平成27年度 卒業論文

PC画面配信システムTreeVNCの NAT への対応

琉球大学工学部情報工学科

125716B 伊波 立樹

指導教員 河野 真治

目 次

第 1 章　画面共有を利用したコミュニケーション　1

第 2 章　TreeVNC の概念　2
2.1 VNC 2
2.2 RFB プロトコル 2
2.3 多人数で VNC を使用する際の問題 2
2.4 TreeVNC の構造 3
2.5 TreeVNC の原理 4
2.6 圧縮形式 5
2.7 通信経路 7
2.8 ノード間で行われるメッセージ通信 7
2.9 MulticastQueue 8
2.10 木の再構成 10
2.11 共有画面切り替え 11
2.12 表示画面サイズの調整 12
2.13 複数のネットワークの対応 12

第 3 章　NAT 対応　13
3.1 NAT 13
3.2 TreeVNC での NAT の問題点 13
3.3 Direct Connection 13
3.4 Direct Connection での木構造 14
3.5 NAT を越えた共有画面切り替え 15

第 4 章　マルチディスプレイ対応　17
4.1 マルチディスプレイ時の問題点 17
4.2 ディスプレイの選択 17

第 5 章　共有画面切り替えの安定化　19
5.1 画面切り替えの問題点 19
5.2 切り替え用スレッドの生成 19
5.3 切り替え用スレッドの動作 20

第 6 章　各 Node へのエラー通知 **21**
6.1 Root Node へのエラーメッセージの表示 21
6.2 ERROR_ANNOUNCE メッセージ 21

第 7 章　TreeVNC の評価 **22**
7.1 画像データ伝達の遅延 22
7.2 実験環境 22
7.3 メッセージを使用した実測 22
7.4 結果 23
7.5 ボトルネックになっている Node への対処 25

第 8 章　結論 **27**
8.1 まとめ 27
8.2 今後の課題 27

図 目 次

2.1 VNC の構造 3
2.2 構成される木構造 4
2.3 ポート一本にかかる負荷 5
2.4 ZRLE での問題点 6
2.5 ZRLEE 6
2.6 node 間で行われるメッセージ通信 9
2.7 LOST_CHILD を検知・再接続 11
2.8 Multi Network Tree 12

3.1 Direct Connection 14
3.2 新しい Node の接続 14
3.3 NAT を越えたネットワークへの画面配信 15
3.4 NAT を越えたネットワークでの画面切り替え 16

4.1 マルチディスプレイでの送信 17
4.2 マルチディスプレイへの対応 18

5.1 切り替え用スレッドの動作 20

6.1 ERROR_ANNOUNCE メッセージの挙動 21

7.1 データサイズと遅延の関係 : 深さ 1 23
7.2 データサイズと遅延の関係 : 深さ 2 24
7.3 データサイズと遅延の関係 : 深さ 3 24
7.4 データサイズと遅延の関係 : 深さ 4 25
7.5 ボトルネックになっている Node への対処 26

表 目 次

2.1 ポート一本あたりの通信量 (N はノード数, d はデータ量) 4
2.2 通信経路とメッセージ一覧 7

第1章　画面共有を利用したコミュニケーション

授業やゼミ等で、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。

通常の授業では先生の用意した資料、 PC 画面を見ながら授業が進むことが多い。ゼミでは発表者を切り替えながら発表を行う。

通常これらの画面を表示するためにプロジェクタが使用されている。しかし、プロジェクタでは通常の授業の際、参加者はプロジェクタに集中するため、手元の PC をほぼ使用することができない。更に手元の PC を使う際はプロジェクタと PC を行き来するため、目に負担がかかってしまう。またゼミの際には発表者を切り替えるたびにプロジェクタにケーブルを差し替える必要がある。ケーブルの差し替えの際に発表者の PC によってアダプターの種類や解像度の設定によって正常に PC 画面を表示できない場合もある。

画面配信システム TreeVNC[1] は発表者の画面を参加者の PC に表示する。そのため、参加者は不自由なく手元の PC を使用しながら授業を受ける事が可能になる。更に発表者の切り替えの際もケーブルの差し替えずに共有する画面の切替を可能としている。

Tree VNC は VNC[2] を使用した画面配信を行っている。しかし通常の VNC では配信側に全ての参加者が接続するため、多人数の際の処理性能が落ちてしまう。Tree VNC では有線でネットワークに接続した参加者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分散させる仕組みになっている。そのため、授業で先生の画面を表示する際、多人数の生徒が参加しても処理性能が下がらない。また、ツリーのルートが参照している VNC サーバーを変更することで、共有する画面の切替が可能となる。

しかし TreeVNC を授業やゼミで使用している中、 様々な問題が発生した。NAT を越えたネットワーク接続に対応しておらず、遠隔地などで授業やゼミに参加することが出来ない。また、ゼミの際に、マルチディスプレイを使用して画面配信を行う際、すべての画面が配信され、不必要な画面まで表示されてしまう。

そこで、本研究では上記の問題点を解決し、TreeVNC の有用性を評価することで授業やゼミを円滑に行えることを目標とする。

第2章　TreeVNC の概念

2.1　VNC

VNC(Virtual Network Computing) は、 RFB プロトコルを用いて遠隔操作を行うリモートデスクトップソフトウェアである。VNC はサーバー側とクライアント (ビューア) 側に分かれている。 サーバを起動し、クライアントがサーバに接続を行い遠隔操作を可能とする。

2.2　RFB プロトコル

RFB(remote frame buffer) プロトコル [3] 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。

2.3　多人数で VNC を使用する際の問題

VNC を使用すればクライアント側にサーバー側の画面を表示することが可能である。しかし、図 2.1 のように多人数のクライアントが 1 つのサーバーに接続してしまうと処理性能が落ちてしまうという問題点がある。

また、 ゼミ等の発表で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。

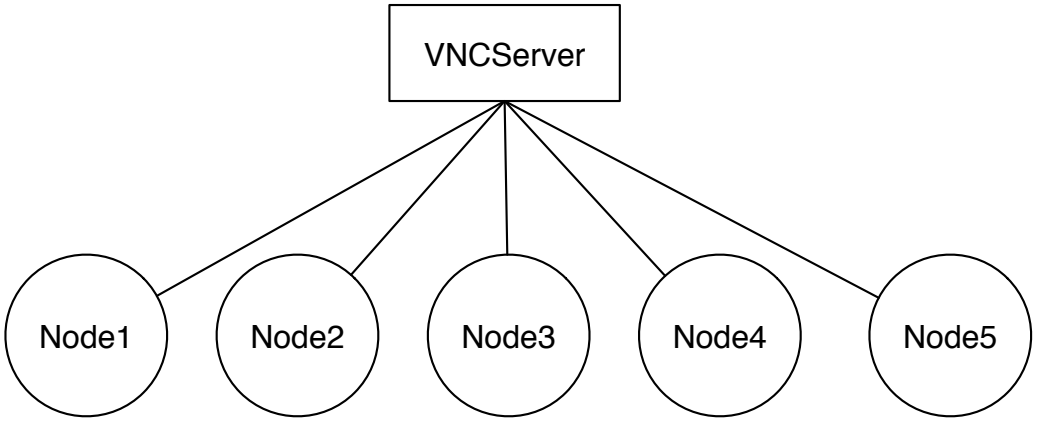

図 2.1: VNC の構造

2.4 TreeVNC の構造

TreeVNC は Java を用いて作成された TightVNC(Tight Virtual Network Computing)[4] を元に作成されている。

TreeVNC は クライアント同士を接続させ、画面描画のデータを受け取ったクライアントが次のクライアントにデータを流す方式を取っている。また、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 2.2)。バイナリツリー状に接続することで、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 では各ノードが 2 回ずつコピーするだけで済む。

TreeVNC で通信される画像のデータ量は大きいため、大きなネットワークスループットが必要である。そのため、有線接続が必須である。

バイナリツリーのルートのノードを Root Node と呼び、 Root Node に接続されるノードを Node と呼ぶ Root Node は子 Node にデータを流す機能に加え、各 Node の管理、VNC サーバーから流れてきた画像データの管理を行う。Node は 親 Node から送られたデータを 自分の子 Node に流す機能、 逆に子 Node から送られてきたデータを 親 Node に流す機能がある。

