Vol.2016-AL-157 No.4
2016/3/6

通信リンク障害に対応する動的コンパクトルーティング方式

光野 泰弘[1]　山田 敏規[1]

概要：ルーティング方式の中で、各ノードに格納する経路情報の大きさが $o(n)$ であるものをコンパクトルーティング方式といい、そのなかでもネットワークトポロジの変化に対応したものを動的コンパクトルーティングという。既存の方式としてメモリサイズ $O(\log n)$ の任意ノードの追加に対応した動的木に対するコンパクトルーティングが Korman[7] により示されている。本稿では，この Korman によるルーティング方式を任意リンクの削除にも対応するように拡張したルーティング方式を示す。

Link Failure Tolerant Dynamic Compact Routing Scheme

Yasuhiro Mitsuno[1]　Toshinori Yamada[1]

1. はじめに

一般にルーティング方式は, ネットワークの各ノードへの個別ラベル (アドレス)・個別ポート番号の割り当て, 各ノードへの経路情報の格納, メッセージの転送手順によって構成される．この経路情報が $\Omega(n)$ であるような方式ではネットワークのノードが増大した際に情報量も膨大になってしまう．そのため，各ノードでの経路情報の大きさを $o(n)$ で抑えることでネットワークのノード数の増大に耐えうるコンパクトルーティングの研究がなされている [3].

ネットワークトポロジの動的変化のない静的コンパクトルーティング方式は非常に多くの提案がなされており，メモリ量の下界やストレッチとのトレードオフも示されている．一方で，動的コンパクトルーティング方式はあまり提案がされておらず，その研究はまだ発展途上である．

先行研究として [1], [6], [8] で動的コンパクトルーティング方式が示されているが，いずれもルーティング正確性の保証は静的な振る舞いのときのみである．そのなかでルーティングの正確性がすべての時間において保証される方式が Korman[7] によって示されている．しかし，この方式もリンクの削除などには対応できていない．そこで本研究ではより多くのトポロジ変化に対応できる動的コンパクトルーティング方式の提案を目指している．

[1] 埼玉大学
Saitama University

2. 動的コンパクトルーティング

コンパクトルーティング方式には各ノードに対しラベルを割り当てることのできるラベリング方式と各ノードに予めラベルが割り当てらている名前非依存方式が存在する．また, 各リンクに対するポート番号の割り当てが可能な設計ポートモデルとポート番号が予め割り当てられている固定ポートモデルの二つが存在する.

本稿ではラベリング方式かつ設計ポートモデル，およびヘッダ情報の書き換えが可能であることを前提としている.

また，動的コンパクトルーティング方式の評価基準は以下である.

- ラベルサイズ
- テーブル (経路情報) サイズ
- ストレッチ
- メッセージ複雑性
- 対応するトポロジ変化
- ルーティング正確性の保証

メッセージ複雑性とは，トポロジ変化が起こった際に発生するメッセージ量であり，正確性の保証とは，トポロジ変化のシナリオのどの時刻においてルーティングが正しく行われるかの保証である.

3. 準備

本稿では，ネットワークを無向連結グラフ $G = (V, E)$ と

してモデル化する．ここで，V は頂点（ノード）集合であり，E はノード間の双方向通信リンク集合を示している．ネットワーク G 上に構成される任意ノード r を根とする全域木を $T = (V, E')$ とする．このとき，ノード v の T 上の隣接ノードのうち，v と r を結ぶパス上にあるノードを v の親ノード，それ以外のノードを子ノードと呼ぶ．また，ノード v の親ノードを $p(v)$ で表す．

木 T における子孫および祖先とは親子関係の推移的閉包である．v を含んだ v の子孫ノードからなる T の部分木を T_v で表し，部分木 T_v の重さ $\omega(v)$ を T_v 内のノード数つまり $\omega(v) = |T_v|$ とする．また，ノード v の兄弟ノードとは，$p(v)$ の v 以外の子ノードである．

4. 動的木に対するコンパクトルーティング

この章では，提案ルーティング方式のもととなる Korman[7] による動的木に対するコンパクトルーティング方式について示す．

静的時の振る舞いである系統ルーティング方式 [5] と，ノードの追加によりバランスが崩れた際の更新処理および転送手順について示す．

4.1 系統ルーティング

系統ルーティングとは，系統ラベルと heavy・light の割当ての二つの要素からなるラベルサイズ $\Theta(\log n)$ のルーティング方式である．ここで，系統ラベルとは以下の子孫判定を簡単に行うことができるラベルである：

