一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SITE2015-64,IA2015-96(2016-03)

Self-Refining Skip Graph: 理想的なSkip Graphに近づいていく構造化オーバレイ

川口　峻史† 　坂野　遼平† 　北條　真史† 　首藤　一幸†

† 東京工業大学　〒152-8552 東京都目黒区大岡山 2-12-1
E-mail: {kawaguchi.t.ae, banno.r.aa, hojo.m.aa}@m.titech.ac.jp, shudo@is.titech.ac.jp

あらまし　構造化オーバレイでは，数学的・論理的な構造を持ったネットワークを構築する．構造化オーバレイの一つである Skip Graph では，ノードそれぞれに割り当てられる membership vector に基づいてノードの挿入や離脱の際に他ノードとリンクを張り経路表構築を行う．しかしながら，membership vector は乱数によって決まるため，理想から離れたトポロジのネットワークが形成され，経路長は悪化することが多い．そこで，我々は理想的な Skip Graph のトポロジに向けて各ノードが自律分散的に経路表を洗練し，トポロジを改良する構造化オーバレイ Self-Refining Skip Graph を提案する．提案手法は，membership vector が持つロバスト性を保ちつつ，理想的なトポロジに近づくことによって，より効率の良いルーティングが可能である．
キーワード　Peer-to-Peer，構造化オーバレイ，Skip Graph，membership vector

Self-Refining Skip Graph: A Structured Overlay Approaching to Ideal Skip Graph

Takafumi KAWAGUCHI†, Ryohei BANNO†, Masashi HOJO†, and Kazuyuki SHUDO†

† Tokyo Institute of Technology　Ookayama 2-12-1, Meguro-ku, Tokyo, 152-8552 Japan
E-mail: {kawaguchi.t.ae, banno.r.aa, hojo.m.aa}@m.titech.ac.jp, shudo@is.titech.ac.jp

Abstract　Structured overlays construct mathematical and logical structural networks. In Skip Graph, one of structured overlays, each node constructs its routing table by establishing or disconnecting links to other nodes based on its membership vector when a node joins or leaves. However, membership vectors are determined randomly, so nodes don't always compose an ideal topology of the network. This causes route lengths to be worse. Therefore, we propose Self-Refining Skip Graph, a structured overlay where each node refines its routing table for an ideal topology of Skip Graph. The proposed method provides more efficient routings by approaching to an ideal topology while keeping the robustness caused by membership vectors.
Key words　Peer-to-Peer, Structured Overlay, Skip Graph, membership vector

1. はじめに

実ネットワーク上に構築したアプリケーションレベルの仮想的なネットワークをオーバレイネットワークと呼び，これはしばしば Peer-to-Peer システムに応用される．オーバレイネットワークはそのトポロジの構造により非構造化オーバレイと構造化オーバレイに分類される．

構造化オーバレイでは，ノードに ID を割り当て，ノードは割り当てられた ID (ノード ID) に基づいて ID 空間内の担当領域が決められる．各ノードは隣接ノードの ID と IP アドレスの組からなる経路表を構築する．ノード間の通信は，目的ノードに向けて自身の経路表から転送先を選択し，その選択ノードへのメッセージ転送を繰り返すことによって実現する．構造化オーバレイは，耐故障性やスケーラビリティに優れているといった特徴を持つ．

構造化オーバレイを利用した技術として，複数のノードで連想配列を管理する分散ハッシュテーブル (Distributed Hash Table, DHT) がある [9], [5], [3]．DHT では，構造化オーバレイのノード上で key と value のペアからなるデータの格納と取得を実現する．ノード ID の割り当てには自身の IP アドレスなどをハッシュ関数にかけたものを用いるため，ノード数が十分大きい場合にはノード ID は一様に分布する．また，データにも key をハッシュ関数にかけて ID を割り当て，データは割り当てられた ID (データ ID) を担当領域として含むノードに保

存される．各ノードの経路表構築は，ノード間の ID 距離に基づいて行うことが多い．この方式では，ノード数が小さい場合やノード ID の割り当てに位置情報などの他の要素を考慮する場合，ノード ID の一様性が崩れることが多く，ルーティングにおける目的ノードまでの経路長 (メッセージの転送回数) は悪化するといった欠点を持つ．

