Multicast Wifi VNCの実装と評価

安田 亮[1,a)]　河野 真治[2,b)]

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。PCごとにアダプターや解像度が異なっており、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。TreeVNCの画像共有は、送信するデータ量が多いために有線LANでの接続に限られている。本稿では無線LANでもTreeVNCを利用可能にするため、Wifi上にシステム制御用の従来の木構造と、画像データ送信用のMulticastの両方を構築を行う。Multicastでは、サーバから送信された画像データUpdateRectangleを小さいパケットに分割し送信を行うよう実装した。

1. 画面配信ソフトウェア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移しながら進行することが多い。ゼミでは発表者を交代する際に、PC画面の切り替えを行う必要がある。

その場合、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。さらに、発表者の交代もケーブルの差し替えを行わずに、全体に共有する画面の切り替えが可能となっている。

しかし、画面共有は送信するデータ量が多いため、無線LAN接続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通信の実装を行い無線LAN接続でもTreeVNCを利用可能にし、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。

1　琉球大学大学院理工学研究科情報工学専攻
2　琉球大学工学部工学科知能情報コース
a)　riono210@cr.ie.u-ryukyu.ac.jp
b)　kono@ie.u-ryukyu.ac.jp

2. TreeVNCの基本概念

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。また、動作にはRFBプロトコルを用いている。

Remote Frame Buffer プロトコル[3](以下RFB)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Framebuffffer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。

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なっているが、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 1)。

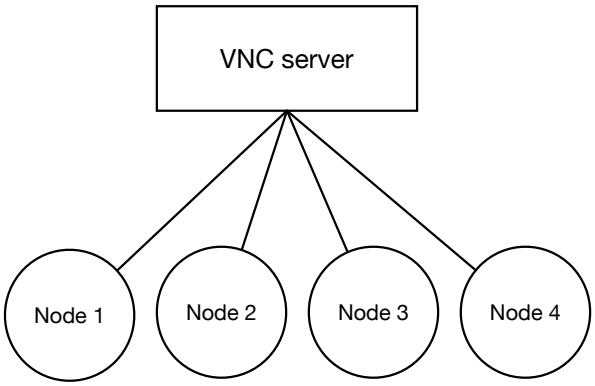

図 1 従来の VNC での接続構造

そこで TreeVNC ではサーバに接続を行な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)。

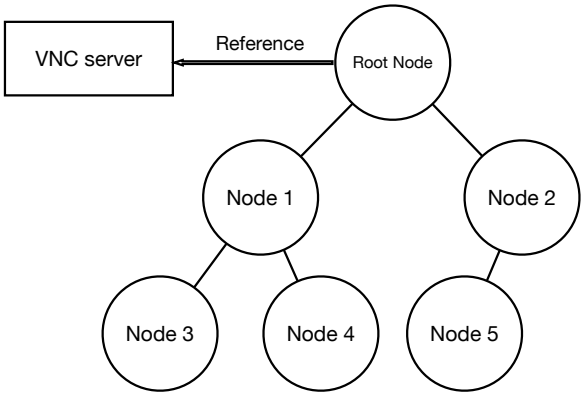

図 2 TreeVNC での接続構造

通信の数は、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。

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なってきた場合、従来の VNC ではサーバ側が N 回のコピーを行な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。

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なっている。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。

3. MulticastQueue

配信側の画面が更新されると、VNC サーバから画像データが FRAME_BUFFER_UPDATE メッセージとして送られる。その際、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。

4. 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 切断の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。

5. データの圧縮形式

TreeVNC では、ZRLEE[5] 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。

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出す (flush) ことができない。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できない。

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、

始めからデータを読み込んでいなくても解凍をできるようになっている (図 3)。

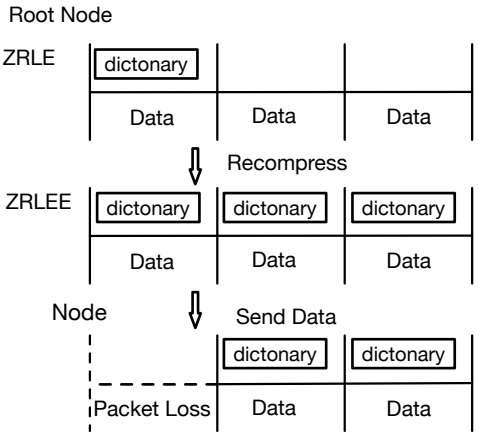

図 3 ZRLEE へ再圧縮されたデータを途中から受け取った場合

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いけないため、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は向上する。

6. ShareScreen

従来の VNC では、配信者が交代するたびに 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 で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せずに、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 Shere 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。

7. 有線接続と無線接続の違い

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。しかし画面配信のッデータ量は膨大なため、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 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 で一度だけ送信すればよい。

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 4)。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影響を及ぼさない。

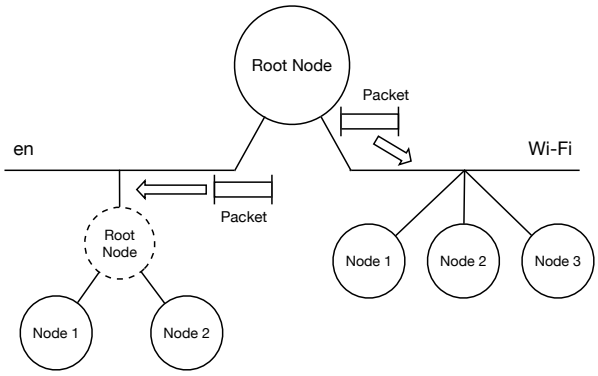

図 4 接続方法の分割

Wifi の Multicast Packet のサイズは 64KB が最大となっている。4K ディスプレイと例にとると、画面更新には 8MB の画素数 * 8B の色情報となり、圧縮前で 64MB 程度となる。

