Vol.2011-IOT-12 No.13
2011/2/28

ブロードキャストが可能な環境においてノードの平衡２分木状接続を自律的に行う分散アルゴリズムとその試験的実装

山之上　卓[†]　今村文紀[†]　小田謙太郎[†]　下園幸一[†]

自律的に，平衡２分木状に，ノード間接続を行う分散アルゴリズムと，それを試験的に実装した構造型 P2P システムについて述べる．このアルゴリズムは，ノード間が IP reachable で，かつ，ブロードキャスト通信が可能な範囲内で利用できる．ある条件が満たされた場合，N をノード数とするとすべてのノードが接続されるのに必要な時間は$O((\log N)^2)$ となる．任意の節ノードが停止してもその下位ノードでこのアルゴリズムを再起動することにより，再接続が行われる．

An Autonomous Distributed Algorithm, which Forms Balanced Binary Tree Structure of Nodes in the Broadcast-able Environment, and its Experimental Implementation

Takashi Yamanoue[†],　Fumiki Imamura[†],
Kentaro Oda[†] and Koichi Shimozono[†]

This paper shows an autonomous distributed algorithm which forms balanced binary structure of nodes, and an experimental structured P2P system which adopts the algorithm. This algorithm can be applied when nodes are IP reachable and broadcast-able. The time complexity of this algorithm is $O((\log N)^2)$, where N is the number of nodes, if some conditions are satisfied. The shape will be fixed by re-starting the algorithm at children of the failure node.

[†]鹿児島大学
Kagoshima University

1. はじめに

企業や教育機関の情報インフラ管理者は，管理している大量の端末に同一のソフトウェアやデータなどの大量の情報を短時間に配布することが必要になる場合がある．このとき，情報を，信頼性を持って配布することが重要になる．近年の MPEG や H.264 などの動画は主にキーフレームとその差分によって構成されており，受信端末がキーフレームの受信を失敗すると，長い間画像に大きな乱れが生じる．このように高品質な動画を端末に放送する場合も通信の信頼性が重要になる．

大量の同一データを大量の端末に短時間で信頼性を持って配布するには，ネットワークにスイッチを使い，ノード間を平衡２分木状にTCPで接続してデータを分割して並列に転送することが有効な手段の一つであることが示されている[2]．

我々は教育支援システム 「SOLAR-CATS[4)5)8)]」を開発している．SOLAR-CATS は１つの端末のディスクトップ画面を大量の他の端末のディスクトップに短時間に信頼性を持って転送する機能を持っており，これを実現するためにノード間の平衡２分木状接続を行っている一種のP2Pシステムである．このような接続を行うため，SOLAR-CATS は「グループマネージャ」と名付けた１つのホストで動作するプログラムを動作させており，新規にグループに参加するノードはグループマネージャに問い合わせを行い，そのノードの接続先を決定していた．グループ内のノードに障害が発生して通信不能になったり，グループから離脱したりした場合も，その周りのノードがグループマネージャに問い合わせ，平衡２分木状の結合を維持するよう，再接続を行っていた．授業でこのシステムを利用する場合，短時間に大量の問い合わせがグループマネージャに届き，グループマネージャの負荷が上がる．このため従来のシステムはノード接続時の動作が不安定な場合があった．また，接続時にグループマネージャのホスト名またはIPアドレスを参加するノードで指定する必要があり，これも本システムを使いにくくしている原因の一つであった．

このような問題に対処するため，ノード間を平衡２分木状に接続するための新たなアルゴリズムを考案し，SOLAR-CATS に試験的に実装した．本稿ではこのことについて述べる．

2. アルゴリズム

このアルゴリズムでは，２分木状に接続されたグループに新たにノード（参加要求ノード）が加わる時，参加要求ノードがグループのすべての葉ノードまたは子を１つしか持っていないノードに対して参加要求メッセージ（データグラム）を送信する．参加要求メッセージを受け取ったノードが参加要求ノードに返事を送る．参加要求ノードでは複数の返事を受け取る場合があるが，最初に返事を届けたノードを接続先ノードとする．参加要求ノードが接続先ノードの子ノードとして，接続先ノードに接続

