2009－MBL－48 (2)
2009／1／29

プラグイン型ルーティングアルゴリズムを用いた複数オーバーレイネットワークの並列検証機構

水谷 后宏† 洞井 晋一† 松浦 知史† 藤川 和利† 砂原 秀樹†‡

† 奈良先端科学技術大学院大学 情報科学研究科
‡ 慶応義塾大学大学院 メディアデザイン研究科

概要

オーバーレイネットワークの研究が盛んに行われる様になるにつれ，多種多様な，経路決定を行うアルゴリズムが提案されてきた．そのため，新たなアルゴリズムの提案において，既存のアルゴリズムとの比較検証が必要になる．しかし既存のアルゴリズムの検証機構では，一つの基盤において，一つのアルゴリズムの検証のみ可能なため，多種多様なアルゴリズムを検証するには，アルゴリズム毎に基盤を再構築する必要があった．そこで，本研究では，一つのオーバーレイの基盤において，様々なアルゴリズムをプラグイン化する事により，複数のアルゴリズムを並列に検証できる機構を提案する．これにより，各アルゴリズムの独立した評価が可能になり，既存の検証機構の実環境上における再現性の問題を解決する事が可能となる．

A system for parallel estimation of multiple overlay network routing algorithms by a plug-in architecture

Kimihiro Mizutani† Shinichi Doi†Satoshi Matsuura†Kazutoshi Fujikawa† Hideki Sunahara†‡

†Nara Institute of Science and Technology Graduate School of Information Science
‡Keio University Graduate School of Media Design

Abstract

As a lot of researchers study overlay network, a kind of routing algorithms are proposed. If you propose a new algorithm, you will need to prove the superiority of your algorithm. Using past estimation systems, you can estimate only one algorithm at the same time. Therefore you have to reconstruct an environment in order to estimate other algorithms. In this research, we propose a new estimation system. Our system is based on one fundamental overlay network and enables a parallel estimation of multiple routing algorithm at the same instant by a plug-in architecture. Our system offers a environment where you can estimate and compare routing algorithms on the same experimental circumstance .

1 はじめに

TCP/UDP の通信を抽象化し，アプリケーション層で仮想的にネットワークを構築するオーバーレイネットワーク（以下 オーバーレイ）が盛んに研究されている．その中でも，特に構造化オーバーレイの技術が研究されており，オーバーレイ上での端末（以下 ノード）間での経路検索を，アルゴリズムによって論理的に決定しようとする試みが存在する．このアルゴリズムの例として，Chord[1],Kademlia[3],Pastry[4] が挙げられる．これらのアルゴリズムは，それぞれ異なった特徴があり，オーバーレイ上での経路決定手法や，ノードの参加・離脱への対処が異なる．また，これらのアルゴリズム以外も，アプリケーションの用途や，様々な環境下（参加離脱の頻度が高い環境，ネットワークトラフィックが多い環境など）[5] に適応するようなアル

