2008－OS－109 (3)
2008／8／6

トポロジの変化を考慮したネットワーク座標系を用いた分散ハッシュテーブル

小島 俊範　浅原 理人　早川 愛　河野 健二

慶應義塾大学 理工学部 情報工学科
E-mail: {kojima, asahara, aiai}@sslab.ics.keio.ac.jp, kono@ics.keio.ac.jp

インターネット上には，分散ハッシュテーブル（Distributed Hash Tables, DHT）によってオーバレイネットワークを構築する Peer-to-Peer システムが数多く存在する．DHT を用いることで，ネットワーク上で提供されているコンテンツを漏れなくかつ高速に探索することが可能となる．しかし，従来の DHT の多くは，物理ネットワークのトポロジを考慮していない．そのため，オーバレイネットワーク上における見かけの位置関係と，物理ネットワーク上における実際の位置関係の間に相関がなく，ノード間の通信経路に無駄が生じてしまう．本論文では，通信経路の無駄を削減した DHT として，*Latency-aware CAN* を提案する．Latency-aware CAN では，ネットワーク座標系によって求めた座標を基に CAN を構築することで，物理ネットワークのトポロジを近似する．また，物理ネットワークのトポロジの変化に対しても，それに追従してオーバレイネットワークの構造を動的に変化させることで，通信遅延の増加を防ぐ．シミュレーションを行った結果，Latency-aware CAN は従来の CAN に比べて通信遅延が 21 %減少するという結果が得られた．また，物理ネットワークのトポロジが変化しても，通信遅延の増加が抑えられることが示された．

Topologically-aware Distributed Hash Table with Network Coordinates

Toshinori Kojima　Masato Asahara　Ai Hayakawa　Kenji Kono

Department of Information and Computer Science, Keio University
E-mail: {kojima, asahara, aiai}@sslab.ics.keio.ac.jp, kono@ics.keio.ac.jp

Many peer-to-peer systems on the Internet are built up with Distributed Hash Tables (DHTs). By using a DHT, we can lookup the contents on the network quickly and completely. However, since most of the proposed DHTs are independent of the underlying physical network, a well-routed message path on such DHT can result in a long delay due to undesirably long distances in some physical links. In this paper we propose a topologically-aware DHT, called *Latency-aware CAN*, which aims at reducing the lookup routing latency. To build a topologically-aware DHT, Latency-aware CAN constructs a particular kind of CAN whose coordinate space is based on the network coordinate systems. To avoid the lookup routing latency increase due to the change of the physical network topology, Latency-aware CAN adaptively changes its overlay to approximate the current topology of the physical network. Experimental results demonstrate that the lookup routing latency on Latency-aware CAN is 21% less than that on the original CAN. In addition, the results also demonstrate that the latency on Latency-aware CAN increases little even if the topology of the underlying physical network changes.

1 はじめに

近年，インターネットの発展と共に，Peer-to-Peer（P2P）システムが広く利用されるようになってきた．P2P の中には，分散ハッシュテーブル（Distributed Hash Table, DHT）を用いてオーバレイネットワークを構築しているものが数多くある [1, 2, 3]. DHT を用いると，ネットワーク上のコンテンツの所在がハッシュ値により一意に求まるため，検索を漏れなくかつ高速に行うことができる．

通常，オーバレイネットワークはその下位トポロジである物理ネットワークとは独立しており，両者の間には相関がない．そのため，例えばあるノード A, B が，物理ネットワーク上では近いところに存在するのに，オーバレイネットワーク上では遠いところに位置しているように見えてしまうことがある．このような場合，本当なら A, B 間の検索遅延は小さいものであるはずなのに，不必要な経路を通ることで通信遅延が大きくなってしまい，結果としてシステムのパフォーマンスが低下してしまう．

こうした問題を解決するためには，物理ネットワークのトポロジを考慮したオーバレイネットワークを構築する必要がある．それにはまずノード間の通信遅延を正確に把握しなくてはならないが，そのための 1 つの手法として，ネットワーク座標系 [4, 5] がある．ネットワーク座標系では，ネットワーク上に存在する各ノードに一意の座標を与える．そして，この座標間のユークリッド距離が物理ネットワーク上での通信遅延の近似となっている．従って，本来ならば実際に通信を行って測定しなくては分からな

い任意のノード間の通信遅延を，ごく簡単な計算だけで予測することができる．通信遅延が正確に予測できると，ノード間の位置関係を把握することができるので，より効率のよい経路を選択できるようになる．しかし，こうした手法を用いてオーバレイネットワークに物理ネットワークのトポロジを反映させるだけでは不十分である．なぜなら，物理ネットワークのトポロジは動的に変化するからである．この点を考慮していない静的な手法では，時間の経過と共にオーバレイネットワークと物理ネットワークのトポロジの間に徐々にずれが生じ，やがてパフォーマンスが低下してしまう．