ノード ID の一様性が崩れていても経路長が悪化しない構造化オーバレイとして，Chord$^{\#}$ [6]，Skip Graph [1]，Mercury [2] などが挙げられる．これらの構造化オーバレイでは，データの連続性を保ったままデータ ID を割り当てることが可能なため，範囲検索が可能な構造化オーバレイとして知られている．

Chord$^{\#}$ と Skip Graph を比較すると，Skip Graph ではノードの挿入や離脱の際にその隣接ノードのみ経路表を更新すれば良いのに対して，Chord$^{\#}$ では定期的に全ノードの経路表を更新する必要がある．そのため，経路表の維持管理コストは Chord$^{\#}$ より Skip Graph の方が低いと言える．しかしながら，両者をルーティング効率の点で比較すると，ノード数が N の時，Chord$^{\#}$ では経路表構築が収束すると経路長が $\log N$ 以下になるのに対して，Skip Graph では経路長が $O(\log N)$ であるものの最大経路長は悪化する傾向にある．これは，Skip Graph においてはノードそれぞれに割り当てられる membership vector (MV) と呼ばれる乱数列に基づいて他ノードとリンクを張り経路表構築を行うため，理想から離れたトポロジのネットワークが形成されることが多く，効率の良いルーティングが実現されないことが原因であると考えられる．

そこで，我々は Skip Graph のように MV に基づいて経路表構築を行いつつ，新たに各ノードが自律分散的に MV の偏りを修正しながら経路表を洗練し，トポロジを改良する構造化オーバレイ Self-Refining Skip Graph を提案する．これにより，経路表の維持管理コストを Chord$^{\#}$ よりも低く抑えながら，Skip Graph のように最大経路長が悪化することのない構造化オーバレイが実現する．提案手法は，MV が持つロバスト性を保ちつつ，理想的なトポロジに近づくことによって，より効率の良いルーティングが可能である．

2. 関 連 研 究

本章では，ノード ID の一様性が崩れていても経路長が悪化しない構造化オーバレイ Chord$^{\#}$ [6]，Skip Graph [1] について述べる．これらの構造化オーバレイでは，ノード ID として key と呼ばれる値をノードに割り当て，ノードは割り当てられた key に基づいて key 空間上に key の昇順に配置されるため，範囲検索が可能である．

2.1 Chord$^{\#}$

Chord$^{\#}$ は，Chord [9] において用いられた Chord リングと呼ばれる円環状の key 空間を持つ．ノードは自身の key に基づいて Chord リング上に配置される．データは自身の key に基づいて Chord リング上に仮想的に配置され，その key から Chord リング上を時計回りに進んで最初に出会う担当ノードに保存される．

Chord$^{\#}$ では，各ノードが successor list，predecessor，finger table の 3 つの経路表を持つ．ある key から Chord リング上を時計回りに進んで最初に出会うノードをその key に対する successor と呼び，successor list は自身の key から Chord リング上を時計回りに進んで出会うノードの情報を順に一定数保持する経路表である．また，ある key から Chord リング上を反時計回りに進んで最初に出会うノードをその key に対する predecessor と呼び，自身の predecessor のノードの情報も経路表に保持する．finger table は $\mathrm{finger}_i (i = 0, 1, 2, ...)$ を要素に持つ経路表であり，$\mathrm{finger}_i \cdot \mathrm{finger}_j$ を finger_i が指すノードの経路表における finger table の要素 finger_j とすると，$\mathrm{finger}_i (i = 0, 1, 2, ...)$ は (1) のように定義される．

$$\mathrm{finger}_i = \begin{cases} \text{successor} & , i = 0 \\ \mathrm{finger}_{i-1} \cdot \mathrm{finger}_{i-1} & , i > 0 \end{cases} \quad (1)$$

(1) により，Chord$^{\#}$ では，自身の key から Chord リング上を時計回りに進んで 2^i 番目 $(i = 0, 1, 2, ...)$ に出会うノードの情報を経路表に保持することが可能である．

Chord$^{\#}$ のルーティングは，目的 key に最も近づくエントリのノードへのメッセージ転送を繰り返す greedy routing によって行われる．各ノードは successor list 及び finger table から転送先を選択する．目的 key の predecessor に到達すると，最後にその successor にメッセージを転送することによって目的 key の担当ノードに到達する．Chord$^{\#}$ では，successor list を用いたルーティングにより，メッセージの担当ノードへの到達性を保証している．また，経路表構築が収束した finger table を用いた greedy routing により，一度のホップで最低でも目的 key までに存在するノード数をおよそ半分にすることができるため，ノード数 N に対して，経路長が $O(\log N)$ になることが証明されている．

