Vol.2021-DPS-186 No.54
Vol.2021-CSEC-92 No.54
2021/3/16

ビザンチン障害耐性を備えるキー順序保存型構造化オーバーレイネットワークの実現に向けて

寺西 裕一[1,a)]　秋山 豊和[2,b)]　安倍 広多[3,c)]

概要：ビザンチン障害耐性 (BFT) を備えるキー順序保存型構造化オーバレイネットワーク (KOPSON) は幅広い応用が期待できる一方，これまでに現実的な実現方法が示されていない．本研究では BFT を備える KOPSON の実現を目指し，Authority より与えられたキーおよび乱数に基づく構造と，同構造のもと冗長化された経路を構成するルーティング方式を提案する．本稿では提案方式，ならびに，その基礎的な性能の解析について報告するとともに，解決すべき課題について議論する．

キーワード：分散システム，セキュリティ，ビザンチン障害耐性，構造化オーバレイネットワーク

1. はじめに

構造化オーバレイネットワークは，ノード間の自律的なルーティングにより，指定されたキー (名前，値など) をもつノードへ効率的にメッセージを配送可能なアプリケーションレイヤのネットワークである．構造化オーバレイネットワークのうち，任意の全順序集合の要素をキーとし，キーの隣接関係を保存した構造を有するキー順序保存型構造化オーバレイネットワーク (Key-Order Preserving Structured Overlay Network, 以下 KOPSON) [1], [2], [3] は，キーの値の分布に偏りがある場合も $O(\log n)$ (n はノード数）ホップで宛先へ到達可能であり，分散ハッシュテーブル (DHT)，マルチキャスト (ALM)，分散データベース，分散 pub/sub などの幅広い応用がある．ブロックチェーンの下位ネットワークとしての応用の検討も進んでいる [4].

既存の KOPSON は，一般的にノードが停止する障害（ストップ障害）に対する耐性を有するが，決められたアルゴリズムを逸脱した任意の動作による障害に対する耐性，すなわち，ビザンチン障害耐性 (Byzantine Fault Tolerance, 以下 BFT) を有するものは知られていない．このような KOPSON は悪意あるユーザ（攻撃者）からの攻撃に対して脆弱であり，容易に破壊され得る．インターネットなどのオープンネットワークにおいて KOPSON を実用的に運用するには BFT が不可欠である．例えばブロックチェーンを用いる暗号通貨のようなアプリケーションでは損害が生じる攻撃が想定され，BFT が欠如した KOPSON の適用は現実的ではない．本稿では，ビザンチン障害（ストップ障害等を含む）を持つノードを **faulty ノード**と呼ぶ.

キー順序保存型ではない構造化オーバレイネットワークにおいて BFT を実現する方法については，これまでに多数の研究がなされてきた.

既存研究の多くは，複数のノードによって構成されたグループ（Quorum）を用いている [5], [6], [7]．この方式は，Quorum 内のノードで合意を取りながら動作を進めることで一定の割合までの faulty ノードの存在に対処する．しかし，オーバーヘッドが大きく，実装も複雑となる.

また，値が一様に分布するノード識別子をキーとして用い，キーの分布を検証することで不正な動作を検出・回避を行う研究も多い [8], [9], [10]．しかし，この方法はキーの値の分布に偏りがあることを想定する KOPSON には適用できない.

本研究では，BFT を有する KOPSON の実現方式の検討を行った．方式検討にあたっては，実現性を鑑み，信頼の起点となる Authority の存在を前提とした.

本稿では BFT を有する実用的な KOPSON の実現に寄与する次の提案を行う.

- Authority により与えられたキーと α 進数の乱数（ベクタデータ）に基づく構造による KOPSON の構成方式

[1] 情報通信研究機構ネットワークシステム研究所
National Institute of Information and Communications Technology, Network System Research Institute
[2] 京都産業大学情報理工学部
Kyoto Sangyo University, Faculty of Information Science and Engineering
[3] 大阪市立大学大学院工学研究科
Osaka City Uniersity, Graduate School of Engineering
[a)] teranisi@nict.go.jp
[b)] akiyama@cc.kyoto-su.ac.jp
[c)] k-abe@osaka-cu.ac.jp

- 上記構造のもと指定されたキーに収束する冗長化経路を構成するルーティング方式

本稿では，提案手法とその基礎的な解析結果を報告する．

2. 攻撃モデル

本研究では，ビザンチン障害のうち，特に KOPSON に対する次の攻撃を想定する．

Random Next Hop faulty ノードが，メッセージをアルゴリズムによって決められたノードではなく，任意のランダムな宛先へ転送する攻撃 [10]．メッセージの破棄も含む．

Eclipse Attack 標的ノードをオーバレイネットワークから分離させる攻撃．攻撃者が標的ノードの経路表を faulty ノードによって占有する攻撃を含む．Eclipse Attack およびその派生攻撃により，暗号通貨における応用では多重支払いや不正マイニング [11] が可能となることが知られており，対処すべき重要課題である．

Sybil Attack 攻撃者が多数の faulty ノードをオーバレイネットワークに参加させ，正常な動作を妨害する攻撃 [12]．

また，faulty ノードが攻撃対象のノードに対してのみ，あるいはある時間帯のみ攻撃を行い，それ以外は正常に動作する場合も想定する．攻撃者が複数の faulty ノードを操作する，いわゆる結託攻撃も想定する．

提案手法では公開鍵暗号を利用するが，攻撃者は他のノードの秘密鍵を知ることができず，署名の偽造等は行なえないものとする．

3. 提案手法

本研究では上記攻撃モデル・前提のもと実用的に運用可能な KOPSON の構成方式，および，ルーティング方式を提案する．

3.1 Authority

本研究では，信頼の起点となる Authority の存在を前提とする．全てのノードは Authority へアクセスするためのホスト名や公開鍵をあらかじめ知っているものとする．各ノードは秘密鍵・公開鍵を持ち，各ノードの公開鍵には Authority が署名を付与しておく．