ゴリズムもある．このように，アルゴリズムは多数存在しており，新たなアルゴリズムを研究・提案する上で，既存のアルゴリズムとの比較検証が必須になってくる．

比較検証では，提案するアルゴリズムと既存のアルゴリズムを同じ環境下で動作させなければならない．同じ環境下とは，シミュレーションやエミュレーションで再現される事が多い．そのため，これらを利用し，アルゴリズムの比較検証を行い，実環境における動作を再現した後，実環境上で試験や運用を行う．しかし，アルゴリズムによっては，実環境上での検証でしか妥当性を検証できない場合がある．例として，実環境上での外部要因 (RTT, ジッタ, 帯域の変化) を考慮に入れたアルゴリズム [6] [7] [8] がある．このアルゴリズムの場合，比較検証は，実環境上で行う必要があるため，既存の外部要因を考慮したアルゴリズムとの比較において，実環境での再現性が必要になってくる．しかし，インターネットのような不安定な実環境では，アルゴリズムの検証を行うたびに環境が異なるため，既存のテストベッド [2] のような安定した環境に比べ，再現性を保証することが困難となる．そこで，本論文では不安定な実環境上でも再現性のある検証を行うために，アルゴリズムを並列して動作する事を可能にする機構を提案する．これにより，同じ環境下で，複数の比較対象となるアルゴリズムを検証する事ができ，再現性の問題を解くことが可能となる．さらに，複数のアルゴリズム間での干渉における，トラフィック異常などの外部要因の検出が可能となり，ネットワーク全体の挙動を考慮した計測が可能となる．また本機構では，アルゴリズムの比較検証を容易にするため，アルゴリズムのプラグイン化機能も提供する．このプラグイン化機能により，アルゴリズムの再利用やアルゴリズムを反映したオーバーレイを簡単に構築する事が可能になる．そして，実環境における検証の問題点であった広域に分散しているノードへのアルゴリズムの適用 (追加・交換) も可能になる．

以降では，まず２章で関連研究を述べ，３章で並列検証機構の構造や，アルゴリズムの追加方法を具体的に示し，４章で本機構の動作確認をエミュレーションで示し，この並列検証機構が実環境上で動作する際の課題を述べる．そして，5 章, で今後の課題を述べ，6 章で結びとする．

2　関連研究

オーバーレイ上で経路決定を行うアルゴリズムの研究に当たり，様々な検証機構が提案されてきた．アルゴリズムのシミュレータにおいて p2psim[9], 複数の端末間でのエミュレーションを可能にする peeremu[10], 実環境上での実時間を peeremu と同様に利用した Overlay Weaver[11] がある．p2psim では疑似コードを利用し，アルゴリズムの実装を可能とする．これに対し，実環境を考慮した peeremu, Overlay Weaver では，アルゴリズムの実用性を考慮した検証機構になっている．peeremu では複数台の端末で，数千規模の実環境での大規模ネットワークの再現を行っている． また，どのようなアルゴリズムにおいても，同一の環境 (ネットワーク遅延, ノードの数，ノードの生存時間) をエミュレートする事で，再現性を達成している．ただし，この同一環境とは，実環境独特のネットワークリソース (ジッタ，RTT，帯域) の変化や各ノードのタスクのスケジューリングの再現性は言及されていない．これに対し，Overlay Weaver は，実環境上での動作実績を持ち，十数万ノードのエミュレーションが可能になっている．しかし，Overlay Weaver は，たしかに実環境上で動作が可能だが，複数のアルゴリズムの検証には，アルゴリズム毎に実験環境の再構築を行わなければならない．また，比較検証する際の問題点として，実環境が動的に変わるため，比較対象のオーバーレイを切り替えると環境が異なってしまうので，peeremu で述べられている再現性の問題もあり，アルゴリズムの公平な比較が困難である．

これらの既存の検証機構を用いてアルゴリズムの実用性を検証する場合に比べ，提案するアルゴリズムの比較検証機構では，実環境上でのネットワークリソースの変化や各ノードのタスクのスケジューリングの再現性を考慮している．これはアルゴリズムの比較において，比較対象となる複数のアルゴリズムを並列に動作させる事により可能となっている．なお，複数アルゴリズムの処理を公平に行うため，OverQoS[6]，CORTH[7] で述べられている，QoS 制御法を参考にしている．これらは，QoS を考慮したオーバーレイネットワークの構築手法に言及しており，後者の CORTH では ALM(Application Level Multicast) アプリケーションにおいて，ストリーム別にネットワークリソースの変化や各端末のスケジューリング，および処理能力に応じた QoS 制御を