しかしながら，(1) に示したように，Chord$^{\#}$ は finger table の要素 finger_i として，finger_{i-1} が指すノードの経路表における finger_{i-1} をエントリとして持つため，あるノードの finger_i が変更されると，連鎖的に全ノードの finger table のいずれかの要素も変更されることになる．これは定期的に全ノードの経路表のいずれかの要素を更新する必要があることを意味し，経路表の維持管理コストが比較的高いという欠点を持つ．

2.2 Skip Graph

Skip Graph は，Skip List [4] と呼ばれるデータ構造に基づいた構造化オーバレイである．ノードは自身の key 及びそれぞれに割り当てられる membership vector (MV) に基づいて階層構造を持ったリスト上に配置される．MV とは，m 進数の乱数列であり，本稿では $m = 2$ とする．図 1 に示すように，レベル 0 においては，全ノードが key の昇順に自身の key の前後にあたるノードと対称なリンクを張り，レベル i においては，MV の接頭辞 i 桁が一致するノード同士が key の昇順に自身の key の前後にあたるノードと対称なリンクを張ることで経路表構築を行う．

Skip Graph では，各ノードが neighbor[R]，neighbor[L] の 2 つの経路表を持つ．neighbor[R] は各レベルにおける自身の key から key の昇順に進んで最初に出会うノードを保持する経

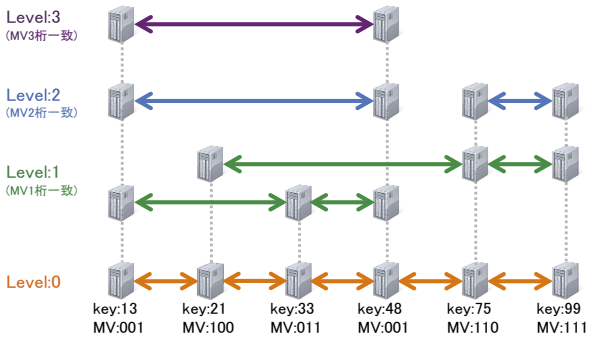

図 1　Skip Graph のトポロジ
Fig. 1　A topology of Skip Graph

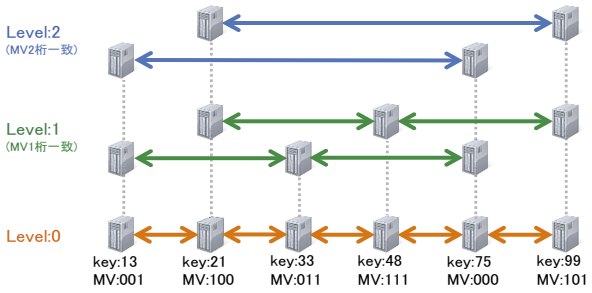

図 2　理想的な Skip Graph のトポロジ
Fig. 2　A topology of ideal Skip Graph

路表である．また，neighbor[L] は各レベルにおける自身の key から key の降順に進んで最初に出会うノードを保持する経路表である．

Skip Graph のルーティングは，目的ノードの key に最も近づくエントリのノードへのメッセージ転送を繰り返すことによって行われ，最後に目的ノードに到達する．各ノードは neighbor[R] 及び neighbor[L] の最上位レベルから転送先を選択し，目的ノードの key に最も近づくエントリが存在しない場合は 1 つ下位のレベルから転送先を選択する．Skip Graph では，レベル 0 における key の昇順に自身の key の前後にあたるノードを含む neighbor[R] 及び neighbor[L] を用いたルーティングにより，メッセージの担当ノードへの到達性を保証しながら，一度のホップで確率的に目的ノードまでに存在するノード数をおよそ半分にすることができるため，ノード数 N に対して，経路長が $O(\log N)$ になることが証明されている．

