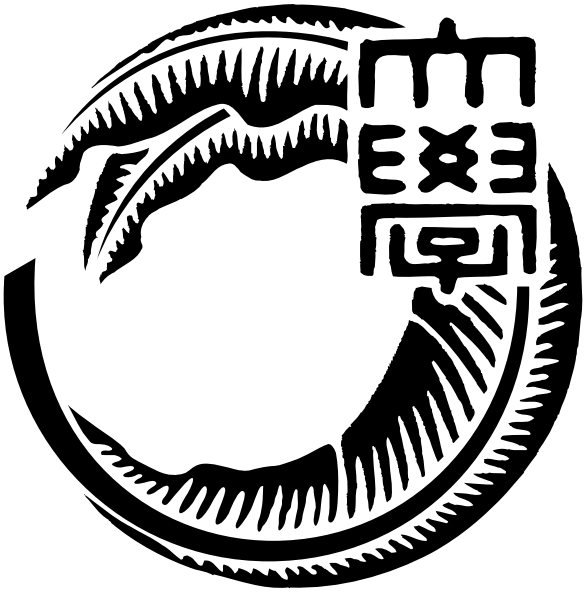
令和元年度 卒業論文

画面配信システム TreeVNC の Multicast 導入



琉球大学工学部情報工学科

165729B 安田亮

指導教員 河野 真治

目 次

第 1 章　画像配信ソフトウェア TreeVNC の活用　1

第 2 章　TreeVNC の基本概念　2

2.1 Virtual Network Computing 2
2.2 Remote Frame Buffer プロトコル 2
2.3 TreeVNC の構造 3
2.4 TreeVNC の通信プロトコル 4
2.5 MulticastQueue 5
2.6 木の再構成 6
2.7 データの圧縮形式 7
2.8 ShareScreen 9
2.9 複数のネットワーク接続時の木の構成 10

第 3 章　Multicast に向けた Blocking の実装　11

3.1 有線接続と無線 LAN 接続との違い 11
3.2 Update Rectangle の構成 12
3.3 TileLoop 14
3.4 Packet Lost 17

第 4 章　TreeVNC 開発環境やソースコードの修正改善　18

4.1 Gradle 6.1 対応 18
4.2 java9 以降の RetinaAPI 対応 18
4.3 デバッグオプションの修正 20

第 5 章　まとめ　21

参考文献　22

図 目 次

2.1 従来の VNC での接続構造 3
2.2 TreeVNC での接続構造 3
2.3 LOST_CHILD の検知・再接続 7
2.4 ZRLE でデータを途中から受け取った場合 8
2.5 ZRLEE へ再圧縮されたデータを途中から受け取った場合 8
2.6 Multi Network Tree 10

3.1 接続方法の分割 11
3.2 Rectangle の分割 13
3.3 ZRLEE の Packet の構成と分割された Rectangle 14
3.4 TileLoop のフローチャート 15

表 目 次

2.1 通信経路とメッセージ一覧 5

3.1 UpdateRectangle による Packet の構成 12
3.2 Rectangle Header の構成 14

ソースコード目次

4.1 java8 以前の Retina ディスプレイ API 19
4.2 Retina ディスプレイの判別関数 19
4.3 Retina ディスプレイの表示倍率を取得する関数 19

第1章　画像配信ソフトウェアTreeVNCの活用

現代の講義や発表、LT などでは PC 画面で用意した資料をプロジェクタに映しながら進行することが多い。ゼミでは発表者が変わるたびに、PC を接続し直す必要がある。

この場合、発表者の PC を接続するたびにケーブルを差し替える必要がある。発表者の PC によっては接続するアダプターの種類や解像度の設定により、正常に PC 画面を表示できない場合がある。また、参加者もプロジェクタに集中を割く必要があり、同時に手元の PC で作業を行う場合、集中の妨げとなってしまう。

当研究室で開発している画面配信システム TreeVNC [1] は、発表者の PC 画面を参加者の PC 画面に表示するソフトウェアである。そのため、参加者は不自由なく手元の PC を操作しながら講義を受けることが可能になる。また java で書かれているため、物理的な制限に左右されることが少ない。さらに、発表者の交代の際もケーブルを差し替えずに、全体に共有する画面の切り替えが可能になっている。

TreeVNC は VNC を使用した画面配信を行なっている。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。TreeVNC ではネットワークに接続を行なった参加者をバイナリツリー状に接続することで、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は下がらない。また、Tree の Root が参照している VNC サーバを変更することで、共有する画面の切り替えが可能となる。

しかし、画面共有は送信するデータ量が多いため、無線 LAN で接続を行なった際に有線接続よりも遅延が大きくなってしまう。そこで本研究では、Multicast でのデータ通信の考察やデータの分割・圧縮方法の実装、評価を行うことにより、無線 LAN での配信環境の向上を目指し、TreeVNC の有用性を評価することで講義やゼミを円滑に行えることを目標とする。

第2章　TreeVNCの基本概念

2.1　Virtual Network Computing

Virtual Network Computing [2](以下 VNC) は、サーバ側とクライアント (ビューワー) 側からなるリモートデスクトップソフトウェアである。遠隔操作にはサーバを起動し、クライアント側がサーバに接続することで可能としている。

2.2　Remote Frame Bufferプロトコル

Remote Frame Buffer(以下 RFB) プロトコル [3] とは VNC 上で使用される、自身の PC 画面をネットワーク上に送信し他人の PC 画面に表示を行うプロトコルである。画面が表示されるユーザ側を RFB クライアントと呼び、画面送信を行うために FrameBuffer の更新が行われる側を 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、最初にプロトコルのバージョンの確認や認証が行われる。その後、RFB クライアントへ向けて 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を送信する。

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 の変更部分を送信する。さらに、RFB クライアントから Framebuffer - UpdateRequest が来るとそれに答え返信する。変更部分のみを送信する理由は、更新があるたびに全画面を送信すると、送信するデータ面と更新にかかる時間面において効率が悪くなるからである。

2.3 TreeVNCの構造

TreeVNC は java を用いて作成された Tight VNC [4] を元に作成されている。 TreeVNC は VNC を利用して画面配信を行っているが、従来の VNC では配信 (サーバ) 側の PC に全ての参加者 (クライアント) が接続するため負荷が大きくなってしまう (図 2.1)。

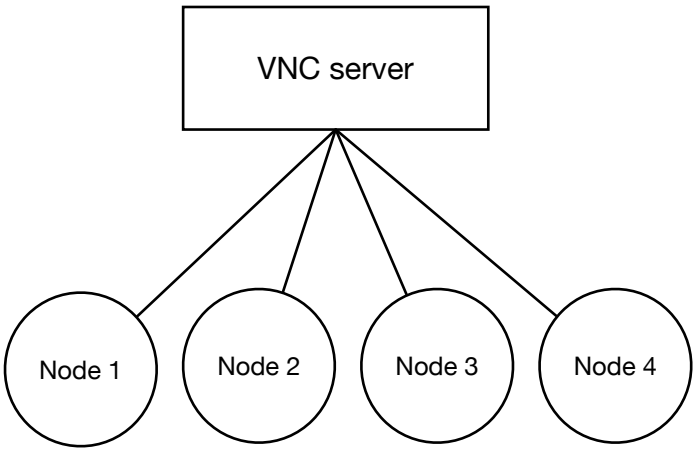


図 2.1: 従来の VNC での接続構造

そこで TreeVNC ではサーバに接続を行ってきたクライアントをバイナリツリー状 (木構造) に接続する。接続してきたクライアントをノードとし、その下に新たなノードを最大 2 つ接続していく。これにより人数分のデータのコピーと送信の手間を分散することができる (図 2.2)。