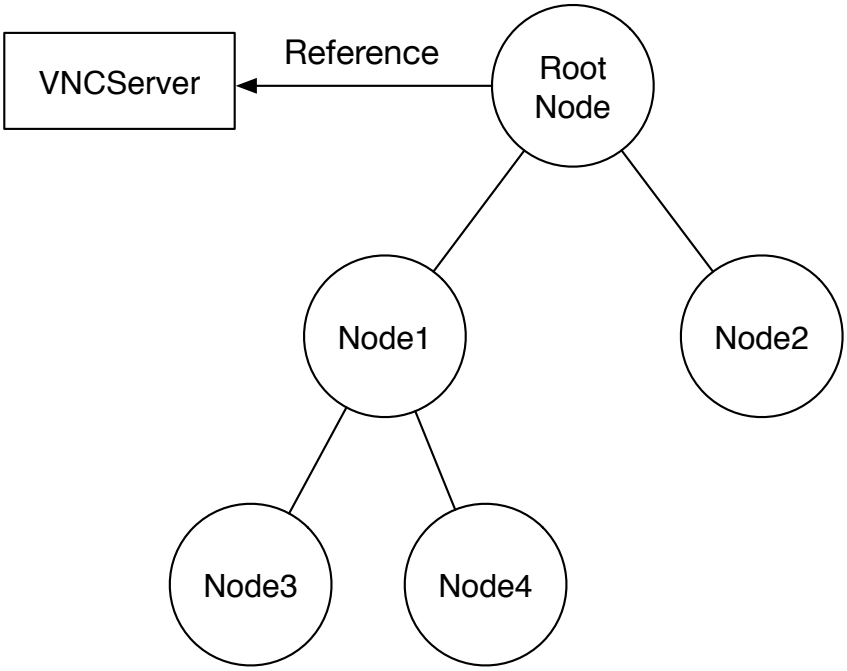

図 2.2: 構成される木構造

2.5 TreeVNC の原理

従来の VNC と TreeVNC の構造を比較を図 2.3 に示す。

表 2.1 はポート一本あたりの通信量である。通常の VNC の場合、クライアント数に比例してポート一本あたりの負荷が増えている。TreeVNC の場合は 1 つの Node に対して 2 台の Node が接続するため、最大でも親から送信されるデータと 2 つの子に送信するデータ分の負荷になる。

送信するデータ量も通常の VNC の場合 Node 数に比例した量のデータを送信する必要があり、 CPU に負荷がかかってしまう。それに対して TreeVNC はクライアントが増えても送信するデータはクライアント毎に分散されているため、1 台の CPU に掛かる負荷は一定となる。そのため、性能が低下せずに画面配信を行うことが出来る。

表 2.1: ポート一本あたりの通信量 (N はノード数, d はデータ量)

	通常の VNC	TreeVNC
通信量	N * d	(2 + 1) * d

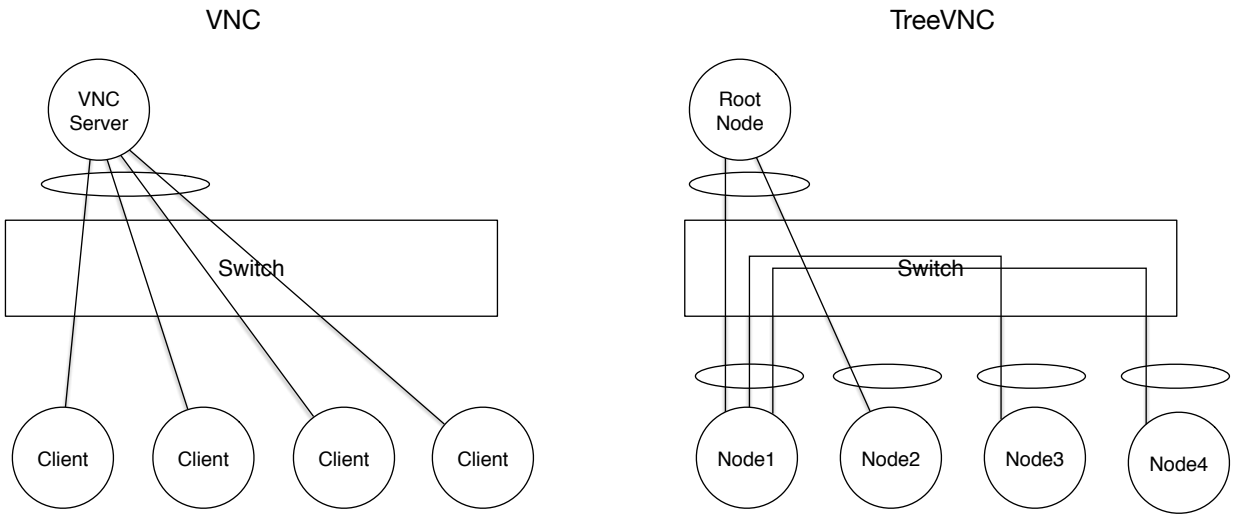

図 2.3: ポート一本にかかる負荷

2.6 圧縮形式

TreeVNC は ZRLEE[5] というエンコードでデータのやり取りを行う。ZRLEE は RFB プロトコルで使えるエンコーディングタイプの ZRLE を元に生成される。

ZRLE は Zlib[6] で圧縮されたデータとそのデータのバイト数がヘッダーとして付けて送られてくる。Zlib は java.util.zip.deflater と java.util.zip.inflater で圧縮と解凍が行える。

しかし、 java.util.zip.deflater は解凍に必要な辞書を書き出す (flush) ことが出来ない。そのため図 2.4 のように、 Zlib 圧縮されたデータを途中から受け取ってもデータを正しく解凍することが出来ない。

そこで ZRLEE は 一度 Root Node で受け取った ZRLE のデータを unzip し、 データを update rectangle という画面毎のデータに辞書を付けて zip し直すことで初めからデータを呼んでいなくても解凍を行えるようにした (図 2.5)。一度 ZRLEE に変換してしまえば子 Node はそのデータをそのまま流すだけで良い。ただし、deflater と inflater では前回までの通信で得た辞書をクリアしないといけないため、 Root Node と Node 側では毎回新しく作る必要がある。

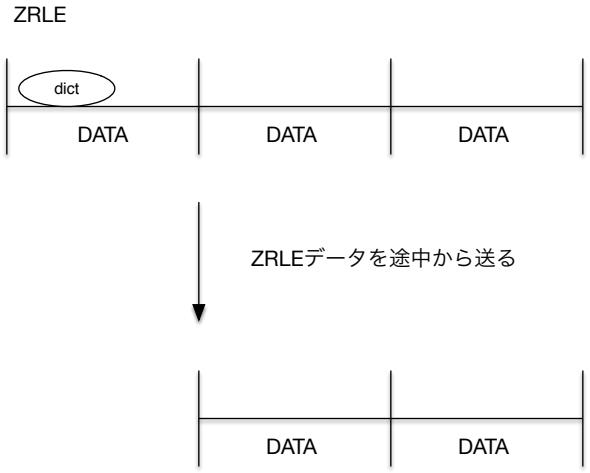

図 2.4: ZRLE での問題点

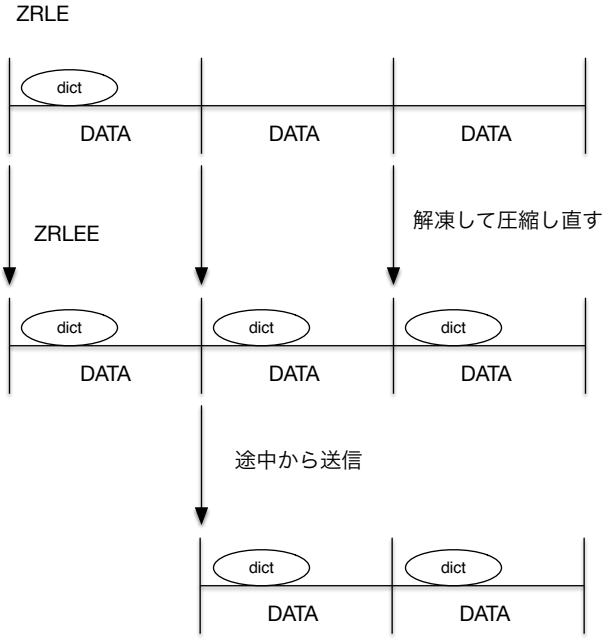

図 2.5: ZRLEE

2.7 通信経路

TreeVNC の通信経路として以下が挙げられる

- ある Node から Root Node に直接通信を行う send direct message (Node to Root)
- Root Node からある Node に直接通信を行う send direct message (Root to Node)
- Root Node から木の末端の Node までのすべての Node に通信を行う messeage down tree (Root to Node)
- ある Node から木構造を上に辿って Root Node まで通信を行う message up tree (Node to Root)
- Root Node から配信者の VNC サーバーへの通信を行う send message (Root to VNCServer)
- VNC サーバーから Root Node への通信を行う send message (VNCServer to Root)