本論文では，物理ネットワークのトポロジの変化に対応したオーバレイネットワークの構築手法として，*Latency-aware CAN* を提案する．Latency-aware CAN は，物理ネットワーク上のノードの位置関係を反映した CAN [6] を構築することで，無駄の少ない経路でクエリを転送することができる．さらに，物理ネットワークのトポロジが変化しても，それに合わせて動的にオーバレイネットワークのトポロジを修正するため，常に優れたパフォーマンスを維持することができる．

オリジナルの CAN では，各ノードやコンテンツに与えられる座標は物理ネットワークのトポロジとは無関係であったが，これに対して，Latency-aware CAN ではネットワーク座標系を用いて座標を求めている．こうすることで，物理ネットワーク上における位置関係を反映した CAN を構築している．また，CAN ではユークリッド空間全体を，各ノード間でゾーンと呼ばれる領域に分割して管理しているが，Latency-aware CAN ではさらに，物理ネットワークのトポロジが変化すると，それに対応して各ノードが管理する領域を動的に修正する．

提案手法の有用性を示すため，Latency-aware CAN をネットワークシミュレータ上に実装し，2 つの評価実験を行った．第 1 の実験では，Latency-aware CAN による通信遅延の向上を検証した．その結果，Latency-aware CAN は通常の CAN と比較して，通信遅延が 21 %減少した．第 2 の実験では，時間経過によるトポロジの変化に対する Latency-aware CAN の適応性を検証した．その結果，適応性がないときはトポロジが変化するにつれて検索にかかる通信遅延が線形に増加したのに対し，Latency-aware CAN ではその増加を抑えた．

以下，2 章で本研究の基礎となっている CAN やネットワーク座標系について説明する．続く 3 章で Latency-aware CAN の構造やアルゴリズムについて述べる．4 章では Latency-aware CAN に対して行った評価実験について述べる．5 章で関連研究を紹介し，最後に 6 章で本論文をまとめる．

2 基礎技術

ここでは，Latency-aware CAN の基礎となっている技術について説明する．まず DHT の 1 つである CAN について述べた後，ネットワーク座標系の手法について述べる．

2.1 Content-Addressable Network

Content-Addressable Network（CAN) [6] は DHT の 1 つである．CAN では，各ノード及びコンテンツに N 次元空間内の座標を与え，その座標を用いてコンテンツの検索を行う．CAN では，座標空間全体をゾーンと呼ばれる複数の領域に分割する．このとき，1 つのゾーンの中にただ 1 つのノードが存在するようにし，このノードがそのゾーンの管理ノードとなる．あるゾーン Z を管理するノードは，その座標が Z 内にあるようなコンテンツを保持するほか，Z と隣接するゾーン及びそれらのゾーンを管理するノードについての情報を持つ．クライアントが送信したクエリは，目的のコンテンツに対応する座標を含むゾーンに近づくように，隣接したゾーンを辿って転送される．

図 1 を用いてクライアントのクエリが転送される様子を説明する．例として，foo.html という HTML ファイルが要求されたものとする．foo.html に与えられた座標が (3.0, 0.5) であったとすると，このコンテンツはノード A のゾーンに配置される．クライアントから foo.html を要求するクエリがノード C に送られたとすると，C は隣接するゾーンの中から (3.0, 0.5) に最も近づくゾーンを選択し，そのゾーンを持つノード B にクエリを転送する．クエリを受け取った B は，隣接するゾーンの中に (3.0,0.5) を含むゾーンが存在するので，そのゾーンを持つノードである A に向けてクエリを転送する．

2.2 ネットワーク座標系

ネットワーク上におけるノードの位置関係，すなわちノード間の通信遅延を効率よく把握するための手法として，ネットワーク座標系 [4, 5] がある．ネットワーク座標系は，ネットワーク全体を 1 つのユークリッド空間とみなし，各ノードの位置を座標で表現する．この座標は，座標間のユークリッド距離が実際の通信遅延に近似となるように設定される．従って，ネットワークに参加しているノードは，

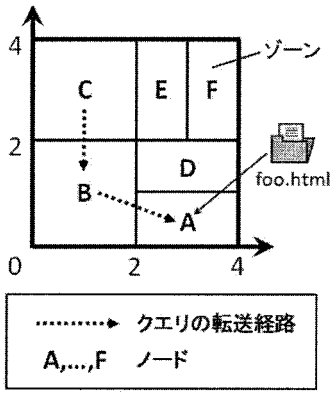


図 1: Content-Addressable Network

一度自身に対応する座標が与えられると，その後はネットワーク上のあらゆるノードとの間の通信遅延を高い精度で予測することができる．2 点間のユークリッド距離を計算するだけでノード間の通信遅延を把握できるので，実測を行う必要がなく，ネットワーク上のトラフィックを増加させることがない．その有用性は，実際にインターネット上で 100 万台以上のマシンを用いた大規模な実験においても示されている [7]．以下では，ネットワーク座標系の例として，Vivaldi について説明する．

