安定したストリーム配信を実現するオーバレイマルチキャストプロトコルの設計とPlanetLab上での実証実験

Thilmee M. Baduge 池田 和史 山口 弘純 東野 輝夫
大阪大学 大学院情報科学研究科
{thilmee,k-ikeda,h-yamagu,higashino}@ist.osaka-u.ac.jp

本稿ではインタラクティブアプリケーションにおける動画配信などの用途を想定したオーバレイマルチキャストプロトコルを提案する．そのようなプロトコルへの性能要求として，インタラクティブ性を実現するために最大遅延がある閾値以下に制約できること，各ノードの動画中継負荷を抑えるためにノードのリンク数（次数）を制限可能であること，ならびにノード離脱による動画中断が生じにくいこと，などが挙げられる．提案プロトコルではこれらの実現を設計目標とし，複数のソースノードからの遅延制約，ならびに各ノードの次数制約を満たしながら，全ノードの受信安定度総和をなるべく高くするオーバレイマルチキャスト木を構築する．提案手法では，各ノードに対し，ノード安定度係数（そのノードのセッション滞留時間，転送能力等）というトポロジ非依存メトリックを定義し，あるノードについてそのすべての上流ノードの安定度の積をそのノードの受信安定度と定義する．シミュレーションとPlanetLab上の実験による性能評価により提案プロトコルの有効性を確認した．

An Application Layer Multicast Protocol for Stable Streaming and its Evaluation on PlanetLab

Thilmee M. Baduge Kazushi Ikeda Hirozumi Yamaguchi Teruo Higashino
Graduate School of Information Science and Technology, Osaka University

In this manuscript, we propose a new protocol designed for inter-active multimedia streaming applications. The protocol considers the heterogeneity of P2P multicast end-hosts and tries to minimize the negative impact (data outage) of end-hosts' un-announced departures. For this purpose, it concentrates on end-hosts' reliability (lifetime for instance) and constructs a shared tree called ms-DDBMSST (multiple-source Degree and Delay Bounded Maximum Stability Spanning Tree) as an overlay network that involves all the participants of the application, in a distributed manner. For a given set of nodes where some of them are senders, ms-DDBMSST is a spanning tree where the receive path stability of the entire tree is maximized while satisfying the delay-from-source constraint and degree constraint for each node. We believe that this is the first approach that defines ms-DDBMSST construction problem and presents a distributed protocol for the purpose. Our performance evaluation is based on experiments in both simulated networks and PlanetLab that strongly shows the efficiency and usefulness of the proposed protocol.

1 まえがき

インターネットの普及により，大規模なマルチメディアアプリケーション（例えば数千人規模で行う動画配信やゲームなど）への需要が増加しつつある．そのようなアプリケーションにおいては，各ユーザへの情報配信を効率良く行うため，ユーザ間のマルチキャスト配信を行なうことが望ましい．これに対し，ユーザ間のユニキャスト接続からなる論理的なオーバレイネットワーク上でマルチキャスト木を構築するプロトコル（アプリケーション層マルチキャスト（ALM）プロトコル）がこれまでに多く提案されてきている（文献[2, 3, 4, 5, 6, 7, 8]）．

ALMでは各ユーザが他のユーザへのデータ中継を行うルータ的な役割も果たすため，あるユーザが離脱するとそのユーザからデータ中継を受けていたユーザへの情報供給が中断される．従って，動画配信アプリケーションなどにALMを適用する場合は，ユーザの離脱が他のユーザへ及ぼす影響をなるべく小さくするようにユーザノードを配置し，より安定性の高い（各ノードの受信経路切断率が低い）マルチキャスト木を実現することが重要である．さらに実時間インタラクション性を持つアプリケーション（例えば講義システムなど）においては，ユーザ（視聴者）からの実時間フィードバックが必要な場合もあり，この観点から，各ユーザへの配信遅延をある閾値以下に抑えることが重要である．また，オーバレイネットワークにおけるマルチキャスト木の各ノードの全てのリンクはそのノードの物理ネットワークインタフェースを利用する．したがって，そのインターフェースの容量などを考慮し，ノードに接続するリンク数（これを以下次数とよぶ）をある一定値以下に抑えることが望ましい．