2.8 ノード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 2.2 に示す。

表 2.2: 通信経路とメッセージ一覧

通信経路	message	説明
send direct message (Node to Root)	FIND_ROOT	TreeVNC 接続時に Root Node を探す。
	WHERE_TO_CONNECT	接続先を Root Node に聞く。
	LOST_CHILD	子 Node の切断を Root Node に知らせる。
send direct message (Root to Node)	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_TO	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 Node)	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 (Node 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。

図 2.6 は TreeVNC で 新しい Node が Root Node と通信し、木構造を形成するためのメッセージ通信の流れである。図 2.6 の手順として

- 接続を行う Node は Multicast 通信で Root Node に対して FIND_ROOT を送信する。
- Root Node が FIND_ROOT を受信し、FIND_ROOT_REPLY を送信する。
- Node 側で、どの Root Node 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 Node 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。
- 受信した Root Node は Node の接続先を CONNECT_TO で送信する。 Node 3 が接続する場合、Root Node には既に 2 台の Node が接続している為、 CONNET_TO で指定する接続先は Node 1 となる。
- Node は Root の指定した接続先に接続しに行く。

を行い、木構造を形成する。

2.9 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 メッセージとして送られる。その際、 画像データの更新を複数の Node に同時に伝えるため MulticastQueue というキューに画像データを格納する。

MulticastQueue は java.util.concurrent.CountDownLatch を用いて実装されている。CountDownLatch は java の並列用の 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が出来るクラスである。CountDownLatch を初期化する際にカウントを設定することができる。このカウントはスレッドを開放する際に使用し、await メソッドでカウントが 0 になるまでメソッドをブロックする事ができる。

MulticastQueue は put メソッドを使用して データを queue に追加する。put の際に CountDownLatch をカウントダウンする。poll メソッドを使用することで Queue のデータを取得することが出来る。poll メソッドの実行の際に await メソッドが使われているため、 次の put でデータが来るまでスレッドをブロックする。スレッドをブロックされた場合 新しいデータが put されると CountDownLatch がカウントダウンされるため、 データの読み込みが再開される。

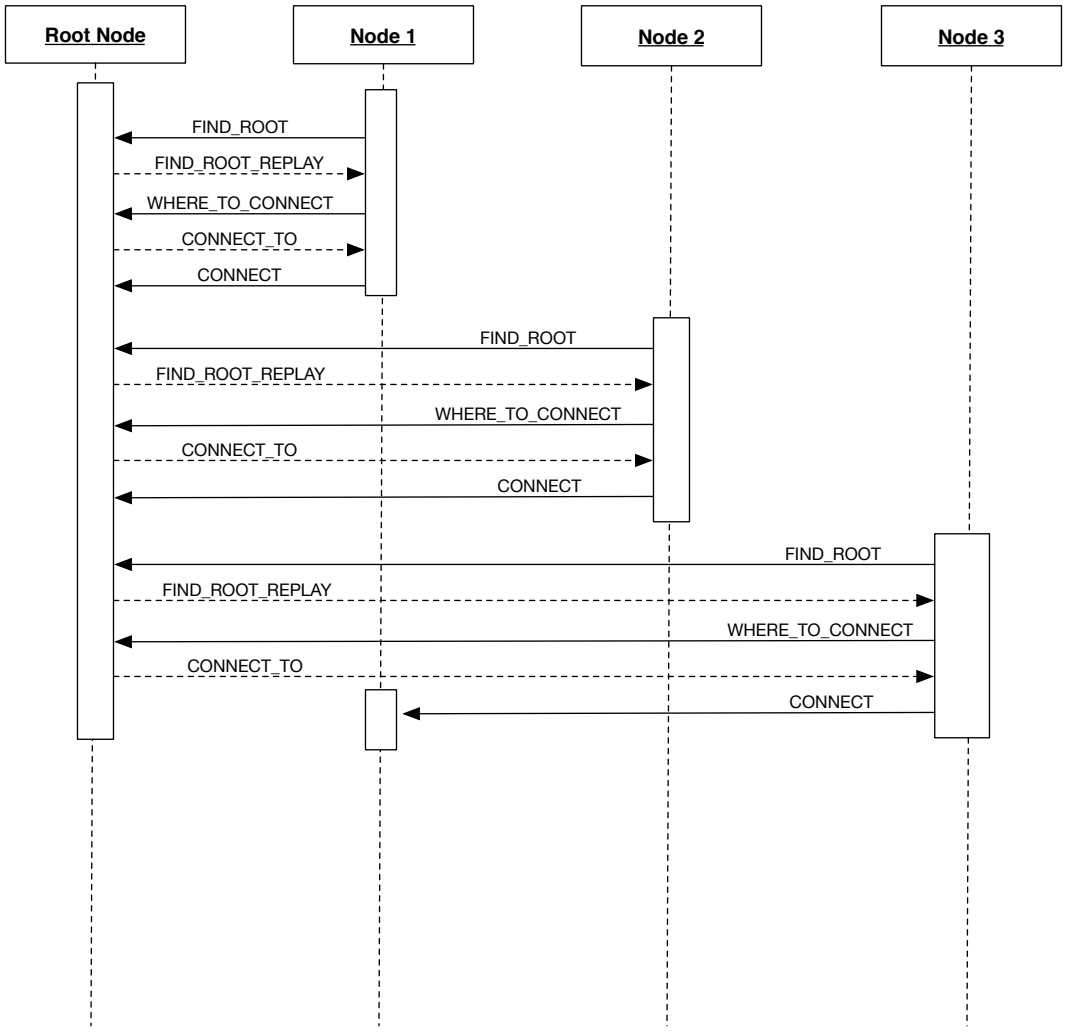

図 2.6: node 間で行われるメッセージ通信

2.10 木の再構成

TreeVNC はバイナリツリーでの接続という特性上 Node が切断されたことを検知できずにいると、Node 同士で構成された木構造が崩れてしまい、新しい Node が接続に来た場合に適切な場所に Node を接続することができなくなってしまう。木構造を崩さないよう、Node 同士の接続を再構成を行う必要がある。

TreeVNC の木構造のネットワークトポロジーは Root Node が持っている nodeList というリストで管理している。つまり、Node の接続が切れた場合、木の再構成を行うため Root Node に知らせなければならない。

TreeVNC は LOST_CHILD というメッセージ通信で Node の切断を検知・木の再構成を行っている。

TreeVNC は VNC サーバーから送られる画像データ (FRAME_BUFFER_Update) を MulticastQueue に蓄積しており、Node はこのキューから画像データを取得し、画面を描画している。LOST_CHILD 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 Memory Over Flow を回避するために Timeout スレッドが用意されている。Timeout を検知した際、Node との接続が切れたと判断する。

図 2.7 は 6 台の Node が接続してる状態で Node3 が切断した場合の木の再構成を示している。

- Node3 の切断を検知した Node1 が Root Node へ LOST_CHILD メッセージを送信する (1: sendLostChild)。
- LOST_CHILD メッセージを受け取った Root Node は nodeList から切断した Node を消し、最後尾の Node6 に切断した Node Number を割り当てる (2: updateNodeList)。
- Root Node は最後尾の Node6 に、切断した子 Node3 が接続していた Node1 に接続する様に CONNECT_TO メッセージを送信する (3: send ConnectTo (Node1))。
- 最後尾の Node が子 Node を失った親 Node へ接続しに行く (4: connect (Node1))。

LOST_CHILD によって、切断された全ての Node を検知することができるため、nodeList の更新が正しく行われる。よって、新しく接続に来た Node を適切な場所に接続することが可能となる。