する．ここで，最も根に近いノードの返事が最も早く参加要求ノードに届くようにすることにより，平衡２分木状にノード間が接続される．図 1 に従来の接続方法と今回の接続方法の比較を示す．

アルゴリズムを構成する具体的な手続きを以下に示す．葉ノードまたは子を１つしか持っていないノードで動作させる手続きが InitialRequestServer である．最初は根ノードのみでこの手続きが実行される．参加要求するノードで動作させる手続きが InitialRequestClient である．ノードがグループに参加すると，そのノードで，この手続きの実行が終了し，このノードで新たに InitialRequestServer が実行される．

すべてのノードがグループに参加し，InitialRequestClient を実行するノードがいなくなったとき，アルゴリズムは停止する．

InitialRequestServer はすでにグループに参加し，２分木を構成しているノード(グループノード)で実行される．ここでは以下を繰り返し実行する．もし，グループノードの左側の子ノードがまだ接続されていない状態であれば，左側に子ノードが接続されるまで，まだグループに参加していないノード(参加要求ノード)からの参加要求メッセージ(IP マルチキャスト)を待つ．参加要求メッセージを受け取ると，そのノードに対して，自ノードのアドレスと左側ポート番号を返事として TCP で返し，子ノードが左側に接続されるまで待つ．

もし，グループノードの右側の子ノードがまだ接続されていない状態であれば，右側に子ノードが接続されるまで，参加要求ノードからの参加要求メッセージ(IPマルチキャスト)を待つ．参加要求メッセージを受け取ると，そのノードに対して，自ノードのアドレスと，右側ポート番号を返事としてTCPで返し，子ノードが右側に接続されるまで待つ．

参加要求ノードは最も早く受信した返事のみを受領し，その返事に書かれたアドレスのノードのポートに接続する．

左右それぞれの場合について，参加要求メッセージを受け取ってから返事を返すまで，根ノードからそのグループノードまでの木の高さに比例した (k*my_node.height) 時間を待つ．このことにより，参加要求を待っている複数のグループノードが同時に１つの参加要求ノードから放送された参加要求を受信した場合，根に近いノードが先に返事を返す．これにより，根に近い側のノードの左右のどちらかが空いている場合は，そのポートに優先して接続が行われるため，平行２分木状にノードが接続されることになる．

InitialRequestCllient はグループにまだ参加していない，参加要求ノードで実行される．自ノードがグループに参加するまで，参加要求メッセージを放送（IPマルチキャスト）し，返事を受け取るスレッドの起動を行い，一定期間待つことを繰り返し行う．