子孫判定

u と v のラベル $L(u)$ と $L(v)$ が与えられたとき，v が u の子孫であるか否かを判定する

このような子孫判定を簡単に行うことができるラベルはアドレス区間の割当てによって実現される（図 1）．これにより，宛先ラベルが自身のラベルの区間内か否かで自身の子孫か判定することができる．また，同様に宛先がどの子ノードの子孫であるかを子ノードの系統ラベルにより判定することができる．

しかしながら，ノード v が全ての子ノードの系統ラベルを保持しようとすると，v のテーブルサイズが $\Theta(\deg(v) \cdot \log n)$ となりコンパクトにならない可能性がある．そこで，子ノードの中で最も重い（子孫ノードの数が多い）子ノードを heavy 子ノードとし，それ以外の子ノードを *light* 子ノードとする heavy・light 木（図 2）を構築し，v は heavy 子ノードの系統ラベルのみを保持し，v から heavy 子ノードへの通信リンクのポート番号を 1 に，light 子ノードへの通信リンクのポート番号を 2 から $\deg(v)$ にそれぞれ設定する．各ノード v から親への通信リンクのポート番号は 0 とする．

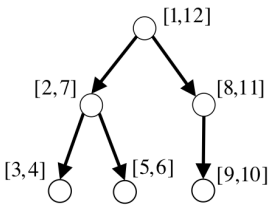

図 1 系統ラベルの割り当て

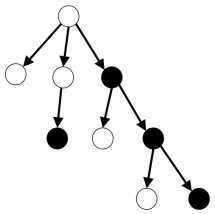

図 2 heavy・light 木

これにより系統ルーティングでは，各ノード v のラベルとして $L(v) = \langle L_{anc}(v), L_{heavy}(v), L_{light}(v) \rangle$ を用いる．ここで，$L_{anc}(v)$ は v 自身の系統ラベルを，$L_{heavy}(v)$ は v の heavy 子ノード $H(v)$ の系統ラベルを表し，$L_{light}(v)$ は以下によって定義されるポート番号の列である：

$$L_{light}(v) = \begin{cases} \varepsilon(\text{空列}) & (v \text{ が根であるとき}) \\ L_{light}(w) & \\ \qquad (v \text{ が } w \text{ の heavy 子ノードであるとき}) & \\ L_{light}(w) \circ Port(w, v) & \\ \qquad (v \text{ が } w \text{ の light 子ノードであるとき}) & \end{cases}$$

ここで，$Port(w, v)$ はノード w でのノード v への通信リンクのポート番号である．

4.2 ノード追加への対応

ネットワークに新たなノード z が追加された際には，[7] に示されるメッセージ複雑性 $O(n \log n)$ の方式により系統ラベル $L_{anc}(z)$ を割り当てる．また，heavy ラベル $L_{heavy}(z)$ 及び light ラベル $L_{light}(z)$ は小節 4.3.3 で示す **Shuffle** 手続きをシミュレートする事により割り当てが行われる．

4.3 更新処理

ノードの追加を繰り返すことにより，heavy・light 木のバランスの崩れることが起こりうる．すなわち，あるノード u のある light 子ノード v の子孫にノードが追加されることにより，v の heavy 子ノードよりも極端に重くなる状況が発生する．この場合，バランスの崩れた部分木に対してラベルの再構築を行うことで heavy・light 関係を修復する．この再構築を行うための三つの手続きについて示す．

4.3.1 Procedure Prepare_Subtree

各ノード v で保持しているカウンタ $c_q(v)$ が 2^{q+2} を超えたときに $T_{light}(v) = T_v - T_{H(v)}$ に対してブロードキャストによりノードの追加を許さない凍結状態にする．その後 $T_{light}(v)$ に対して **Shuffle** 手続きを実行する．

4.3.2 Procedure Find_Pivot

ノード追加の際に新たなノード z へ割り当てる light ラベル $L_{light}(z)$ のサイズが $2^5 \log n'_0$ を超えるときに実行される．ここで，n'_0 は前回全体木で **Shuffle** 手続きが行われたときの総ノード数である．

この手続きでは，ピボットと呼ばれる light ラベルのサ

イズが $2^4 \log n'_0$ 以下であり heavy・light の崩れた部分木の頂点 ρ を条件に従い z からアップキャストによって探しだす．このピボットを根とする部分木 T_ρ に対して **Shuffle** 手続きを実行する．