Skip Graph は MV に基づいて他ノードと対称なリンクを張り経路表構築を行うため，ノードの挿入や離脱がある場合にその隣接ノードのみ経路表を更新すれば良く，経路表の維持管理コストは比較的低い．しかしながら，乱数によって決まる MV の偏りにより，理想から離れたトポロジのネットワークが形成され，最大経路長は悪化するという欠点を持つ．MV の偏りとは，隣接ノード間における MV の接頭辞の共通部分列の長さを指す．図 2 に示すように，理想的な Skip Graph では，レベル i の隣接ノードがレベル $i-1$ の隣接ノードの隣接ノードであることが期待されている．これにより，自身の key から進んで 2^i 番目 $(i = 0, 1, 2, ...)$ に出会うノードの情報を経路表に保持することが可能となり，効率の良いルーティングが実現する．しかしながら，実際には隣接ノード同士で MV の接頭辞の共通部分列が長くなり，レベル i の隣接ノードがレベル $i-1$ の隣接ノードそのものであることも少なくなく，各ノードの経路表は隣接レベル間で重複するエントリを持ってしまう．したがって，Skip Graph は理想から離れたトポロジのネットワークが形成されることが多く，常に効率の良いルーティングを実現しているとは言えない．

3. 提案手法

我々は，Skip Graph [1] のように MV に基づいて経路表構築を行いつつ，新たに各ノードが自律分散的に MV の偏りを修正しながら経路表を洗練し，トポロジを改良する構造化オーバレイ Self-Refining Skip Graph を提案する．

3.1 概　　要

提案手法では，key 及び MV の割り当て方法や経路表構築方法などは Skip Graph と同様にする．これは経路表の維持管理コストを Skip Graph のように低く抑えるためである．仮に，MV の割り当てを乱数生成ではなく，理想的な Skip Graph のトポロジになるよう計算することにすれば，ノード数 N に対して，経路長を $\log N$ 以下にすることは可能である．しかしながら，この場合，ノードの挿入や離脱がある度に理想的なトポロジを維持するため，全ノードが MV の修正とそれに伴う経路表の更新を強いられ，経路表の維持管理コストが Chord$^\#$ [6] のように高くなってしまう．したがって，MV の割り当て方法も Skip Graph と同様にし，経路表構築後に MV の偏りを修正しながら経路表を洗練し，トポロジを改良することを考える．

提案手法では，経路表構築後に各ノードが定期的に MV の偏りの検知と修正，それに伴う経路表の更新を行う．これにより，各ノードの経路表から隣接レベル間で重複するエントリが減り，レベル i の隣接ノードがレベル $i-1$ の隣接ノードの隣接ノードであることが実現し，全体として理想的な Skip Graph のトポロジに近づいていく．この MV の偏りの検知と修正を十分な時間行えば，理想的なトポロジに収束するが，決して Chord$^\#$ のように理想的なトポロジを保つことがアルゴリズムの設計方針ではない．理想的なトポロジに向けてトポロジを改良しながら通常のルーティングを行うことで，提案手法は Chord$^\#$ よりも経路表の維持管理コストを低く抑えながら，Skip Graph よりも効率の良いルーティングが実現するのである．

以下，具体的にどのノードの MV をどのように修正すれば経路表が洗練され，理想的な Skip Graph のトポロジに近づいていくかについて 3.2 で述べる．また，このような MV の修正を各ノードが自律分散的に行うため，各ノードが定期的に実行するアルゴリズムとして我々が考えたものを MV 偏り検知プロトコルと呼び，その詳細を 3.3，3.4 で述べる．

3.2 membership vector の修正

MV の修正は，MV の接頭辞のある桁のビットを反転することによって実現する．MV の修正が必要なノードは，MV 偏りノード群の偶数番目のノードである．MV 偏りノード群とは，図 3 の赤枠に示すような隣接ノード間において MV の接頭辞 i 桁目も $i-1$ 桁目も一致しているノードの集まり，つまりレベ