ノード v は KOPSON に参加する際，Authority にアクセスし，

- $key(v)$
- $TMV(v)$
- $\langle key(v), TMV(v)\rangle$ に対する Authority の署名

を得る（これら 3 つをまとめて**認証済参加情報**と呼ぶ）．

$key(v)$ はノード v が KOPSON に参加するためのキーである．提案方式では，各ノードは Authority から与えられたキーを用いる想定とし，faulty ノードが意図的にキーを選択することで特定のノードの経路表に影響を与えるといった攻撃はできないものとする．

$TMV(v)$ は，Authority が生成する L 桁の α 進数の乱数である（Trustable Membership Vector(TMV) と呼ぶ）．TMV は KOPSON の構造決定に用いる．L は後述の経路表の高さよりも十分大きいものとする．

ノード間で通信路を確立する際，Authority の署名付き公開鍵と認証済参加情報を交換し，検証する．また，ノード間で送受信されるメッセージには送信元ノードの署名を付与し，受信側は改ざんが無いことを検証する．検証に失敗した場合はメッセージは破棄する．これにより，faulty ノードが Authority により承認されていないキーや TMV を使用して KOPSON に参加することを防ぐ．

Sybil Attack を回避するために，Authority は 1 ユーザに与える認証済参加情報の数を制約する必要がある．そのための方法としては，Authority がユーザのアカウントを管理し，アカウントごとに与える認証済参加情報の数を制限する方法が考えられる．このとき，ユーザがアカウントを無制限に取得できないようにする必要があるが具体的な方法は本研究の対象外とする．

Authority には，(1) キーと TMV の生成を（ノードではなく）Authority が行い，さらに署名を付与することで，KOPSON の構造に制約を課し，攻撃者が経路表を操作する Eclipse Attack を防ぐ，(2) 認証済参加情報の発行をコントロールすることで，攻撃者が多数のノードを参加させる Sybil Attack を防ぐ，という役割がある．なお，認証済参加情報を取得していても faulty ノードである可能性はあることに注意されたい．

3.2 構造

提案する KOPSON の構造について述べる．

3.2.1 定義

全てのノードを含む順序付き集合を N，ノード数 $|N|$ を n と表記する．ノード間の順序関係は各ノードが持つキーの大小によって決まる． 以下，key(v) と v は同義に扱う．

N をキーの順序関係に従ってリング状に並べたものを**リング**と呼ぶ．ここで，最大値のキーを持つノードの正方向に最も近い隣接ノードは最小値のキーをもつノードとする（反対方向も同様）．

また，2 つのノード $v, u \in N$ があるとき，N のリング上で v から数えて u が何個目のノードにあたるかを，v, u の**ノード間距離**と呼ぶ．例えば v, u の間に 2 個のノードが存在するならば，v, u のノード間距離は 3 である．

提案手法は，冗長度に関するパラメータ k を用いる．k は，k 個のノードを無作為に選んだときに，少なくとも 1 個は正常である確率が十分高くなるように選ぶ（$k \geq 1$）．提案手法では，KOPSON を用いるアプリケーションが，キーの近傍 k 個のノード（高確率で正常なノードを少なく

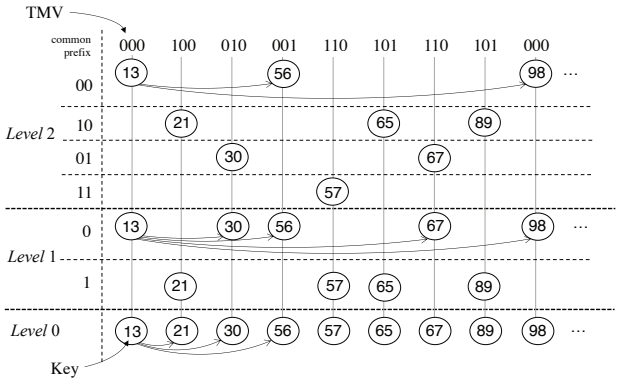


図 1 隣接ノードリストの例 ($k = 3, \alpha = 2$)

とも 1 つ含む）にデータや機能を複製配置することを想定し，宛先として指定されたキー（**目標キー**）の近傍 k 個のノードを検索する機能を提供する．

3.2.2 経路表

各ノードが保持する経路表について述べる．

まず，複数のレベルで階層化されたリングを考える．レベル i（$i \geq 0$）のリングは，TMV の接頭 i 桁が一致するノードから構成される．レベル 0 のリングはすべてのノードから構成される．

各ノード v は，複数のレベルで構成される経路表を保持する．経路表のレベル $i\,(i \geq 0)$ は，正方向および負方向の**隣接ノードリスト** $v.R_i$，$v.L_i$ から構成される．$v.R_i$ は，v が所属するレベル i のリングにおいて，v の正方向に近接する範囲 $(v, w]$ に含まれるノードのリストである．ただし，w は，v から正方向にリングを辿ったときに，v と TMV が接頭 $i+1$ 桁以上一致するノードの $k-1$ 番目である．$v.L_i$ は負方向で同様に考える（範囲は $[w, v)$）．正方向（あるいは負方向）の隣接ノードリスト内の各ノードは v から正方向（あるいは負方向）にキーが近い順に並べておく．また，ノード v における，$v.L_i$ を逆順にしたリスト，v，および，$v.R_i$ を連結したリスト（$\mathrm{rev}(v.L_i) + \{v\} + v.R_i$，ただし rev はリストを逆順としたものを表す）をノード v の**レベル i ノードリスト**と呼び，$v.U_i$ で表す．