本稿では，このような設計要求に基づき，データ配信を行うと予想される複数ノード（複数ソース）からの遅延制

約，ならびに各ノードの次数制約を満たしながら，全ノードの受信安定度総和をなるべく高くするオーバレイマルチキャスト木を構築する．具体的には，ソースノード集合 S を含むユーザノードの集合 V，遅延がわかっているユーザ間ユニキャスト（無向辺）集合 E からなる論理的な完全グラフネットワーク（オーバレイネットワーク）$G=(V,E)$ 上で，各ノード $v \in V$ に与えられた次数制約 $d(v)$ と S からの遅延制約 D_{max} を満たし，かつ全ノードの受信安定度総和（木の総受信安定度）がなるべく高い G の被覆木 T を構築するプロトコルを提案する．ここで，各ノード x には安定度係数（例えば，そのノードがどの程度の期間セッションに留まりデータ中継に貢献できるかを表すセッション滞留時間や転送能力等）$sc(x)$ が与えられているとし，あるノードについてそのすべての上流ノード（各ソースノードからの経路上のノード）の安定度の積をノードの受信安定度と定義する．したがって，T の総受信安定度 $stab_T$ は以下で定義できる．

$$stab_T = \sum_{(s,r)\in S\times R} stab_T(s,r)$$

$$stab_T(s,r) = \begin{cases} 1 & (s=r) \\ sc(r')\cdot stab_T(s,r') & (s\neq r) \end{cases}$$

ただし，r' は s から r への経路上の r の隣接ノードである．

2 提案プロトコルの概要

木 T におけるのソースノードからの遅延の最大値を $depth_T$ で表すと，一般的に，$stab_T$ は各ノードの安定度係数と次数に，$depth_T$ は各ノード間のリンク遅延と各ノードの次数に依存する．これはノードの安定度係数とノード間のリンク遅延は，順に $stab_T$，および $depth_T$ を最適化する上で重要な役割を果たすことを意味する．一方，ノードの滞留時間とリンク遅延の間に相関がないことは文献 [9] 等で知られており，同様にノードの転送能力とリンク遅延の間にも相関がないと考えられ，結果的に $stab_T$ と $depth_T$ は相関のないメトリックであると言える．よって，従来の遅延最小木構築アルゴリズム [2] の目的関数を単純にこれらのメトリックスの線形和で置き換えるといった手法は利用できない．

このような観点から，それぞれ $stab_T$ と $depth_T$ の最適化を分離できるよう，下記の2つのステップからなる木構築手法を適用する．まず (a) $depth_T$ を最適化するための初期木構築を行う．次に (b) $stab_T$ を最適化するための木改善処理を行う．$depth_T$ が D_{max} により制限されていることを考慮し，初期木構築に既存の遅延最小木構築アルゴリズム [2] を適用する．これにより，$depth_T$ を極力小さくすることができ，その値が与えられた遅延制約 D_{max} より小さくなる可能性が高いと考えられる．次に，木改善処理を開始し，$stab_T$ が改善するようにノードが移動する．この処理は，$depth_T$ に対して最適化されていた初期木を $stab_T$ に対して最適化し直す目的で実行されるため，本処理によって $depth_T$ が増加してしまう可能性が高いことが容易にわかる．よって，本改善処理における各ノードの移動は $depth_T$ への遅延制約が満たされるように行われることが重要である．なお，D_{max} に小さすぎる値を与えると初期木における $depth_T$ が D_{max} より大きくなる可能性があり，この場合には木の改善処理は実行されない．従って，ここでは D_{max} は実際のアプリケーションを想定して与えられ，初期木における $depth_T$ より大きいことを仮定する．

ここで一般に，多くの参加ノードはセッション開始時刻より前に到着すると仮定できるため，初期木を集中アルゴリズムによって構築することは現実的であると言える．一方，木改善に関してはセッション開始後に実行されるため，分散処理が要求され，これを分散処理により実現する．なお，セッション開始後のノード参加，および離脱への配慮も重要であるが，これらに対する処理を [12] 等の既存手法により実現し，本稿では主に初期木構築処理と木改善処理に関して議論する．木改善処理の概要を以下に述べる．あるノード x が，x をルートノードとする部分木の総受信安定度を増加させるため，x から下流へ K ホップ以内のノードからなる部分木（これを $RefT_{x,K}$ で表す）内でノードの組み換え操作（再構築）を行う．ただし，この再構築はそれに伴う再構築後の木全体の新しい遅延が S からの遅延制約 D_{max} を越えない場合のみ行う．この改善処理を葉ノードから親ノードに向けてボトムアップ的に行うことにより，各ノードの次数制約，ならびに S からの遅延制約を満たしながら，木の総受信安定度をなるべく高くする．

3 プロトコル設計

3.1 初期木構築処理

初期木はソースノード集合 S からの最大遅延（$depth_T$）が最小になるように構築される．全てのソースノードがお互いのオーバレイリンク遅延が比較的に短くなるように配置されている場合には，それらを単一グループとし，そのグループをルートノードとみなすことができ，単純に従来の最小遅延被覆木構築アルゴリズム [2] が適用可能である．しかし，一般的にはソースノードがお互いに離れている場合もあるため，それらを単一グループとして扱うと余分な遅延が発生し，$depth_T$ が大きくなる可能性がある．よって，従来の最小遅延被覆木構築アルゴリズムを複数ソースノード用に拡張した下記に示すアルゴリズムを適用する．ここで，T_{node}，T_{edge} は順に初期木のノードと辺，R は T_{node} 以外のノードを表し，まず S の中から一つのソースノード（s_0）をランダムに選択し，それを初期木とする．