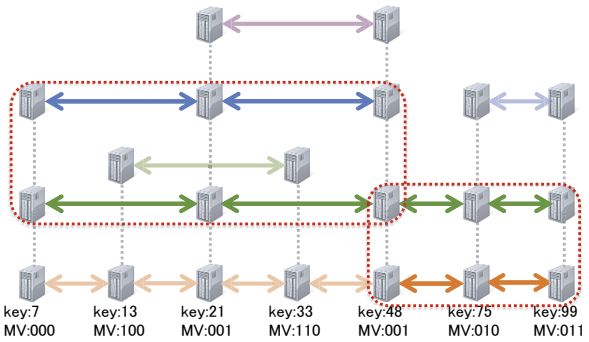

図 3　Skip Graph における MV 偏りノード群
Fig. 3　MV deviated node groups in Skip Graph

ル i とレベル $i-1$ の隣接ノードが一致しているノードの集まりを指しており，順序関係は key の昇順で定義される．この時，MV 偏りノード群の偶数番目のノードの MV の接頭辞 i 桁目のビットを反転すると，いずれの隣接ノード間においても MV の接頭辞 i 桁目が一致しなくなる．そのため，レベル i とレベル $i-1$ の隣接ノードが異なるものとなり，MV 偏りノード群を解消することができる．このような MV の修正を一度行うと，理想的な Skip Graph に期待されているレベル i の隣接ノードがレベル $i-1$ の隣接ノードの隣接ノードであることが部分的に実現する．これを各ノードが十分な回数行えば，全ノードにおいてレベル i の隣接ノードがレベル $i-1$ の隣接ノードの隣接ノードであることが実現し，理想的な Skip Graph のトポロジに収束するのである．

3.3　MV 偏り検知プロトコル

MV 偏り検知プロトコルとは，各ノードが定期的に実行するもので，自身が MV の偏りの一部であるか確認し，MV の偏りの一部であれば隣接ノードと協調し合いながら MV の修正を行うアルゴリズムである．

まず，自身が MV の偏りの一部であるか確認する．そのため，自身の経路表のいずれかのレベル間で隣接ノードが一致していないか調査することで，MV 偏りノード群に含まれているか確認する．次に，自身が MV 偏りノード群に含まれていれば，隣接ノードと協調し合いながら自身の MV を修正する必要があるか判断する．3.2 より，自身がこのノード群の偶数番目に位置するか把握すれば，自身の MV を修正するすべきか判断できるが，1 番目のノードを除き自身ではこのノード群の何番目に位置するか把握できない．そこで，このノード群の 1 番目でないノードは，1 番目のノードに MV 偏り検知メッセージを送信してもらうことによって，自身の位置を把握することにする．MV 偏り検知メッセージとは，転送回数が記録されたメッセージであり，このメッセージを受信したノードはこのノード群の 1 番目のノードから key の昇順に何回転送されてきたかを知ることによって，自身の位置を把握できる．つまり，このノード群の 1 番目のノードは MV 偏り検知メッセージを MV 偏りノード群に含まれるノードに向けて key の昇順に送信し，このメッセージを受信したノードはその転送回数から自身の位置を把握し，偶数番目であれば自身の MV を修正すべきと判断すれば良い．最後に，自身の MV を修正すべきと判断すれば，MV の接頭辞の適切なビットを反転することによって MV の修正を行う．

したがって，提案手法では，MV 偏り検知メッセージを受信すると，その転送回数から MV 偏りノード群における自身の位置を把握し，偶数番目である場合のみ自身の MV を修正してから，このノード群の次のノードに転送する (最後尾のため次のノードが存在しなければメッセージを破棄する) 機能をあらかじめ各ノードが備える．その上で，自身が MV 偏りノード群に含まれているか確認し，MV 偏りノード群に含まれていて，かつ 1 番目に位置していれば，MV 偏り検知メッセージをこのノード群の次のノードに送信する MV 偏り検知プロトコルを定期的に実行することによって，MV の修正は実現する．

なお，自身が複数の MV 偏りノード群に含まれている場合，MV 偏り検知プロトコルにおいて MV 偏り検知メッセージを送信するかの判断は最小のレベルにおける MV 偏りノード群でのみ行う．MV の接頭辞 i 桁目のビットを反転すると，3.4 で述べるように，経路表のレベル i 以上を更新する必要が出てくる．例えば，自身がレベル i と $j(i<j)$ における MV 偏りノード群に含まれている場合，先に MV の接頭辞 j 桁目のビットを反転すると，経路表のレベル j 以上を更新することになるが，次に MV の接頭辞 i 桁目のビットを反転した際に経路表のレベル j を含むレベル i 以上を更新することになり，以前にレベル j 以上を更新したことが無駄になってしまう．したがって，このような無駄が出ないよう最小のレベルからリンクを張り替えるように経路表を更新するため，MV 偏り検知メッセージを送信するかの判断は常に最小のレベルにおける MV 偏りノード群において行うようにする．