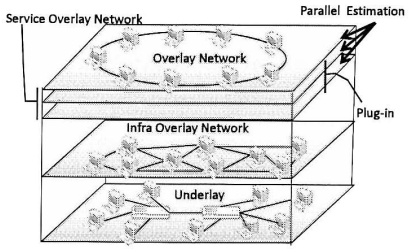

図 1: 並列検証機構の概念図

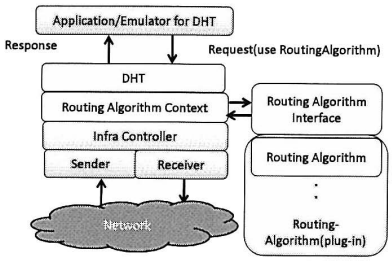

図 2: 提案システムの構成図

行っている．この QoS 制御を参考にし，本機構では，特定のアルゴリズムに対する処理の偏りを防ぐため，各アルゴリズムの処理制御を行う．このアルゴリズムの処理制御では，CPU やメモリ等の端末のリソースを公平に割り当てるため，複数のアルゴリズムをスレッドにより並列で動作させる．また，非効率なアルゴリズムがネットワークリソースを独占しないように，帯域制御 (パケット数/パケットの総量による制御) 等も必要となる．

3 並列検証機構の構造

本研究で構築する検証機構の概念図，および提案システムの構成図を図 1,2 に示す．Infra Overlay Network は，上層の Service Overlay Network の管理・制御を行う基盤となっている．また Service Overlay Network では，様々なアルゴリズムを実装したオーバーレイの動作を可能とし，容易にアルゴリズムの変更や追加も可能となっている．本章では各機構についての詳細を述べる．

3.1 Infra Overlay Network

Service Overlay Network における経路情報は，Infra Overlay Network のノード情報を利用している．そのため，Infra Overlay Network では，ノード間の接続保証を行うため，Service Overlay Network を維持する機能 (ノードの参加/離脱に対処する機能) を持つ．なお，Service Overlay Network に適用されるアルゴリズム毎に経路決定が異なるため，Infra Overlay Network で担保するノード情報は冗長である必要がある．そのため，Infra Overlay Network では，全てのノード情報を全てのノードが互いに管理するフルメッシュ型オーバーレイを構築し，維持する方針をとった．また，Service Overlay Network 上での通信を行う際の API を用意し，さらに，NAT 内にいるノードへの通信手法 (逆向き接続/UDP Hole Punchiung) の抽象化も行っている．

3.1.1 システム構成

Infra Overlay Network は，図 2 の Infra Controller と，Routing Algorithm Context のシステムに該当する．Infra Controller では，上層からのデータの送信命令や，受信したデータを上層に反映を行う．また，Service Overlay Network の接続維持も本システムで行う．なお，フルメッシュ型オーバーレイの効率的な管理のため，Infra Overlay Network 上の全ノードに対して，効率的にデータを伝搬させる N-ary-flooding 機能を持っている．

Routing Algorithm Context は，Service Overlay Network を管理しているシステムである．これは，Service Overlay Network のアルゴリズムに ID を付けて，保存しておき，上層からのアルゴリズムの呼び出しに応じて，切り替わるシステムになっている．

3.1.2 N-ary-flooding

Infra Overlay Network 上でのノード情報の交換において，1 対多 (全ノード) での通信 (Flooding) が発生する．これは，ノードがオーバーレイに参加/脱退を行った際に，全ノードへその情報を伝える必要があるためである．しかし，オーバーレイの規模が肥大化するにつれ，Flooding を行うノードの負荷は莫大になる．そのため，ノードの負担軽減のために，効率的な Flooding が必要になる．本機構で

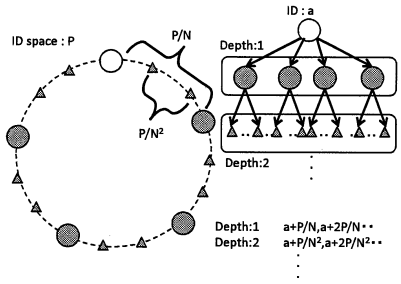

図 3: N-ary-flooding

はFloodingの方法として，各ノードを節・子に見立て，N分木を構築し，送信者をrootノードとした，マルチキャストを行う手法 (以下　N-ary-flooding) をとる．具体的には，図3に示す通り，ノードのID空間 (ID 総数:P) を，root ノードの ID:a を始点として，まずN分割する．root ノードから P/N の倍数番目のノードに対して，root ノードはデータを送信する．データを受信したノード (P/N) は，次にデータを受信したノード (2P/N) までの，ID 空間を再度 N 分割し，データを送信する．これを再帰的に繰り返すことにより，マルチキャストを用いたFloodingを行うことが可能となる．