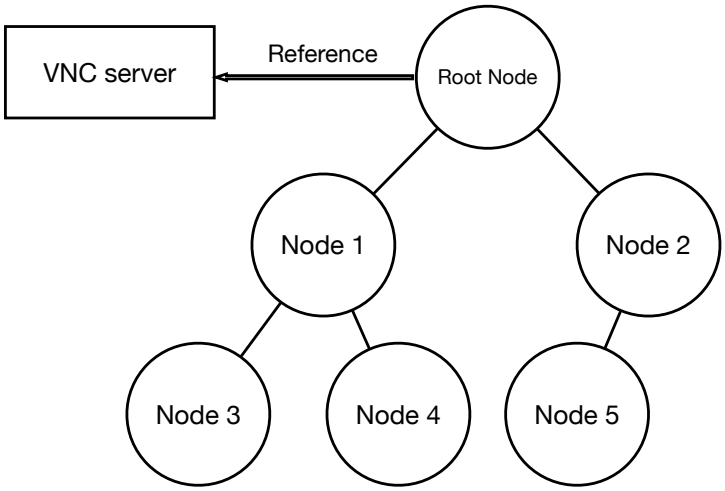


図 2.2: TreeVNC での接続構造

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を行ってきた場合、従来の VNC ではサーバ側が N 回のコピーを行って画面配信する必要があるが、TreeVNC では各ノードが最大 2 回ずつコピーするだけで画面配信が可能となる。

送信されるデータは従来の方法だと N 個のノードに対して N-1 回の通信が必要である。これはバイナリツリー状の構造を持っている TreeVNC でも通信の数は変わらない。

木構造のルートのノードを Root Node と呼び、そこに接続されるノードを Node と呼ぶ。Root Node は子 Node にデータを渡す機能、各 Node の管理、VNC サーバから送られてきたデータの管理を行っている。各 Node は、親 Node から送られてきたデータを自身の子 Node に渡す機能、子 Node から送られてきたデータを親 Node に渡す機能がある。

2.4 TreeVNC の通信プロトコル

TreeVNC の通信経路としては以下が挙げられる。

- 任意の Node から Root Node に直接通信を行う send direct message (Node to Root)
- Root Node から任意の Node に直接通信を行う send direct message (Root to Node)
- 任意の 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 から木の末端の Node までの全ての Node に通信を行う message down tree (Root to Node)
- Root Node から配信者の VNC サーバへの通信を行う send message (Root to VNC Server)
- 配信者の VNC サーバから Root Node への通信を行う send message (VNC Server to Root)

Node 間で行われるメッセージ通信には 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。

表 2.1: 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.

2.5 MulticastQueue

配信側の画面が更新されると、VNC サーバから画面データが FRAME_BUFFER_UPDATE メッセージとして送られる。その際親 Node が受け取った画像データを、同時に複数の子 Node に伝えるために MulticastQueue というキューにデータを蓄積を行う。

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 の並列実行用に用意された 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が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、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 から画像データが削除される。

2.6 木の再構成

TreeVNC はバイナリツリー状での接続のため、Node が切断されたことを検知できずにいると構成した木構造が崩れてしまい、新しい Node を適切な場所に接続できなくなってしまう。そこで木構造を崩さないよう、Node 同士の接続の再構成を行う必要がある。

TreeVNC の木構造の接続形態は Root Node が持っている nodeList というリストで管理している。nodeList 内の treeNum という値が各 Node に割り当てられており、これによって木構造のどの位置に Node が接続されているかの判別が可能である。よって、Node の接続が切れた場合 Root Node に切断を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 切断の検知および木構造の再構成を行っている。

LOST_CHILD の検出方法には、MulticastQueue を用いている。親 Node が MulticastQueue を持っているが、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 の中に入っている画像データを削除することができない。子 Node が MulticastQueue から画像データを取得せずに、画像データが溜まり続けると親 Node が MemoryOverFlow を起こしてしまう。この問題を回避するために Timeout スレッドが用意されている。Timeout を検知した際に Node との接続が切れたと判断する。LOST_CHILD の検知と木構造の再構成の手順を以下に示す。

- 子 Node の切断を検知した Node が Root Node へ LOST_CHILD メッセージを送信する (図 2.3 中、1: lostChild())
- LOST_CHILD メッセージを受け取った Root Node は nodeList の更新を行う (図 2.3 中、2: updateNodeList())
- 切断した Node を nodeList から削除し、nodeList の最後尾の Node に切断した Node と treeNum を割り当てる
- Root Node は最後尾の Node に、切断した子 Node が接続していた親 Node に接続するよう、COMMECT_TO メッセージを送信する (図 2.3 中、3: connectTo(1))
- 最後尾の Node が子 Node を失った親 Node へ接続を行う (図 2.3 中、4: connectToParent(1))

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、新しく接続を行ってきた Node を適切な場所に接続することが可能となる。

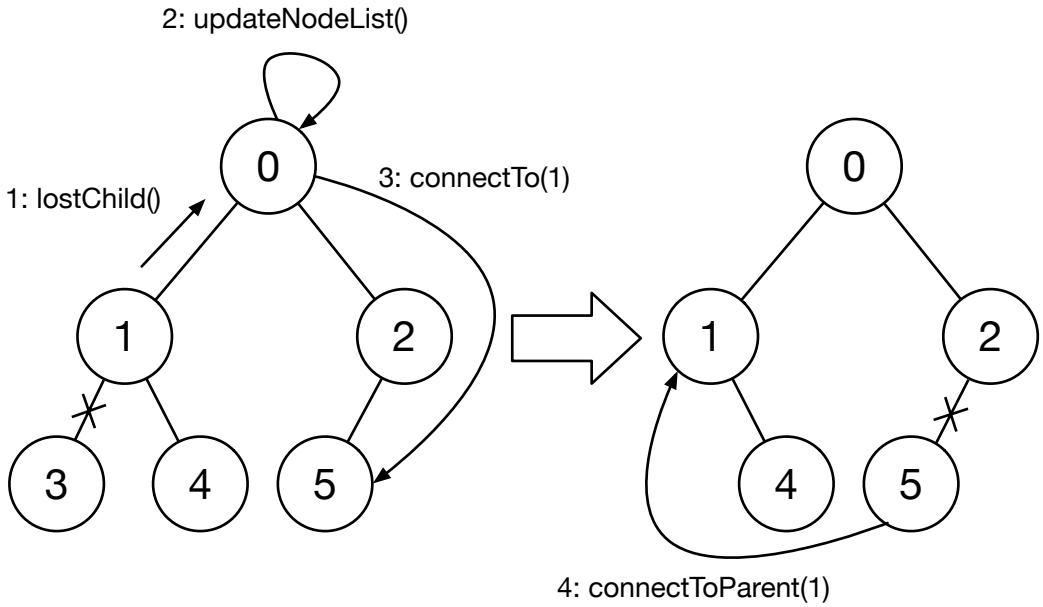


図 2.3: LOST_CHILD の検知・再接続

2.7 データの圧縮形式

TreeVNC は ZRLEE [5] というエンコードタイプでデータのやり取りを行う。ZRLEE は RFB プロトコルで使用できる ZLRE というエンコードタイプを元に生成される。