隣接ノードリストの各要素を経路表エントリと呼ぶ．経路表エントリは当該ノードのロケータ（IP アドレス，ポート番号など），認証済接続情報，公開鍵を保持する．

ノード v の，レベル i の正方向（あるいは負方向）の隣接ノードリストの最後の要素は，レベル $i+1$ の同じ方向の隣接ノードリストの $k-1$ 番目の要素と等しくなる．**図 1** に例を示す．図はノード 13（キーの値が 13 のノード）の正方向隣接ノードリストが含むノードを矢印で示している（ただし $k = 3, \alpha = 2$）．ノード 13 は，レベル 0 では，レベル 1 で $k-1(=2)$ 個目となるノード 56 まで ($(13, 56]$)，レベル 1 では，レベル 2 で $k-1$ 個目となるノード 98 まで ($(13, 98]$) の各ノードを隣接ノードリストに含んでいる．

この経路表には次の性質がある：任意のノード v のレベル $i\,(i > 0)$ ノードリストから，連続する k 個のノード $\{n_1, \ldots, n_k\}$ を選ぶと，各 $n_j\,(1 \leq j \leq k)$ のレベル $i-1$ ノードリストは $[n_1, n_k]$ の範囲をカバーする（すなわち，$\forall j([n_1, n_k] \subseteq [\mathrm{first}(n_j.U_{i-1}), \mathrm{last}(n_j.U_{i-1})])$，ただし first および last はリストの先頭および最後の要素を表す）．例えば図 1 の，接頭 2 桁 (common prefix) が 00 のレベル 2 リングで連続する $k = 3$ 個のノード $\{13, 56, 98\}$ を考える．これらのノードのレベル 1 ノードリストは $[13, 98]$ の範囲をカバーしている（例えばノード 98 のレベル 1 ノードリストは $[13, 30, 56, 67, 98, \ldots]$）．後で述べるルーティングアルゴリズムでは，この性質を利用して，各ホップで k 個の冗長化された転送経路を形成する．

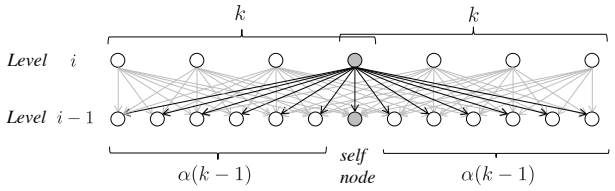


図 2 レベル間の関係

隣接ノードリストは少なくとも $k-1$ 個の要素を含む．レベルが上がるとリング中のノード数は減少するため，いずれノード v の正負各方向の隣接ノードリストの要素に重複が生じる．このレベルを v の経路表の最大レベルとする．

3.2.3 経路表サイズ

以下，上記構造における経路表サイズ（各ノードが保持する経路表エントリの数）を見積もる．

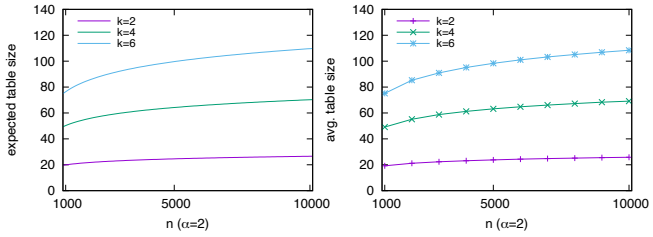
図 2 は，本構造におけるあるノード (図中の self node) が有する経路表のレベル i と $i-1$ の隣接ノードリストの関係を示している．

任意のノード v のレベル $i-1$ のリングに所属する各ノードは，$\frac{1}{\alpha}$ の確率で v のレベル i のリングにも所属する．このため，隣接ノードリストの要素数の期待値は片方向で $\alpha(k-1)$，双方向で $2\alpha(k-1)$ となる（ただし，v のレベル i ノードリストがレベル i のリング全体をカバーする場合はこれよりも小さくなる）．

経路表の最大レベルの期待値を h とする．最大レベルの定義から，h は $v.R_i \cap v.L_i \neq \emptyset$ を満たす最小の i の期待値となる．$v.R_i, v.L_i$ に含まれるノードにおける，v とのノード間距離の最大値の期待値は，$\alpha(k-1)\alpha^i = (k-1)\alpha^{i+1}$ となる．$v.R_i$ と $v.L_i$ における上記期待値の和がノード数 n と等しくなる i において $i \approx h$ となる．$n = 2(k-1)\alpha^{h+1}$ より，$h \approx \log_\alpha \frac{n}{2(k-1)} - 1 = \log_\alpha \frac{n}{2\alpha(k-1)}$ となる．

経路表の高さの期待値は，$h + 1 \approx \log_\alpha \frac{n}{2(k-1)}$ であり，これと各レベルの経路表エントリ数の期待値の積が経路表サイズの期待値となる．すなわち経路表サイズの期待値は，$2\alpha(k-1)\log_\alpha \frac{n}{2(k-1)}$ にほぼ等しい．

提案手法の経路表では，複数のレベルで重複する経路表



(a) 期待値　(b) シミュレーション

図 3　重複を除いた経路表サイズ

エントリが存在する．重複するエントリ間ではデータや通信路は共有できるため，経路表上で重複しないエントリの数（これを**重複を除いた経路表サイズ**と呼ぶ）が，実質的に管理を要するエントリの数となる．