2.2.1 Vivaldi

Vivaldi [5] は，物理ネットワークのトポロジの変化に動的に対応できるネットワーク座標系手法である．Vivaldi は，座標間のユークリッド距離と実際の通信遅延との間の誤差を徐々に修正していくことで座標の精度を高い水準に保つ．修正の度に通信遅延を測定する必要があるが，アプリケーションレベルの通信に piggy-back して情報を取得するので，測定のためにトラフィックが増加することはない．

Vivaldi では，あるノード i が別のあるノード j との間の通信遅延の実測値 rtt_{ij} を取得すると，i,j 間のフォースベクトル $\boldsymbol{F}_{ij}$ を以下の式から求める．

$$\boldsymbol{F}_{ij} = (rtt_{ij} - ||\boldsymbol{x}_i - \boldsymbol{x}_j||) \times u(\boldsymbol{x}_i - \boldsymbol{x}_j)$$
$$\text{ただし, } ||u(\boldsymbol{x}_i - \boldsymbol{x}_j)|| = 1$$

ここで，$\boldsymbol{x}_i, \boldsymbol{x}_j$ はそれぞれ i,j の座標を表すベクトルである．そして，i は自身の座標 $\boldsymbol{x}_i$ を以下の式に従って修正する．

$$\boldsymbol{x}_i = \boldsymbol{x}_i + \delta \times \boldsymbol{F}_{ij}$$

ここで，δ は座標の移動の重みづけとなる値である．δ が大きいと座標の収束までの時間が短くなる反面，座標が振動してしまう可能性が高くなる．一方，δ が小さいと座標が振動する可能性は低くなるが，収束まで時間がかかってしまう．Vivaldi では，ノード間のユークリッド距離と実際の通信遅延との間の誤差の大きさに合わせて δ の値を動的に変更することで，座標が素早く収束し，かつ振動してしまうことがないようにしている．

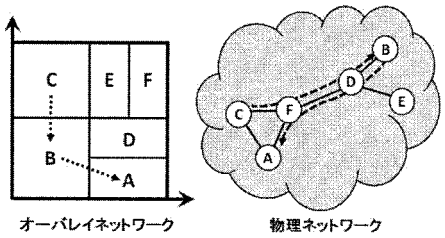


図 2: 無駄な転送経路を通ってしまう例

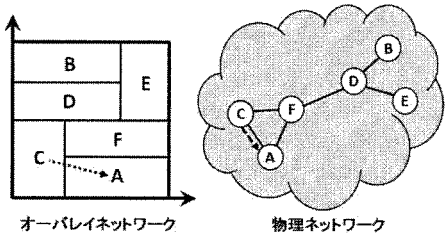
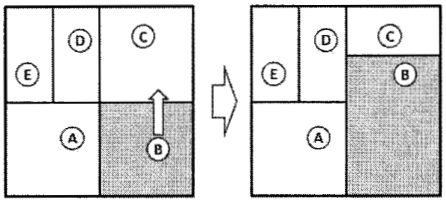


図 3: Latency-aware CAN

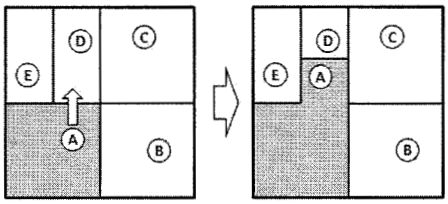
3 Latency-aware CAN

本章では，本論文の提案手法である Latency-aware CAN について述べる．Latency-aware CAN は，代表的な DHT の 1 つである CAN をベースとしている．通常の CAN において構築されるオーバレイネットワークは，物理ネットワークのトポロジを考慮していない．そのためクエリの転送経路に無駄が生じてしまう．図 2 は，無駄な転送経路が生じる例である．ここではノード C からノード A にクエリが送られている．このとき，物理ネットワーク上の位置関係を見ると，C と A は近くに存在しており，通信遅延も小さい．しかし，CAN のオーバレイネットワーク上ではこのクエリはノード B を介して転送されるため，点線の矢印で示したような無駄の多い経路を通ることになってしまう．