3.1.3　フィードバック機構

Service Overlay Networkの接続保証は，全てInfra Overlay Network 上で行う事を本節で述べてきた．本項では，Infra Overlay Network における接続保証（ノードの参加/離脱に対しての対処）の方法を述べる．

- **ノードの参加：** 本機構にノードが参加する際に，既存ノードに対して join-message を送信する．join-message を受信したノードは，自身がもつ Infra Overlay Network に存在している全てのノード情報を参加ノードに対して送信する．参加ノードは，受け取ったノード情報を使用し，Infra Overlay Network に存在している全ノードに対して addAddress-message を送信し，自身のノード情報の登録をしてもらう．

- **ノードの離脱：** Infra Overlay Networkの各ノードは，存在しているノードの中で，ID 空間上で隣に位置するノードに対して，ある一定時間の周期で stub-message を送信する．stub-message を受信したノードは，送信ノードに対して responseStub-message を返す．もし一定時間内に responseStub-message を受信できなかった場合，離脱が起きたと判断し，stub-message を送信したノードは Infra Overlay Network に存在している全ノードに対して lostNode-message を送信し，離脱ノードのノード情報を削除してもらう．

3.2　Service Overlay Network

本章1節での説明の通り，Service Overlay Network は，Infra Overlay Network 上で，構築できるオーバーレイである．なお，本節3項で説明するプラグイン型ルーティングアルゴリズムを使用する事で，オーバーレイを容易に構築する事が可能になっている．

3.2.1　システム構成

Routing Algorithm Interface を用いた Routing Algorithm が，Service Overlay Network 上のオーバーレイのアルゴリズムに該当する．なお，Interface では，Infra Overlay Network 側からのフィードバック機構によるノード情報を利用する事が可能となっている．また Infra Overlay Network における，通信 API の使用も可能となっている．

3.2.2　プラグイン型ルーティングアルゴリズム

本機構では，Infra Overlay Network の通信 API やノード情報を用いて，Service Overlay Network を構築する．つまり，Infra Overlay Network の持つノード情報を用いてオーバーレイ毎の各アルゴリズムが持つ経路情報 (以下　ルーティングテーブル) を作成し，上層から指定されたアルゴリズムをルーティングに適用する．なお，各アルゴリズムのルーティングテーブルを作成するためには，本機構で提案する API を用いなければならない．この API はアルゴリズムの作成以外に，ノードの参加/離脱に対しての，アルゴリズムの動作を指定する事も可能となっている．これらの API を実装した Java の Class を，本機構ではプラグイン型ルーティングアルゴリズムと呼び，これを Infra Overlay Network

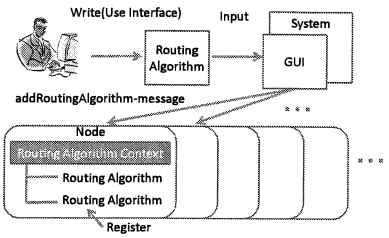

図 4: アルゴリズムの追加

の N-ary-flooding を用いて，全ノードに伝搬させ，読み込ませる事により，Service Overlay Network を容易に構築する事が可能となる．