01: $T_{node} \Leftarrow \{s_0\}$; $T_{edge} \Leftarrow \emptyset$, $S \Leftarrow s_0$, $R \Leftarrow R - s_0$;
02: while ($R \neq \emptyset$)
03: find $r \in R \backslash T_{node}$ and $r' \in T_{node}$ that minimize $depth_{(T_{node}\cup\{r\},T_{edge}\cup\{(r',r)\})}$
04: $T_{node} \Leftarrow T_{node} \cup \{r\}$;
05: $T_{edge} \Leftarrow T_{edge} \cup \{(r',r)\}$;
06: $R \Leftarrow R\backslash\{r\}$;
07: if (r is a source node) then $S \Leftarrow S \cup \{r\}$;
08: endwhile;
09: return (T_{node}, T_{edge});

3.2 安定度改善処理

3.2.1 情報収集

次に，ルートノード（s_0）は情報メッセージとよばれるメッセージを子ノードに送信し，初期木の構築が終了したことを通知する．情報メッセージには，(i) s_0 から他のソースノードへの各パス上のノード ID のリスト，(ii) s_0 から他のソースノードへの遅延値，(iii) 最大遅延制約値（D_{max}），および (iv) 後述する安定度改善処理の対象となる部分木の高さ（K）が含まれる．D_{max} と K はアプリケーションに依存する定数であり，ソースノードにより指定されるものとする．情報メッセージを受信した各ノード v は，(v) ソースノード S から自身への遅延（以下 $D(s_0,v)$ で表す），(vi) ソースノード S から自身へのホップ数（以下 $H(s_0,v)$ で表す），をそのメッセージに追加して自身の子ノードに転送する．これにより，各ノードは (i) から (vi) までの情報を獲得できる．

また，情報メッセージを受信することにより，各ノード v は，同時に隣接ノードへの ID 割り当ても行う．ルートノード r は自身にノード ID 0 を割り当て，隣接ノードに

は $1,\ldots,d(r)$ ($d(x)$ はノード x の次数を表す) のノード ID を情報メッセージにより指定する．同様に，あるノード v が隣接ノードからノード ID i を割り当てる情報メッセージを受信した場合，ノード v はその他の隣接ノードに $i\times d_{max}+1,\ldots,i\times d_{max}+d(v)-1$ のノード ID を割り当て，情報メッセージを送信する．ここで d_{max} は全ノードの次数制約の最大値を示す．これにより，各ノードに一意な ID が割り当てられる．

3.2.2 ノード入れ換え

各ノードが自身の子ノードから入れ換え候補ノードを選択する[1]．一般的に，ノード u は部分木 $RefT_{u,K}$ の総受信安定度が増加する場合にのみ，子孫ノード w を自身との入れ換え候補として選択する．u，w の入れ換えによって得られる部分木の総受信安定度増分は下記のように表現できる．

$$\Delta_{u,w}=\begin{cases}\sum_{n=1}^{d(u)}\Big(sc(w)-sc(u)\Big)\cdot stab_{T_{x_n}}+\\ \sum_{n=d(u)+1}^{d(w)}\Big(sc(w)-sc(u)\cdot R\cdot sc(w)\Big)\cdot stab_{T_{y_n}}\\ \qquad(\text{if } d(w)\geq d(u),\ \text{Fig.1(b)})\\ \sum_{n=1}^{d(w)}\Big(sc(w)-sc(u)\Big)\cdot stab_{T_{x_n}}+\\ \sum_{n=d(w)+1}^{d(u)}\Big(sc(u)\cdot R\cdot sc(w)-sc(w)\Big)\cdot stab_{T_{x_n}}\\ \qquad(\text{if } d(w)<d(u),\ \text{Fig. 1(c)})\end{cases}$$

ここで，$R=sc(v)\cdots sc(x_1)$ (v が w の親ノードである) であり，T_{x_1} は例外的に子孫部分木である $T_{y_1}\cdots T_{y_{d(u)}}$ 以外の部分木を表すものとする．

また，$RefT_{u,K}$ 内のノード入れ換えを木全体の遅延制約が満たされるように実行する必要がある．このために各部分木に遅延増分上限というものを割り当て，ノード入れ換えに伴う部分木の遅延上昇が遅延増分上限以内で納まるような交換候補を選択することで木全体の遅延制約を満たす．ここで，ノード入れ換えに関る全てのノード (x と記す) を考慮し，各 x に割り当てられる遅延増分上限 $\delta(x)_{max}$ を下記の式で求める．例えば，図 2(a) の u，w_j の入れ換えにおいては，u, w_j (u の子ノード)，および z (w_j の子ノード) を x で表すことになる．

$$\delta(x)_{max}=\{D_{max}-D(s_f,x)-depth(T_x)\}/H(s_f,x)\qquad(x\neq z)$$

$$\delta(z)_{max}=\delta(w_j)_{max}$$

ここで，$depth(T_x)$ は x をルートノードとする部分木の最大遅延，および $H(s_f,x)$ は s_f から x への経路上のホップ数である．この式は x 上で利用可能な最大遅延増分を s_f までの経路上の各ノードへ均等に分割するものである．