レベル 0 には片方向あたり $\alpha(k-1)$ 個のエントリが存在する．また，隣接するレベル i と $i+1$ の間で重複エントリ数の期待値は片方向あたり $k-1$ 個であり，重複しないエントリ数の期待値は $\alpha(k-1)-(k-1)=(\alpha-1)(k-1)$ となる．このため，正負方向合わせて $2\{\alpha(k-1)+h(\alpha-1)(k-1)\}=2(h(\alpha-1)+\alpha)(k-1)\approx 2(\log_\alpha \frac{n}{2\alpha(k-1)}(\alpha-1)+\alpha)(k-1)$ が，ノードあたりの重複を除いた経路表サイズの期待値となる．α を定数とすると，経路表サイズ，および，重複を除いた経路表サイズのオーダーはいずれも $O(k\log\frac{n}{k})$ となる．

上記を確認するため，提案手法のシミュレータを実装し，ノード数を 1,000 から 10,000 に変化させたときの重複を除いた経路表サイズの平均値を求めた．**図 3** は，$\alpha=2, k=2,4,6$ の時の，重複を除いた経路表サイズの期待値 (a) とシミュレーション結果 (b) を示している．x 軸はノード数，y 軸は重複を除いた経路表サイズである．期待値とシミュレーション結果はほぼ一致した．

3.3　検索

3.3.1　アルゴリズム

与えられた目標キー（s とする）の近傍 k 個のノードを検索するアルゴリズムについて述べる．

任意のノード v の，あるレベル i ノードリストにおける連続する k 個のノード（$n_1, n_2, \ldots, n_k$）を考える．この k 個が s を中央に含む場合（$s \in [n_{\lceil k/2\rceil}, n_{\lceil k/2\rceil+1}$）となるとき**中央に含む**という），3.2.2 節で述べた性質により，これら k 個のノードのレベル $i-1$ ノードリストは $[n_1, n_k]$ をカバーする．また，レベル $i-1$ でこの範囲に少なくとも k 個のノードが存在するため，レベル $i-1$ ノードリストにも s を中央に含む連続する k 個のノードが存在する．このため，あるノード v が s を中央に含むレベル i ノードリストの連続する k 個のノード（$v.M_{i,s}$ とする）にメッセージを送信し，次に $v.M_{i,s}$ の各ノードが s を中央に含むレベル $i-1$ の k 個のノードに送信し…という動作を繰り返すことで，レベル 0 で s を中央に含む k 個のノードにメッセージを送信できる．

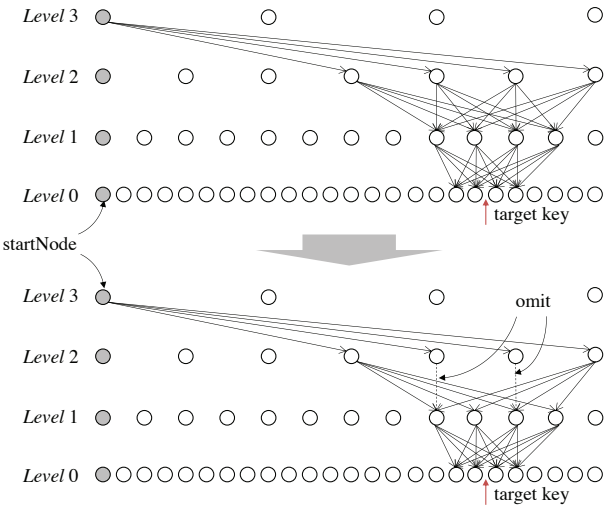


図 4　上図：目標キー (target key) への転送経路 ($k=4, \alpha=2$)，下図：不要な転送経路の省略

$v.M_{i,s}$ の各ノード $n_{i_1}, \ldots, n_{i_k}$ は個々にレベル $i-1$ の転送先の k 個のノードを選ぶが，各 n_{i_j} $(1 \le j \le k)$ が選ぶ k 個のノード集合はすべて同じとなる（$v.M_{i,s}$ の各ノードはレベル $i-1$ でも同一のリングに所属しているため）．このため，$v.M_{i,s}$ のすべてのノードが正常である場合，レベル $i-1$ の k 個のノードはそれぞれレベル i の k 個のノードから同じメッセージを重複して受信する．これによって，$v.M_{i,s}$ の一部が faulty ノードであったとしても，少なくとも 1 つのノードが正常ならばレベル $i-1$ の転送処理を継続できる．

なお，重複して受信したメッセージは転送せずに破棄する．このため，各メッセージには固有の識別子（メッセージ ID）を付与し，各ノードでは受信済みのメッセージ ID を一定期間保持しておく．

図 4 の上図は，startNode から target key（s）へのメッセージの転送経路を示している．レベル 2 で s を中央に含む $k=4$ 個のノードに送信している．また，これら k 個のノードはレベル 1 でより s に近い k 個のノードに転送している（同一ノードに送信していることに注意）．最終的にレベル 0 で s の近傍 k 個のノードにメッセージが配送されることがわかる．

擬似コードを Algorithm 1 に示す[*1]．検索要求を行なうノード (startNode) が，対象キー s と，自身の経路表の最上位レベル+1 を指定して `searchOp` メッセージを自ノードに送信することで検索を開始する．3〜4 行目は，`searchOp` メッセージをレベル $l=0$ として受信した場合の動作である．本ルーティングではレベル 0 のメッセージを受信したノードは，目標キーの隣接 k 個にあたり，検索結果として自ノード v を `resultOp` メッセージを startNode へ返却す

[*1] メッセージの検証および重複メッセージを破棄する処理は省略している．

Algorithm 1: ノード v における検索

```
1  upon receiving ⟨searchOp, startNode, s, l⟩:
2  // s: target key to search, l: level
3  if l = 0 then
4      send ⟨resultOp, v⟩ to startNode
5  else
6      foreach x ∈ v.M_{l−1,s} do
7          send ⟨searchOp, startNode, s, l − 1⟩ to x
```