3.2.3 API

CoreAPI 及び，SubAPI が利用可能となっている，CoreAPI とは，アルゴリズムの作成において必ず実装しなければならない API で，ルーティングテーブルの作成，ルーティング情報の提供を行う API である．makeRoutingTable の引数に当たる infra とは Infra Controller の機能であり，Infra Controller の通信 API を利用や，Infra Overlay Network におけるノード情報 (lifetime,ID) の利用が可能となっている．また，SubAPI はアルゴリズムの作成において，ユーザが任意で実装することができる API である．表 1 に API の一覧を示す．なお SubAPI は，返り値が boolean となっている．もし，true を返した場合，ユーザがノードの参加/離脱の処理を Infra Controller の API を用いて記述する必要がある．false の場合は，Infra Controller がノードの参加/離脱に応じて，自動的に Service Overlay Network のルーティングテーブルを書き換える．このように，検証項目に対して API を割り当てる事で，アルゴリズムの経路決定方法の検証やノードの参加/離脱についての検証を切り分ける事が可能となる．

3.2.4 ルーティングアルゴリズムの追加

アルゴリズムの追加の工程を図 4 に示す．API を用いて，Java のソースファイルを作成し，コンパイルを行い，class ファイル (プラグイン型ルーティングアルゴリズム) を作成する．この class ファイルを addRoutingAlgorithm-message と共に Infra Overlay Network の N-ary-flooding を用いて，全

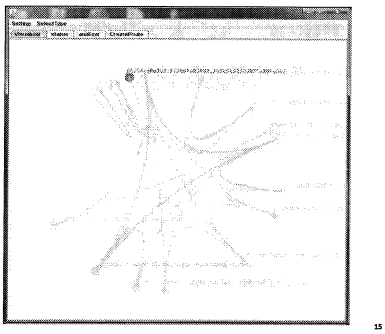

図 5: ビジュアライザ

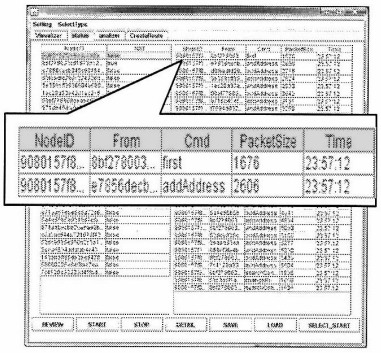

図 6: パラメータ表示

ノードに伝搬させることで，アルゴリズムを追加し，Service Overlay Network を構築する事が可能となる．本機構では，Infra Overlay Network によるアルゴリズムの追加機能を提供する GUI を用意してある．そのため，この GUI を用いて class ファイルを伝搬させる事ができ，容易に Service Overlay Network を構築する事が可能となる．

3.2.5 評価可能なパラメータ

評価可能なパラメータは GUI を通して閲覧する事ができる．本機構の GUI では，リアルタイムでルーティングの状況を閲覧できるビジュアライザや各ノード毎のメッセージログ，各 Service Overlay Network におけるパケットの総量 (bytes)，メッセージ総数なども閲覧できる．図 5，6 に GUI を示す．

表 1: API一覧

API	API Name	説明
CoreAPI	void makeRoutingTable(infra)	Infra Overlay Network のノード情報を利用しルーティングテーブルを作成
	ID getNextRoute(ID target)	target の ID に対し，ホップ先のノード ID を返す
SubAPI	boolean copeWithNodeJoin (ID joinNodeID)	ノードの参加を検知した際の，動作を記述
	boolean copeWithNodeLost(ID lostNodeID)	ノードの離脱を検知した際の，動作を記述

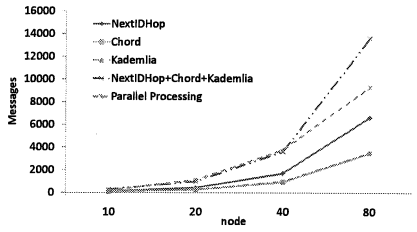

図 7: メッセージ数

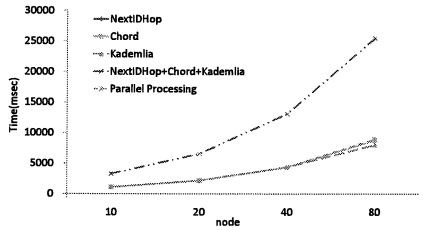

図 8: 処理時間

4 実験と評価

本研究では，並列検証の妥当性や，並列検証におけるアルゴリズム間の干渉を調査し，実環境上において，実験結果の再現性を向上させる事が目的である．本章では，特定のコンテンツの検索/収得におけるアルゴリズム毎の動作と複数のアルゴリズムの並列動作についてのエミュレーション実験を行った結果を示す．そして，結果から明らかになったアルゴリズム間の干渉についての考察を述べ，実環境への適用について議論する．