最後に，ノード u がこれらの計算に基づき，各 x の遅延増分 $\delta(x)$ が $\delta(x)_{max}$ 以内であり，かつ $\Delta_{u,w}$ を最大にする子ノード w を入れ換え候補として選択する．

$$\delta(x)=D(s_f,x)_{new}-D(s_f,x)_{prev}$$

$$D(s_f,w_j)_{new}=D(s_f,u)_{prev}-D(v,u)+D(v,w_j)$$

$$D(s_f,u)_{new}=D(s_f,w_j)_{new}+D(w_j,u)$$

$$D(s_f,w)_{new}=\begin{cases}D(s_f,w_j)_{new}+D(w_j,w)\ \text{if } d(u)\geq d(w_j)\\ \qquad(w\text{ が }u\text{ に付随して移動しない})\\ D(s_f,u)_{new}+D(u,w)\\ \qquad\text{else }(w\text{ が }u\text{ に付随して移動する})\end{cases}$$

[1] 本稿では $K=1$ の場合のみを議論し，$K>1$ については今後の課題とする．

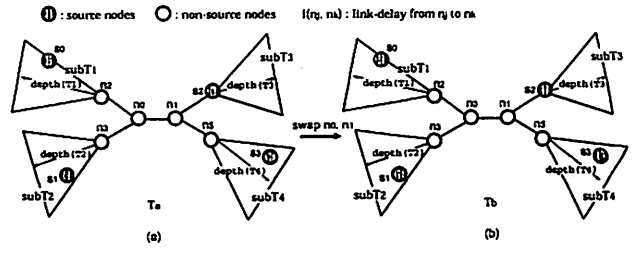

図 3: 複数ソースへの対応: ソースノード間の経路上のノードの入れ換え (a) 交換前 (b) 交換後

$$D(s_f,z)_{new}=\begin{cases}D(s_f,u)_{new}+D(u,z)\ \text{if } d(w_j)\geq d(u)\\ \qquad(z\text{ が }w_j\text{ に付随して移動しない})\\ D(s_f,w_j)_{new}+D(w_j,z)\\ \qquad\text{else }(z\text{ が }w_j\text{ に付随して移動する})\end{cases}$$

ノード入れ換えが終了した時点で，w_j が自身の新しい親ノード v に下記の情報を含めた収集メッセージを送信する．(i) $sc(w_j)$, (ii) $depth(T_{w_j})$ および，(iii) $D(w_j,v)$．ノード入れ換えが実現されなかった場合には，同様なメッセージが u によって v へ送信される．$RefT_{v,K}$ の改善処理は v が全ての子ノードから収集メッセージを受信した時点で開始される．また，この改善処理は木のルートノード s_0 に至るまで繰り返される．

3.2.3 複数ソースへの対応

入れ換えに関るノードがソースノード間の経路上に位置していない (入れ換え対象となるノードを根とする部分木内にソースノードが存在しない) 限り，複数ソースノードの存在を意識せずに上記のノード入れ換え処理が実行できる．これは，このような部分木内のノード入れ換えに対し，全てのソースノードは部分木からみて上流にいるため，あたかも一つのソースであると見なすことができるからである．しかし，改善処理が上流に伝搬されるにしたがい，この条件を満たさないノードが現われるため，特別な処理が必要となる．本稿ではこのようなノードを不規則ノードをよぶ．

以降，このようなノードに対する入れ換え処理を図 3 をもとに述べる．ノード n_0，n_1 はそれらを交換することで，木全体の総受信安定度が上昇し，かつ遅延制約が満たされる時にのみ交換する．なお，ここで得られる総受信安定度増分 Δ_{n_0,n_1} は以下のように計算できる．

$$\Delta_{n_0,n_1}=stab_{T_b}-stab_{T_a}$$

$$stab_{T_a}=\sum_{0\leq k\leq 4}\sum_{r\in R}stab_{T_a}(s_k,r)$$

$$\sum_{r\in R}stab_{T_a}(s_0,r)=sc(s_0)\cdot sc(n_2)\cdot sc(n_1)\cdot sc(n_0)\cdot(stab_{subT_2}+sc(n_1)\cdot(stab_{subT_3}+stab_{subT_4}))$$

$\sum_{r\in R}stab_{T_a}(s_i,r)$ $(i=1,2,3)$ も同様に表現でき，それによって $stab_{T_a}$ が求まる．$stab_{T_b}$ も同様な方法で計算でき，これらを用いて求まる Δ_{n_0,n_1} によって n_0, n_1 を入れ換えるかどうかを判断できる．ただし，この入れ換えは T_b のソースからの最大遅延増分が遅延制約以内であるかどうかを検討してから行うものであり，この値は $depth(T_{subT_k})$ $(k=0,1,2,3)$ と n_0，n_1 周辺のリンク遅延を用いることで容易に計算できる．