Latency-aware CAN では，このような無駄な経路を減らすために，例えば図 3 のようなオーバレイネットワークを構築する．同じようにノード C からノード A にクエリを送る場合，ここでは C と A はオーバレイネットワーク上において隣接しているので，1 ホップで送られる．よって，点線の矢印で示



(a) ノードの移動に合わせて領域を修正する



(b) 修正により領域の形状が複雑化する

図 4: ノードの持つ領域の動的な修正

したような無駄のない経路でクエリを送ることができる．Latency-aware CAN は，ネットワーク座標系手法によって算出された座標を用いて CAN を構築することで，このようなオーバレイネットワークを実現する．さらに，Latency-aware CAN は物理ネットワークのトポロジの動的な変化に対しても対応可能である．トポロジの変化に合わせてルーティングテーブルを動的に修正することで，物理ネットワークとオーバレイネットワークとの間の近似精度を高い水準に保つ．

3.1 エリアとブロックの概念の導入

Latency-aware CAN で各ノードに与える座標は，動的なネットワーク座標系手法である Vivaldi を用いて求める．従って，物理ネットワークのトポロジが動的に変化すると，それに合わせて各ノードの座標が移動する．その移動に応じてノードが管理する領域を動的に修正することで，ルーティングテーブルを常に適切な状態に保つ．具体的には，ノードの座標が，そのノードが管理する領域を外れて他のノードの管理する領域に入ったとき，そのノードとの間で領域の修正を行う．

図 4 の (a) に領域の修正を行っている様子を示す．ここではノード B がノード C の領域に移動したので，B と C の間で領域を修正している．しかし，修正の結果，ノードの管理する領域が複雑な形状になる場合がある．図 4 の (b) にそのようなケースの例を示す．ここでは，ノード A がノード D の領域に移動したので，A と D の間で領域を修正している．

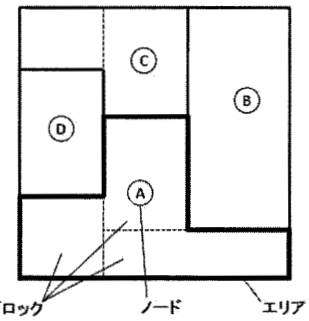


図 5: エリアとブロックによる座標空間の分割

その結果，A の領域がグレーで示した複雑な形状となっている．このような形状の領域は，従来の CAN のゾーンの概念においては存在しえない．そこで，Latency-aware CAN では，エリアとブロックという新たな概念を導入する．

従来の CAN において，ゾーンの形状は必ず超直方体となっている．超直方体とは，2 次元空間では長方形，3 次元空間では直方体といった具合に，直線のみから成る，N 次元空間上の異なる 2 点を決めると一意に定まる図形のことである．一方 Latency-aware CAN では，各ノードが管理する領域の形状は必ずしも超直方体になるとは限らない．しかし，それらの形状は複数の超直方体をつなぎ合わせたものと見ることができる．すなわち，ある 1 つのノードが管理する領域は，1 つ以上の超直方体の集合として表現できる．よって Latency-aware CAN では，1 つのノードが管理する領域全体をエリア，そのエリアを構成するそれぞれの超直方体をブロックと呼び，従来の CAN におけるゾーンの代わりにこの 2 つの概念を用いてルーティングテーブルを構築する．図 5 に，エリアとブロックによる座標空間の分割の例を示す．ここでは，ノード A のエリアは太線で縁取られた領域で，点線で区切られた 3 つのブロックによって構成されている．以下同様に，ノード B, C, D はそれぞれ 1 つ，2 つ，1 つのブロックから構成されている．

3.2 アルゴリズム

3.2.1 エリアの修正

Latency-aware CAN では，通常の CAN と同じように，ノードの座標はそのノードの持つエリアの内部に存在するようにする．よって，移動によりノードの座標が自身のエリアから外れてしまったとき，そのノードは移動後の座標が含まれるようにエリアを修正する．