4.1 実験概要

複数のアルゴリズムを評価するために，Chord[1],Kademlia[3],NextIDHop(次の経路は常に successer と言うアルゴリズム) の 3 種類のアルゴリズムを用いてエミュレーション実験を行った．エミュレーション実験では，これから行う予定の実機実験のため，1 ノード毎に送受信用のスレッドを作成し，データの送受信は全て UDP を用いて，実環境を想定した実験を行った．

実験のシナリオは，Infra Overlay Network のノード数を 10,20,40,80 と変化させ，全ノードが Infra Overlay Network を構築し，特定の ID を持つコンテンツの検索/収得を行う事にした．そして，コンテンツの収得が完了した際に費やした時間 (msec) や，Service Overlay Netwrok の各オーバーレイにおけるパケットの数を，アルゴリズム毎とアルゴリズムを並列動作させた際に測定した．なお，図 7,8 の凡例にある NextIDHop+Chord+Kademlia は各アルゴリズムの実験結果の和を示し，Parallel Processing は 3 種類のアルゴリズムを本機構を用いて並列で動作させた際の実験結果を示す．ノード ID には IP アドレス+ポート番号を SHA-1 を用いてハッシュ化したものを使用し，実験環境は　OS:Windows Vista Home Premium，CPU:Core 2 Duo T8100 2.1GHz，メモリ 2GB の PC を用いた．

4.2 実験結果と考察

本章 1 節で述べた実験シナリオを用いて，図 7 に示す総メッセージ数と，図 8 に示す実験シナリオを消化した際の時間の計測を行った．図 7 では，40 ノードま

でのメッセージ数がNextIDHop+Chord+Kademliaと Paralell Processingにおいて，あまり変化は見られないが，80ノードに規模を拡張した際に，NextIDHop+Chord+Kademliaでは13620，Parallel Processingでは8941となっており，メッセージ数が約35% 減少した．また図8では，10-80ノードの規模にかけて，NextIDHop+Chord+Kademliaと Parallel Processingでは，約70% の処理時間の差が見られた．

実験結果より，アルゴリズム並列動作を行った検証の方が，各アルゴリズムを独立して動作させて検証するより，メッセージ数の削減と検証時間を短縮できる事を確認できた．これは，アルゴリズム毎にオーバーレイネットワークの基盤構築を行わず，一つの基盤で複数のアルゴリズムを動作させるためである．また，Parallel Processingと各アルゴリズムでの処理時間の差異は2% - 11% となっており，これが並列検証の際に発生するアルゴリズム間の干渉となっている．

本実験により，少ないノード数で構成される検証環境では，並列動作が可能である事が分かった．同時に，プロトタイプ実装が正しく動作している事を確認できた．また，本検証機構では，各オーバーレイを独立して検証でき，プラグイン型ルーティングアルゴリズムの再利用が可能となっており，比較検証も容易に行う事ができた．しかし，大規模な実験環境における，マシンリソース (CPU, メモリ，ソケット)/ネットワークリソース (帯域) が異なるノード間での検証では，公平性を阻害するノードが現れる可能性がある．そのため，大規模な実験環境を用いて，どの程度の規模（ノード数）まで，アルゴリズム毎に割り当てられるマシンリソース/ネットワークリソースの，公平性の担保が可能かどうかを検証する必要がある．