ここで，不規則ノードは最大遅延と受信安定度の面からお互いに影響し合うため，それらの入れ換えを異なる場所で同時に行うのは不可能であることに注目されたい．よっ

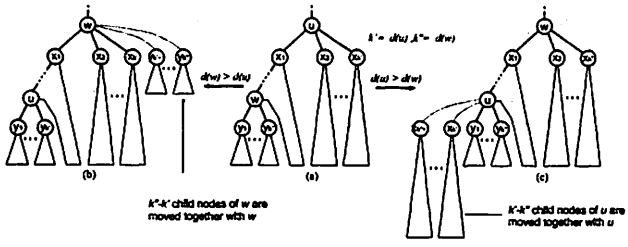

図 1: 入れ換え候補選択方法の概要: (a) 初期木 (b) $d(w) > d(u)$ 時に u, w を入れ換え後の木 (c) $d(w) < d(u)$ 時に u, w を入れ換え後の木

て，改善処理が不規則ノード上で行われる前に，そのノードがルートノードに改善処理許可を得る仕組みを導入し，同時実行を避けるようにする．ルートノードは，全ての通常ノード（非不規則ノード）の改善処理が終了した時点から，これら不規則ノードの改善処理を順に許可する．各不規則ノードは改善処理の開始前と終了後に自身に関する情報をルートノードに送信することで，ルートノードは改善処理に必要な情報を提供することが可能になる．[2]．

4 実験結果

4.1 シミュレーション実験

構築されたマルチキャスト木上でストリーム配信を行った時に各ノードがどれだけ安定してそれを受信できるかを表す指標である総受信安定度，$stab_T$（$stab_T$ が高いことをノードの平均受信安定度が高いことを意味する）を用いて提案プロトコルの評価を行った．シミュレーション実験は最大 1000 オーバレイノードを用いて行った．

以下シミュレーション設定を示す：ノード間のオーバレイリンク遅延（フルメッシュ）を $\mu = 100[ms]$, $\sigma = 20[ms]$ (i.e. $N(100, 20^2)$) となる標準正規分布に従い，ノードの次数をパレート分布 [10] に従うものとする．このために GNU 科学演算ライブラリ（GSL：GNU Scientific Library）で提供されている分布を用いた．ここでの分布は $p(x) = (a*b^a)/x^{(a+1)}$ $(x \geq b)$ で定義されており，$a = 0.6$ および $b = 20$ に設定した．実験においてノード安定度係数としてノード滞留時間を利用し，$a = 1.2$ および $b = 1$ に設定したパレート分布を適用した．

[最大遅延制約による影響] 提案プロトコルの最大遅延制約（D_{max}）による影響を調べた．本プロトコルが与えられた D_{max} 以内で $stab_T$ を改善するため，D_{max} が小さいと得られる $stab_T$ の改善も小さくなる．表 1 にこの様子を示す．D_{max} が上昇するにしたがい $stab_T$ も上昇していることがわかる．また，表 1 によるとノード数が増えるにしたがい $stab_T$ が下がっていることがわかる．これは，一般的にノード数に比例して初期木の最大遅延制約の数も増えるため，$stab_T$ の改善に利用できる遅延の余裕が少くなるためと考えられる．

[2] 一般的に不規則ノードの数は少いため，ここでのロック機構は長時間を要するものではない．

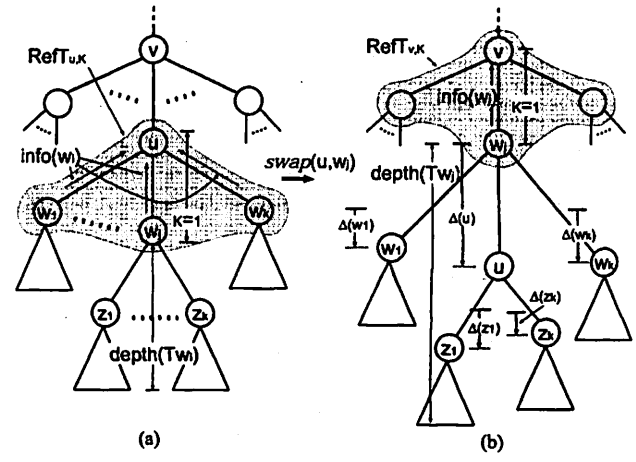

図 2: $RefT_{u,K}$ の改善の様子 (a) 改善前 (b) 改善後

表 1: 異なる遅延制約における提案プロトコルの振舞い

ノード数		100	200	500	1000
D_{max}= 500[ms]	$depth_T$[ms]	480	472	481	463
	$stab_T$	0.46	0.46	0.38	0.34
D_{max}= 1[s]	$depth_T$[ms]	751	879	901	934
	$stab_T$	0.55	0.53	0.48	0.43
D_{max}= 1.5[s]	$depth_T$[ms]	1207	1324	1372	1402
	$stab_T$	0.62	0.62	0.58	0.52