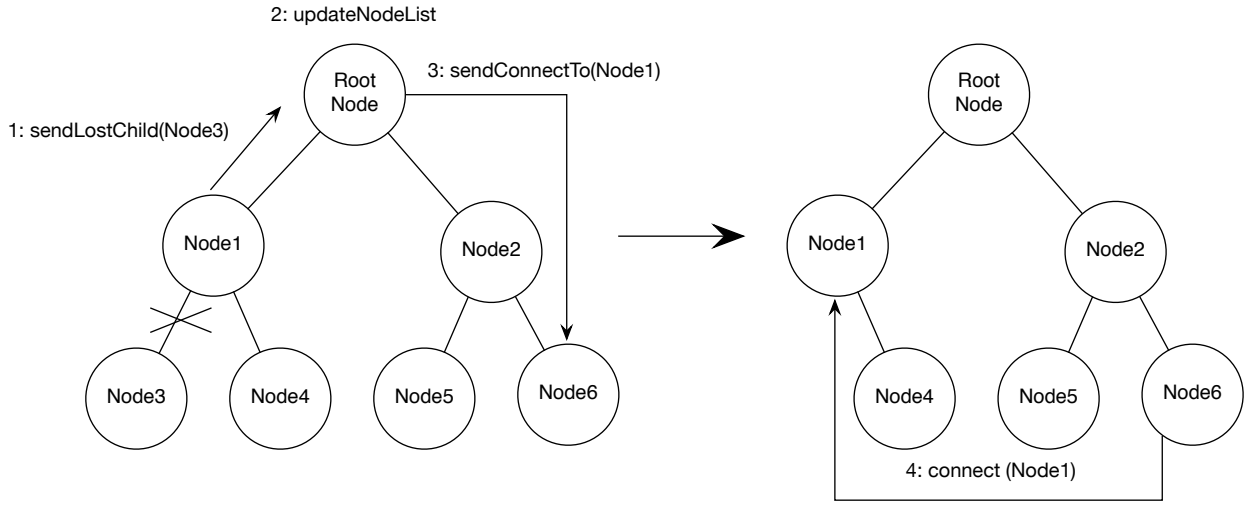

図 2.7: LOST_CHILD を検知・再接続

2.11 共有画面切り替え

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。

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う必要がある。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 VNC サーバーへと再接続を行う必要がある。

TreeVNC はユーザが VNC サーバーへの再接続を行うことなく、ビューアの Share Screen ボタン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 Node は配信者の VNC サーバーと通信を行っている。VNC サーバーから画面データを受信し、そのデータを 子 Node へと送信している。配信者切り替え時に Share Screen ボタンが押されると、SERVER_CHANGE_REQUEST メッセージに Node 番号やディスプレイ情報 を付加し、Root Node に送信する。SERVER_CHANGE_REQUEST メッセージを受け取った Root Node は配信を行う Node の VNC サーバーと通信を始める。この切り替え動作は別のスレッドを生成して行うため、 切り替え作業が終了したときに現在配信している画面が切り替わる。

そのため 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。更にプロジェクタ等も使用しないため、ケーブルの差し替えも行わずとも画面配信を行うことが出来

2.12　表示画面サイズの調整

配信側 PC によって配信される画面サイズが変化する。配信側と Node で画面サイズに差がある場合、画面に入りきらない、表示画面が小さすぎる等の問題が生じる。

TreeVNC では ビューワに fit screen ボタンを設置することでこの問題を解決している。fit screen ボタンが押されると PC の画面サイズに合わせてフルサイズで拡大・縮小する。

2.13　複数のネットワークの対応

TreeVNC は Root Node が複数のネットワークに接続している場合、 図 2.8 の様に、ネットワーク別に木構造を形成する。

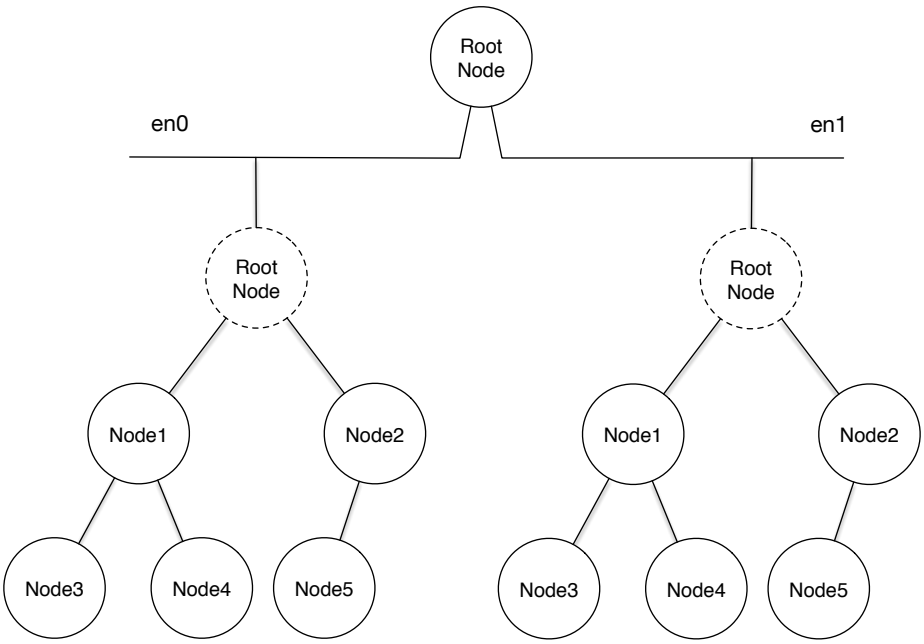

図 2.8: Multi Network Tree

TreeVNC は Root Node 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、切断検出時の接続の切り替え等を行う。

Root Node の保持しているネットワーク毎に TreeManager を生成する。新しい Node が接続してきた際、 interfaces から Node のネットワークと一致する TreeManager を取得し、 Node 接続の処理を任せる。

第3章　NAT 対応

3.1　NAT

NAT(Network Address Translation) は ローカルなネットワークではプライベート IP アドレスを設定し、外のネットワークへ接続するときにグローバル IP アドレスに変換する技術のことである。また、NAT に変換にポート番号を付け加えることで 1 つのグローバル IP アドレスで複数のホスト間の通信も可能となる。この変換は NAPT(Network Address Ports Translator) と呼ばれる

3.2　TreeVNC での NAT の問題点

TreeVNC は NAT の変換を経由した通信を行う場合様々な問題が発生する。

まず、Multicast 通信を行うことが出来ないという点である。 Multicast は同一ネットワーク内の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。TreeVNC は Root Node を探す際に FIND_ROOT メッセージを Multicast で送信する。しかし、NAT を越えたネットワークは同一ネットワークではないので、Root Node を探すことができない。

また、Root Node の木構造を管理している nodeList に NAT を越えたネットワークのアドレスを追加していくと IP アドレスが重複する可能性があるため、通常の TreeVNC の構成では NAT を越えたネットワークからの接続を行うことが出来ない。

3.3　Direct Connection

遠隔地からでもゼミや授業に参加できるよう、NAT を越えたネットワークから TreeVNC への接続を可能 [7] にした。

図 3.1 に NAT を越えた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 Node に接続を行う。この接続を Direct Connection と呼ぶ。

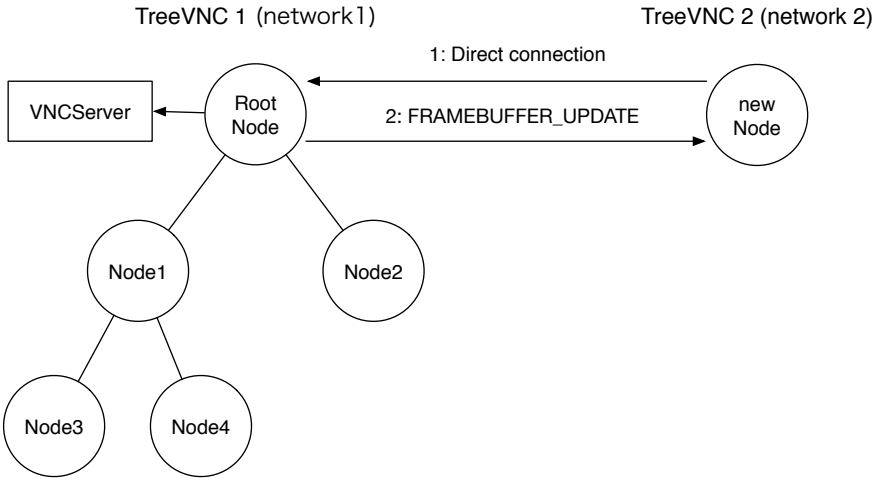

図 3.1: Direct Connection

3.4 Direct Connectionでの木構造

Direct Connection した Node はそのネットワークの Root Node になり、接続先である TreeVNC の nodeList に追加されない。つまり、ネットワーク毎に木構造をもつことになる。