4.3.3 Procedure Shuffle

Shuffle 手続きを行うにあたり以下の要素がラベル L に追加される．

追加されるラベル

- ポートラベル: L_{port}（cL_{port} と fL_{port} で構成）
- future ラベル: fL_{light} と fL_{heavy}
- ファーストビットラベル：β
- ビジーラベル：L_{busy}

根 ρ の部分木 T' に対する **Shuffle** 手続きは以下の通りである．

まず，根 ρ からのブロードキャストと ρ へのアップキャストを行うとこにより，以下のことが保証される．

保証される内容

(1) 葉でないノード $w \in T'$ は重さ最大の子ノード $fH(w)$ への（現在の）ポート番号を知っている．
(2) 各ノード $w \in T'$ は，重さ $\omega(w)$ と新しい light 重み $f\omega_{light}(w)$ を知っている．
(3) 各ノード $w \in T'$ の全てのカウンター $c_q(w)$ は 0 である．

ρ で **Shuffle** のためのエージェントが作られ，エージェントは各ノード $w \in T'$ において現在のポート番号が大きい順に DFS ツアーを行う．

(1) ノード w は，エージェントが到達したら直ちに $L_{busy}(w) \leftarrow 1$ とする．

(2) エージェントがノード w から w の新しい light 子ノード z へ向かうとき，$cL_{port}(w) \leftarrow Port(w,z)$，$fL_{port}(w) \leftarrow 2^{q+2} + c_q(w)$ として，$c_q(w)$ の値を 1 増やす．ここで，q は $w \in J_q(f\omega_{light}(w))$ を満たす整数で，$J_q(m) = \left[\frac{m}{2^{q+1}}, \frac{m}{2^q}\right)$ である．また，このときポート番号の最下位ビットには β が付加される．**Shuffle** 手続きを通じて $0 \le c_q(w) < 2^{q+1}$ であるので，$2^{q+2} + c_q(w) \in I_q$ を満たす．ここで，$I_q = \left[2^{q+2}, 2^{q+3}\right)$ である．

ただし，$T' = T(u) \cup p$ であるときは ρ が $f\omega_{light}(\rho)$ の値を知る保証がないため，代わりに前回計算された ρ の light 重み $\omega^0_{light}(\rho)$ を用いる．

(3) $fL_{light}(w)$ を以下のように設定する：

$$fL_{light}(v) \leftarrow \begin{cases} fL_{light}(p(w)) \\ \qquad (w = fH(p(w)) \text{ のとき}) \\ fL_{light}(p(w)) \circ fL_{port}(p(w)) \\ \qquad (w \neq fH(p(w)) \text{ のとき}) \end{cases}$$

ただし，$p(w)$ は w の親を表す

(4) エージェントが子ノード z から親 $w \in T'$ へ戻ってきたとき，$Port(w,z) \leftarrow fL_{port}(w)$ とする．

(5) エージェントが全ての子ノードを訪れた後に $w \in T'$ へ戻ったとき，以下の 5 つを実行する：

- $w \neq \rho$ であるならば $L_{light}(w) \leftarrow fL_{light}(w)$ とする．
- $H(w) \leftarrow fH(w)$ とする．
- $Port(w, H(w)) \leftarrow 1$ とする．
- w の新ラベルを空にする．
- ファーストビットラベル β をビット反転する．

4.4 転送手順

ノード u が宛先ラベル $L(v)$ を受け取ったとき，以下の転送手順に従いメッセージの転送を行う．

転送手順 1

宛先 $L(v) = L(u)$ のとき，ルーティングは完了．
系統ラベルにより v が u の子孫であるかどうか判定．
(1) 子孫でない場合．ポート 0 へ転送．
(2) $H(u)$ の子孫である場合，ポート 1 へ転送．
(3) その他
  (a) u のビジーラベルが 0 の場合．
    $Ext(L_{light}(u), L_{light}(v))$ へ転送
  (b) u のビジーラベルが 1 の場合．
    転送手順 2 に従う

転送手順 2

(1) v が $fH(u)$ の子孫である場合．
  $fL_{heavy}(u)$ のポート番号へ転送
(2) $fL_{light}(v)$ が空でない場合．
  $cL_{port}(u)$ のポート番号へ転送
(3) $fL_{light}(v)$ が空の場合．
  (a) v のビジーラベルが 0 の場合．
    $Ext(L_{light}(u), L_{light}(v))$ へ転送
  (b) v のビジーラベルが 1 の場合．
    $Port \leftarrow Ext(fL_{light}(u), L_{light}(v))$
    ($fL_{light}(u)$ が空のときは $L_{light}(u)$ を用いる)
    $Port = fL_{port}(u)$ のとき，$cL_{port}(u)$ へ転送
    それ以外のとき，$Port$ へ転送