3.4　MV の修正に伴う経路表の更新

提案手法では，ノードの挿入時に MV に基づいて経路表構築を行うため，ノードの挿入後に MV を修正した場合には新しい MV に基づいて経路表を更新する必要がある．そこで本節では，MV 偏り検知プロトコルにより，あるノード A の MV の接頭辞 i 桁目のビットを反転した場合に新しい MV に基づいて経路表を更新する動作について述べる．この動作はノード A をエントリとして持つノードの経路表の更新とノード A 自身の経路表の更新の順に 2 ステップで行われる．

まず，ノード A をエントリとして持つノードの経路表の更新について述べる．ノード A をエントリとして持つノードのうち経路表の更新が必要なノードは，ノード A の経路表のレベル i 以上に含まれるエントリのノードである．そこで，これらのノードに新しい MV を通知する MV 通知メッセージを送信し，新しい MV の下で経路表を更新してもらうことを考える．したがって，提案手法では，MV 通知メッセージを受信すると，自身の経路表のレベル i 以上に含まれるエントリのうちメッセージの送信元ノードのエントリを一度削除し，新しい MV の下で削除したレベルにおける経路表構築を行う機能をあらかじめ各ノードが備える．その上で，ノード A の経路表のレベル i 以上に含まれるエントリのノードに MV 通知メッセージを送信することによって，ノード A をエントリとして持つノードの経路表

の更新は実現する．

次に，ノード A 自身の経路表の更新について述べる．ノード A の経路表のうち更新が必要なエントリは，レベル i 以上に含まれるエントリである．したがって，ノード A の経路表のレベル i 以上に含まれるエントリを一度削除し，新しい MV に基づいて削除したレベルにおける経路表構築を行うことによって，ノード A 自身の経路表の更新は実現する．

4. 評　　価

オーバレイネットワーク構築ツールキットである Overlay Weaver [7] [8] 上に提案手法である Self-Refining Skip Graph を実装し，シミュレーションによりその評価実験を行った．実験環境は表 1 の通りである．

評価実験では，オーバレイネットワークのノード数を 1000 とする．各ノードの key は一様分布に従うようにし，全ノードがランダムに選択した 10 ノードへの探索を繰り返した時の経路長を計測する．

4.1 MV 偏り検知プロトコルの実行回数比較

各ノードが自律分散的に自身が MV 偏りノード群に含まれているか確認する MV 偏り検知プロトコルを 0 回，5 回，500 回実行した時の経路長分布，平均経路長及び最大経路長を比較する．MV 偏り検知プロトコル 0 回とは，MV の偏りの修正を一切しないことを意味し，Skip Graph [1] と同様の動作をするため，その結果は Skip Graph を動作した時の結果ともみなせる．そのため，これらの結果から Self-Refining Skip Graph を Skip Graph と比較することができる．

図 4，5，6 に示すように，経路長分布は MV 偏り検知プロトコルの実行回数の増加と共に左に移動しており，全体として経路長が減少していることが分かる．それぞれの平均経路長を算出してみても 8.38，6.65，4.52 と減少している．同様に，それぞれの最大経路長も 28，20，9 と減少しており，これは Skip Graph において最大経路長が悪化していた状況を徐々に改善していることを表している．よって，MV 偏り検知プロトコル 0 回の Skip Graph と比較すると，MV 偏り検知プロトコルを実行する提案手法は，より効率の良いルーティングを行っていると言える．

4.2 理想的な Skip Graph のトポロジへの近づき方

全ノードが自律分散的に MV 偏り検知プロトコルを 1 回実行する度に，全ノードの経路表から隣接レベル間で重複するエントリの組数を数え上げ，その総数の変化を考察する．例えば，あるノードにおいて経路表のレベル 0，1，2 のエントリが一致している場合，レベル 0 - 1 間と 1 - 2 間に重複するエントリの組があるため，隣接レベル間で重複するエントリの組数は 2 と数えられる．これは MV 偏り検知プロトコルの実行により，どれほど理想的な Skip Graph のトポロジに近づいたかを表す指標になっており，全ノードのエントリ重複総数が 0 に到達することが理想的なトポロジに収束したことを表す．

図 7 に示すように，MV 偏り検知プロトコルの実行回数に対する全ノードのエントリ重複総数はおおよそ線形に減少している．MV 偏り検知プロトコル 0 回の場合は，全ノードのエントリ重複総数が 5000 であったのに対して，MV 偏り検知プロトコルを数回実行すると 5000，4526，4099，3825，3782，…とその総数は大幅に減少していく．これは Skip Graph のトポロジに対して MV 偏り検知プロトコルを数回実行するだけで大きく理想的な Skip Graph のトポロジに近づくことを表している．この結果は，図 4 と 5 も合わせて参照すれば，MV 偏り検知プロトコルを数回実行するだけで全体として経路長が減少してい