Algorithm 2: ノード v における検索（転送経路の省略あり）

```
1  upon receiving ⟨searchOp, startNode, s, l⟩:
2  // s: target key to search, l: level
3  if l = 0 then
4      send ⟨resultOp, v⟩ to startNode
5  else
6      for i ← 0 to l − 1 do
7          if v.M_{i,s} exists then
8              foreach x ∈ v.M_{i,s} do
9                  send ⟨searchOp, startNode, s, i⟩ to x
10             break
```

る．6〜7 行目にて，s を中央に含むレベル $l-1$ の各ノードへメッセージを転送している．

通常, 検索が完了すると s の隣接 k 個のノードが `resultOp` メッセージを返却する．ただし，faulty ノードは承認済みの離れたキーを含む偽の `resultOp` メッセージを返却する可能性がある．startNode は，得られた `resultOp` の中から s の近傍 k 個を採用することで，高確率でこのような攻撃の影響から免れることができる（偽の `resultOp` でも Authority が署名したキーしか返却できないため，正常な `resultOp` が少なくとも一つ返却されれば，s の実際の近傍ノードが返すキーのほうが s に近くなる）．

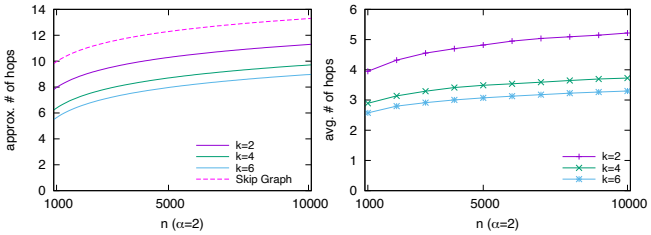
faulty ノードが Random Next Hop 攻撃を行った場合，不要なメッセージの転送によりメッセージ数は増加する．しかし，この場合も，各レベルで転送を行う k 個ノードのうち，少なくとも一つのノードが正常であれば，s の隣接 k 個のノードに辿り着く．

3.3.2 不要な転送経路の省略

ノード v がレベル i からレベル $i-1$ に転送する際，転送先ノードに v 自身が入る場合がある．このとき，レベル $i-1$ の k 個のノードへの転送処理は省略し，レベル $i-2$ での転送に進んで良い（v のレベル $i-1$ の転送先の k 個のノード（$v.M_{i-1,s}$）は，結局 v のレベル $i-2$ の転送先の k 個のノード（$v.M_{i-2,s}$）に転送することになるため）．レベル $i-2$ 以下でも同様のことが言える．このため，v は s を k 個の中央に含む最低のレベルで転送すればよい（なお，v が s のレベル 0 の近傍 k 個に入る場合は，近傍 k 個に転送する）．これによってメッセージ数を削減できる．なお，この省略によって検索成功率が下がることはない．

例えば，図 4 の上図において，レベル 2 の右から 2 つ目と 3 つ目のノードはレベル 1 での転送先に自分自身が含まれている．これらのノードはレベル 1 での転送処理を省略し，レベル 0 で転送処理を行う（図 4 下図）．

Algorithm 2 は上記動作に対応する疑似コードである．Algorithm 1 との主な違いは，6〜10 行目にあり，v は s を k 個の中央に含む最低のレベル i へ転送する動作となっている．



(a) 最大ホップ数の上限 (理論値)　(b) シミュレーション

図 5 ホップ数

3.3.3 ホップ数

検索にかかるホップ数が最大となるのは，最上位レベルから 0 までの全てのレベルで転送が行われる場合である．このため，最大ホップ数は最上位レベルの期待値 h で抑えられる．$h \approx \log_\alpha \frac{n}{2\alpha(k-1)}$ であることから，α が定数とすると，ホップ数のオーダーは $O(\log \frac{n}{k})$ となる．**図 5** は，上記最上位レベルの期待値 (a) とシミュレーション結果 (b) を示している．シミュレーション結果は，始点ノードと目標キーを乱数で選び検索を行った際のホップ数の平均値である（1 回の検索で k 個のホップ数が得られるが，それらすべてを平均している）．試行はノード数の 4 倍の回数行った．また，すべてのノードは正常とした．

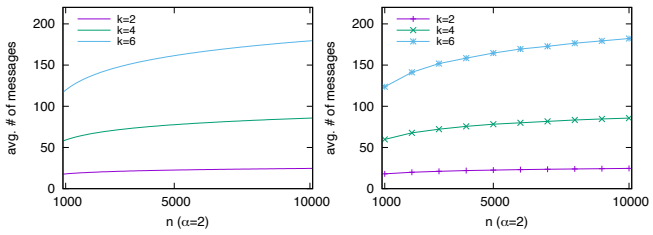
図より，(1) ホップ数 (y 軸) はノード数 (x 軸) の増加に対して対数的な増加に抑えられること，(2)k が増加するほど経路長が短くなること，が確認できる．シミュレーションでのホップ数は h の概ね半分となっているが，これは転送の省略によって転送回数が削減されるためである．

図 5(a) は, 既存の代表的 KOPSON である Skip Graph[1] における平均ホップ数も示している（Skip Graph のメンバシップベクタの基数は 2 としている）．提案方式は Skip Graph よりもホップ数は少なくなっている．