ここで，ポート番号の抽出関数 Ext は

$$Ext(p_1 \circ \ldots \circ p_i, p_1 \circ \ldots \circ p_i \circ p_{i+1} \circ \ldots) = p_{i+1}$$

である．

Korman のルーティング方式がメッセージを常に正しくルーティングすることは [7] の第 5 節により示されており，以下の補題を得る．

補題 1. *Korman のルーティング方式は常に正しくメッセージを転送する [7].*

5. 提案ルーティング方式

前章で示した Korman[7] による動的木に対するコンパクトルーティング方式を一般トポロジへ拡張し，リンクの削除に対応させたルーティング方式を示す．復旧の手順及びメッセージの転送手順について示す．

5.1 一般トポロジへの拡張

木構造に対するルーティング方式から一般トポロジに対するルーティング方式へと拡張させるために，ポート番号に以下のビットを付加する．

α ビット　全域木を構成するリンクか判定するビット．
β ビット　**Shuffle** 手続きで使用するビット
γ ビット　復旧用のリンクを区別するためのビット．

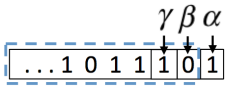

図 3　ポート番号

ノードがリンクへ割り当てるポート番号は割り当てる番号と α, β, γ のすべてを含んだビット列であるが，light ラベルへ連結するポート番号は図中の破線で囲んだ α を取り除いたビット列となる．また，light ラベルは可変長であり，単純なポート番号の連結ではポート番号の抽出ができないため，図 4 のようにポート番号ごとの切れ目を判定する為のビット列が付与される．

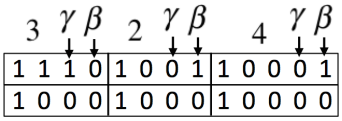

図 4　light ラベルのビット構造（3 ∘ 2 ∘ 4 の例）

5.2 復旧情報の構成・格納

ネットワークに新たな辺 (u, w) が追加されたとする．この時，ノード u と w でそれぞれ辺 (u, w) に対して固有のポート番号 $Port(u, w)$ と $Port(w, u)$ を割り当てる．

このとき，ノード u と w はともに u のラベル $L(u)$，ポート番号 $Port(u, w)$，w のラベル $L(w)$，ポート番号 $Port(w, u)$ を u と w の最も近い共通の祖先に向かってアップキャストをする．アップキャスト中のノード x は，$L_{anc}(x)$ と $L_{anc}(u)$ と $L_{anc}(w)$ を用いて u と w の共通の祖先であるか否かを判定し，共通の祖先であるならばアップキャストを終了する．ことのきのノードが最も近い共通の祖先（NCA: Nearest Common Ancestor[2]）であり，これを $nca(u, w)$ とする．

u と $nca(u, w)$ の間のノード v（ただし $nca(u, w)$ を除く）は以下で構成される復旧情報 $R(v)$ を持つ：

復旧情報 $R(v)$ の構造

- v のラベル L_v と v 重さ $\omega(v)$
- w のラベル L_w，ポート番号 $Port(w, u)$
- u のラベル L_u，ポート番号 $Port(u, w)$

ただし，復旧情報をすでに持っている場合には．復旧情報の書き換えは行わない．

また，v の兄弟ノードにも $R(v)$ を持たせる．すなわち，あるノードの子ノードをポート番号の昇順に $v_1, v_2, \ldots, v_m$ とすると，v_i は $R(v_i)$ と $R(v_{(i \bmod m)+1})$ を格納する（図 6）．ただし，$m = 1$ の時には $p(v_1)$ に $R(v_1)$ を持たせる．

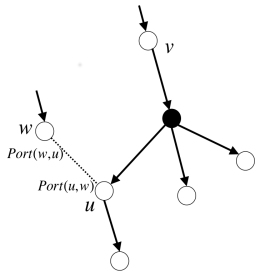

図 5　復旧情報

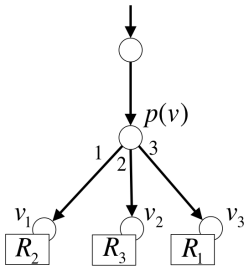

図 6　復旧情報の格納方法

5.3 復旧の方法

全域木のあるリンクが切断された際に全域木を復旧する手順について説明する．復旧の流れは以下のようになっており，それぞれのステップについて説明を行う．以下ではノード v とその親ノード $p(v)$ の間のリンクが切断されたと仮定する．