表 2: ノード入れ換え手法の性能比較

	$stab_T$	$depth_T$[ms]	リンク張替数
提案	0.64	1238	843
$s \cdot d$ 優先	0.62	1165	818
s 優先	0.61	1330	827

表 3: 異なるソースノード数におけるプロトコルの振舞い

ソース数	1	2	3	4	5
$stab_T$	0.65	0.63	0.62	0.58	0.56
$depth_T$[ms]	1427	1398	1390	1452	1421

[ノード入れ換え手法による影響] 提案プロトコルの性能を従来手法である $s \cdot d$ 優先，および s 優先のノード入れ換え方法と比較した．これらの既存手法では順に $s \cdot d$ 乗算または s が高いノードを無条件に木の上流に移動させる．これに対し，提案手法はある下流ノードを木全体の総受信安定度が上昇する場合にのみ上流に移動させる．$s \cdot d$ 優先，および s 優先の手法は [10]，[11] 等のノード滞留時間に触れている既存研究で利用されている．ここで，提案プロトコルにおいて D_{max} を十分に大きく（=2[s]）することで自由にノード移動を可能にした上で，さらに，ソースノード数を 1 にすることで，遅延制約，および複数ソースを扱ってない既存手法と同じ環境を実現した．表 2 に実験結果を示しており，提案プロトコルはより高い $stab_T$ を実現していることがわかる．

[ソースノード数による影響] ソースノード数によるプロトコルへの影響を評価した結果（異なるソースノード数に対する $stab_T$，及び $depth_T$ の値）を表 3 に示す．D_{max} は 1.5[s] に，オーバレイノード数は 500 に設定した．これらの結果からソースノード数が増加するにしたがい，$stab_T$ は下がる傾向にあることがわかる．これは，ソースノード

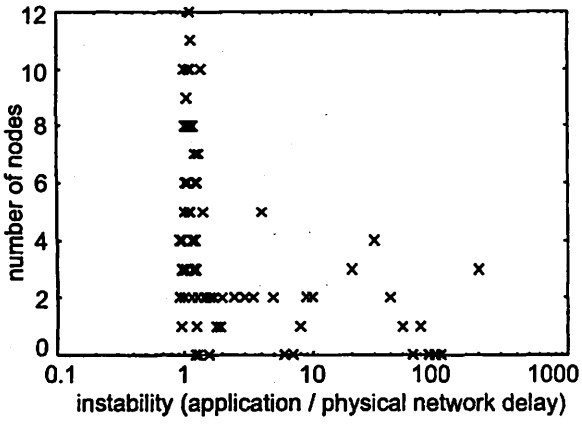

図 4: PlanetLab ノードの不安定度分布

数の増加に伴い，各ノードの移動時の遅延制約がより厳しくなるからである．

4.2 PlanetLab 上での実験

提案プロトコルの性能を実環境である PlanetLab 上での実験により評価した．実験には我々の研究グループが提案しているミドルウェア [13] を利用した．このミドルウェアは様々な ALM プロトコルの実装および PlanetLab 上での性能評価実験を支援する．提案プロトコルとグリーディな遅延最小木を構築するアルゴリズムをミドルウェア上に実装し，マルチメディアストリーミング配信時の性能を比較した．

4.2.1 実験環境

実験環境の詳細を以下に示す．PlanetLab には世界各地から端末が提供されており，実験に使用した端末は各地域から提供されている端末数に応じて均一に選んだ．端末数:320，端末性能：Pentium 3 (1.2GHz)〜Pentium 4 (3.4GHz) の CPU 能力，および 512MB〜3.6GB の RAM．OS: Linux OS version 2.6.12-1.1398_FC4.5.planetlab，その他:JRE1.6 (Java version) とした．

PlanetLab ノードの不安定さを評価するために，ping コマンドを実行したときの OS レベルでの RTT(round trip time) と Java プログラム上で ping アプリケーションを実行したときのアプリケーションレベルでの RTT を比較した．OS レベルでの ping は他のアプリケーションよりも優先的に実行され，ネットワーク遅延のみを評価するのに対し，アプリケーションレベルでの ping はネットワーク遅延に加え，ノードの状態やパケットの生成，受信イベントの待ち時間など，実際にアプリケーションでパケットを送信した際に遅延を発生させる様々な要因が含まれる．ここでは RTT(Application level)/RTT(OS level) を不安定度とし，PlanetLab 上の各ノードにおいて，他の全ノードからの不安定度の平均を評価した．使用した PlanetLab ノードの不安定度の分布を図 4 に示す．この図から半数以上のノードは不安定度は 0.9-1.2 に含まれるのに対し，一部のノードは非常に大きな不安定度をもっていることがわかる．以降では不安定度が 1.5 以上のノードを不安定なノードとして扱う．