ZLRE(Zlib Run-Length Encoding) とは可逆圧縮可能な Zlib 形式 [6] と Run-Length Encoding 方式を組み合わせたエンコードタイプである。

ZLRE は Zlib 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与され送信される。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だす (flush) ことが出来ない。従って、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 (図 2.4)。

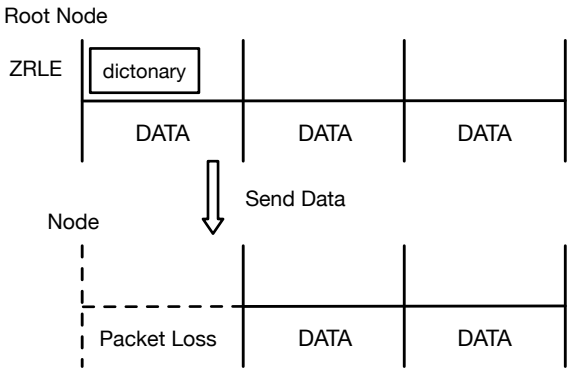


図 2.4: ZRLE でデータを途中から受け取った場合

そこで ZRLEE は一度 Root Node で受け取った ZRLE のデータを unzip し、後述する Update Rectangle と呼ばれる画面ごとのデータに辞書を付与して zip し直すことで、初めからデータを読み込んでいなくても解凍を出来るようになっている (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よい。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けないため、Root Node 側と Node 側では毎回新しく作る必要がある。辞書をクリアすることで短時間で解凍され画面描画されるという、適応圧縮を実現していることになり圧縮率が向上する。

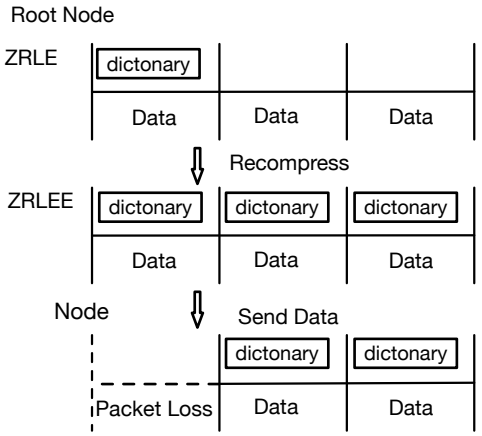


図 2.5: ZRLEE へ再圧縮されたデータを途中から受け取った場合

TreeVNCではRFBプロトコルによって配信側の画面の変更部分はFRAME_BUFFER_UPDATEメッセージとして送られてくる。メッセージの中には変更部分の原点のx,y座標と縦横の幅等が含まれており、長方形として展開される。この長方形をUpdate Rectangleと呼ぶ。

2.8 ShareScreen

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たび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、発表者が変わるたびにケーブルの抜き差しを行う必要がある。その際に、PCとプロジェクタを接続するための変換アダプタが必要になる場合や、接触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では、配信者が切り替わるたびにVNCの再起動、サーバ・クライアント間の再接続を行う必要がある。TreeVNCは配信者の切り替えのたび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。画面の切り替えについてはユーザがVNCサーバへの際接続を行うことなく、ビューワー側のShare Screenボタンを押すことで配信者の切り替えが可能となっている。

TreeVNCのRoot Nodeは配信者のVNCサーバと通信を行なっている。VNCサーバ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を実行すると、Root Nodeに対しSERVER_CHANGE_REQUESTというメッセージが送信される。このメッセージにはShare Screenボタンを押したNodeの番号やディスプレイ情報が付与されている。メッセージを受け取ったRoot Nodeは配信を希望しているNodeのVNCサーバと通信を始める。そのためTreeVNCは配信者切り替えのたびにVNCを終了し再接続する必要がない。

2.9 複数のネットワーク接続時の木の構成

TreeVNC は Root Node が複数のネットワークに接続している場合、図 2.6 のようにネットワーク別にを形成する。TreeVNC は Root Node が TreeManager を持っている。TreeManager は TreeVNC の接続部分を管理しており、木構造を管理する nodeList の生成を行う。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なっている。

TreeManager は Root Node の保持しているネットワーク毎に生成される。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 Tree Manager を取得し、Node 接続の処理を任せる。

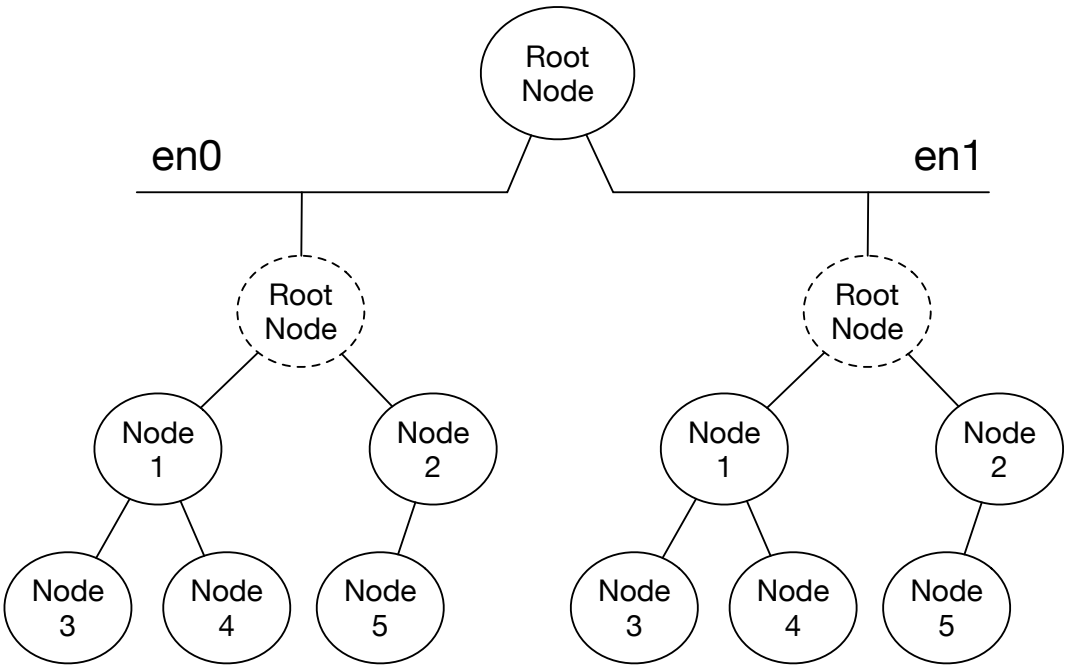


図 2.6: Multi Network Tree

第3章　Multicastに向けたBlockingの実装

3.1　有線接続と無線LAN接続との違い

現在のTreeVNCでは有線接続と無線LAN接続のどちらでも、VNCサーバから画面配信の提供を受けることが可能である。しかし画像配信のデータ量は膨大なため、現在のTreeVNCでVNCサーバに無線LAN接続を行なった場合、画面配信の遅延が大きくなってしまう。