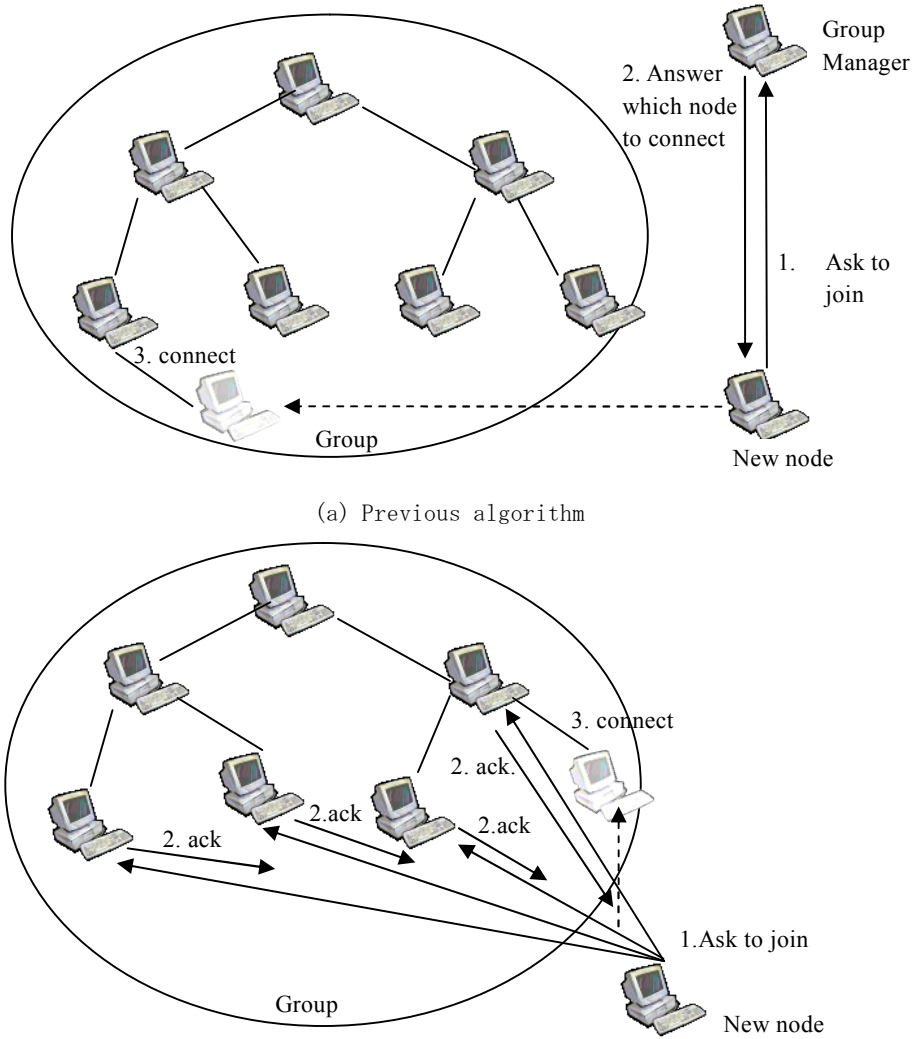

図1．従来の接続方法と本接続方法の比較

```
Procedure InitialRequestServer
begin
  this_multicast_socket.join(“mcast_port”);
  repeat
    if  my_node.left_is_not_connected then
    begin
      message←this_multicast_socket.receive_a_request_message;
      wait_time( k*my_node.height);  // In order to construct balanced binary tree.
      the_remote_requester_address←message.source_address;
      this_tcp_socket.connect_to_(the_remote_requester_address , “recv_port”);
      this_tcp_socket.send( my_node.“address” ,  my_node.“left_port”);
      ack←this_tcp_socket.receive_ack;
      if ack== “accepted” then
        wait_until my_node.left_is_connected ;
    end;
    else
    if  my_node.right_is_not_connected then
    begin
      message←this_multicast_socket.receive_a_request_message;
      wait_time( k*my_node.height); // In order to construct balanced binary tree.
      the_remote_requester_address←message.source_address;
      this_tcp_socket.connect_to_(the_remote_requester_address , “recv_port”);
      this_tcp_socket.send( my_node.“address” ,  my_node.“right_port”);
      ack←this_tcp_socket.receive_ack;
      if ack== “accepted” then
        wait_until my_node.right_is_connected ;
    end;
    else wait_time(for_a_while)
  forever;
end
```

Procedure 1. InitihalRequestServer

```
Procedure InitialRequestClient
begin
  this_multicast_socket.join(mcast_port);
  while my_node.up_node_is_not_connected
  begin
    this_multicast_socket.send(“request_to_join_in_the_group”);
    accept_first;
    wait_time( request_term);
  end;
end

Procedure accept_first
begin
  this_server_socket.start_receive_connection_at_port(recv_port);
  this_receive_socket←this_server_socket.accept;
  message←this_receive_socket.read;
  this_receive_socket.send_ack(“accepted”);
  my_node.connect_to_upper_node(message.“address”,  message.“port”);
  while enough_term_to_receive_the_message_from_all_nodes
  cobegin
    this_receive_socket←this_server_socket.accept;
    message←this_receive_socket.read;
    this_receive_socket_send_ack(“rejected”);
  coend
end
```