4.2.2 実験シナリオ

提案プロトコルと遅延最小木を構築するグリーディアルゴリズムについて，ストリーミング配信時の平均ジッタと平均利用帯域を (1) トポロジサイズが変化した場合，(2) 全体に占める不安定なノードの割合が変化した場合について，それぞれ評価した．

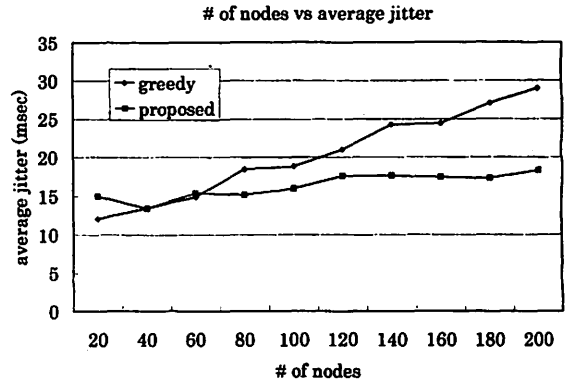

図 5: ノード数とストリーミングのジッタ

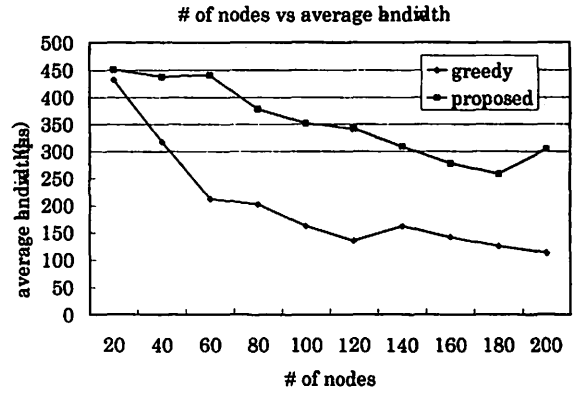

図 6: ノード数と各ノードの平均利用帯域

実験 (1) のシナリオを示す．始めに 20 ノードをプロトコルに従った方法でセッションに参加させ，トポロジを形成する．始めに参加したノードをソースノードとし，ソースノードから 500kbps のストリーミングを配信し (マルチメディア配信アプリケーションなどを想定)，プロトコルに従ってストリーミングをマルチキャストし，全ノードへ配信すると同時に，各ノードでのストリーミングのジッタと利用帯域を評価する．続いて，20 ノードを新たに参加させ，40 ノードが参加している状態を作り，同様にソースノードからストリーミングを配信する．以下同様に，200 ノードまでストリーミングの配信性能を評価した．

実験 (2) ではノード数を 200 に固定し，その中で図 4 から求められた不安定なノードの占める割合を 0%〜25%まで 5%ずつ変化させ，各割合でプロトコルに従ってトポロジを構築し，ソースノードから 500kbps のストリーミングを配信し，ジッタとビットレートを評価した．

それぞれのプロトコルにおいて PlanetLab 上で受信したストリームの品質を http://www-higashi.ist.osaka-u.ac.jp/software/stable_multicast.html より確認できる．

4.2.3 実験結果

図 5，および 図 6 に各プロトコルにおける平均ジッタと平均利用帯域を示す．グリーディアルゴリズムにおいてノード数に比例してジッタの上昇，および帯域の低下が激しくなっているのに対し，提案プロトコルではそれらの振舞いがより緩やかになっていることが容易にわかる．グリーディ手法ではソースノード近くに存在し得る不安定なノードが木全体の性能を悪化させているが，これに対して提案プロトコルではこのような不安定なノードを木の比較的に下流の位置に移動させるため，よりよいストリーミングを提供

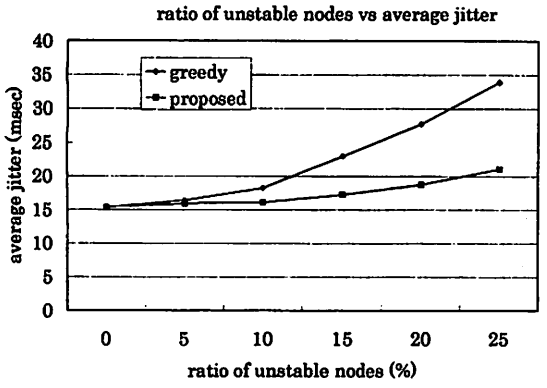

図 7: 不安定ノードの割合とストリーミングのジッタ

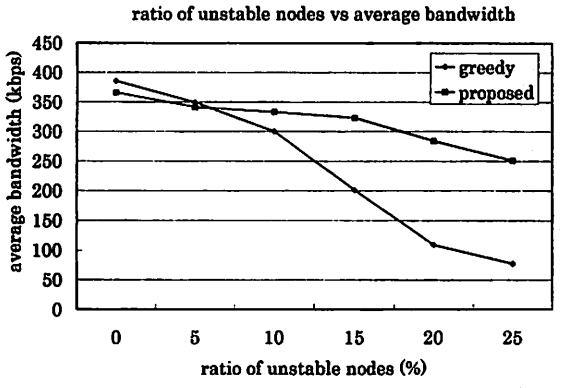

図 8: 不安定ノードの割合と各ノードの平均利用帯域

できている．さらに，図 7，および図 8 より，グリーディプロトコルが不安定ノードの割り合いに大きく影響され，ストリーミングの品質が落ちていることがわかる．結果的に提案プロトコルは，ノードの不安定係数を考慮することでより高品質なストリーミングを実現していると言える．