5 今後の課題

現状として，複数のオーバーレイを実環境上で動作させた際に，各オーバーレイ間でのマシンリソース/ネットワークリソースの干渉が，どのように影響を与えるか分かっていない．そこで，提案機構を用いて，これらが，どう言った環境下で起こるのかを明らかにしていく．その上で，マシンリソース/ネットワークリソースをモニタリングするような補助ツール（GUI）を用いて，これらのアルゴリズム間の影響を詳細に検証できるように拡充していく．また，複数のID空間を使用するマルチオーバーレイ [13] の検証に適用できるように実装を行っていく予定である．

6 おわりに

本提案では，複数のアルゴリズムの並列検証を行う機構を用いて，既存の検証機構における実環境上での再現性の問題に着目し，公平な比較検証を可能にするための議論を行った．また，アルゴリズムをプラグイン化する事により，容易にオーバーレイを構築する事が可能になり，複数のアルゴリズムの比較検証を容易に行う事が可能となった．また，PC１台を用いてUDP通信を用いたエミュレーションにより，アルゴリズムの挙動や性質を調べることができた．そして，それらを並列で検証する事で効率的な検証が可能である事を確認した．

参考文献

[1] Stoica, I., Morris, R., Karger, D., Kaashoek,M. F. and Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications,Proc. ACM SIGCOMM 2001 , pp.149-160 (2001).

[2] Miyachi, T., Chinen, K. and Shinoda, Y.: StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software, Proc. International Conference on Performance Evaluation Methodologies and Tools (Valuetools) 2006 (2006).

[3] Maymounkov, P. and Mazi'eres, D.: Kademlia:A Peer-to-peer Information System Based on the XOR Metric, Proc. IPTPS ' 02 (2002).

[4] Rowstron, A. and Druschel, P.: Pastry: Scalable,decentralized object location and routing for large-scale peer-to-peer systems, Proc. Middleware (2001).

[5] Prakash Linga,Indranil Gupta,Ken Birman.:Cornell University, NYA Churn-Resistant Peer-to-Peer Web Caching System,Proceedings of the 2003 ACM workshop

on Survivable and self-regenerative systems(2003).

[6] L. Subramanian, I. Stoica, H. Balakrishnan and R. Katz: “OverQoS: Offering internet QoS using overlays”, in Proceedings of HotNets-I(2002).

[7] 川田雅人, 正岡直也, 森川博之, 青山友紀,CORTH: 端末特性の異種性を考慮したオーバーレイネットワーキング, 社団法人電子情報通信学会信学技報 (2003).

[8] Tutomu Murase,Hideyuki Shimonishi,Masayuki Murata.”Overlay Network Technologies for QoS Control”,IEICE TRANS.COMMUN., vol.e89-b,No.9 September,2006(2006)

[9] J.J. Li, J. Stribling, T. M. Gil, R. Morris, and M. F.Kaashoek. Comparing the performance of distributed hash tables under churn. In Proc. of the 3rd International Workshop on Peer-to-Peer Systems(IPTPS04), February 2004(2004).

[10] Daishi Kato,Toshiyuki Kamiya.Evaluating DHT Implementations in Complex Environments by Network Emulator .iptps,paper(2008).

[11] 首藤一幸, 田中良夫, 関口智嗣:オーバーレイ構築ツールキット Overlay Weaver, 情報処理学会論文誌:コンピューティングシステム,Vol.47,No.SIG12(ACS15),pp358-367(2006)

[12] Chun B.,Culler D.,Roscoe T.,Bavier A.,Peterson L.,Wawrzoniak M.and Bwman M.:PlanetLab: An Overlay Testbed for Broad-Coverge Services,ACM SIGCOMM Computer Communication ,Vol.33,No.3,pp3-12(2003)

[13] 吉田幹, 寺西裕一, 春本要, 下條真司, ”マルチオーバーレイと分散エージェントの機構を統合化した P2P プラットフォーム PIAX,” 情報処理学会研究報告 2006-DPS-128, pp.43-48 (2006.9).