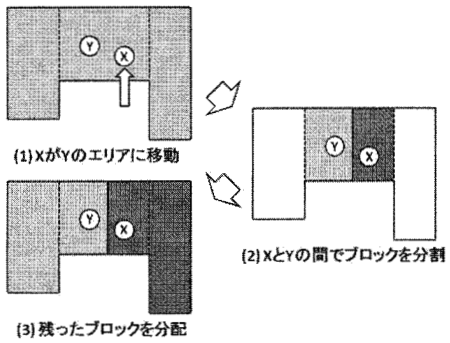


図 6: エリアの分割

ここでは，ノード X の座標 P_X が，X のエリアからノード Y のエリアへ移動したものとする．このとき，まず初めに X は Y に対してエリアの分割を要求する．Y はそれに従い自身のエリアの一部を A_{give} として X に渡す．Y の修正前のエリアから A_{give} を引いたものを A_{stay} とする．このとき，A_{give} には P_X が，A_{stay} には Y の座標 P_Y が含まれるようにする．A_{give} 及び A_{stay} は次のようにして決定する．まず，修正前の Y のエリアにおいて，P_X と P_Y が互いに異なるブロックに含まれていた場合，それぞれが含まれるブロックを A_{give}，A_{stay} に加える．同じブロックに含まれていた場合，そのブロックを，P_X と P_Y が分かれるように分割し，それぞれ A_{give} と A_{stay} に加える．そして，どちらのエリアにも含まれていない残りのブロックを A_{give} と A_{stay} に分配する．図 6 にエリアの分割の例を示す．ここでは，P_X と P_Y は同じブロックに含まれているので，このブロックを分割した後，残りのブロックを分配している．

こうして A_{give} 及び A_{stay} が決定したら，Y は A_{stay} を自身の新たなエリアとすると同時に，A_{give} を X に渡す．これを受け取った X は，A_{give} を自身の新たなエリアとする．このとき，X がそれまで持っていたエリアの処理は，そのエリアと A_{give} が接しているかどうかで異なる．接している場合，それまで持っていたエリアは A_{give} に加えられ，X の新たなエリアの一部となる．接していない場合，それまで持っていたエリアは，そのエリアと隣接するエリアの管理ノードにブロック単位で譲渡される．以上のようにして，エリアの修正は完了する．

3.2.2 同時に移動する場合

特殊なケースとして，X の座標が動いて Y のエリアに入るのと同時に，Y の座標も動いて別のノードのエリアに入ってしまう場合が考えられる．この

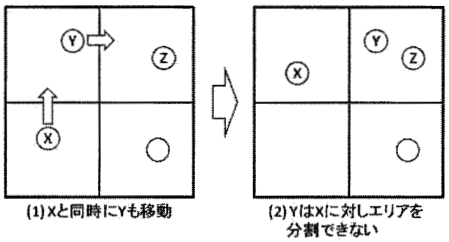


図 7: 移動先のエリアのノードが同時に移動する

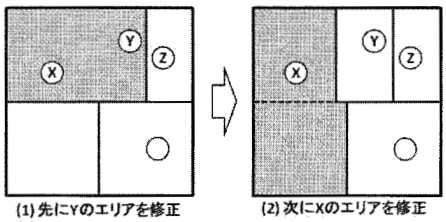


図 8: エリアの修正順を調整する

場合，Y のエリアの修正が先に行われ，その後に X のエリアの修正が行われるのならば問題はない．しかし，もし逆に，X のエリアの修正が先に行われる場合，Y がエリアを分割しようとしても，その段階では一時的に Y の座標は自身のエリアのどこにも含まれていない状態であるため，図 7 のように，Y がエリアを分割できなくなってしまい，不具合が生じてしまう．

この問題を解決するため，Latency-aware CAN では次のような手順を踏む．まずノード X からエリアの分割を要求されたノード Y は，分割を行う前に自身の座標 P_Y が自身の持つエリアに含まれているかどうか調べる．ここでもし含まれていれば，そのまま Y のエリアの分割を行う．もし含まれていなければ，そこで一度 X のエリアの修正を中断し，先に Y のエリアの修正を行う．そして Y のエリアの修正が終わってから，X のエリアの修正を再開し，Y のエリアの分割を行う．このアルゴリズムに従うと，先の例は図 8 のように処理される．ここでは，X が先にエリアの修正を行おうとすると，その前に Y がノード Z との間でエリアの修正を行う．そしてその後に X と Y の間でエリアの修正が行われる．

さらに特殊なケースとして，P_X が Y のエリアに入るのと同時に，P_Y が X のエリアに入る場合が考えられる．上述のアルゴリズムでは，このような場合，X は Y の修正を待ち，Y は X の修正を待つことになり，デッドロックが生じてしまう．これを避けるため，Latency-aware CAN では，このような場

```
if ( 自身の座標 ∉ 自身のエリア ) {
  エリアの分割を要求;
  if ( 返信メッセージが「分割」 ) {
    分割されたエリアを新たなエリアとする;
    if ( 分割されたエリアと以前のエリアが隣接 ) {
      以前のエリアを新たなエリアに加える; }
    else {
      以前のエリアを周辺のノードに譲渡; } }
  else if ( 返信メッセージが「交換」 ) {
    自身の持つ情報を相手ノードに送信;
    返信メッセージから自身の情報を更新; } }
```

(a) 能動側

```
if ( 他ノードから分割要求メッセージを受信 ) {
  if ( 自身の座標 ∉ 自身のエリア ) {
    if ( 循環が発生 ) {
      エリアの交換メッセージを送信;
      返信メッセージから自身の情報を更新;
      return; }
    else {
      自身のエリアを修正; } }
  要求してきたノードに対しエリアを分割; }
```