8. RFB の UpdateRectangle の構成

表 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

RFB の Update Rectangle によって送られてくる Packet は以下の表 1 のような構成となっている。

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 detalengths 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 5 中 Tile)。

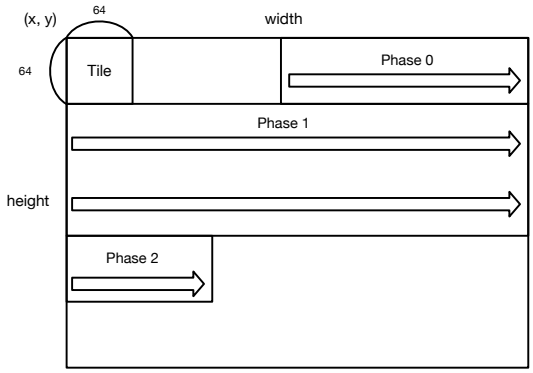

図 5 Rectangle の分割

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 Multicast のためにデータを 64KB に収まる最大 3 つの Rectangle に再構成する。(図 5)。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、Packet を構成する必要がある。

64KB 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 5 中 Phase0)
- 行の初めから最後までを構成する Rectangle(図 5 中 Phase1)
- 行の初めから、行の途中までを構成する Rectangle(図 5 中 Phase2)

9. TileLoop

TileLoop は VNC サーバから受け取った ZRLE を図 5 のように Rectangle を分割し、ZRLEE に再構成を行った Packet を生成する。

以下の図 6 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。

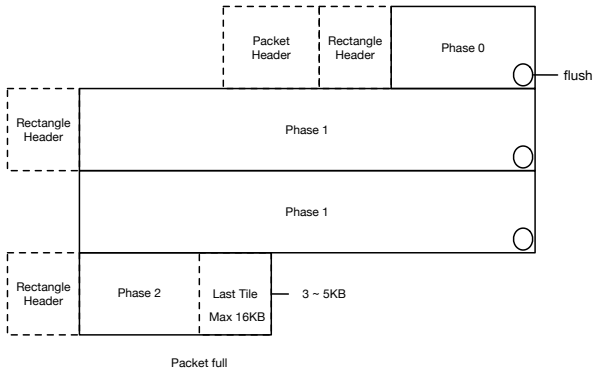

図 6 ZRLEE の Packet の構成と分割された Rectangle

Packet Header には表 1 に示した messageID、padding、n of rectangle が核にのうされている。また、分割された Rectangle にはそれぞれ表 2 に示した Rectangle Header を持っている。

表 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 7 は TileRoop のフローチャートである。

図 7 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 Rectangle 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**??**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なっている。Packet のサイズは 60KB としているが、一旦の制限として 42KB 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効率で圧縮を行う。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 に圧縮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

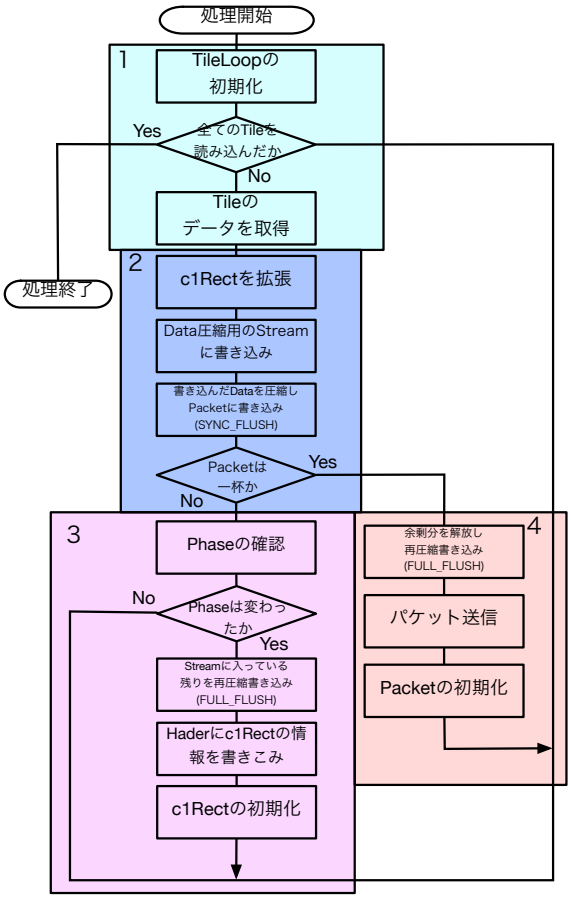

図 7 TileLoop のフローチャート

- FULL_FLUSH : SYNC_FLUSH 同様、これまでに Stream に格納されたデータ圧縮を行う。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 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 60KB ではなく、42KB に制限を行なっている。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 60KB 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 の上限を超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 7 中 3 では Packet の上限までいかなかった場合の分岐である。分岐の初めでは Rectangle の構成を行なっている 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 6 の各 Rectangle の行末にあるように、FULL_HLUSH 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 7 中 4 の処理は、Packet が一旦の上限 42KB 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 60KB にすることで、確実に書き込みを可能とする。図 6 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 16KB と考えられる。これを圧縮すると 3 - 4KB 程度であるので、その分のマージンを持っていくことで、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。

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。

10. Multicast 用のシステム構成

11. まとめ

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] 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) (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* (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).