5 まとめ

本稿では，エンドユーザ間のユニキャスト接続からなる論理的なオーバレイネットワーク上でマルチキャスト木（被覆木）を構築するプロトコルを提案した．提案手法では，インタラクティブ性を実現するために最大遅延がある程度抑制されること，各ノードの動画中継負荷を抑えるためにノードのリンク数を制限可能であること，ならびにノード離脱による動画中継が生じにくいこと，を考慮し，複数のソースノードからの遅延制約，ならびに各ノードの次数制約を満たしながら，全ノードの総受信安定度をなるべく高くするオーバレイマルチキャスト木を構築する．シミュレーション実験により，提案プロトコルは従来手法より高い総受信安定度を実現可能であることが確認できた．さらに，PlanetLab 上の実験によって本プロトコルは遅延制約のみを考慮する既存グリーディプロトコルより高品質なストリーミングを実現していることを確認した．それぞれのプロトコルにおいて得られたストリームファイルを，http://www-higashi.ist.osaka-u.ac.jp/software/stable_multicast.html よりダウンロードできる．

参考文献

[1] S. Paul, K. K. Sabnani, J. C.-H. Lin and S. Bhattacharyya, "Reliable Multicast Transport Protocol (RMTP)," *IEEE Journal of Selected Areas in Communications*, Vol. 15, No. 3, pp. 407–421, 1997.

[2] S. Shi, J. Turner and M. Waldvogel, "Dimensioning Server Access Bandwidth and Multicast Routing in Overlay Networks," *Proc. of Network and Operating System Support for Digital Audio and Video (NOSSDAV'01)*, pp. 83–91, 2001.

[3] V. Roca and A. El-Sayed, "A Host-Based Multicast (HBM) Solution for Group Communications," *Proc. of 2001 IEEE Int. Conf. on Networking (ICN'01)*, 2001.

[4] D. Pendarakis, S. Shi, D. Verma and M. Waldvogel, "ALMI: An Application Level Multicast Infrastructure," *Proc. of 3rd USENIX Symp. on Internet Technologies and Systems (USITS)*, pp. 49–60, 2001.

[5] Y. -H. Chu, S. G. Rao and H. Zhang, "A Case for End System Multicast," *Proc. of ACM SIGMETRICS*, pp. 1–12, 2000.

[6] S. Banerjee, B. Bhattacharjee and C. Kommareddy, "Scalable Application Layer Multicast," *Proc. of ACM SIGCOMM 2002*, pp. 205–217, 2002.

[7] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee and S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications," *Proc. of IEEE INFOCOM 2003*, 2003.

[8] T. M. Baduge, 廣森 聡仁, 梅津 高朗, 山口 弘純, 東野 輝夫, "オーバレイネットワーク上で遅延最小木を動的に構築する分散型プロトコル MODE の提案と評価," **情報処理学会論文誌**, vol. 46. no. 2, pp 482–492, 2005.

[9] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The feasibility of supporting large-scale live streaming applications with dynamic application end-points," in *Proc. of ACM SIGCOMM 2004*, pp. 107–120, 2004.

[10] M. Bishop, S. Rao, and K. Sripanidkulchai, "Considering priority in overlay multicast protocols under heterogeneous environments," in *Proc. of IEEE INFOCOM 2006*, pp. 1–13, 2006.

[11] G. Tan and S. A. Jarvis, "Improving the fault resilience of overlay multicast for media streaming," *IEEE Transactions on Parallel and Distributed Systems* **18**, pp. 721–734, June 2007.

[12] T. M. Baduge, A. Hiromori, H. Yamaguchi, and T. Higashino, "Design and implemetation of overlay multicast protocol for multimedia streaming," in *Proc. of 34th IEEE Int. Conf. on Parallel Processing (ICPP2005)*, pp. 41–48, 2005.

[13] K. Ikeda, T. M. Baduge, T. Umedu, H. Yamaguchi, and T. Higashino, "A middleware for implementation and evaluation of application layer multicast protocols in real environments," *Proc. of the 17th International workshop on Network and Operating Systems Support for Digital Audio & Video (NOSSDAV 2007)*, pp. 125–130, 2007. Tools are provided: http://www-higashi.ist.osaka-u.ac.jp/software/ALM/middlewareAPI/.