復旧の流れ

(1) 手続きの中断
(2) 新系統ラベルの付与
(3) 新ラベル・ポートの付与
(4) ラベル・ポートの切り替え
(5) 更新済みラベルを削除
(6) **Shuffle** 手続き
(7) 手続きの再開

ノード v とその親ノード $p(v)$ とのリンクが切断された際には，$p(v)$ は v の復旧情報 $R(v)$ を格納されている場所，v の兄弟ノードから取得を行う．ただし，$m = 1$ のときには $p(v)$ が $R(v)$ を持っている．また，v の次兄弟ノード v^+ の復旧情報 $R(v^+)$ を v の前兄弟ノード v^- に書き込む（$R(v)$ に上書きする）．

(1) 手続きの中断

$p(v)$ および v が全ノードに対しアップキャストおよびブロードキャストによって **Prepare_Subtree**・**Find_Pivot**・**Shuffle** 手続きの中断を通達する．これらの手続きを行っ

ている（エージェントが存在している）ノードでは手続き中断が通達されたら手続きを停止する．

(2) 新系統ラベルの付与

ノード $p(v)$ でエージェントを生成し w へ向かい DFS ツアーを開始する．エージェントを受け取ったノード x では新たな系統ラベル $newL_{anc}(x)$ の計算を行い，各ノードに一時的に保存する．新たな系統ラベルはノード v の系統ラベル $L(v)$ から計算可能である．ここで，$L_{anc}(i)$ を $[a_i, b_i]$ として表現し．$d_i = b_i - a_i + 1$ とする．また，$newL_{anc}(i)$ を $[a'_i, b'_i]$ として表す．

ノード x では，まず始めに $[a'_x, b'_x] = [a_x, b_x]$ とする．

エージェントが親ノードから到達した場合には，

$$[a'_x, b'_x] \leftarrow \begin{cases} [a'_x, b'_x + d_v] & (a_w < a_{p(v)}) \\ [a'_x - d_v, b'_x] & (a_w > a_{p(v)}) \end{cases}$$

とし，エージェントを親ノードへ転送する場合には，

$$[a'_x, b'_x] \leftarrow \begin{cases} [a'_x + d_v, b'_x] & (a_w < a_{p(v)}) \\ [a'_x, b'_x - d_v] & (a_w > a_{p(v)}) \end{cases}$$

とする．また，エージェントが heavy 子ノード $H(x)$ から到達した際には，$newL_{heavy}(x) = newL_{anc}(H(x))$ とする．

エージェントが w へ到達したら，辺 (w, u) を全域木のリンクとして採用，すなわち，$Port(w, u)$ と $Port(u, w)$ の α ビットを 1 とする．

部分木 T_v を，ノード u を根とした部分木 T_u として再構築する．u の新たな系統ラベルは，

$$[a'_u, b'_u] \leftarrow \begin{cases} [b_w + 1, b_w + d_v] & (a_w < a_{p(v)}) \\ [a_w - d_v + 1, a_w] & (a_w > a_{p(v)}) \end{cases}$$

とする．

T_v 内のノード x では，エージェントが親ノードから到達した際には，$a'_x \leftarrow a'_{p(x)} + 1$ とし，エージェントを親ノードへ転送する際には，b'_x を子ノード y 内の b'_y の最大の値とする．ただし，x が葉ノードである場合は $b'_x \leftarrow a'_x$ とする．

(3) 新ラベル・ポート番号の付与

T_u 内のノードの heavy ラベル及び light ラベルの計算を DFS ツアーによって行う．

ノード u と v の間のノード x では，親ノードの変更が起こり u へと向かう隣のノード y が新たな親ノードとなる．また，復旧情報の $\omega(v)$ を元に新たな heavy 子ノードの選定を行う．T_v を根 u により再構築した際の，x の新たな子ノード z（現在の x の親ノード）の重みは $\omega(v) - \omega(x)$ である．ノード x では現在の heavy 子ノード $H(x)$ の重みと新たな子ノードの重みを比較して，新たな heavy 子ノード $fH(x)$ を決定する．ただし，$H(x) = y$ であり，$fH(x) = H(x)$ となった場合には，x の子ノードのうちその子ノードへのリンクへ割り当てられているポート番号が一番小さいノードを $fH(x)$ とする．そして，$newL_{heavy}(x) \leftarrow newL_{anc}(fH(x))$ とする．