(b) 受動側

図 9: エリア修正アルゴリズム

合には X と Y の間でエリアを丸ごと交換する．3 つ以上のノード間で修正待ちが循環してしまう場合も，同様に処理する．以上のようなエリア修正のアルゴリズムを，能動側と受動側についてそれぞれ擬似コードにまとめたものを，図 9 の (a) 及び (b) に示す．

4 実験

本章では，Latency-aware CAN の評価のために行った実験について述べる．オーバレイネットワークの開発支援ツールである Overlay Weaver [8] 上に Latency-aware CAN を実装し，シミュレーションにより 2 つの評価実験を行った．

第 1 の実験では通信遅延のストレッチを評価の指標とする．ストレッチとは，1 つのクエリが転送される際の，物理ネットワーク上における最小の通信遅延と，実際にクエリの転送にかかった通信遅延の比である．これはオーバレイネットワークがどの程度物理ネットワークに近似しているかを測る指標となり，近似精度が高いほどストレッチは小さくなる．従って，ストレッチは小さいほどよい．第 2 の実験では，オーバレイネットワークの変化に伴うクエリあたりの通信遅延の変化を見るので，初期状態における通信遅延を基準値 100 として標準化した通信遅延を評価の指標とする．クエリあたりの通信遅延はコンテンツの提供にかかる時間を意味する．すなわち，小さければ小さいほどよい．

4.1 検索にかかる通信遅延の減少

Latency-aware CAN はノードの座標にネットワーク座標系を用いているため，従来の CAN に比べ検索にかかる通信遅延を減らすことができる．これを確認する実験を行った．評価用のネットワークトポロジには Transit-Stub モデル [9] を用いた．Transit-Stub モデルとは，インターネットのような大規模ネットワークを階層的なツリー構造としてモデル化したものである．各種パラメータは，Transit ドメイン数を 228，ドメインあたりの Transit ノード数を 5，Transit あたりの Stub ドメイン数を 4，Stub ドメインあたりのノード数を 2 として，計 9,140 個のノードを生成した．この Transit-Stub モデルは，比較的大規模なバックボーンを備え，末端のネットワークが比較的小規模であるようなトポロジを想定している [10]．用意した Transit-Stub モデルの全ノードに対し，ネットワーク座標系手法を用いて 6 次元座標を与えた．その中から同一の座標をもつノードを除いた 2,862 ノードを選択し実験に使用した．

本実験では，DHT を用いて構築したオーバレイネットワーク上において，ノードを検索するクエリを発行した．クエリは，ランダムに 2 つのノードを選択し，一方のノードがもう一方のノードを要求するものとする．このようなクエリを 6 万件発行し，DHT として CAN を用いたときと Latency-aware CAN を用いたときとでストレッチを比較した．

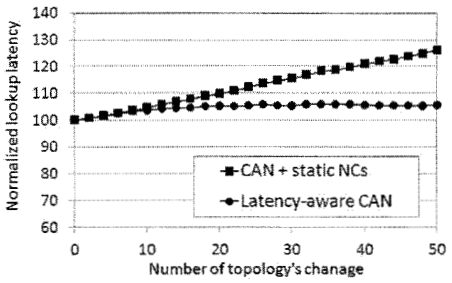
その結果，通常の CAN ではストレッチが 5.40 だったのに対し，Latency-aware CAN では 4.28 と，21 %減少した．この結果より，物理ネットワークのトポロジを考慮している Latency-aware CAN は，考慮していない CAN に比べてストレッチが小さくなっていることが分かる．

4.2 トポロジの変化に対する適応性

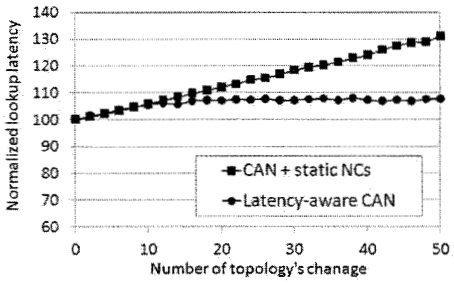
物理ネットワークのトポロジの変化に対する適応性の有無によるパフォーマンスの違いを検証した．比較対象として，静的なネットワーク座標を用いて構築した CAN を用意した．静的なネットワーク座標は，初期状態から座標が移動しない．本実験では，一定時間ごとに物理ネットワークのトポロジが少し