表 1　実 験 環 境
Table 1　Experimental environment

シミュレータ	Overlay Weaver 0.10.4
OS	Mac OS X 10.10.3
CPU	Intel Core i7-5557U 3.1GHz
メモリ	16GB
Java	Java SE 8 Update 45

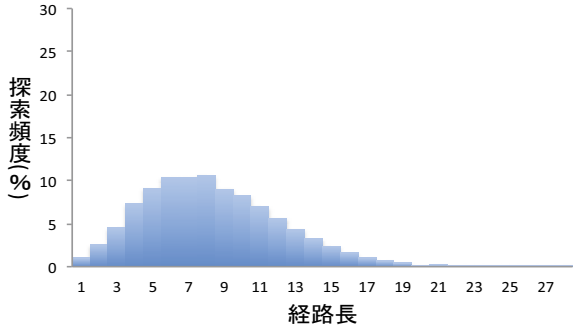

図 4　MV 偏り検知プロトコル 0 回の経路長分布
Fig. 4　Route lengths after 0 times proposal protocol applied

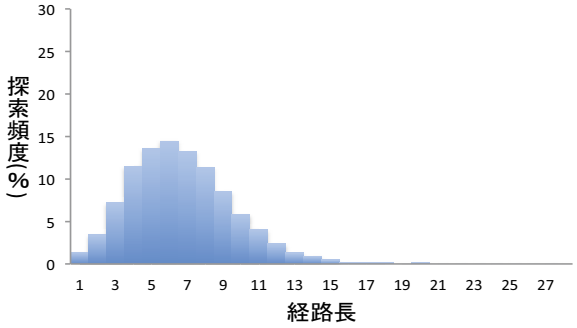

図 5　MV 偏り検知プロトコル 5 回後の経路長分布
Fig. 5　Route lengths after 5 times proposal protocol applied

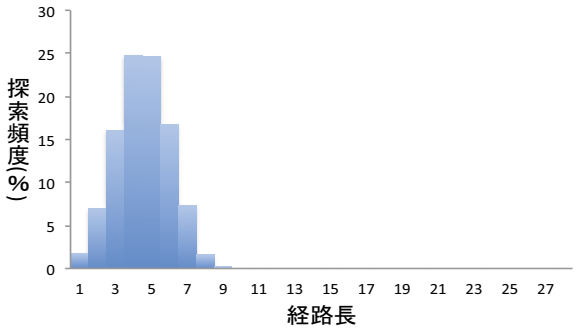

図 6　MV 偏り検知プロトコル 500 回後の経路長分布
Fig. 6　Route lengths after 500 times proposal protocol applied

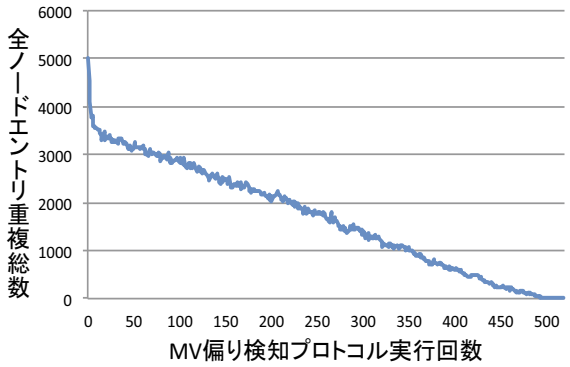

図 7　理想的なトポロジへの近づき方
Fig. 7　Approaching to ideal Skip Graph

ることにも表れている．その後，MV 偏り検知プロトコルを実行し続けると，全ノードのエントリ重複総数は緩やかに減少していき，500 回実行した時点で全ノードのエントリ重複総数は 0 に到達し，理想的なトポロジに収束する．

MV 偏り検知プロトコルの実行回数に対する全ノードのエントリ重複総数の減少は単調ではない．これは MV 偏り検知プロトコルの実行により，時には全ノードのエントリ重複総数がわずかに増加することもあることを表している．あるノードの MV の修正による重複するエントリの削減に成功しても，それが他ノードのエントリの重複につながってしまうことは少なくない．ただ，全体としては減少に向かっているため，特に問題はないと考えられる．