無線LAN接続時の場合でも画面切り替えの機能は有効であるため、VNCサーバ側が無線LANで接続を行い、クライアント側は有線接続を行うことで画面配信が可能となる。ここで、Wifiの Multicastの機能を用いてクライアント側でもWifiを使用することが可能であると考えられる。Root Nodeは無線LANに対して、変更するUpdate Rectangleを Multicastで一度だけ送信する。

有線接続の場合は従来通り、VNCサーバ、Root Node、Nodeからなるバイナリツリー状に接続されるため、有線接続時と無線LAN接続時でのVNCサーバの接続方法を分割することが可能である (図3.1)。こうすることにより、新しいNodeが無線LAN接続であっても有線接続の木構造には影響を及ぼさない。

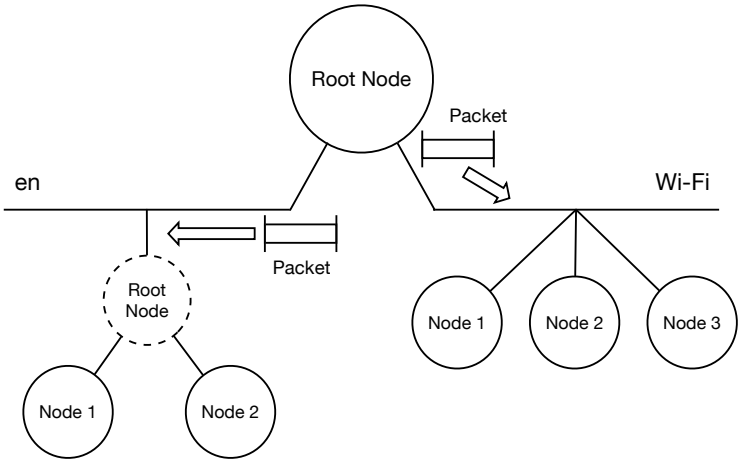


図 3.1: 接続方法の分割

Wifi の Multicast Packet のサイズは 64KByte が最大となっている。4K ディスプレイを例にとると、4K ディスプレイの大きさの画面更新には 8MByte(画素数) 8Byte(色情報)で圧縮前で、64MByte 程度となる。

3.2 Update Rectangle の構成

RFB の Update Rectangle によって送られてくる Packet は以下の表 3.1 の構成となっている。

表 3.1: UpdateRectangle による Packet の構成

1 byte	messageID
1 byte	padding
2 byte	n of rectangles
2 byte	U16 - x-position
2 byte	U16 - y-position
2 byte	U16 - width
2 byte	U16 - height
4 byte	S32 - encoding-type
4 byte	U32 datalengths
1 byte	subencoding of tile
n byte	Run Length Encoded Tile

1 つの Update Rectangle には複数の Rectangle が入っており、さらに 1 つ 1 つの Rectangle には x,y 座標や縦横幅、encoding type が含まれている Rectangle Header を持っている。ここでは ZRLE で圧縮された Rectangle が 1 つ、VNC サーバから送られてくる。Rectangle には Zlib 圧縮されたデータが、datelengths と呼ばれる指定された長さだけ付いてくる。このデータは、さらに 64x64 の Tile に分割されている (図 3.2 中 Tile)。

Tile内はパレットなどがある場合があるが、通常はRun Length encodeされたRGBデータである。これまでのTreeVNCではVNCサーバから受け取ったRectangleを分割せずにZRLEEへ再構成を行なっていた。これをMluticastのためにデータを64KByteに収まる最大3つのRectangleに再構成する (図3.2)。この時にTile内部は変更する必要はないが、Rectangleの構成は変わる。ZLREを展開しつつ、Packetを構成する必要がある。

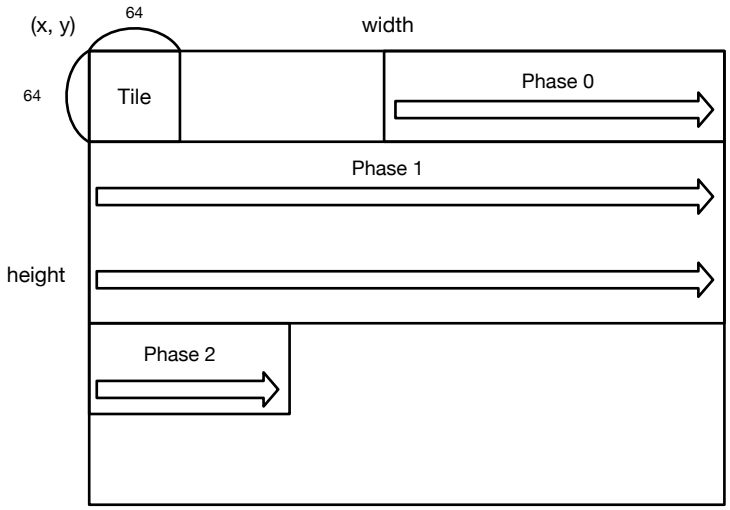


図 3.2: Rectangleの分割

64KByteのPacketの中には複数のTileが存在するが、連続してRectangleを構成する必要がある。3つのRectangleの構成を下記に示す。

- 行の途中から始まり、行の最後までを構成するRectangle(図3.2中 Phase0)
- 行の初めから最後までを構成するRectangle(図3.2中 Phase1)
- 行の初めから、行の途中までを構成するRectangle(図3.2中 Phase2)

3.3 TileLoop

TileLoop は VNC サーバから受け取った ZRLE を図 3.2 のように Rectangle を分割し、ZRLEE に再圧縮を行った Packet を生成する。

以下の図 3.3 に TileLoop で生成される Packet 全体と、分割される各 Phase の Rectangle を示した。

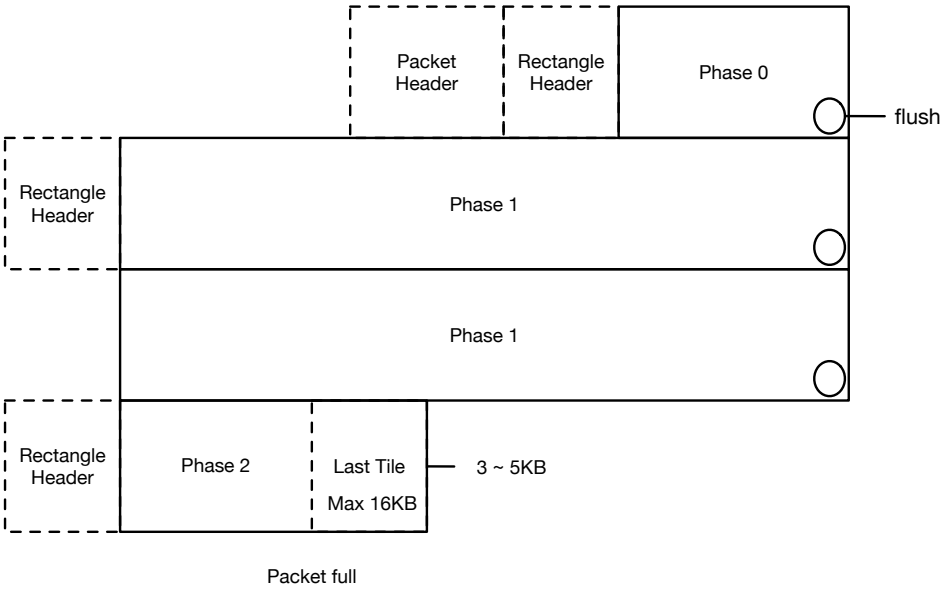


図 3.3: ZRLEE の Packet の構成と分割された Rectangle

Packet Header には表 3.1 に示した messageID、padding、n of rectangles が格納されている。また、分割された Rectangle にはそれぞれ表 3.2 に示した Rectangle Header を持っている。