新しく接続する Node は図3.2のようにそのネットワークの Root Node に FIND_ROOT メッセージを送信し、接続処理を行う。

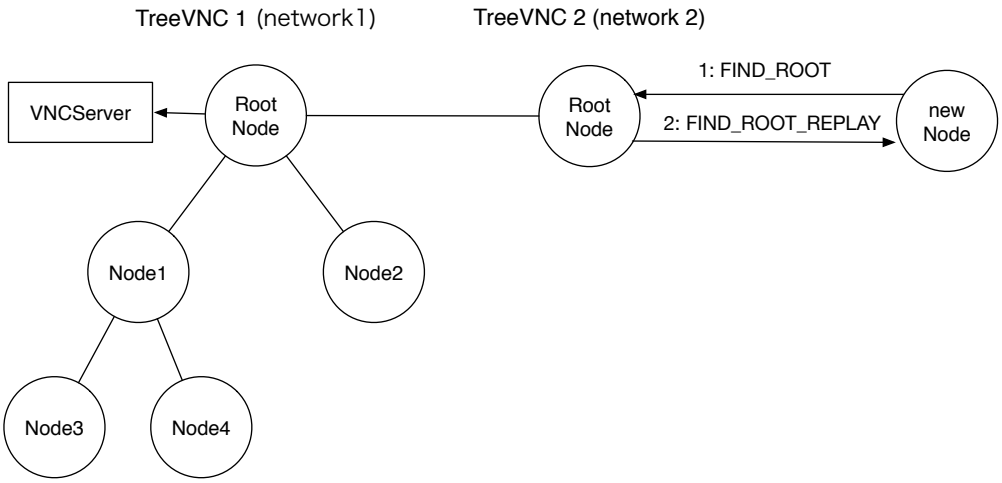

図 3.2: 新しい Node の接続

配信側の Root Node はDirect Connection で接続された Node に対して通常の子 Node と同じように FRAMEBUFFER_UPDATE メッセージで画像データを送信する。FRAMEBUFFER_UPDATE メッセージを受けった Root Node はそのネットワークの Node に対して FRAMEBUFFER_UPDATE メッセージを送信する。

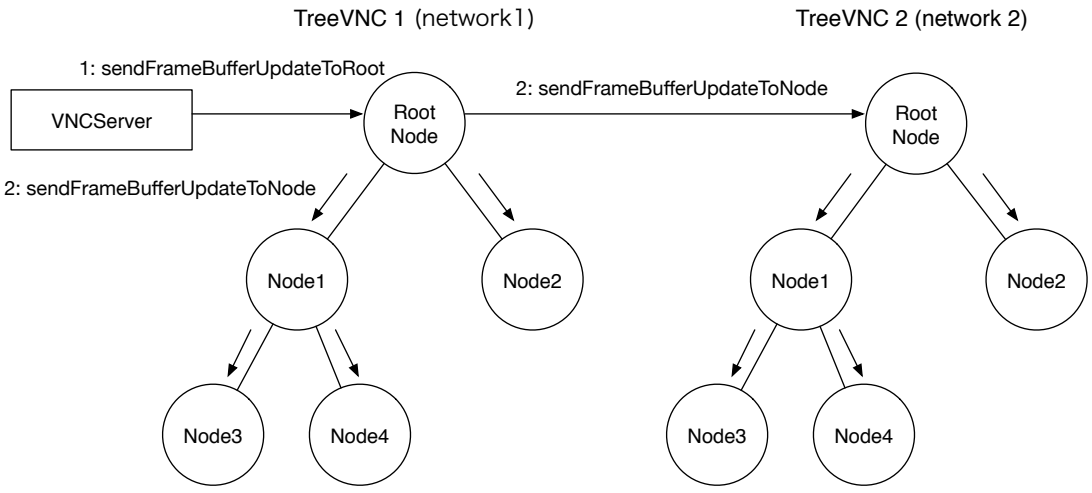

図 3.3: NAT を越えたネットワークへの画面配信

3.5 NAT を越えた共有画面切り替え

図 3.4 は Direct Connection での画面切り替えの実装案である。まず Direct Connection した ネットワークのクライアントから SERVER_CHANGE_REQUEST メッセージが送信される。SERVER_CHAGEN_REQUEST メッセージを受け取った そのネットワークの Root Node は Direct Connection した先の Root Node へ SERVER_CHANGE_REQUEST を -1 を付加して送信する。-1 を送信することで、 受け取った Root Node は別のネットワークからの画面切り替え要求ということを判断することができる。別ネットワークからの切り替え要求を受け取った Root Node は読み込み用と書き込み用のソケットを入れ替える。ソケットを入れ替えることで、 切り替え要求した Root Node のデータを受け取ることが可能となる。

図 3.4 の実装案ではなく、画面切り替えが起こったネットワークに対して繋がっているネットワークが DirectConnection をすることで 容易に実装することができる。だが、NAT を越えた接続のため、再接続する際に接続先が見つからないという問題が発生する。そのため接続状況を維持したまま送受信状態を入れ替える必要がある。

しかし、接続状態を維持したままの入れ替えは実装が複雑になっており、今回追加した Direct Connection では NAT を越えたネットワークの画面の配信を行うのみであり、 画面切り替えを行うことが出来ない。

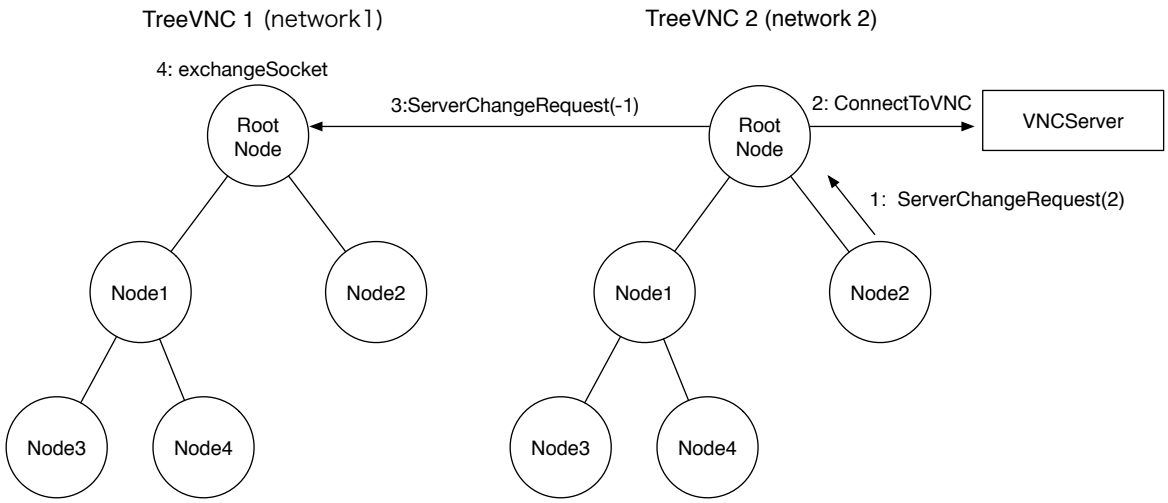

図 3.4: NAT を越えたネットワークでの画面切り替え

第4章　マルチディスプレイ対応

4.1　マルチディスプレイ時の問題点

画面配信側の PC がマルチディスプレイの場合、 図 4.1 のように VNC サーバー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。

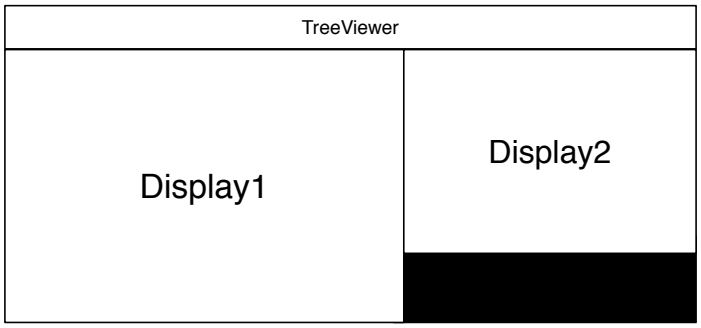

図 4.1: マルチディスプレイでの送信

4.2　ディスプレイの選択

マルチディスプレイでの問題を解決するため、画面を共有する際に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 [7]。

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。そのため、配信側は画面の切り替えを行う際に、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 Node への画面切り替えを要求する SERVER_CHANGE_REQUEST メッセージに付加させる。Root Node は 配信側の VNC サーバー に画像データを要求する FRAMEBUFFER_UPDATE_REPLY メッセージに送信された座標を付加する。VNC サーバーは要求された座標内の画像データを FRAMEBUFFER_UPDATE メッセージで Root Node に送信する。これにより、一画面のみの表示が可能になる。