Procedure 2. InitialRequestClient

ここで accept_first の中では，返事を受け取るための TCP サーバソケットを準備し，そのソケットに，最初に接続のあったグループノードから返事（メッセージ）をうけとり，受理したことを返事し，その返事に書かれているアドレスのノード（そのグループノード）のポートに対して，自ノードを接続させる．このグループノードが親ノードとなる．それ以降に接続のあったグループノードに対しては，拒否を返事する．while enough_term_to_receive_the_message_from_all_nodes cobegin … coend は，すべての返事を受け取るのに十分な時間 cobegin…coend 間が並行実行されることを表す．

このアルゴリズムで 2 分木状の接続を行う場合，すべての接続処理は分散して行わ

れるため, 一か所に負荷が集中することはない. また, あるノードで, 上位ノードへの接続が切れた時, そのノードを根とする木のすべてのノードの InitialRequestServer の動作を停止させ, その根ノードでもう一度 InitialRequestClient の処理を実行することにより, 自動的に修復が行われる. 修復後, 停止していた InitialRequestServer を動作させる. この修復の結果, ２分木の平衡性はくずれる場合があるが, その後もできるだけ平衡２分木が構成されるよう, ノードのグループへの参加処理が行われる.

3. アルゴリズムの正当性の証明と理論的な実行時間

このアルゴリズムを実行する環境として, 以下を仮定する.

- InitialRequestServer の repeat ループ内の左右のどちらかの
  message←this_multicast_socket.receive_a_request_message;
  でメッセージ受信を待っているすべてのノードは, InitialRequestClient が送信する参加要求メッセージを p 回以内に必ず受け取る.
- 任意のノード間の TCP 接続が行われるとき, その接続は必ず成功し, 通信も確実に行われる. 複数の TCP 通信はお互いに他に影響を与えない(スイッチの利用を前提とする).
- InitialRequestServer と InitialRequestClient は実行中に停止することはない.

3.1 停止性の証明

ノード数を N とし, 最初は根ノードでのみ InitialRequestServer を起動し, 他のすべてのノードで InitialRequestClient を起動した場合, 有限時間内でアルゴリズムは停止する. アルゴリズムが停止しないと仮定すると, 永遠に InitialRequestClient が動作しているノードが存在することになる. InitialRequestClient はそれが動作している間, 参加要求メッセージの送信を繰り返し行う. 以下で述べるようにグループは必ず２分木状にノード間が接続され, その後, このノードで InitialRequestServer が起動するため, 必ず, InitialRequestServer が動作している葉ノードまたは子ノードを一つしか持たないノード(接続待機ノード)が存在する. 接続待機ノードでは, いつか必ず, 左または右のどちらかで,

message←this_multicast_socket.receive_a_request_message;

を実行する. InitialRequestClient から参加要求メッセージが p 回送信されると, 接続待機ノードのこの部分で必ずどれかのメッセージの受信が行われる. 参加要求メッセージを送信したノードは, 必ず, 接続待機ノードのどれかからの返信を最初に受信する. その後, 必ずこの参加要求ノードが接続待機ノードに接続され, InitialRequestClient の実行が終了し, このノードで InitialRequestServer が起動される. InitialRequestServer が動いているノードがある限り, 必ず, いつかは, InitialRequestClient を動作させているノードでその動作が停止する. InitialRequestServer が動いていない場合は, 根ノードで InitialRequestServer を最初に動かさなかった場合だけであり, それは前提と矛盾する. したがって, このアルゴリズムは必ず停止する.

3.2 ノード間接続が平衡２分木状に行われることの証明

ノードは根ノードから順番に１つずつグループに参加していくものとする. ノード数 N が 1 の場合, 根ノードのみが存在し, そこでは InitialRequestClient は起動されず, InitialRequestServer のみが起動される. 根ノードのみ存在する木は２分木である. i 番目のノードがグループに接続するとき, そのノードは i-1 番目までのノードで構成された木の葉または子が１つしかいない節にしか接続できない. したがって, この木の任意の節は, 子を持たないか, １つ持つか, ２つ持つかのどれかの場合しかない. また, 根ノードでは InitialRequestClient は起動されないため, 根が他のノードの子ノードに接続されることはない. また, グループに参加していないノードでは InitialRequestServer は動作していないため, 自分自身に接続することもない. したがってループは存在しない. また他のグループに未参加のノードに接続されることもない. グループの, このような節の接続構造は, ２分木である.