表 3.2: Rectangle Header の構成

2 byte	U16 - x-position
2 byte	U16 - y-position
2 byte	U16 - width
2 byte	U16 - height
4 byte	S32 - encoding-type
4 byte	U32 datalengths
1 byte	subencoding of tile
n byte	Run Length Encoded Tile

次に TileLoop の処理について説明する。以下の図 3.4 は TileRoop のフローチャートである。

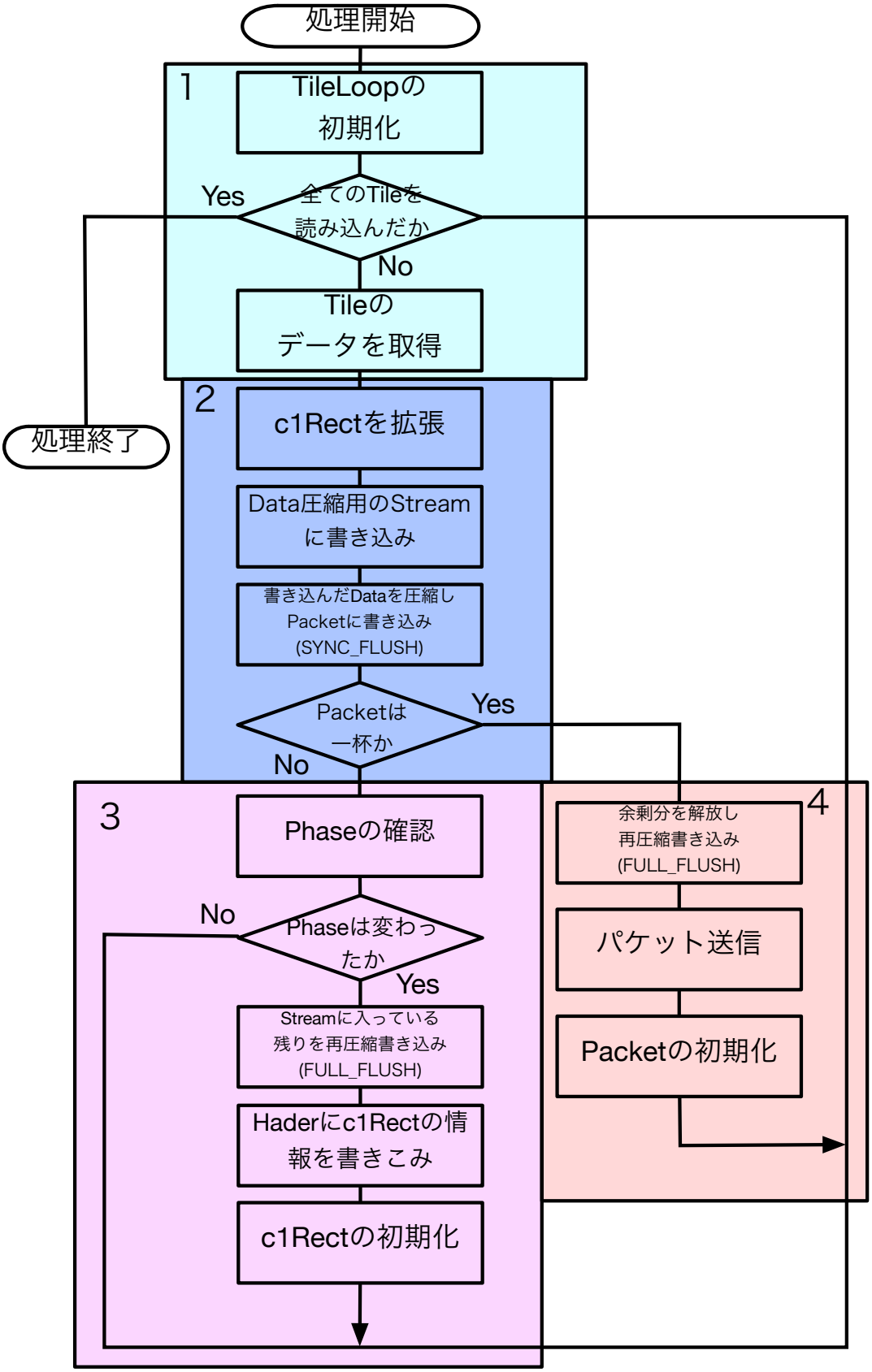


図 3.4: TileLoop のフローチャート

図 3.4 中 1 にて、TileLoop の初期化で Blocking と構築する Packet の準備を行う。Loop 本体では ZRLE で受け取った Recganle を 1Tile 64x64 に分割し、1Tile ずつ処理を行う。そして受け取った ZRLE より処理を行う Tile のデータを取得し、圧縮段階に入る。

TileLoop には c1Rect と呼ばれる Rectangle を持っている。これは読み込んだ Tile 分だけ縦横を拡張していくことによって Rectangle の再構成を行なっている。図 3.4 中 2 の圧縮段階では、読み込んだ Tile のデータを圧縮用の Stream に格納し、java.util.zip.deflater を利用して圧縮を行っている。Packet のサイズは 60KByte としているが、一旦の制限として 42KByte までを格納可能としている。

java.util.zip.deflater には下記の 3 種類の圧縮方法がある。

- NO_FLUSH : Stream に格納されたデータを最高率で圧縮を行う。Stream にある入力データが規定量に満たない場合は圧縮されない
- SYNC_FLUSH : これまでに Stream に格納されたデータの圧縮を行う。ただし圧縮率が低下する可能性がある
- FULL_FLUSH : SYNC_FLUSH 同様、これまでに Stream に格納されたデータの圧縮を行う。異なる点はこれまでの辞書情報がリセットされるため、圧縮率が極端に低くなる可能性がある

ZRLE と java.util.zip.deflater を使用した圧縮では、圧縮後のデータ長を予測することができない。Packet が満杯になってしまうと、圧縮書き込みの途中であっても圧縮書き込みが中断する。そのため、Packet のサイズを余分に確保する必要がある。したがって最初から最大の 60KByte ではなく、42KByte に制限を行っている。TileLoop ではデータの圧縮に NO_FLUSH を利用していたが、圧縮後のデータが Packet の上限である 60KByte を超えてしまうことが多発した。

これは圧縮されるための入力データの規定量が想定以上に多く、圧縮後のデータ長が Multicast Packet の上限を越えてしまったためである。

そこで圧縮率は悪くなるが、確実に Packet に書き込まれる SYNC_FLUSH を利用し、ZRLEE の生成を行う。

図 3.4 中 3 では Packet の上限まで行かなかった場合の分岐である。分岐の初めでは Rectangle の構成を行っている c1Rect と、ZRLE から送られてきた Rectangle などと比較を行い、Phase の確認をする。

