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

# 図 目 次

# 表 目 次

# 第1章　画面共有を利用したコミュニケーション

# 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 [?] 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、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。

## 2.4 TreeVNC の構造

TreeVNC は Java を用いて作成された TightVNC(Tight Virtual Network Computing)[?] 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 2.1)。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。

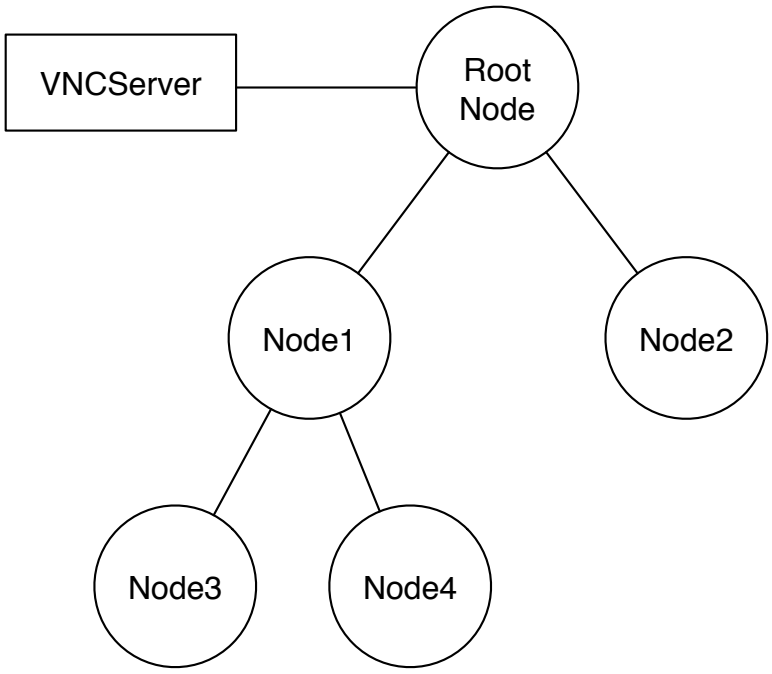


図 2.1: 構成される木構造

## 2.5 圧縮形式

TreeVNC は ZRLEE[?] 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[?] 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 2.2 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 zip し直して最後に finish() をいれる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 2.3)。