3.3.4 メッセージ数

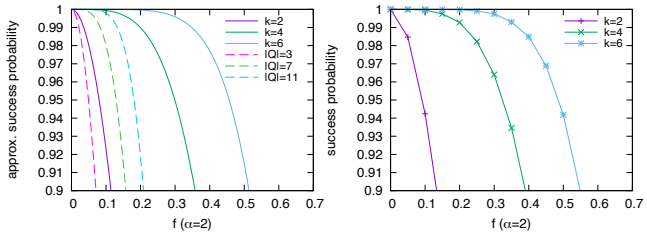
1 回の検索に要するメッセージ数の期待値 m を考える．

レベル i からレベル $i-1$ $(1 \le i)$ への転送では，平均 k/α 個のノードが転送を省略できる．このため平均 $k(1-\frac{1}{\alpha})$ 個のノードから k 個のノードへメッセージが転送される．したがって 1 ホップあたり合計 $k^2(1-\frac{1}{\alpha})$ 個のメッセー



(a) 期待値　(b) シミュレーション

図 6　メッセージ数



(a) 式 (1)　(b) シミュレーション

図 7　検索成功率

ジが転送される．レベル 1 から 0 へは，転送の省略は行われない[*2]．したがって，$m \approx k^2(1-\frac{1}{\alpha})(h-1)+k^2 \approx k^2\{(1-\frac{1}{\alpha})(\log_\alpha \frac{n}{2\alpha^2(k-1)})+1\}$ が成り立つ．α は定数とすると，メッセージ数のオーダーは $O(k^2 \log \frac{n}{k}+k^2)$ となる．

図 6 は，メッセージ数についての上記期待値 (a) とシミュレーション結果 (b) の対比を示している．シミュレーション結果は，上記期待値とほぼ同じ値となった．

3.3.5　検索成功率

あるレベルにおける転送先ノード集合のうち少なくとも 1 つのノードが faulty ノードでなければ，1 ホップのメッセージの転送は成功する．ノード全体ので faulty ノードが占める割合を f とすると，1 ホップの転送に成功する確率は $1-f^k$ である．

ホップ数が増えるにしたがい検索成功率は低下し，i ホップの検索が成功する確率（i ホップ目の k 個のノードのうち少なくとも 1 つがメッセージを受信する確率）は $(1-f^k)^i$ となる．最大ホップ数を h とすると，この時の検索成功率は

$$(1-f^k)^h \approx (1-f^k)^{\log_\alpha \frac{n}{2\alpha(k-1)}} \qquad (1)$$

である．実際のホップ数は転送の省略によって h よりも小さくなるため，平均検索成功率は式 (1) よりも大きくなる．**図 7** はノード数 1,000 の場合の式 (1) とシミュレーションの対比を示している．x 軸は faulty ノードの割合（f），y 軸は検索成功率であり，(a) では式 (1), (b) ではシミュレーション結果をプロットしている．シミュレーションでは，始点ノードと目標キーを乱数で選ぶ検索を 4,000 回試行した際の全経路について，ノードを f に従ってランダムにストップ故障させた場合の検索成功率を測定し，その平均値を求めた．想定した通り，シミュレーション結果は式 (1) と比べやや高い値となった．

$f=0.3$ のとき，$k=2,4,6$ における検索成功率 (シミュレーション結果) は，それぞれ，$0.612, 0.964, 0.997$ であった．k を大きくするに従って高い検索成功率が得られるが，メッセージ数の増加とのトレードオフの関係にある．

図 7 は，文献 [6] の方式に相当する Quorum ベースの構造化オーバレイネットワークにおける検索成功率の理論値も参考値として示している (図中の破線)．図では Quorum のサイズを $|Q|$ と表記している ($|Q|=3,7,11$)．文献 [6] の方式では Quorum の過半数のノードが正常である必要がある．検索成功率を上げるには Quorum のサイズを大きくする必要があるが，faulty ノードの割合が増えると検索成功率を維持することが難しくなる．一方，本ルーティング方式では，k 個のノードのうち少なくとも 1 つが正常であればよく，faulty ノードの割合が多い場合も k を大きくすることで検索成功率を高く保つことができる．本ルーティング方式において $k=4$ とした時の検索成功率は，Quorum において $|Q|=11$ とした時の検索成功率よりも高い．

3.4　ノードの参加と離脱

各ノードは，既に本提案の KOPSON に参加済みのノード (Introducer) を介して同 KOPSON に参加する．Introducer は正常である必要がある．参加ノードは 1 つ以上の Introducer をあらかじめ知っているものとする．

参加ノードを j とする．まず，j は Authority から認証済参加情報 ($key(j)$, $TMV(j)$, $\langle key(j), TMV(j)\rangle$ の署名) を得る．次に j は，Introducer からの検索によってキー j 近傍の k 個のノード集合を得る．j は，得られたノード集合を並び替え，レベル 0 の隣接ノードリスト $j.R_0$ および $j.L_0$ の初期値とする．この段階で，高確率で少なくとも 1 つの正常な j の近傍ノードが経路表エントリに得られた状態となる．

次に j はレベル 0 から経路表を構築する．以下はその動作の概略である．

(1) j は，レベル i $(i \geq 0)$ において，$j.R_i$ および $j.L_i$ の各ノードへ更新メッセージを送信する．
(2) 更新メッセージを受け取ったノードは，全ての経路表エントリを j へ返信するとともに，j をレベル i の隣接ノードリストに追加する．
(3) j は，(2) で得られる経路表エントリを j のレベル i の隣接ノードリストに追加する．
(4) 各ノードは，経路表の各レベルについて，自ノードから近い順に正負双方向に k 個目以降となる経路表エントリを適宜削除する．
(5) j は，(1) から (4) を，レベル i で TMV の接頭 $i+1$ 桁以上一致するノードが，j から近い順に正負双方向に $k-1$ 個得られるまで行う．また，TMV の接頭 $i+1$