ずつ変化したときの，検索にかかる通信遅延の変化を比較した．適応性の有無によるパフォーマンスの変化のみを比較するため，評価用のネットワークトポロジには次のような仮想的なモデルを用いた．まず，初期状態では 4,096 個のノードが座標空間上において格子状に整列した状態であり，かつノード間のユークリッド距離がノード間の通信遅延に等しい．一定時間ごとにトポロジの変化が起こり，ノード間の通信遅延が変化する．このとき，Latency-aware CAN では通信遅延の変化に伴いノードの座標が移動するが，静的なネットワーク座標は移動しない．なお，実験中にノードの新規参加及び離脱は生じない．以上のような環境下で，トポロジを 50 回変化させながら，検索にかかる通信遅延の変化を測定した．検索の際に発行するクエリは第 1 の実験と同様のものとした．

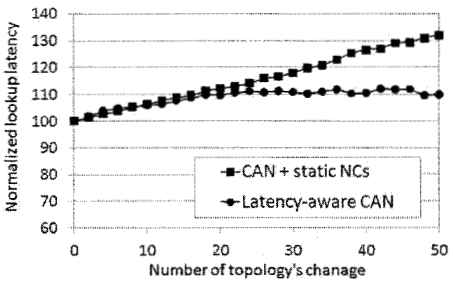
実験結果を図 10 に示す．本実験は次元数 d を変えて 3 通り行った．$d = 2, 3, 4$ のときの結果が，それぞれ (a), (b), (c) である．これらを見ると，静的座標を用いた CAN の通信遅延がいずれもほぼ線形に増加していることが分かる．一方 Latency-aware CAN では，初めのうちこそ静的座標を用いた CAN と同じように増加するものの，途中からはほぼ横這いとなった．$d = 2$ のとき，トポロジが 50 回変化した段階で，静的座標を用いた CAN が初期状態に比べて遅延が 26 %増加したのに対し，Latency-aware CAN では 5 %の増加に留まった．同様に，$d = 3, 4$ のときも，静的座標を用いた CAN がそれぞれ 31 %，32 %増加したのに対し，Latency-aware CAN ではそれぞれ 7 %，10 %の増加に留まった．これらの結果から，トポロジの変化に対し適応性があることによって，通信遅延の増加を抑えることができると言える．



(a) 次元数 2 のとき



(b) 次元数 3 のとき



(c) 次元数 4 のとき

図 10: 次元数を変えたときの通信遅延の推移

5 関連研究

Latency-aware CAN と同じように，CAN においてより効率よくクエリの転送を行うための手法として，SAT-Match [11] や eCAN [12] などがある．また，CAN 以外の DHT をベースにし，物理ネットワークのトポロジを考慮した DHT として Coral [13] や DHash++ [14] がある．

SAT-Match は，実測によりオーバレイネットワークを物理ネットワークに近似させる手法である．SAT-Match では，各ノードが付近のノードとの間の通信遅延を定期的に測定し，その結果に基づいてオーバレイネットワークの構造を動的に変化させる．しかし，修正の度に複数のノードとの間で通信を行わなくてはならないので，ネットワークトラフィックが増加してしまうという問題点がある．Latency-aware CAN でも座標の修正に通信遅延の実測値が必要となるが，その測定は他の通信に piggy-back して行うので，測定のためにトラフィックが増加することはない．

eCAN は，クエリが目的のノードに到達するまでにかかるホップ数を減らすことで通信遅延を短縮する手法である．これは，ホップあたりの通信遅延を減らすことで全体の通信遅延を短縮する Latency-aware CAN とは異なるアプローチ手法であると言える．eCAN は，粒度の異なる複数の階層の CAN を構築することで，クエリのホップ数を削減する．物理ネットワークのトポロジを考慮しないオーバレイネットワークの場合，通信遅延は概ねホップ数に比例するので，ホップ数を減らすことで通信遅延の短

縮が期待できる．eCANは，オーバレイネットワーク上において離れたところにある座標を検索する際には効果が高い反面，近いところにある座標を検索する際には通常のCANとあまり効率が変わらない．Latency-aware CANでは，目的となる座標の遠近を問わず，通常のCANよりも通信遅延を短縮することができる．

Coralは，eCANと同様に，粒度の異なる複数階層のDHTを構築することでホップ数を削減する手法である．ベースとするDHTはChord [15]またはKademlia [16]である．また，各階層でDHTを構築する際，通信遅延が小さいノード同士でクラスタリングすることにより，物理ネットワークのトポロジを反映している．しかし，Coralではトポロジの動的な変化までは考慮していない．また，考慮しようとしても，各ノードが複数のルーティングテーブルを保持しているため，修正にかかるコストはLatency-aware CANよりも大きくなってしまうと考えられる．