ノード u では，ノード w のラベルの状態（図 7）に従い $newL_{busy}(u) \leftarrow L_{busy}(w)$ とし，以下のように u の新たな light ラベル $newL_{light}(u)$ を計算する．

$$newL_{light}(u) \leftarrow \begin{cases} fL_{light}(w) \circ Port(w, u) & (fL_{light}(w) \neq \phi) \\ L_{light}(w) \circ Port(w, u) & (fL_{light}(w) = \phi) \end{cases}$$

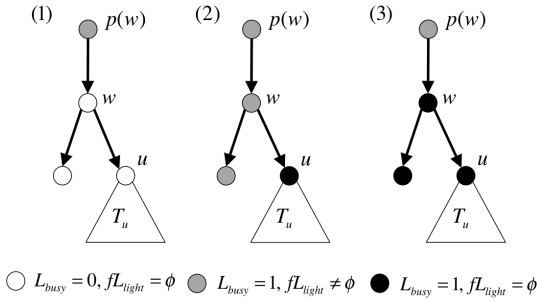

図 7 ラベル状態

u 以外の T_u 内のノード x は自身が新たな heavy 子ノードである場合には，$newL_{light}(x) \leftarrow newL_{light}(p(x))$ とし，そうでない場合には，$newL_{light}(x) \leftarrow newL_{light}(p(x)) \circ Port(p(x), x)$ とする．

また，各ノード x では，ノード v のラベル $L(v)$ を格納する．

(4) ラベル・ポートの切り替え

T 内のラベルの切り替えを u から DFS ツアーで登る際に行う．エージェントを受け取ったノード x では $newL(x)$ を現行ラベルとし，更新済みラベルを 1 とする．

また，(3) で決定したノード y，z，$fH(x)$ をもとにポート番号を次のように切り替える．

$$Port(x, z) \leftarrow \begin{cases} Port(x, fH) & (z \neq fH(x) \text{ かつ } y = H(x)) \\ Port(x, y) & (z \neq fH(x) \text{ かつ } y \neq H(x)) \end{cases}$$

$$Port(x, H) \leftarrow \quad Port(x, y) \quad (y \neq H(x) \text{ かつ } z = fH(x))$$

$$Port(x, y) \leftarrow 0, Port(x, f(H)) \leftarrow 1$$

T 内のラベル切り替えが完了したら w から $p(v)$ へ同様に DFS で登る際に切り替えを行い更新済みラベルを 1 とする．

(5) 更新済みラベルの削除

v と w のもっとも近い共通の祖先から 更新済みラベル・v のラベル $L(v)$ の削除を DFS によって行う．

(6)Shuffle 手続き

T 内の新ラベルを計算する途中でラベルが $2^5 \log n'_0$ を超えるものがあった場合には，u から **Find_Pivot** を行い．**Shuffle** の対象となる部分木を求める．このとき，対象となる部分木の根 ρ が凍結状態，つまり復旧処理の最初に中断した **Shuffle** 手続きの動作する部分木 T' の内部である

場合にはその部分木の頂点 ρ' をピボット ρ とする．ピボット ρ を根とする部分木 T_ρ 内部に T' が含まれるならば先に T' 内での **Shuffle** 手続きを完了させたのち T_ρ の **Shuffle** 手続きを行う．

(7) 手続きの再開

アップキャスト・ブロードキャストによって復旧作業終了を全ノードに通知する．(1) において手続きを中断したノードでは手続きの再開を行う．

・**Shuffle** 手続きの再開

復旧手続きにおいて **Shuffle** した部分木に中断した **Shuffle** 手続きの部分木が含まれていた場合には前フェーズにて手続きが完了している．

手続きを中断した際の **Shuffle** のエージェントの位置によって場合分けを行う．エージェントが復旧する部分木 T_v の内部にある場合には T_v 内の全ノードは新ラベルへと移行しているため無視することができる．また，$T\rho \in T_v$ である場合には $p(v)$ と v が切断された際にエージェントが $p(v)$ に戻ったとして扱う．これは切断されたリンクに対応したポート番号が $p(v)$ の持つ cL_{port} と等しいか否かで判定することで可能である．

・**Find_Pivot**・**Prepare_Subtree** の再開

復旧作業終了の通達により，停止していた重さの取得および凍結処理の再開を行う．

5.4 転送手順

ルーティング手順

(1) 自身の更新済みラベルが 0 の場合
  (a) 宛先の更新済みラベルが 0 の場合
    (i) $L(v)$ を持ち，宛先が v の子孫でない
      新たな親へ転送
    (ii) その他
      現行ラベルで転送手順 1 を実行
  (b) 宛先の更新済みラベルが 1 の場合
    新ラベルがあれば新ラベル，なければ現行ラベルで転送手順 1 を実行．
(2) 自身の更新済みラベルが 1 の場合
  (a) 宛先の更新済みラベルが 0 の場合
    親へ転送 (※)
  (b) 宛先の更新済みラベルが 1 の場合
    現行ラベルで転送手順 1 を実行