図 4.2 は Display1 のみを画面共有する例を示している。

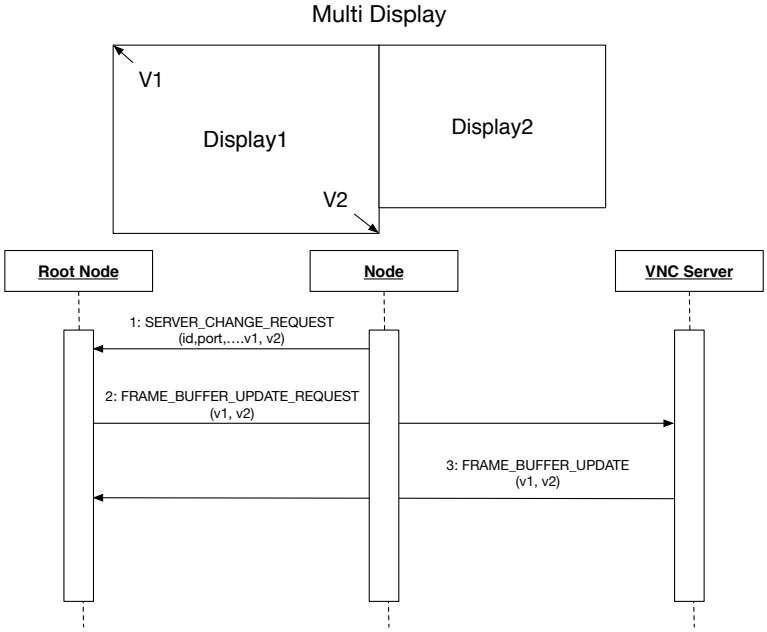

図 4.2: マルチディスプレイへの対応

第5章　共有画面切り替えの安定化

5.1　画面切り替えの問題点

TightVNC では VNC サーバーへの接続を 1 つのスレッドで行っていた。そのため、Tree VNC でも画面切り替えを行う際 配信中の画面を停止した後に VNC サーバーへの接続を行っていた。サーバーしかし、 切り替え先の VNC サーバーへの接続に時間がかかったり、 切り替え先のクライアントに共有確認した際にキャンセルすると TreeVNC が停止するという問題があった。

5.2　切り替え用スレッドの生成

そこで、 画面切り替えを行う際に新しく切り替え用のスレッドを生成し処理する変更を行った。実際に切り替えのスレッドを生成している inhelitClients メソッドを Code 5.1 に示す。

inhelitClients(Code 5.1) は 3 行目で ConnectionPresenter クラスを新しく作っている。ConnectionPresenter クラスは切り替えに必要な情報や処理が入っているクラスである。

8〜13 行目は新しくスレッドを生成して実行している。このスレッドでは Connection-Presenter クラスの startVNCConnection メソッドを読んでいる。

```
1  @Override
2  public void inhelitClients(String hostName, short newVNCServerId, int x, int y, int
       width, int height, int scale) {
3      final ConnectionPresenter connectionPresenter = createNewConnectionPresenter(
           hostName, newVNCServerId, x, y, width, height, scale);
4      isApplet = true;
5      this.setNoConnection(false);
6      final Viewer v = this;
7
8      new Thread(new Runnable() {
9          @Override
10         public void run() {
11             connectionPresenter.startVNCConnection(v, false, null, null);
12         }
13     }, "ServerChangeThread").start();
14 }
```

Code 5.1: 切り替え用スレッドの生成

5.3 切り替え用スレッドの動作

図 5.1 は画面切り替えの際に切り替え用スレッドを用いたときの通信の流れである。まず共有先の Node から切り替え用の SERVER_CHANGE_REQUEST メッセージが送信される。メッセージを受け取った Root Node は 切り替え用のスレッドを生成し、切り替えの処理を任せて、現在の配信の続行する。切り替え用のスレッドではまず切り替えに必要なデータの設定、 VNC サーバー への接続など通常の切り替え処理を実行する。切り替えが完了した後に、 現在配信中の画面を停止し、画面の切替を行う。

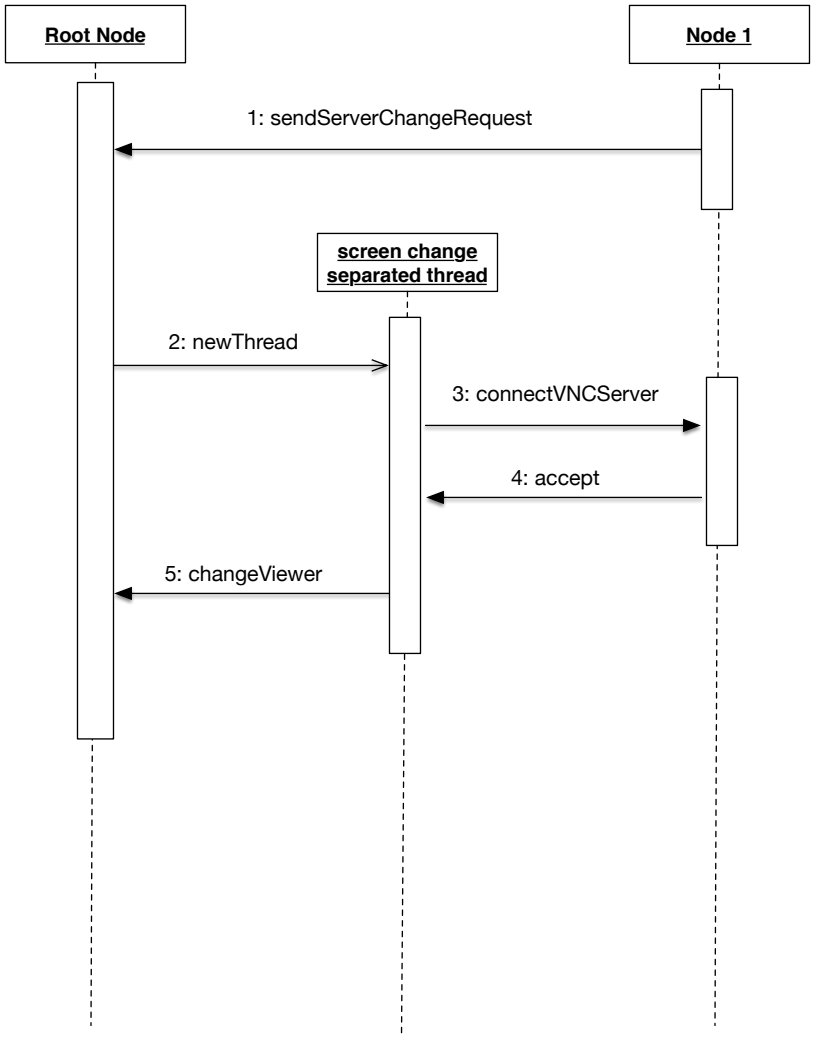

図 5.1: 切り替え用スレッドの動作

第6章　各 Node へのエラー通知

6.1　Root Node へのエラーメッセージの表示

TreeVNC には接続している各 Node へのエラーの通知を行うことが出来なかった。そのため、 画面切り替えを行う際に切り替え先が VNC サーバーの共有設定をしていない場合 Root Node に接続エラーのダイアログが表示されるという実装になっており、切り替えを行った Node には一切通知されなかった。

6.2　ERROR_ANNOUNCE メッセージ

この問題を解決するために新しく ERROR_ANNOUNCE というメッセージを追加した。図 6.1 は Node3 に対してエラー通知を行っている例を示している。ERROR_ANNOUNCE メッセージは Root Node から木構造を辿りながら末端の Node に通信を行うメッセージで、エラー通知したい Node の Node Nuber と エラー内容の文字列を付加して送信する。付加した Node Number に一致する Node がメッセージを受け取ると、 付加されたエラーの文字列をダイアログで表示する。エラー通知用のメッセージを追加することで、 各々のクライアントに対して通知を行うことが可能になった。

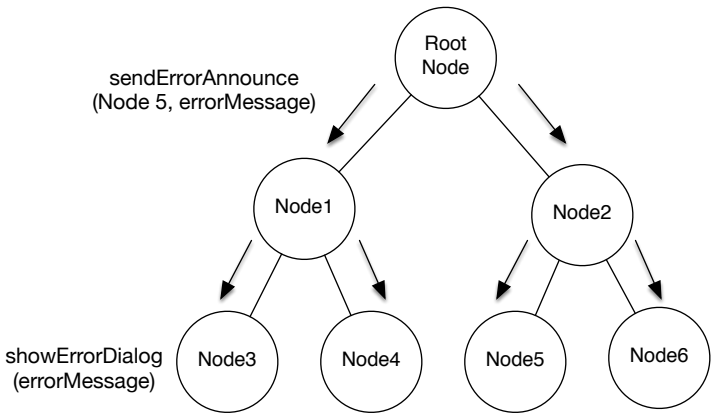

図 6.1: ERROR_ANNOUNCE メッセージの挙動

通知を行えるようになると、ユーザーに対して共有設定の方法などが簡単に通知することができる。

第7章　TreeVNC の評価

7.1　画像データ伝達の遅延

VNC サーバー 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 Node から木の末端の Node までの画像データ伝達の遅延を検証する実験を行った。