また, 新たにノードが加わる度に, そのノードは, 木の中で最も根に近く, かつ, 葉か, 子を１つ持つノードに接続される. 接続されてグループに参加したノードは, InitialRequestServer を起動する. これを繰り返して構成された木は, 根からすべての葉までの距離の差は高々１であり, 平衡２分木状の木となる.

3.3 理論的な実行時間

最初に N 個のノードは互いに接続されておらず, 1 個の根ノードで InitialRequestServer が起動され, N-1 個のノードで InitialRequestClient が起動されているとする. この状態から, N 個の節を持つ２分木が接続されるまでの理論的実行時間を考える.

ここで今までのものに加えて以下を仮定する.

- 複数の参加要求ノードが参加要求メッセージをそれぞれ送信したとき, １つのメッセージがどれかのノードに同時に届くときは他のノードにもそのメッセージが同時に届く. あとで述べるように, この条件は, 実行時間を短くする上で厳しい条件である. 届く順番は, メッセージが送信された順番とする.
- メッセージのノード間通信に必要な時間は無視する.
- グループノードから返信メッセージが参加要求ノードに送られ, 参加要求ノードからグループノードに受理か拒否の返信が行われるまでの時間を t_{ack} とする.
- グループノードから返信メッセージが参加要求ノードに送られ, 受理の場合, 参加要求ノードとグループノードが接続されるまでの時間を$t_{connect}$ とする.

- InitialRequestClient が参加要求メッセージを送信する間隔を，$t_{request\_term}$ とする．
- 以上以外の実行時間は無視する．

接続要求を受け付け可能なグループノードは，InitialRequestServer が起動されたのち，または参加要求メッセージを受信してその処理を行った後，左右のどちらかで，参加要求メッセージの受信を待つ．１つの参加要求ノードが参加要求メッセージの送信を開始した後，２分木を構成するグループの，高さ h のグループノードのどれかがそれを受け取ることができた場合，InitialRequestServer では，参加要求メッセージを受信後，kh の時間たってから返信を行う．ここで，根ノードは高さ 1 とする．その後，ノード間接続が行われる．したがって，参加要求メッセージを出力されてからそのグループノードに接続するまでの時間 T_{h_1} は

$$T_{h_1} \leq pt_{request\_term} + kh + t_{connect}$$

となる．

2 文木が 3 ノード以上で構成されていて，2 つの参加要求ノードが参加要求メッセージをほぼ同時に繰り返し送信している場合，このとき，すべての，参加要求を受け付け可能なグループノードは p 回以内で１つの参加要求ノードが送信した参加要求メッセージを同時に受け取る．この後，メッセージを受けっとったグループノードは，同時に kh の時間を待ち，返信を，このメッセージを送った参加要求ノードに送る．最初に返信メッセージを届けたノードは受理の返信を受け取り，参加要求ノードからそのノードに対して接続が行われる．次のグループノードは拒否の返信を受け取った後，もう一度，参加要求メッセージの受信を待つ．InitialRequestClient 側では，$t_{request\_term}$ 間隔で次の参加要求メッセージが送信されるため，次のグループノードが次の参加要求メッセージを受け取るまで，それが拒否メッセージを受け取ってから，最悪，$t_{request\_term}$ 待つことになる．従って，参加要求ノードの２つとも，２分木を構成するグループの，高さ h のグループノードの２つのグループノードに接続するとき，参加要求メッセージが出力されてから，２つの接続が完了するまでの時間 t_{h_2} は

$$T_{h_2} \leq pt_{request\_term} + kh + t_{ack} + pt_{request\_term} + kh + t_{connect}$$