Phase の変更がなかった場合は Loop の先頭に戻るが、Phase の変更があった場合、これまで構成していた Rectangle として c1Rect をその Phase の Rectangle Header に書き出す。c1Rect は次の Phase に向けて初期化される。またこの時、図**??**の各 Rectangle の行末にあるように、FULL_FLUSH を行う。FULL_FLUSH を行う理由は、次の行に移る際、圧縮用の Stream にデータを残さないためである。

図 3.4 中 4 の処理は、Packet が一旦の上限 42KByte まで達したことで分岐する。

java.util.zip.deflater を使用した圧縮では、Packet が一杯になってしまうと、圧縮書き出し中でも中断してしまう。そこで Packet の余剰分を解放し上限を 60KByte にすることで、確実に書き込みを可能とする。図 3.3 中 Last Tile とは、Packet が一杯になった際に読み込まれている Tile のことを指す。Last Tile は圧縮前で最大 16KByte と考えられる。これを圧縮すると 3 - 4 KByte 程度であるので、その分のマージンを持っておくことで、読み込んだ最後の Tile まできちんと Packet に書き込むことができる。

Packet に書き込み後は ZRLEE での Multicast Packet が完成したため、子 Node への送信のために MulticastQueue へ Packet が格納される。その後、次の Packet 生成のために Packet が初期化される。

以上のルーチンを ZRLE で受け取った Rectangle 内の Tile 全てで行うことによって、VNC サーバから受け取った ZRLE の画像データを ZRLEE として生成を行うことができる。

3.4 Packet Lost

Wift の Multicast Packet は確実に送られることが保証されていない。データに通し番号をつけて、欠落を検出することはできるが、再送処理は複雑であることが予想される。そこで、一定時間ごとに全画面のデータを送信することによって Packet Lost しても画面共有に影響はないと考える。

第4章　TreeVNC開発環境やソースコードの修正改善

4.1　Gradle 6.1対応

TreeVNCはソースのbuildにGradleを使用している。しかし使用しているGradleのバージョンは4.8であった。これを現行の最新バージョンであるGradle 6.1でbuildを実行すると、build faildとなってしまう。

build faildの原因となっていたのは、MacOS用のapplicationに書き出すためのプラグインedu.sc.seis.macAppBundleのバージョンが古かったことが原因だった。これまで使用していたedu.sc.seis.macAppBundleのバージョンは2.1.7であったが、これを最新の2.3.0に変更することで、Gradle 6.1でソースコードのbuildが可能となった。

4.2　java9以降のRetinaAPI対応

現在のMacにはRetinaディスプレイが搭載されている。Retinaディスプレイとはこれまでに使用されていた液晶ディスプレイよりも画素が細かく、画面がより鮮明に描画されるディスプレイである。画素数が異なるため、画面配信のためには接続されているディスプレイが、液晶ディスプレイか、Retinaディスプレイかの判別が必要となる。

javaにはRetinaディスプレイ判別のためのAPIが提供されている。しかしjava9より大きな仕様変更があり、TreeVNCで使用しているRetinaのAPIは非推奨となってしまったため、書き換えを行なった。非推奨となったコードを以下のソースコード4.1、書き換え後のコードを以下のソースコード4.2、4.3に示す。

ソースコード 4.1: java8 以前の Retina ディスプレイ API

```
1  public static int getRetinaScale(int shareScreenNumber) {
2      int scale = 1;
3      GraphicsEnvironment env = GraphicsEnvironment.getLocalGraphicsEnvironment();
4      final GraphicsDevice[] devices = env.getScreenDevices();
5
6      try {
7          Field field = devices[shareScreenNumber].getClass().getDeclaredField("scale")
               ;
8
9          if (field != null) {
10             field.setAccessible(true);
11             Object retinaScale = field.get(devices[shareScreenNumber]);
12
13             if (retinaScale instanceof Integer) {
14                 scale = (Integer) retinaScale;
15                 return scale;
16             }
17         }
18     } catch (Exception ignore) {}
19     return scale;
20 }
```

ソースコード 4.2: Retina ディスプレイの判別関数

```
1   public boolean getIsRetinaDisplay(int shareScreenNumber) {
2         GraphicsEnvironment env = GraphicsEnvironment.getLocalGraphicsEnvironment();
3         final GraphicsDevice[] devices = env.getScreenDevices();
4         GraphicsConfiguration conf = devices[shareScreenNumber].
              getDefaultConfiguration();
5         return ! conf.getDefaultTransform().isIdentity();
6     }
```

ソースコード 4.3: Retina ディスプレイの表示倍率を取得する関数

```
1  public static int getRetinaScale(int shareScreenNumber) {
2         int scale = 1;
3         GraphicsEnvironment env = GraphicsEnvironment.getLocalGraphicsEnvironment();
4         final GraphicsDevice[] devices = env.getScreenDevices();
5         GraphicsConfiguration conf = devices[shareScreenNumber].
              getDefaultConfiguration();
6         scale = (int)conf.getDefaultTransform().getScaleX();
7        return scale;
8     }
```

java8 以前では表示倍率を取得し、その値より Retina ディスプレイかを判断していた (ソースコード 4.1 中 14 行目)。java9 以降では Retina ディスプレイの判断を行える API が提供された (ソースコード 4.2 中 5 行目)。またソースコード 4.3 の 6 行目にて、接続しているディスプレイの表示倍率を取得する API も提供されため、例外を考える必要がなくなり、コード量が減少した。

4.3 デバッグオプションの修正

TreeVNC のオプションの 1 つに-p オプションがある。通常、TreeVNC のデバッグを行うにはサーバ側とクライアント側の 2 つを同じ PC 上で実行する必要があった。また、ソースコードを変更後には bulid を行わないとデバッグできないという煩わしさがあった。

-p オプションはその煩わしさを解消するために、自らとソケット通信を行うことでデバッグが可能となるオプションとして設計されている。しかし、TreeVNC の仕様変更等の理由により画面データとしての Rectnagle がうまく構成されず使用できないという状態だった。

Blocking の実装中は何度もデバッグを行う必要があったため、前述の煩わしさの解消のために-p オプションの修正を行った。

-p オプションが正常作動しなかった原因として、ディスプレイの選択を行なっていなかったことが挙げられる。設計当初の-p オプションは CUI での動作を想定しており、ディスプレイがない場合もあるという想定だった。そのため、画面配信用の画面が選択されておらず、画面データがうまく生成されない状況が発生していた。

これを、確実に GUI でビューワーを表示するという前提でディスプレイ選択を行うことで、正確に画面データの生成を行わせ-p オプションでのデバッグが可能となった。また、自身で画像データの圧縮し Multicast Queue にデータを送っているため、Blocking のルーチンがデバッグ可能となっている。

第5章　まとめ

本研究では TreeVNC に Wifi 上の Multicast Packet を用いる手法の考察と実装と、Gradle6.1 対応、java9 以降の RetinaAPI 対応、デバッグオプションの修正を行った。

画面圧縮に Hardware supported な MPEG4 などを用いることができればより効率的な転送が可能であるが、ここでは Java 上で実装できる安易な方法をあえて選択した。Wifi の速度と Multicast の信頼性が高ければこれでも実用になる可能性がある。

Blocking 後の Multicast 送信は実装中であるが、Multicast Packet の Packet loss 率は、接続環境に依存すると思われるのでさらなる実験が必要だと思われる。

有線接続時よりも、画面共有の質が落ちるのはある程度はやむお得ないが、再送が必要である場合には、必要なプロトコルを実装する。

VNC サーバへの接続方法の分割についても、Node 接続時の Network Interfaces から有線接続か無線 LAN 接続かを完全に区別できない。接続時にユーザが選択するか、接続時にある程度区別する処理を実装する必要がある。

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] 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.