[*2] 厳密には自ノードへの転送は省略される．

桁以上一致するノードをレベル $i+1$ の隣接ノードリストに加える．

レベル i の正負の隣接ノードリストに重複するエントリが現れた場合，そのレベルが最上位レベルとなり，経路表構築処理を終了する．

各ノードが，レベル i の隣接ノードリストにノードを追加する際，TMV の接頭 i 桁が自身の TMV と一致すること，追加ノードが正常に通信可能であることを確認する．また，追加ノードが持つ Authority の署名付き公開鍵，認証済参加情報の検証を行う．

ノードが KOPSON を離脱するときは，以下を実行する．

(1) 離脱するノードは，経路表の各エントリに対応するノードへ，離脱通知メッセージを送信する．
(2) 離脱通知メッセージを受信した各ノードは，経路表から対象ノードを削除する．各ノードは，離脱ノードや faulty ノードが経路表から欠落した状態から回復するため，参加時と同様の動作により経路表を更新する．

上記動作では，各レベルの正負方向それぞれの隣接ノードリスト $k-1$ 個のうち少なくとも一つのノードが正常であれば，正しい経路表を得ることができる．

Eclipse Attack では，偽の隣接ノードリストや虚偽の離脱通知メッセージが j に与えられる可能性がある．しかし，本提案では，Authority の承認を得たキーを用いていなければ faulty ノードとして扱われるため，経路表は変更されない．faulty ノードが，承認済みの離れたキーをもつノード情報を不正に j に与える攻撃は可能であるが，正常なノードが与える正しい（キーの値が近い）隣接ノード情報があれば，j の経路表を不正に占有することはできない．

4. 考察

本提案の KOPSON は，キーの値とは別の乱数 (TMV) によって構造を決定する．このため，キーの分布に偏りがあったとしても構造や検索性能に影響を与えない．この KOPSON の性質を実現するための基本的な考え方は Skip Graph と同様である．

提案手法は，冗長化された検索経路を構成可能な経路表を生成する点に特徴がある．冗長経路によって検索成功率は向上するが，経路表のサイズや検索時のメッセージ数は比較的大きい．例えば $k=6, \alpha=2, n=10{,}000$ の時の重複を除いた経路表サイズの平均値は約 108，検索時のメッセージ数の平均値は約 183 であった．これらの削減方法は今後の課題の一つである．

本稿では，単一キーの検索を行う方式を示したが，KOPSON としてはキーの範囲を指定した範囲検索が行えることが望ましい．配信経路の冗長性を保ちつつ効率的な範囲検索を実現する方法も今後検討すべき課題の一つである．

また，提案方式では，各ノードは Authority からキーおよび TMV の認証を受ける必要があり，基本的には各ノードが自由にキーを決定できない．pub/sub 等の KOPSON の応用を考慮した場合，この点は改善すべき点となる．

また，3.4 節で述べたノードの参加手順は，初期検討段階のものであり，改善の余地がある．例えば，経路表更新時の隣接ノードへの問い合わせの並列化，キャッシュ等を活用した問い合わせ回数の削減，検証結果の再利用などの改善が考えられる．

BFT の観点での課題としては，リング上で faulty ノードが偶然 k 個以上連続すると，構造やルーティングの動作を正しく維持できなくなる課題があげられる．また，faulty ノードの挙動によっては大きなオーバヘッドが生じうる課題もある．後者の課題に対しては，例えば，Random Next Hop 攻撃が行われた場合，メッセージを受信したノードが，目標キーと自ノードのキーを比較する等により不正動作を検出できれば，以降の不要なメッセージ転送を行わずに済ませられる可能性がある．こうしたオーバヘッドの削減方法や，faulty ノードが連続する場合への対処方法は今後の検討課題である．

提案手法から Authority と証明書関連の処理を削除し，キーと TMV は各ノードで生成するように変更することで，ビザンチン障害を含まない一般的な障害に対して耐性がある KOPSON を実現できる．これは BFT を考慮しなくてもよいクローズドな環境に適している．

5. 関連研究

キー順序保存型ではない構造化オーバレイネットワークにおいて BFT を実現する方法として，Quorum ベースの方式に関する研究が数多くなされてきた [5][6][7]．Quorum ベースの方式では，複数のノードによって Quorum と呼ばれる集合を形成する．1 つの Quorum を構成するノードは同一の経路表を保持し，これらのノードの多数決合意によって構造化オーバレイネットワークのアルゴリズムのステップを進める．この方式では，Quorum 内の faulty ノードが一定の割合を超えない限り，正常なノードは faulty ノードの影響を受けない．しかし，この方式は，Quorum 内の合意を得るために Quorum を形成するノード間で同期しながら動作するオーバヘッドがあり，ルーティングにかかる時間も長くなる．また，Quorum ベースの方式では，Quorum のメンバシップ管理，Quorum 内部での多数決処理，Quorum 間の通信等が必要であり，実装が複雑となる．これに対し，提案方式はそのようなオーバヘッドがなく，かつ，比較的シンプルに実装できる利点がある．

Quorum の実現手法として，文献 [9] は，Cuckoo rule と呼ばれるルールによる Quorum の構成方法を提案している．Cuckoo rule は，キーの値が一様分布に従うことを前提に，隣接ノードの位置を調節する．この方法は，ルールを満たすまで繰り返す必要があり，オーバヘッドが大きい．また，一様分布に従わないキーを利用する KOPSON に適

用できない．

ノードの動作を相互に監視して評価値を与え，評価が低いノードをブラックリスト等に管理することで BFT を実現する方式も提案されている [10]．しかし，この方式は正常な動作と不正な動作を繰り返す faulty ノードへの対処が難しい．一方，提案方式は，同時刻に一定の割合で正常な動作をするノードが存在すれば良く，上記の faulty ノードにも対応可能である．

