライアントをバイナリツリー状に接続し、配信コストを分散させる仕組みを取っている。そのため, 多人数が参加しても処理性能が下がらない。また、ツリーのRootが見ているVNCサーバーを変更することで、ケーブルの差し替えなしに画面の切替が可能となる。今研究ではTreeVNCの改良として、WANへの対応、 マルチディスプレイへの対応を行う。

キーワード：

1. 研究背景と目的

ゼミや授業等で、それぞれがPC端末を持っている場合では、PCの機能を活かしたコミュニケーションが可能である。教員が操作する画面をそのまま学生に配信したり、ゼミなどで、発表する学生の画面を切り替えたりすることを可能にしたい。

TreeVNC画面配信システムは、参加したクライアントをバイナリツリー状に接続し、配信コストをクライアントにバランスさせる仕組みになっている。なので、多人数が参加しても処理性能が下がらない。また、RFBプロトコルを用いているので、ケーブルの差し替えなしに共有している画面の切り替えが可能になっている。

今研究では、 WAN、 マルチディスプレイへの対応を行った。WANへの対応として、新しい接続方法を提案し、実装を行った。また、マルチディスプレイへの対応としては配信する際に、配信するディスプレイ情報を取得し、配信を行うことで、対応した。

[1] 琉球大学工学部情報工学科

2. 画面配信システム TreeVNC

[**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。

[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。

[多人数で VNC を使用する時の問題点]

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。

[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 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。

[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 度データの送信を行う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 度で済むため、負荷分散のために木構造を形成する必要もなくなる。

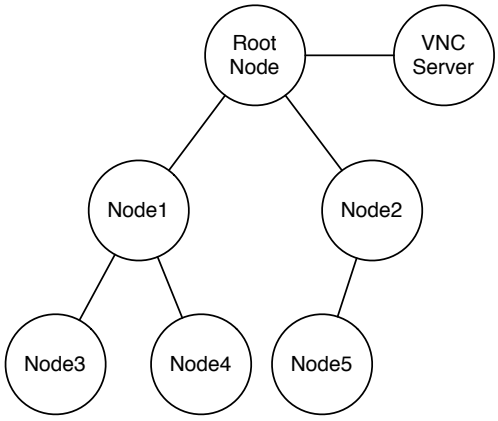

図 1 構成される木構造

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、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、消失を検知することが可能である。もしパケットが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 の実装にはもう更なる工夫が必要である。

[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 1 に示す。

通信経路	message	説明
send direct message (child to root)	FIND_ROOT	TreeVNC 接続時に root を探す。
	WHERE_TO_CONNECT	接続先を root に聞く。
	LOST_CHILD	子 node の切断を root に知らせる。
send direct message (root to child)	FIND_ROOT_REPLY	FIND_ROOT への返信。
	CONNECT_TO_AS_LEADER	左子 node として接続する。接続先の node が含まれている。
	CONNECT	右子 node として接続する。接続先の node が含まれている。
message down tree (root to child)	FRAMEBUFFER_UPDATE	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 up tree (child to root)	CHECK_DELAY_REPLY	CHECK_DELAY への返信。
	SERVER_CHANGE_REQUEST	画面切り替え要求。
send message (root to VNCServer)	FRAMEBUFFER_UPDATE_REPLY	画像データの要求。
	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	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 1 通信経路とメッセージ一覧

図 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。

手順として

- node は Multicast 通信で root に対して FIND_ROOT を送信する (1:findRoot())
- root が FIND_ROOT を受信し、FIND_ROOT_REPLY を送信する (2:findRootReplay())
- node 側で、どの root に接続するかを選択するパネルが表示される
- node はパネルで接続する root を選択し、root に対して接続先を要求する 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しに行く
- 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 (5:framebufferUpdate())

を行っている。

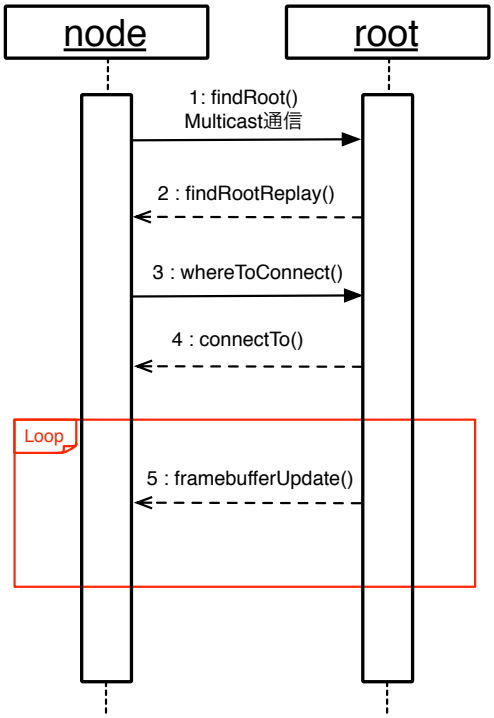

図 2 node 間で行われるメッセージ通信

[配信画面切り替え]