[8] 立樹伊波, 真治河野. 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予稿集, 第 2016 巻, pp. 29–37, jan 2016.

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したします。

数々の貴重な御助言と細かな御配慮を戴いた並列信頼研究室の外間政尊さん、桃原優さん、清水隆博さん、東恩納琢偉さんに深く感謝致します。

また一年間共に研究を行い、暖かな気遣いと励ましをもって支えてくれた並列信頼研究室の一木貴裕君、坂本昂弘君、福田光希君に感謝致します。

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。

2020 年 2 月
安田亮

付録

情報処理学会研究報告
IPSJ SIG Technical Report

画像配信システム TreeVNC のマルチキャストの導入

安田 亮[†1,a)]　大城 由也[,b)]　河野 真治[†1,c)]

概要：講義やゼミでは PC 画面で用意した資料を見ながら進行することが多い．参加者もプロジェクタに集中を割く必要があり，手元の PC と相互に参照する場合，負担になる場合がある．発表者が交代する場合は発表者の PC 画面の切り替えでケーブルを差し替えを行う必要があるが, PC と接続するアダプターによって正常に PC 画面を表示できない場合がある．当研究室で開発している TreeVNC は，発表者の PC 画面を参加者の PC に表示する画面配信システムである．サーバに接続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ることで，多人数が接続しても処理性能が下がらないような設計になっている．また，発表者の画面を自由に切り替える仕組みが存在し，ゼミなどでの発表に便利なものになっている．現在，TreeVNC 画面共有は送信するデータ量が多いため有線 LAN での使用に限られている．広く使われている無線 LAN に対応するために multicast でのデータ通信の実装やデータの分割・圧縮方法の評価を行い，TreeVNC の multicast の可能性を評価する．

Introduction of multicast of screen delivery software TreeVNC test

Ryo Yasuda[†1,a)]　Yukiya Oshiro[,b)]　Shinji Kono[†1,c)]

Abstract: In lectures and seminars, the materials prepared on the PC screen is used. Participants need to concentrate on the projector, which can be a burden when cross-referencing with the PC at hand. When the presenter is replaced, the cable needs to be replaced by switching the presenter's PC screen, depending on the adapter connected to the PC, the PC screen may not be displayed properly. TreeVNC, which is being developed in our laboratory, is a screen distribution system that displays the presenter's PC screen on the participant's PC. By connecting clients connected to the server in the form of a binary tree and distributing the delivery cost, It is designed so that the processing performance does not decrease even if many people connect. In addition, there is a mechanism for freely switching the screen of the presenter, which is convenient for presentations at seminars and the like. Currently, TreeVNC screen sharing is limited to wired LANs because of the large amount of data. To support widely used wireless LAN, we evaluate the implementation of data communication in multicast and the data division and compression method, and evaluate the possibility of multicast in TreeVNC.

1. 画面配信ソフトウェア TreeVNC の活用

現代の講義や発表，プレゼンなどでは PC 画面で用意した資料を見ながら進行することが多い．ゼミでは発表者の PC 画面を切り替えを行いながら発表を行う場合もある．

通常このような場面では資料やスライドを表示するためにプロジェクタが利用される．その際，発表者の PC 画面を切り替えるたびにケーブルを差し替える必要がある．発表者の PC によっては接続するアダプターの種類や解像度の設定により，正常に PC 画面を表示できない場合がある．また，参加者もプロジェクタに集中を割く必要があり，手元の PC と相互に参照する場合，負担になる場合がある．

当研究室で開発している画面配信システム TreeVNC[1] は，発表者の画面を参加者の PC に表示するソフトウェアである．そのため，参加者は不自由なく手元の PC を操作しながら講義を受けることが可能になる．更に発表者の切り替えの際もケーブルを差し替えずに，共有する画面の切り替えが可能になっている．

[†1] 現在，琉球大学工学部情報工学科
Presently with Information Engineering, University of the Ryukyus.
a) riono210@cr.ie.u-ryukyu.ac.jp
b) oshiro@cr.ie.u-ryukyu.ac.jp
c) kono@ie.u-ryukyu.ac.jp

- 配信者の VNC サーバから Root Node への通信を行う send message （VNCServer to Root）

3.1 メッセージ通信

TreeVNC の各 Node と VNCServer 間で通信されるメッセージを表 1 に示す．

3.2 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に Multicast Queue というキューに画像データを格納する．

3.3 木構造の再構成

TreeVNC はバイナリツリーでの接続のため，Node が切断されたことを検知できないと構成した木構造が崩れてしまい，新しい Node を適切な場所に接続できなくなってしまう．そこで木構造を崩さないよう，Node 同士の接続の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 で管理している．Node の接続が切れた場合，Root Node に切断を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 の切断を検知および木構造の再構成を行なっている．LOST_CHILD の検出方法には MulticastQueue を使用しており，ある一定時間 MulticastQueue から画像データが取得されない場合，MemoryOverFlow を回避するために Timeout スレッドが用意されている．そして，Timeout を検知した際に Node との接続が切れたと判断する．

3.4 ZRLEE

TreeVNC では，ZRLEE[4] というエンコード方法でデータの圧縮を行う．ZRLEE は RFB プロトコルで使用できる ZRLE というエンコードタイプを元に生成される．

ZRLE は Zlib[5] 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送信される．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 はデコードに必要な辞書を書き出す（flush）ことが出来ない．従って，圧縮されたデータを途中から受け取ると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 と呼ばれる画面ごとのデータに辞書を付与して zip し直すことで初めからデータを読み込んでいなくても解凍できるようにした（図 3）．辞書をクリアすることにより adaptive compression を実現していることになり圧縮率はむしろ向上する．

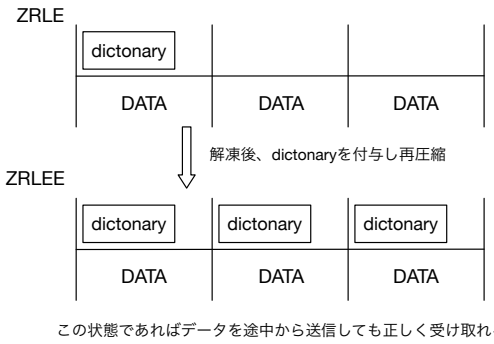


図 3 ZRLEE

3.5 ShareScreen

従来の VNC では，配信者が切り替わるたびに VNC の再起動，サーバ，クライアント間の再接続を行う必要がある．TreeVNC は配信者の切り替えのた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．画面の切り替えについてはユーザが VNC サーバへの再接続を行うことなく，ビューワー側の Share Screen ボタンを押すことで配信者の切り替えが可能になっている．

TreeVNC の Root Node は配信者の VNC サーバと通信を行なっている．VNC サーバ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 を実行すると，Root Node に対し SERVER_CHANGE_REQUEST というメッセージが送信される．このメッセージには Share Screen ボタンを押した Node の番号やディスプレイ情報が付加されている．メッセージを受け取った Root Node は配信を希望している Node の VNC サーバと通信を始める．

3.6 ネットワーク複数時の接続

TreeVNC は Root Node が複数のネットワークに接続している場合，図 4 のようにネットワーク別に木構造を形成する．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おり，木構造を管理する nodeList を生成する．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．

Tree Manager は Root Node の保持しているネットワー

表 1 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.