（ただし，中継ラベルがある場合には中継ラベルを宛先として判定する）

$p(v)$ へ到達した T_v 内のノード宛メッセージに対して，ヘッダに中継ラベル L_w とポート番号 $Port(w,u)$ を付与し，v へ到達した T_v 外のノード宛メッセージに対し，ヘッダに中継ラベル L_u とポート番号 $Port(u,w)$ を付与する．

6. ルーティング方式の分析

6.1 ラベルサイズ

通常時のラベルサイズが $O(\log n)$ であることは [7] および [4] により証明されている．

補題 2. *Korman* のルーティング方式において *light* ラベルのサイズは高々 $2^5 \log n$ である *[7]*.

定理 1. 提案ルーティング方式の *light* ラベルサイズは $O(\log n)$ である．

証明. 補題 1 より，light ラベルは通常時には $2^5 \log n$ 以下である．そのため，全域木の復旧によって復旧される部分木の新たな親の light ラベルは $2^5 \log n$ 以下である．新たに連結される部分木の light ラベルの連結は葉ノードから根へ向かい別の葉ノードへと連結される場合が一番長く，高々 $2^6 \log n$ となる．また，連結する際に新たに加わる復旧ポートは高々 $\log n$ である．よって復旧時にポート番号を連結して与えられる light ラベルは最大で $3 \cdot 2^5 \log n'_0 + \log n$ である．また，復旧中に light ラベルが $2^5 \log n'_0$ を超えた場合には **Shuffle** 手続きを行い $2^5 \log n$ 以下に抑えるため light ラベルのサイズは $O(\log n)$ である． □

6.2 ルーティングテーブルサイズ

定理 2. ルーティングテーブルのサイズは $O(\log n)$ である．

証明. 復旧情報の各ラベルは $O(\log n)$ であり，重さ・ポート番号は高々 $\log n$ で表現可能であるため，復旧情報一つあたりのメモリサイズは高々 $6 \log n$ である．子ノードの復旧情報を格納する場合，または親ノードにより隣の子ノードの格納を任される場合があり，最大で 3 個の復旧情報を格納するため高々 $18 \log n$ である．また復旧中に v の系統ラベルを与えられる場合があるため，ルーティングテーブルサイズは高々 $19 \log n$ であり $O(\log n)$ となる． □

6.3 ルーティングの整合性

定義 1. ルーティングが正しいとは，現在割当て中のラベル宛のメッセージが最終的に宛先へと到達することである．

定理 3. 提案ルーティング方式は常に正しくメッセージを転送する

証明. 復旧のフェーズ，メッセージの送信元・宛先で分けて証明する．ここで，リンクの切断により非連結になる部分木を T_v とし，ノード u とノード w の最も近い共通の祖先を根とする部分木を $T_{nca(u,w)}$ とする．

平常ルーティング時（リンク削除前）

すべてのノードの更新ラベルは 0 である．そのため現行のラベルが用いられ，補題 1 により正しく転送される．

手続き停止処理時

Prepare_Subtree と **Find_Pivot** は重さの取得やノードの凍結処理なのでルーティングに関わる変更は行っておらず，**Shuffle** 手続きの実行中には，ルーティングが正常に動くことを担保しながらラベルの更新を行っているため，途中で手続きを停止してもルーティングは正常に行われる．

リンク削除後から新ラベルの切り替え前

（1）T_v 内から T_v 内宛てのメッセージ

補題 1 より，現在のラベルを用いて正しく転送される．

（2）T_v 外から T_v 外宛てのメッセージ

（1）と同様に現在のラベルを用いて正しく転送される．

（3）T_v 外から T_v 内宛てのメッセージ

現行ラベルにより転送され，T_v の共通の祖先である $p(v)$ に到達する．$p(v)$ にて中継ラベル w とポート番号 $Port(w,u)$ を得る．中継ラベルをもとに中継ノード w へ転送され，ポート番号 $Port(w,u)$ により u へと到達する．$u \in T_v$ であり，このとき T_v 内では現在のラベルを用いてメッセージが転送され宛先へと到達する．