全ノードのエントリ重複総数が 0 に到達するまでの MV 偏り検知プロトコルの実行回数は，ノード数に比例することが実験的に分かっている．前述の通り，ノード数 1000 の場合は，MV 検知プロトコルを 500 回実行した時点で全ノードのエントリ重複総数は 0 に到達した．同様の実験をノード数 100 と 10000 の場合で行うと，それぞれ 50 回，5014 回実行した時点でその総数は 0 に到達した．よって，全ノードのエントリ重複総数が 0 に到達するまでの MV 偏り検知プロトコルの実行回数は，ノード数の半数程度になることが分かる．これは理想的な Skip Graph のトポロジに収束するまでの時間が $O(N)$ であることを表している．ただ，アルゴリズムの設計方針は理想的なトポロジを保つことではないため，理想的なトポロジに収束するまでの時間が $O(N)$ であることは提案手法の性能には影響しない．

以上の結果から注目すべき点は，Skip Graph のトポロジに対して MV 偏り検知プロトコルを数回実行するだけで大きく理想的な Skip Graph のトポロジに近づくことである．これはノードの挿入や離脱がある状況でも MV 偏り検知プロトコルを数回実行すれば，Skip Graph よりも短い経路長でルーティングを行うことを表しており，実用的な結果であると言える．

5. まとめと今後の課題

我々は，理想的な Skip Graph のトポロジに向けて各ノードが自律分散的に経路表を洗練し，トポロジを改良する構造化オーバレイ Self-Refining Skip Graph を提案した．提案手法は，MV が持つロバスト性を保ちつつ，理想的なトポロジに近づくことによって，より効率の良いルーティングが可能である．ノード ID の一様性が崩れていても経路長が悪化しない構造化オーバレイは範囲検索など様々な応用が存在し，我々の提案により，このような範囲検索が経路表の維持管理コストを Chord$^{\#}$ よりも低く抑えながら，Skip Graph よりも効率の良いルーティングで行うことが可能になった．

評価実験により，Skip Graph のように MV に基づいて経路表構築を行っても，提案手法の MV 偏り検知プロトコルを数回実行するだけで大きく理想的な Skip Graph のトポロジに近づき，全体として経路長が減少することを確認した．そのため，ノードの挿入や離脱が頻繁にある状況でも MV 偏り検知プロトコルを実行し続けていれば，Skip Graph よりも短い経路長でルーティングを行うことが分かった．また，仮に理想的な Skip Graph のトポロジに収束させることを考えると，MV 偏り検知プロトコルは $O(N)$ 回実行する必要があることも分かった．

今後の課題として，提案手法の MV 偏り検知プロトコルのメッセージ転送によるコストの定量的な評価が挙げられる．また，その定量的な評価を受けて，メッセージ転送によるコストを低く抑えた効率的なアルゴリズムについて考察し，より早く理想的な Skip Graph のトポロジに収束するようにする．理想的なトポロジを保つことがアルゴリズムの設計方針ではないものの，より早く収束させることは MV 偏り検知プロトコル 1 回の実行に対する経路長の減少率の向上につながるはずであるからである．

謝辞　本研究は JSPS 科研費 25700008，26540161 の助成を受けたものである．

文　献

[1] J. Aspnes and G. Shah. Skip Graphs. *ACM Transactions on Algorithms*, Vol. 3, No. 4, Article 37, 2007.
[2] A.R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. *ACM SIGCOMM Computer Communication Review*, Vol. 34, No. 4, pp. 353–366, 2004.
[3] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *In Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, pp. 53–65, 2002.
[4] W. Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees, 1990.
[5] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *In Proceedings of IFTP/ACM Middleware 2001*, pp. 329–350, 2001.
[6] T. Schutt, F. Schintke, and A. Reinefeld. Range Queries on Structured Overlay Networks. *Computer Communications*, Vol. 31, No. 2, pp. 280–291, 2008.
[7] K. Shudo. Overlay Weaver. http://overlayweaver.sourcegorge.net, 2006.
[8] K. Shudo, Y. Tanaka, and S. Sekiguchi. Overlay Weaver: An Overlay Construction Toolkit. *Computer Communications*, Vol. 31, No. 2, pp. 402–412, 2008.
[9] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *Networking, IEEE/ACM Transactions on*, Vol. 11, No. 1, pp. 17–32, 2003.