DHash++は，Chordをベースとした DHTである．DHash++の各ノードはVivaldiによる座標を持つ．DHash++では，あるノードがクエリを他のノードに転送する際，複数の候補ノードの中から最も座標が近いノードを選択することで，検索にかかる通信遅延を削減する．しかし，Chordの構造自体はトポロジを考慮したものではないので，候補となるノードの中に必ずしも次の転送先として適切なノードが含まれているとは限らない．Latency-aware CANでは，CANの構造自体がトポロジを考慮したものになっており，通常のCANの転送プロトコルに従えば自然と適切なノードに転送されるようになっている．

6 まとめ

本論文では，物理ネットワークのトポロジを考慮し，かつその変化に適応するDHTとしてLatency-aware CANを提案した．Latency-aware CANでは，ネットワーク座標系手法による座標を用いてCANを構築することで，物理ネットワークのトポロジをオーバレイネットワークに反映させている．さらに，トポロジの変化によってその座標が移動しても，それに対応してオーバレイネットワークの構造を動的に変化させることでパフォーマンスの低下を防ぐ．

シミュレーションを行うことによりLatency-aware CANを評価した．その結果，オリジナルのCANに比べ，検索にかかる通信遅延は平均で21 %減少した．また，トポロジの変化に合わせてその構造を動的に変化させることにより，通信遅延の増加を抑えた．

今後の課題として，より現実に即した動的なネットワークトポロジを生成し，そのような環境下における実験を行う必要がある．また，実際にインターネット上で評価できるとよい．

参考文献

[1] Cohen, B.: Incentives Build Robustness in BitTorrent. (2003). http://bitconjurer.org/BitTorrent/bittorrentecon.pdf.

[2] Wang, L., Park, K., Pang, R., Pai, V. and Peterson, L.: Reliability and Security in the CoDeeN Content Distribution Network, *Proceedings of USENIX Annual Technical Conference* (2004).

[3] Freedman, M. J., Freudenthal, E. and Mazières, D.: Democratizing Content Publication with Coral., *Proceedings of Symposium on Networked Systems Design and Implementation(NSDI)*, pp. 239–252 (2004).

[4] Ng, T. S. E. and Zhang, H.: Predicting Internet Network Distance with Coordinates-Based Approaches., *Proceedings of IEEE Infocom*, pp. 170–179 (2002).

[5] Dabek, F., Cox, R., Kaashoek, F. and Morris, R.: Vivaldi: A Decentralized Network Coordinate System., *Proceedings of ACM Special Interest Group on Data Communications (SIGCOMM)*, pp. 15–26 (2004).

[6] Ratnasamy, S., Francis, P., Handley, M., Karp, R. M. and Shenker, S.: A Scalable Content-Addressable Network., *Proceedings of ACM Special Interest Group on Data Communications (SIGCOMM)*, pp. 161–172 (2001).

[7] Ledlie, J., Gardner, P. and Seltzer, M.: Network Coordinates in the Wild, *Proceedings of USENIX Symposium on Networked Systems Design and Implementation(NSDI)*, pp. 299–311 (2007).

[8] 首藤一幸, 田中良夫, 関口智嗣: オーバ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 (ACS 15), pp. 358–367 (2006).

[9] Zegura, E. W., Calvert, K. L. and Bhattacharjee, S.: How to Model an Internetwork., *Proceedings of IEEE Infocom*, pp. 594–602 (1996).

[10] Xu, Z., Tang, C. and Zhang, Z.: Building Topology-Aware Overlays Using Global Soft-Stat., *Proceedings of IEEE International Conference on Distributed Computing Systems(ICDCS)*, pp. 500–508 (2003).

[11] Ren, S., Guo, L., Jiang, S. and Zhang, X.: SAT-Match: A Self-Adaptive Topology Matching Method to Achieve Low Lookup Latency in Structured P2P Overlay Networks., *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2004).

[12] Xu, Z. and Zhang, Z.: Building Low-maintenance Expressways for P2P Systems., Technical report, Hewlett-Packard Laboratories Palo Alto (2002).

[13] Freedman, M. J. and Mazières, D.: Sloppy Hashing and Self-Organizing Clusters, *Proceedings of International Workshop on Peer-to-Peer Systems(IPTPS)*, pp. 45–55 (2003).

[14] Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoes, M. F. and Morris, R.: Designing a DHT for low latency and high throughput, *Proceedings of USENIX Symposium on Networked Systems Design and Implementation(NSDI)*, pp. 85–98 (2004).

[15] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. and Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, *Proceedings of ACM Special Interest Group on Data Communications (SIGCOMM)* (2001).

[16] Maymounkov, P. and Mazières, D.: Kademlia: A Peer-to-peer Infomation System Based on the XOR Metric, *Proceedings of International Workshop on Peer-to-Peer Systems(IPTPS)* (2002).