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。

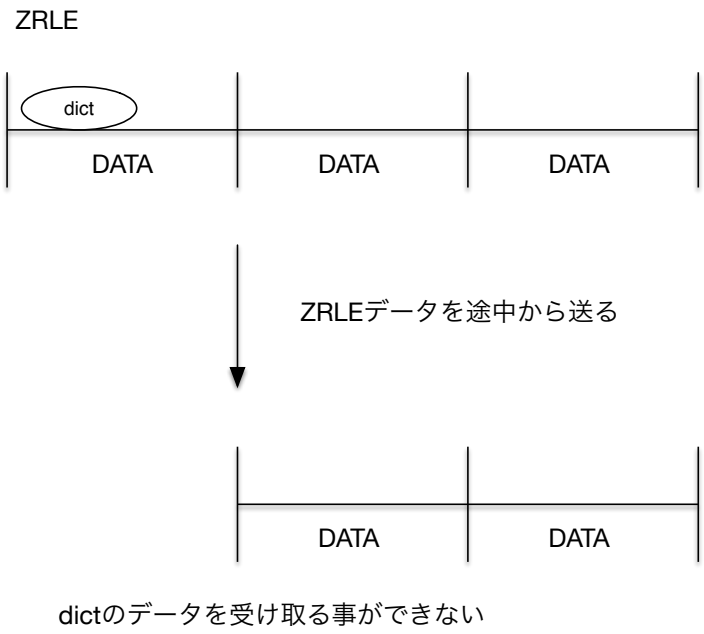


図 2.2: ZRLE での問題点

## 2.6 通信経路

TreeVNC の通信経路として以下が挙げられる

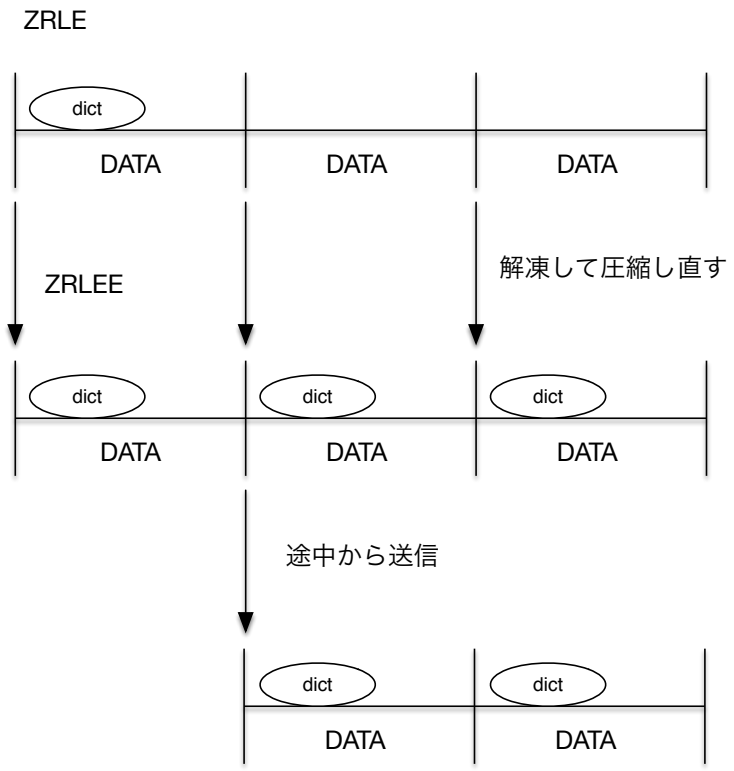
ZRLE
dict
DATA
DATA
DATA
ZRLEE
解凍して圧縮し直す
dict
dict
dict
DATA
DATA
DATA
途中から送信
dict
dict
DATA
DATA
途中から受け取っても辞書がある

図 2.3: ZRLEE

- 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)

## 2.7 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 2.1 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 | 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 2.4 は TreeVNC で Node が Root Node 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 2.4 の手順として

- 接続を行う Node (以下 Client Node) は Multicast 通信で Root Node に対して FIND_ROOT を送信する (1:findRoot())
- Root Node が FIND_ROOT を受信し、FIND_ROOT_REPLY を送信する (2:findRootReplay())

- Client 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
- Client 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 (3:whereToConnect())
- 受信した Root Node は Client Node の接続先を CONNECT_TO で送信する (4:connectTo)
- Client Node は Root の指定した接続先に接続しに行く
- Root Node, Client Node 間の接続が確立後、Root Node から Clinet 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 2.4: node 間で行われるメッセージ通信

## 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 $\mathrm{FRAME}_{B}UFFER_{U}PDATE$ 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。

## 2.9 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。

LOST_CHILD の検知・木の再構成を以下に示す。

- 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 2.5 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 2.5 中, 2:updateNodeList())
- 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 Node は最後尾の Node に、切断した子 Node が接続していた親 Node に接続する様に CONNECT_TO メッセージを送信する (図 2.5 中, 3:connectTo(1))
- 最後尾の Node が子 Node を失った親 Node へ接続しに行く (図 2.5 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 2.5: LOST_CHILD を検知・再接続

## 2.10 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 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、Root Node は Share Screen ボタン を押したクライアントの VNC サーバーと通信を始める。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。

## 2.11 複数のネットワークの対応

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、Root Node が複数のネットワークに接続していても、 単一のネットワーク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 2.6 の様に、ネットワーク別に 木構造を形成するように設計した。