ゼミでは発表者が順々に入れ替わる。発表者が入れ替わる度に共有する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、画面の切り替えをスムーズに行いたい。

画面の共有にプロジェクタを使用する場合、発表者が変わる度にケーブルの抜き差しを行わないとならない。その際に、ディスプレイ解像度を設定し直す必要が出たり、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、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、ケーブルを使用する必要なしに、各参加者の手元の PC に発表者の画面を共有することができる。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、share button を押すことによって、配信者の切り替えを行うことができる。

TreeVNC の root は配信者の VNCServer と通

信を行っている。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。配信者切り替え時に share button が押されると、root は share button を押したクライアントの VNC-Server と通信を始める。TreeVNC は、配信者切り替えの度に VNC を終了し、再接続する必要がない。

しかし、配信者と受信者の画面サイズの違いや、マルチディスプレイ全体を共有してしまう問題があるので、それらを解決する必要がある。

[**MulticastQueue**]

配信側の画面が更新されると、VNCServer から画像データが FRAME_BUFFER_UPDATE message 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。MulticastQueue は複数のスレッドから使用される。

MulticastQueue は、java.util.concurrent.CountDownLatch を用いて実装されている。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 MulticastQueue を持っており、接続されている子 node の数だけ画像データにカウントを設定する。子 node が画像データを取得すると、そのカウントが減る。接続している全ての子 node が画像データを取得するとカウントが 0 になり、MulticastQueue から画像データが消される。

親 node は、接続している全ての子 node が画像データを取得するまで MulticastQueue の中に持っている画像データを削除することができない。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ると、Memory Over Flow を起こしてしまう。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。Timeout スレッドは、ある一定時間取得されない画像データがある場合、そのデータを node の代わりに poll するという仕組みである。Timeout スレッドにより、Memory Over Flow を防ぐことができる。

3. 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は描画処理に時間がかかってしまうことがある。配信者の画面をリアルタイムに取得するため、描画処理に時間のかからないモードを追加する。

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。

今回、どちらのモードを使用するかを Viewer 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。

4. マルチディスプレイ対応

画面配信側の PC がマルチディスプレイの場合、VNCServer からは複数の画面全体の画像データが送信されてしまう。

授業やゼミ等で TreeVNC を使用する場合、複数画面の表示は必要ない。そこで、画面を共有する際、ディスプレイを選択させ、画面共有を行う機能を追加した。

ディスプレイの情報は個々のクライアントでしか取得ができない。なので、配信側は画面の切替を行う際に、ディスプレイを選択し、そのディスプレイの左上と右下の座標を取得する。その座標を root への画面切り替えを要求する SERVER_CHANGE_REQUEST message に付加させる。root は配信側の VNC-Server に画像データを要求する FRAME-

BUFFER_UPDATE_REPLY message に送信された座標を付加する。VNCServer は要求された座標内の画像データを FRAMEBUFFER_UPDATE message で root に送信する。これにより、一画面のみの表示が可能となる。

図 3 は Display1 のみを画面共有する例を示している。

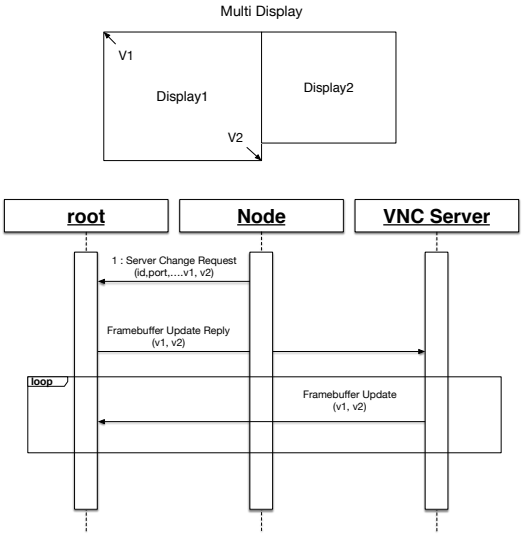

図 3 マルチディスプレイへの対応

5. 無線 LAN への対応

授業で TreeVNC を使用する場合、有線を使用するか否かは学生によって違う。TreeVNC を有線・無線の両方からの接続に対応したい。

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、単一のネットワークインターフェースでしか使用することができなかった。