となる．参加要求ノードからの接続が可能な，高さ h のグループノードが m 個あり，2m 個の参加要求ノードが参加要求メッセージ送信をほぼ同時に開始した後，2m 個すべてが接続を完了するまでの時間 T_{h_m} は

$$T_{h_m} \leq 2pmt_{request\_term} + 2mkh + (2m-1)t_{ack} + t_{connect}$$

である．ここで m は，

$$m \leq 2^{h-1}$$

である．従って T_{h_m} は以下を満たす．

$$T_{h_m} \leq p2^h t_{request\_term} + 2^h kh + (2^h - 1)t_{ack} + t_{connect}$$

h を N 個のノードで構成される平衡２分木の高さとしたとき，N-1 個のノードが参加するために必要な時間 T は

$$T \leq \sum_{h=1}^{\lfloor \log N \rfloor} \{p2^h t_{request\_term} + 2^h kh + (2^h - 1)t_{ack}\} + t_{connect}$$

となる．ここで，

$$N < 2^h,\ h \leq \log N + 1,\ \sum_{h=1}^{\lfloor \log N \rfloor} 2^h \leq N,\ \sum_{h=1}^{\lfloor \log N \rfloor} h2^h < (N+1)\log(N+1)$$

であるので，

$$T < pNt_{request\_term} + kN(\log N + 1) + (N-1)t_{ack} + t_{connect}$$

である．これは，$O(N \log N)$ となり，O(N)より大きくなる．しかしながら，multicast の前提を，１つのメッセージが同時に届かない（局所的に，近くのマルチキャストは，早く届く）とすると，並列処理が行われ，O(N)より早くなる可能性がある．

4. アルゴリズムの改良

3章で述べたアルゴリズムの場合，InitialRequestServer の左または右で待っているすべてのグループノードは，どれかの参加要求ノードが送信する参加要求メッセージを受信する度に，それぞれが同時にkh時間以上待つため並列性を失っている．このアルゴリズムに並列性を加え，高速化する手段の一つは，グループノードすべてが参加要求メッセージを受信後kh時間待たないようにすることである．これを実現するため，以下のようにアルゴリズムを改良する．なお，ここでは，複数の参加要求ノードから一斉に送信された参加要求メッセージが届く場合，すべてのグループノードに順番に届くが，その時間差は0と仮定する．

それぞれの参加要求ノードにおいて，InitialRequestClient が参加要求メッセージを送信する度に，乱数等を用いてノード間でできるだけぶつからないような数をメッセージにつける．グループノードがそのメッセージを受信したとき，その数を２進数で表したものの下の桁から，それが動いているノードの高さ(根からの距離)の桁 $h-1$ だけ見て，その値が自分のidの，同じ桁の部分と一致したときだけ返信し，それ以外は次のメッセージを受け取りなおすようにする．ここで，グループノードのid は，自分の親ノードから見て左側であれば，親ノードのidを２倍したものとし，自分の親ノードからみて右側であれば，親ノードのidを２倍したものに1を加えたものとする．根ノードのidは1とする．参加要求メッセージに付加される数によって，待ったり，待たずにすぐに次のメッセージを受け取ったりすることにより，並列性が加わり，すべての

ノードがグループに参加するまでの時間が早くなる．すべてのグループノードは参加要求メッセージをq回以内で受け取るとした場合，高さhの葉がm個並んだ時の並列数は 2^h であるため，

$$T_{h_m} \leq q\frac{p2^h t_{request\_term} + 2^h kh + (2^h - 1)t_{ack}}{2^h} + t_{connect}$$

$$T_{h_m} \leq pqt_{request\_term} + qkh + q\left(1 - \frac{1}{2^h}\right)t_{ack} + t_{connect}$$

となる．N-1 個のノードが参加するために必要な時間 T は

$$T \leq \sum_{h=1}^{\lfloor \log N \rfloor} q\left\{pt_{request\_term} + kh + \left(1 - \frac{1}{2^h}\right)t_{ack}\right\} + t_{connect}$$

となる．

$$\sum_{h=1}^{\lfloor \log N \rfloor} h = \frac{\lfloor \log N \rfloor(\lfloor \log N \rfloor + 1)}{2}$$