図 2.6: 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 を生成する用に変更を行った。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。そのため、 TreeVNC が複数のネットワーク別に木構造を構成することが可能となる。

# 第3章　TreeVNC の追加機能

## 3.1　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、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。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。そのため、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 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となる。

図 3.1 は Display1 のみを画面共有する例を示している。

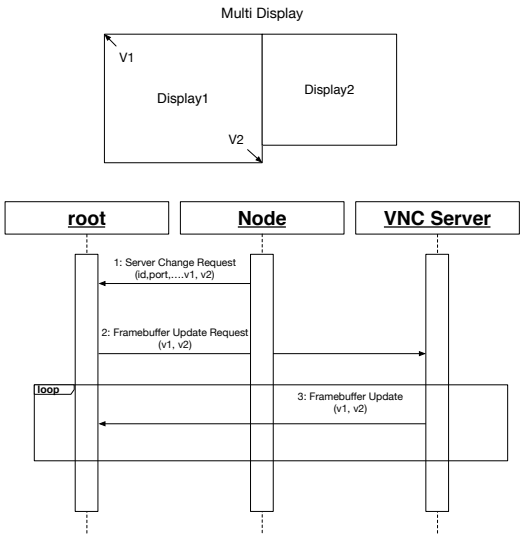


図 3.1: マルチディスプレイへの対応

## 3.2 NAT越え

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 への接続を可能にした。

図 3.2 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。

Direct Connection した Node はそのネットワークの Root Node になる。そのネットワークの他の Node はそのネットワークの Root Node に接続し、木構造を生成する。

配信側の Root Node は Direct Connection で接続された Root Node に対して Framebuffer Update で 画像データを送信する。Framebuffer Update が送信された Root Node は そのネットワークの Node に対して Framebuffer Update を送信する。

これにより、NAT を越えたネットワークの画面共有が可能となる。

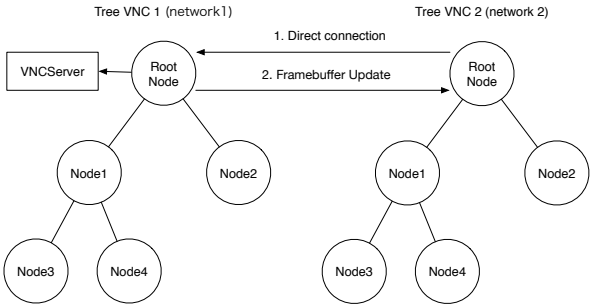


図 3.2: 遠隔地 Node からの接続

## 3.3 ネックになっているノードへの対処

## 3.4 別 Thread での画面切り替え

# 第4章　TreeVNC の評価

## 4.1　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。

## 4.2　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。

## 4.3　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 4.1 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 4.1: 遅延時間の計算方法

## 4.4 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 4.2 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場合がある。そのため、遅い Node を検出して、木の最後尾に移動させる機能が必要である。