7.2　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 17 名が接続していた。

7.3　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 Node から 末端の Node まで伝達するメッセージと画像データ、 CHECK_DELAY_REPLY は各 Node から Root Node まで伝達するメッセージである。

CHECK_DELAY メッセージは送信時刻を付けて送信する。Root Nodeから CHECK_DELAY 送信し、末端の Node まで各 Node を伝いながら伝達して行く。

CHECK_DELAY_REPLY は CHECK_DELAY から受け取った送信時刻をそのままに、画像データのサイズを付けて送信する。CHECK_DELAY を受け取った各 Node は CHECK_DELAY_REPL を接続している親 Node に送信する。

CHECK_DELAY_REPLY を受け取った Root Node はメッセージと画像データの伝達にどれだけの時間がかかったかを計算する。データ計算方法を以下の Code 7.1 に記述する。 この変数 time は CHECK_DELAY_REPLY に付いている CHEKC_DELAY の送信時刻である。

```
1 Long delay = System.currentTimeMillis() - time;
```

Code 7.1: 遅延時間の計算方法

7.4 結果

バイナリツリーで木を構成した場合、 Node 数が 17 台だと深さが 4 となる。各木構造の階層毎に、画像データの伝搬にかかった時間を測定した。

図 7.1, 7.2, 7.3, 7.4 は遅延の分布を示した散布図である。X 軸はメッセージ伝達にかかった秒数 (ms)、 Y 軸は画像データのサイズ (Byte) である。

画像データの伝達はほぼ 1 秒以内に収まっているが、容量が小さい場合でも時間がかかる場合がある。それはその送信の前に大容量の画像を送信した後の回線の Delay が残っているためだと考えられる。

また、深さ 3 で極端に遅い場合がある。遅い原因として、1 つの Node がボトルネックになっていることが判明している。このような極端に遅い Node をそのまま木に配置した場合、その Node の子 Node 以下に影響を及ぼすおそれがある。