であり，

$$\sum_{h=1}^{\lfloor \log N \rfloor} \frac{1}{2^h} \leq 1 - \frac{1}{N}$$

であるので，

$$T \leq p\{q\lfloor \log N \rfloor t_{request\_term} + k\frac{\lfloor \log N \rfloor^2 + \lfloor \log N \rfloor}{2} + (\lfloor \log N \rfloor - 1 + \frac{1}{N})t_{ack}\} + t_{connect}$$

である．したがって T は $O((\log N)^2)$ となり，Nがある程度以上大きくなると O(N) より小さくなる．

5. 試験的に実装したシステムの評価

本アルゴリズムを SOLAR-CATS に実装し，40 人程度の授業や８人程度のゼミで利用している．各ノードでグループマネージャを指定せずに接続を開始することが可能になり，より簡単に利用できるようになった．また，従来は，ノード数が 20 を超えると木の構成に失敗する場合が多く発生し，不安定であった．これに対して新しいアルゴリズムは以前と比較して，安定して木を構成することができている．しかしながらノード数が多い状態で一斉に接続を行うとグループに参加できないノードも存在している．現在，この原因を追及している．

6. 関連研究

信頼性のある IP マルチキャストについては，TMTP(Tree-based Multicast Transport Protocol)[1]，RMTP(Reliable Multicast Transport Protocol)[3]，Reliable Multicast Tree construction algorithm[7] などの研究があるが，これらは平衡木の構成方法については述べていない．
「自律的に平衡 2 分木を保とうとする構造型 P2P システム [6]」は，平衡２分木を構成する手法について述べており，マルチキャストが使えない環境でも利用可能であるが，根ノード以外のノードは，根ノードの IP アドレスを知る必要がある．また，根ノードに最初の問い合わせが集中する．

7. おわりに

ブロードキャストが可能な環境においてノードの平衡２分木状接続を自律的に行う分散アルゴリズムとその試験的実装について述べた．今後，このアルゴリズムの性質の実証を行ったり，バグを修正したりする予定である．大量の端末の管理システムへの応用も検討している．

参考文献

1) R. Yavatkar, et. al. “A Reliable Dissemination Protocol for Interactive Collaborative Applications”, Proc. ACM Multimedia, 1995.
2) Takayuki Hirahara, Takashi Yamanoue, Hiroyuki Anzai and Itsujirou Arita, "SENDING AN IMAGE TO A LARGE NUMBER OF NODES IN SHORT TIME USING TCP", Proceedings of the ICME2000, IEEE International Conference on Multimedia and Expo, pp.987-990, New York City, USA, July 30-Aug.2 ,2000.
3) Paul, S., Sabnani, K.K., Lin, J.C.-H., Bhattacharyya, S., “Reliable Multicast Transport Protocol (RMTP)”, IEEE Journal on Selected Areas in Communications, Vol. 15, Issue 3, pp.407-421 2002.
4) 山之上 卓: P2P 技術を利用した分散システム上の実時間操作共有システム，情報処理学会論文誌, vol.46, No.2, p.392-402, 2005.
5) Yamanoue, T., “Sharing the Same Operation with a Large Number of Users Using P2P”, The 3rd International Conference on Information Technology and Applications (ICITA'05), IEEE CS Press, pp.85-88, July 2005.
6) 山之上卓, 中森 武, "自律的に平衡２分木を保とうとする構造型 P2P システム", 電子情報通信学会技術研究報告, IA2007-19, pp.55-60, Jul. 2007.
7) Choonsung Rhee, Jungwook Song, Eujiun Kim, Sunyoung Han, “Reliable Multicast Tree Construction Algorithm”, Mobility ’08 Proceedings of the International Conference on Mobile Technology, Applications & Systems; 1-5, Ilan, Taiwan, 10-12 Sep. 2008.
8) Takashi Yamanoue, “A Casual Teaching Tool for Large Size Computer Laboratories and Small Size Seminar Classes”, Proceedings of the 37th annual ACM SIGUCCS conference on User services, pp.211-216, St.Louis, Missouri, US.. 11-14 Oct. 2009.