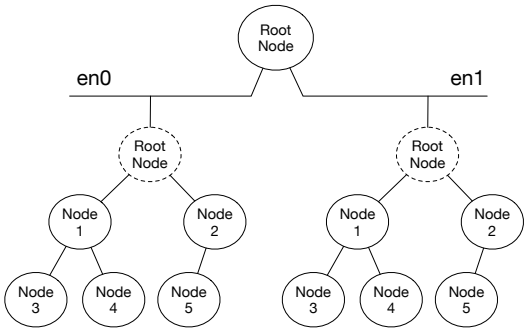


図 4 Multi Network Tree

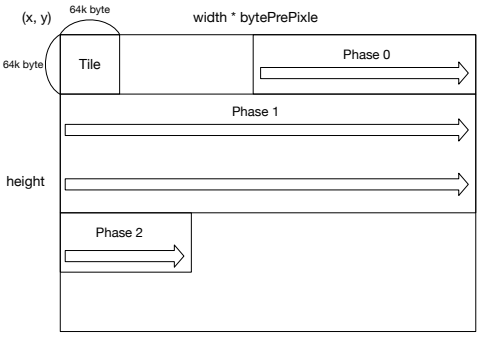


図 5 Rectangle の分割

ク毎に生成される．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 Tree Manager を取得し，Node 接続の処理を任せる．

4. 有線接続との接続形式の違い

画面配信のデータ量は膨大なため，現在の TreeVNC で VNCServer に無線 LAN 接続を行なった場合，画面配信の遅延が大きくなってしまう．

この場合でも画面切り替えの機能は有効である．つまり，画面を提供する PC のみを無線経由で接続し，配信を希望する側は有線を使用することができる．ここで，Wifi の Multicast の機能を用いて配信側にも Wifi を使用することが可能であると考えられる．Tree Root は無線 LAN に対して，変更する UpdateRectangle を Multicast で一度だけ送信する．

Wifi の Multicast packet のサイズは 64kbyte が最大となっている．HD や 4K の大きさの画面更新は 8Mb * 8byte で圧縮前で 64MB 程度になる．

これを圧縮しつつ，64kbye 毎のパケットに変換して送る必要がある．Wifi の Multicast packet は確実に送られること保証されていない．通し番号を付けて欠落を検出することはできるが，再送処理は複雑であることが予想される．

ここではまず Blocking について考察と実験を行う．

5. RFB の UpdateRectangle の構成

RFB の UpdateRectangle は以下の表 2 の構成になっている．一つの updateRectangle には複数の Rectangle は入っていて，さらに一つ一つの Rectangle の encoding type がある．ここでは ZRLE で encode された Rectangle が一つサーバから送られてくる．Rectangle には zlib で圧縮されたデータが指定された長さだけ付いてくる．このデータは，64x64 の tile にさらに分割されている（図 5）．tile 内はパレットとかがある場合があるが，Run length encode された RGB データである．

このパケットを 64kbyte に収まる三つの Rectangle に再構成する．この時に，tile 内部は変更する必要はないが，Rectangle の構成は変わる．ZRLE を展開しつつ，パケットを構成する必要がある．

zlib はちょうど良い所で圧縮を flush する必要がある．このためには，zlib の API を用いて，適当なタイミングで flush を呼ぶ．この時に 1 tile ずづ flush してしまうと圧縮率を下げる可能性がある．

64kbyte の packet の中には複数の tile が存在するが，連続して Rectangle を構成する必要がある．行の途中から始まり，途中で終わる可能性があるので，三つの Rectangle が必要になる．

表 2 updateRectangle の構成

1 byte	messageID
1 byte	padding
2 byte	n of rectangles
2 byte	U16 - x-position
2 byte	U16 - y-position
2 byte	U16 - width
2 byte	U16 - height
4 byte	S32 - encoding-type
4 byte	U32 datalengths
1 byte	subencoding of tile
n byte	Run Length Encoded Tile

6. 木構造とマルチキャストの共存

現在の TreeVNC では有線接続と無線 LAN 接続のどちらでも，VNC サーバから画面配信の提供を受けることが可能である．しかし無線 LAN で接続している Node が存在すると，バイナリツリーを形成している全体の配信遅延に繋がってしまう．この問題点を解決するために，有線接続時と無線 LAN 接続時での VNC サーバの接続方法の分割を提案する．

有線接続の場合は従来通り，VNC サーバ，Root Node，Node からなるバイナリツリー状に接続される．無線 LAN 接続の場合は，Multicast で接続を行う．こうすることに

より，新しい Node が無線 LAN 接続出会っても有線接続のバイナリツリーには影響を及ぼさない（図 6）．

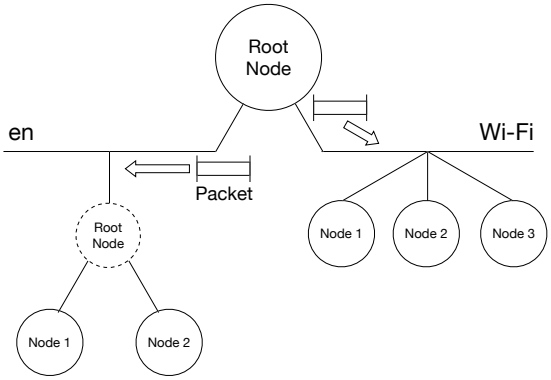


図 6 接続方法の切り替え

7. まとめ

Tree VNC に Wifi 上の Multicast packet を用いる手法について考察した．画面圧縮に Hareware supported な MPEG4 などを用いることができればより効率的な転送が可能であるが，ここでは Java 上で実装できる安易な方法をあえて選択した．Wifi の速度と Multicast の信頼性が高ければこれでも実用になると可能性がある．

Blocking は実装中であり，再圧縮の時間は前画面の時でも実用的な時間ですむことが予想されている．Wifi 上の Multicast packet の drop 率は，接続環境に依存すると思われるのでさらなる実験が必要だと思われる．

有線使用時よりも，画面共有の質が落ちるのはある程度はやむを得ないが，再送が必要である場合には，必要なプロトコルを実装する．

VNC サーバへの接続方法の分割についても，Node 接続時の interfaces から有線接続か無線 LAN 接続かを完全に区別出来ない．接続時にユーザが選択するか，接続時にある程度区別する処理を実装する必要がある．

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 (2011).
[2] RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K. R., AND HOPPER,: A. Virtual Network Computing (1998).
[3] RICHARDSON, T., AND LEVINE, J.: The remote framebuffer protocol. RFC 6143 (2011).
[4] 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) (2012).
[5] LOUP GAILLY, J., AND ADLER, M.: zlib: A massively spiffy yet delicately unobtrusive compression library., http://zlib.net.
[6] TightVNC Software: http://www.tightvnc.com.
[7] 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* (2012).
[8] 立樹伊波，真治河野：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予稿集，Vol. 2016, pp. 29–37 (2016).