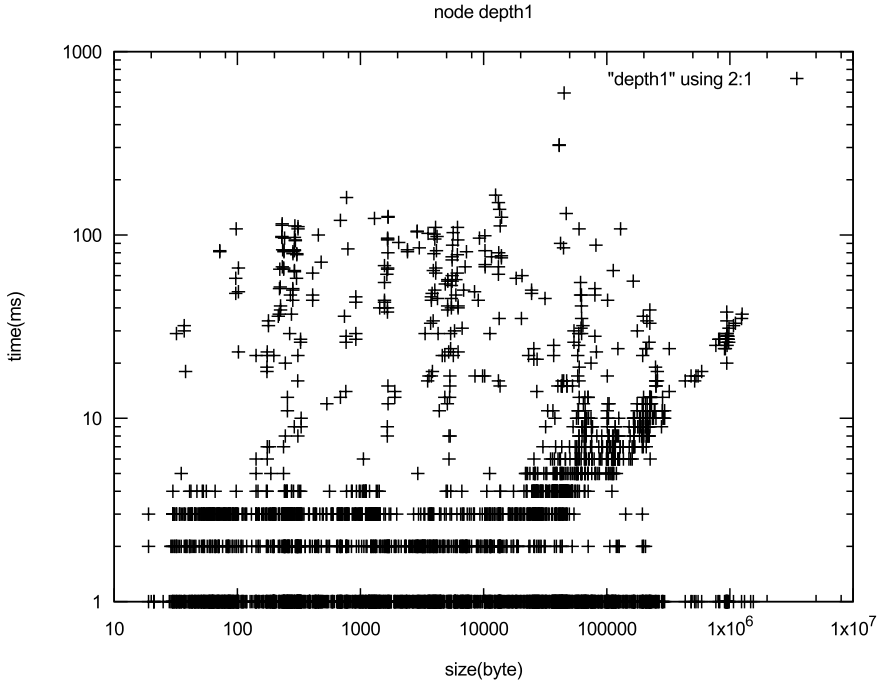

図 7.1: データサイズと遅延の関係：深さ 1

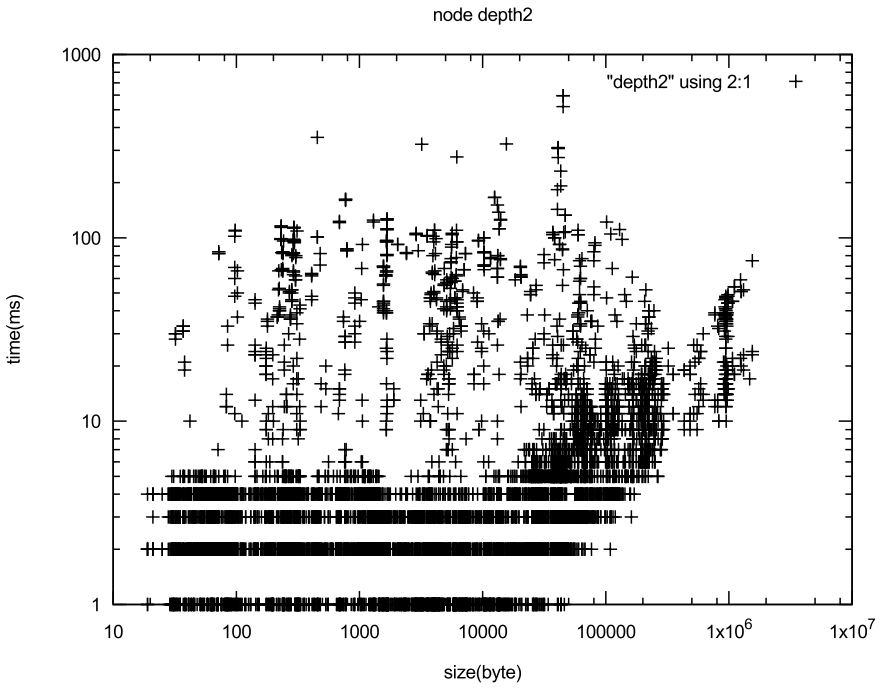

図 7.2: データサイズと遅延の関係：深さ 2

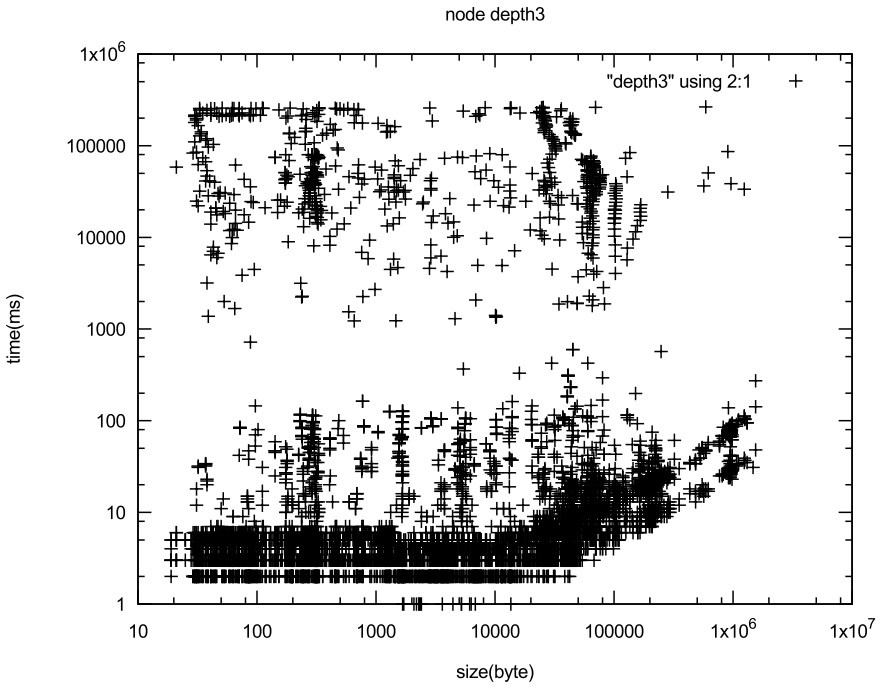

図 7.3: データサイズと遅延の関係：深さ 3

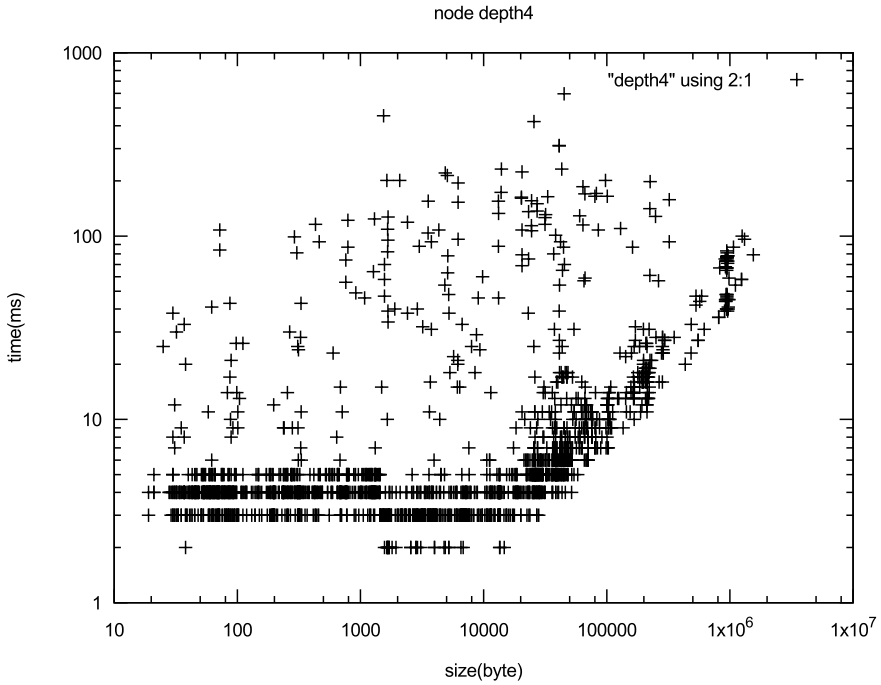

図 7.4: データサイズと遅延の関係：深さ 4

7.5 ボトルネックになっている Node への対処

画像データを受け取る時間が遅い Node をそのまま木構造に配置しているとその子 Node 以下に影響を及ぼす。そのためネックになっている Node への対処が必要である。

ボトルネックになっている Node への対処として CHECK_DELAY_REPLAY メッセージを使用している。図 7.5 は 6 台の Node が接続している状態で Node1 がネックになっている場合の木の再構成を示している。CHECK_DELAY_REPLAY メッセージを受け取った Root Node はその Node がボトルネックなっているかの判断を行う。ボトルネックになっているなら Root Node の nodeList からその Node を削除する。削除した Node の場所には末端の Node を配置するように CONNECT_TO メッセージを送信する。nodeList から削除された Node は接続されたままなので、データの受信は行うが、 木構造には入らないため、ネックになっている Node の下に新しい Node が繋がることはない。そのためネックになっている Node 以下に影響を及ばすことがない。

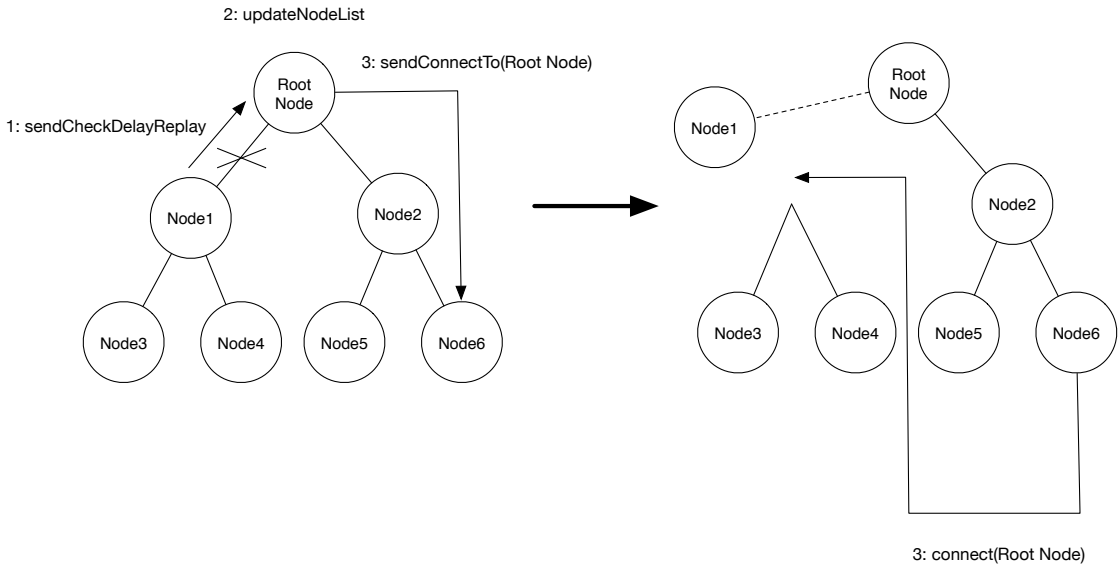

図 7.5: ボトルネックになっている Node への対処

第8章　結論

8.1　まとめ

本研究では画面配信システム TreeVNC での NAT 対応、 リファクタリングによる機能改良、 TreeVNC の評価を行った。

NAT を越えに対応した Direct Connection という接続方法を確立し実装した。これにより、NAT を越えた別ネットワークのユーザーが TreeVNC に参加することが可能となった。

マルチディスプレイの場合に画面を選択して配信することが可能になった。

画面切り替えの際に切り替え用のスレッドを用意することでスムーズな画面切り替えが可能。

新しくメッセージを追加することでクライアントにエラー通知を行える様になった。

今回の画像データの遅延実験を行い、 ボトルネックになっているノードがあることがわかった。また、ボトルネックになっているノードへの対処を行った。

8.2　今後の課題

Direct Conenction により、NAT を越えたの画面配信が可能となった。しかし、 遠隔地からゼミや授業等に参加する場合は画面の表示のみではなく、 音声の配信も行いたい。そのため音声配信機能の実装を行う必要がある。

今回実装したマルチディスプレイ選択の画面配信は PC の画面サイズに合わせてフルサイズで拡大・縮小する fit screen ボタンに対応しておらず、一画面のサイズの調節を行えない。そのためマルチディスプレイの fit screen 機能を実装する必要がある。

現在の TreeVNC は TightVNC を直接編集しているため、コードが複雑になっている。そのため新しい機能を入れるにしてもリファクタリングから始める必要があった。当研究室で開発している分散フレームワーク Alice[8] の例題として記述された AliceVNC は TightVNC のコードの変更点を最小限に抑え、 複雑度も低いことがわかっている。そのため TreeVNC で今回追加した機能を AliceVNC で再設計することが望ましい。

参考文献

[1] 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実装と設計. 日本ソフトウェア科学会第 28 回大会論文集, sep 2011.

[2] RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K. R., AND HOPPER,. A. virtual network computing, jan 1998.

[3] RICHARDSON, T., AND LEVINE, J. The remote framebuffer protocol. rfc 6143, mar 2011.

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

[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. 情報処理学会システムソフトウェアとオペレーティング・システム研究会 (OS), may 2012.

[6] LOUP GAILLY, J., AND ADLER, M. zlib: A massively spiffy yet delicately unobtrusive compression library. `http://zlib.net`.

[7] Tatsuki IHA and Shinji KONO. 有線 lan 上の pc 画面配信システム treevnc の改良. 第 57 回 プログラミング・シンポジウム, jan 2016.

[8] Nozomi TERUYA and Shinji KONO. 分散フレームワーク alice の pc 画面配信システムへの応用. 第 57 回 プログラミング・シンポジウム, jan 2016.

[9] Surendar Chandra, Jacob T. Biehl, John Boreczky, Scott Carter, Lawrence A. Rowe. Understanding screen contents for building a high performance, real time screen sharing system. *ACM Multimedia*, Oct 2012.

謝辞

本研究の遂行，また本論文の作成にあたり、御多忙にも関わらず終始懇切なる御指導と御教授を賜わりました河野真治准教授に深く感謝致します。

数々の貴重な御助言と細かな御配慮を戴いた小久保翔平さん、 徳森海斗さん、 比嘉健太さん並びに並列信頼研究室の皆様に深く感謝致します。

最後に、有意義な時間を共に過ごした情報工学科の学友、並びに物心両面で支えてくれた両親に深く感謝致します。

2016 年 3 月
伊波立樹