この問題を解決するために、図 4 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。

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

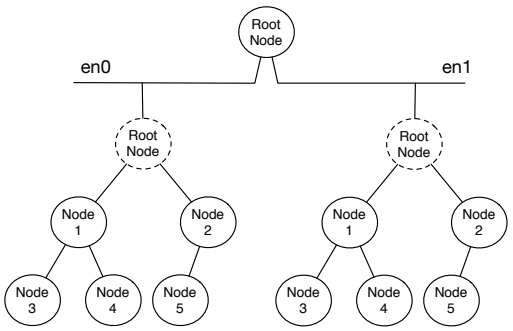

図 4 Multi Network Tree

切断検出時の接続の切り替え等を行う。

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 1 に、nodeList を生成する部分を示す。

- 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 (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 (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 (6,7 行目)
- TreeManager を生成する (8 行目)
- TreeManager にネットマスクとネットアドレスを追加する (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 (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。

6. WAN への対応

遠隔地からでもゼミや授業に参加できるよう、

```
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  }
```

Code 1　TreeManager の生成

別ネットワークから TreeVNC への接続を可能にした。

図 5 に別ネットワークからの接続を示す。別ネットワークから TreeVNC に参加する際、 直接配信側のネットワークの root に接続を行う。この接続を Direct Connection と呼ぶ。

Direct Connection した node はそのネットワークの root になり、node はそのネットワークの root に接続し、木を生成する。

配信側の root は Direct Connection で接続された node に対して Framebuffer Update で 画像データを別ネットワークの node に送信する。Framebuffer Update が送信された node はそのネットワークの root なので、子 node に対して Framebuffer Update を送信する。

これにより、別ネットワークでの画面共有が可能となる。

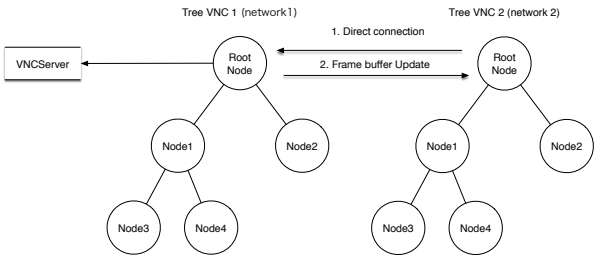

図 5　遠隔地 node からの接続

7. 評価

[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。

[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。

[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 6, 左)、CHECK_DELAY_REPLY は各 node から root まで伝達するメッセージ (図 6, 右) である。

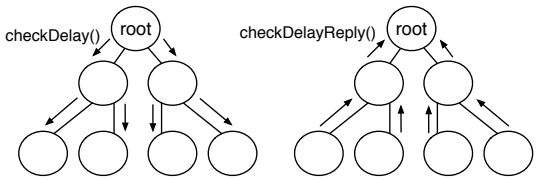

図 6 CHECH_DELAY, CHECK_DELAY_REPLY

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 2 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。

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

Code 2 遅延時間の計算方法

[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 7 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。

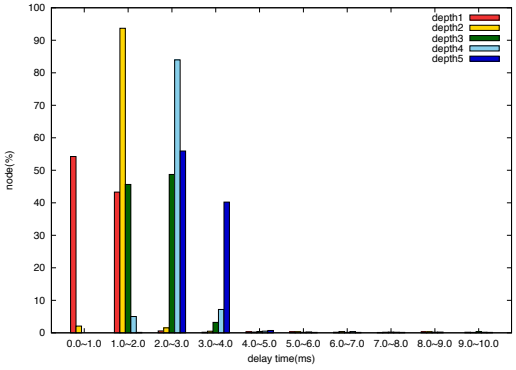

図 7 遅延の分布

8. まとめ

本研究では画面配信システム TreeVNC をマルチディスプレイ、WAN に対応させた。

マルチディスプレイに対応したことで、配信者が配信したいディスプレイを選択し、画面配信することが可能となった。

WAN に対応することで別ネットワークにいるユーザーが TreeVNC に参加することが可能となった。

今後の課題として、画面切り替えの安定化、ユーザビリティの向上、 共有機能の追加を行う。

現在の TreeVNC では、share button を押すと、その時配信されている画面から、自動的に画面が切り替わってしまうという問題がある。それを防ぐために share button が押されるとその時の配信者に切り替え確認を行う処理を追加する。

また、今回追加した Direct Connection などの一部の機能はコマンドラインオプションで指定する必要があるため、一般ユーザーでは操作するのが困難である。そこで、 今までコマンドラインオプションで指定していた機能を Viewer で操作するように変更を行う。

共有機能の追加としては、音声、質問・意見 等が上げられる。

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] Miwa OSHIRO and Shinji KONO: 授業やゼミ向けの画面配信システム TreeVNC の拡張機能，琉球大学工学部情報工学科平成 26 年度学位論文 (学士) (2014).

[4] Yu TANINARI and Shinji KONO: 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) (2013).

[5] Yu TANINARI and Nobuyasu OSHIRO and Shinji KONO: 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 (2012).

[6] Yu TANINARI and Nobuyasu OSHIRO and Shinji KONO: 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 (2011).