Sybil Attack に対する対策として，hash puzzle によってノードの識別子を取得することを防止する方法が提案されている [13]．この方法によって，分散環境において，少ない攻撃者が大量のキーを取得することは困難となる．hash puzzle は，計算量が大きい処理であり，膨大な電力消費が問題となり得る．提案方式では Authority がキーの承認を行うため，このような対策は不要となっている．

Eclipse Attack を回避する方法として，キーの値が一様分布に従うことを前提に経路表エントリに制約を設ける方法が提案されている [8], [10]．しかし，この方法は一様分布に従わないキーを利用する KOPSON に適用できない．

冗長化によってメッセージ到達性を担保する方法として，KadCast[14] がある．KadCast は，Kademlia に基づくオーバレイネットワーク構造を用いてメッセージの転送を冗長化するブロードキャスト手法である．KadCast は Kademlia の構造に基づいて冗長化するため，障害耐性は Kademlia に依存する．Kademlia は障害に対する耐性が比較的高く，ある程度のビザンチン障害耐性を実現する Kademlia の実装もある．しかし，Sybil Attack を想定したビザンチン障害耐性が十分とは言えない．例えば，Ethereum[15] が用いている Kademlia[16] アルゴリズムに基づく構造化オーバレイ実装では，Eclipse Attack を回避するため，同一サブネットワーク内のノードを隣接ノードとする数に上限を設けるなどの制限を設けている [17]．しかし，攻撃者同士が結託し，faulty ノードが複数のサブネットワークに分散して存在する場合に対処できない．

6. おわりに

本稿では，ビザンチン障害耐性を備えた新しい KOPSON の実現方式を提案した．提案は，Authority による参加ノードの認証を前提とし，faulty ノードの影響を受けにくい冗長経路を含むオーバーレイの構成法，冗長経路を用いたルーティング方式，ノードの参加離脱方法を含む．アルゴリズムは比較的単純で，実装は容易である．

4 章にて述べた通り，本提案は検討の途上にあり，実用的な手法としていくには，さらなる検討・検証が必要である．今後，上記検討を進めるとともに，プロトタイプ設計・実装を行ない，分散ハッシュテーブル，pub/sub メッセージング，ブロックチェーン等の各種応用への適用性も検証していく．

謝辞 本研究は JSPS 科研費 JP20H04186 の助成を受けたものである．ここに記して謝意を表す．

参考文献

[1] Aspnes, J. and Shah, G.: Skip graphs, *ACM Transaction on Algorithms*, Vol. 3, No. 4, pp. 1–25 (2007).
[2] Schütt, T., Schintke, F. and Reinefeld, A.: Range queries on structured overlay networks, *Computer Communications*, Vol. 31, No. 2, pp. 280–291 (2008).
[3] Abe, K. and Teranishi, Y.: Suzaku: A Churn Resilient and Lookup-Efficient Key-Order Preserving Structured Overlay Network, *IEICE Transactions on Communications*, Vol. E102.B, No. 9, pp. 1885–1894 (2019).
[4] Hassanzadeh-Nazarabadi, Y., Küpçü, A. and Özkasap, Ö.: LightChain: A DHT-based Blockchain for Resource Constrained Environments, *arXiv preprint arXiv:1904.00375* (2019).
[5] Naor, M. and Wieder, U.: A Simple Fault Tolerant Distributed Hash Table, *Proc. of International Workshop on Peer-to-Peer Systems*, pp. 88–97 (2003).
[6] Fiat, A., Saia, J. and Young, M.: Making Chord Robust to Byzantine Attacks, *Proc. of European Symposium on Algorithms*, pp. 803–814 (2005).
[7] Young, M., Kate, A., Goldberg, I. and Karsten, M.: Towards Practical Communication in Byzantine-resistant DHTs, *IEEE/ACM Transactions on Networking*, Vol. 21, No. 1, pp. 190–203 (2012).
[8] Puttaswamy, K. P., Zheng, H. and Zhao, B. Y.: Securing Structured Overlays Against Identity Attacks, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 20, No. 10, pp. 1487–1498 (2008).
[9] Awerbuch, B. and Scheideler, C.: Towards a Scalable and Robust DHT, *Theory of Computing Systems*, Vol. 45, No. 2, pp. 234–260 (2009).
[10] Needels, K. and Kwon, M.: Secure Routing in Peer-to-Peer Distributed Hash Tables, *Proc. of the 2009 ACM Symposium on Applied Computing*, pp. 54–58 (2009).
[11] Nayak, K., Kumar, S., Miller, A. and Shi, E.: Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack, *Proc. 2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 305–320 (2016).
[12] Douceur, J. R.: The Sybil Attack, *Proc. of International Workshop on Peer-to-Peer Systems*, pp. 251–260 (2002).
[13] Borisov, N.: Computational Puzzles as Sybil Defenses, *Proc. of the 6th IEEE International Conference on Peer-to-Peer Computing (P2P'06)*, pp. 171–176 (2006).
[14] Rohrer, E. and Tschorsch, F.: Kadcast: A Structured Approach to Broadcast in Blockchain Networks, *Proc. of the 1st ACM Conference on Advances in Financial Technologies*, pp. 199–213 (2019).
[15] Wood, G.: Ethereum: A Secure Decentralised Generalised Transaction Ledger, *Ethereum Project Yellow Paper*, Vol. 151, No. 2014, pp. 1–32 (2014).
[16] Maymounkov, P. and Mazieres, D.: Kademlia: A Peer-to-Peer Information System based on the XOR Metric, *Proc. of International Workshop on Peer-to-Peer Systems*, pp. 53–65 (2002).
[17] Henningsen, S., Teunis, D., Florian, M. and Scheuermann, B.: Eclipsing Ethereum Peers with False Friends, *Proc. of 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 300–309 (2019).