今回 4 段分のデータでは 30 名程度の遅延のみしか判断することができないため、更に大人数での実験を繰り返し行う必要がある。

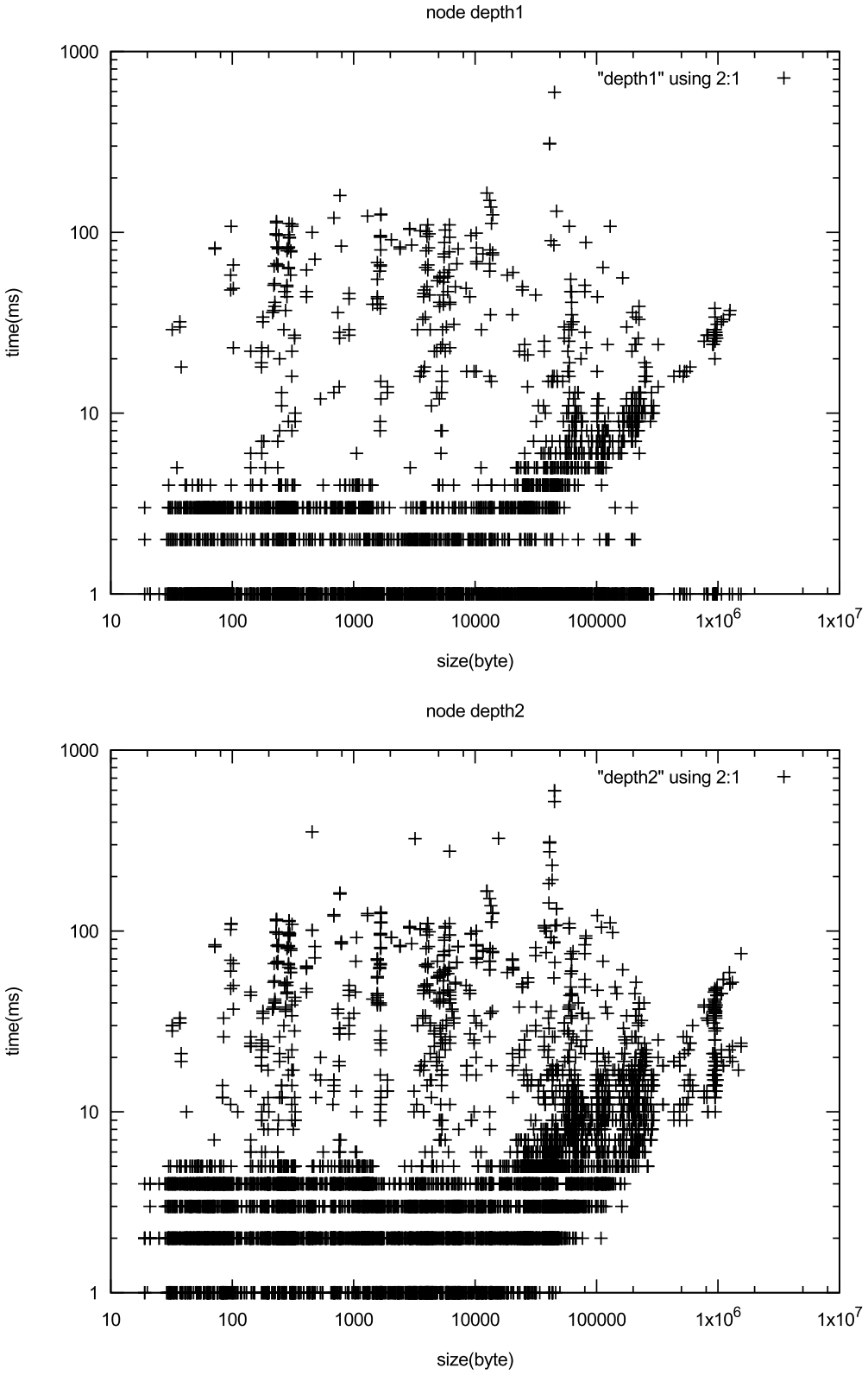


図 4.1: 深さ 1,2 のデータサイズと遅延の関係

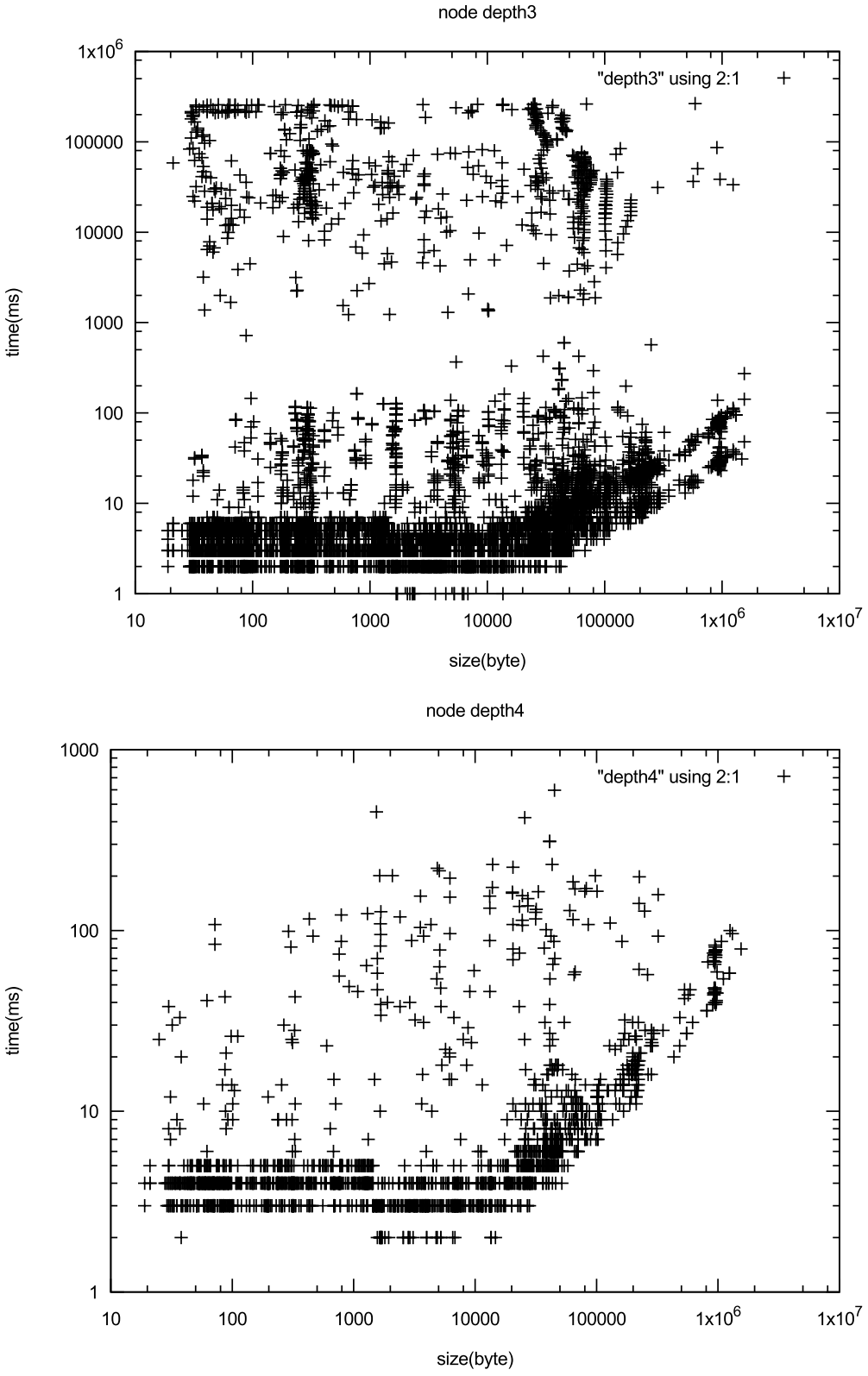


図 4.2: 深さ 3, 4 のデータサイズと遅延の関係

# 第5章　まとめ

## 5.1　NATを越えた画面切り替え

## 5.2　新機能の評価

# 参考文献

[1] hoge

# 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 hoge 助教授に深く感謝したします。

また、本研究の遂行及び本論文の作成にあたり、日頃より終始懇切なる御教授と御指導を賜わりました hoge 教授に心より深く感謝致します。

数々の貴重な御助言と細かな御配慮を戴いた hoge 研究室の hoge 氏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 hoge 研究室の hoge 君、hoge 君、hoge さん並びに hoge 研究室の hoge、hoge 君、hoge 君、hoge 君、hoge 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。

2010 年 3 月

hoge