（4）T_v 内から T_v 外宛てのメッセージ

現行ラベルにより転送され v へと到達する．v にて中継ラベル u とポート番号 $Port(u,w)$ を得る．中継ラベル・ポートを元に w へと転送される．このとき，w から宛先へは現在のラベルを用いてメッセージが転送され宛先へと到達する．

T_v 内の新ラベルへの切り替え

（1）T_v 内から T_v 内宛てのメッセージ

宛先が旧ラベルの場合，自身が更新済みであるならば親へと転送され更新済みでないノードへと転送される，T_v 内の更新済みでないノードは連結であるため，現行ラベルによって宛先へと転送される．一方，宛先が新ラベルの場合には，保持している新ラベルまたは更新済みである自身のラベルによって正しく転送される．

（2）T_v 外から T_v 外宛てのメッセージ

ラベルの変更はないため現在のラベルによって正しく転送される．

（3）T_v 外から T_v 内宛てのメッセージ

宛先が旧ラベルの場合には，新ラベル切り替え前と同様に宛先へルーティングが可能である．なぜならば，T_v 内の更新前のノードは連結であるためである．宛先が新ラベルの場合には，v と w のもっとも近い共通の祖先を根とする部分木 $T_{(v,w)}$ に到達することは自明である．$T_{(v,w)}$ 内のノードはすべて新ラベルを保持しているため，新ラベルを用いて転送することができる．

（4）T_v 内から T_v 外宛てのメッセージ

v の系統ラベル $L_{anc}(v)$ から宛先が T_v 内のノードでないことが判定され，新しい親へと転送され w へと到達する．T_v 外のノードはすべて更新前であるため現行のラベルによって正しく転送される．

T_v 外の新ラベルへの切り替えから更新ラベル削除前

T_v 内のノードで旧ラベルを用いているノードは存在しないことに注意．

（1）T_v 内から T_v 内宛てのメッセージ

すべてのノードが新ラベルを用いており，新ラベルで正しく転送される．

（2）T_v 外から T_v 外宛てのメッセージ

T_v の外部で切り替わるラベルは系統ラベルと heavy ラベルのみである．DFS で下から切り替えを行っており，また更新されるノードで更新前のノードはすべて新ラベル（新系統ラベル，新 heavy ラベル）を持っているため，これに従い正しくルーティングをすることができる．

（3）T_v 外から T_v 内宛てのメッセージ

宛先ラベルは新ラベルであり，u と w のもっとも近い共通の祖先 $nca(u,w)$ を根とする部分木 $T_{nca(u,w)}$ に含まれないノードは親へと転送する．u と w の共通の祖先へと到達すると $nca(u,w)$ へ向かい転送される．$nca(u,w)$ に到達すると $T_{nca(v,w)}$ の全ノードは新ラベルを保持または現行ラベルとして利用しており，これを元に正しく転送される．

（4）T_v 内から T_v 外宛てのメッセージ

宛先ラベルが旧ラベルの時には v のラベルを保持しているため新しい親へ転送される．T_v 外へと転送され，（2）より現行ラベルによって正しく転送される．宛先ラベルが新ラベルの時には，更新済みノードはすべて連結であるため現行ラベルで正しく転送される．

更新済みラベル削除から Shuffle 手続き開始前

前回のフェーズの終了時点で，更新が必要なすべてのノードが更新済みになっており，自身の更新済みラベルが 1 で宛先の更新済みラベルが 0 であり v のラベルを持っている場合を除き，現行ラベルによってルーティングすることに気をつける．更新が必要なノードはすべてラベルの更新が済んでいるためすべてのノードが現行のラベルによってルーティングされた場合にはそれは正しいルーティングである．また，更新済みラベル・v のラベルの削除は v と w のもっとも近い祖先から DFS によって行われる．そのため，必ず親から更新済みラベルが 0 になるためルーティング手順（※）において親へ転送されるルーティングは正しい状態を保ちながら更新済みラベルおよび v のラベルの削除がなされる．

Shuffle 手続き中

シャッフル手続きはルーティングの整合性を保ちながら行われる．また，**Shuffle** の対象が T_v の外部まで及び，**Shuffle** 手続きが中断されている部分木まで **Shuffle** 手続きが行われる際には，中断中の **Shuffle** 手続きのルーティングの正確性を保ちながら **Shuffle** しラベルを更新することが可能である．

□

6.4 ストレッチ

ストレッチの評価については，通常のルーティングを行っている場合と全域木の切断による復旧を行っている場合によってルーティングが異なるため分けて評価をする．

静的時には，Korman によるルーティング方式と同様の転送を行う．Korman のルーティング方式は構成する全域木において最短経路ルーティングを行うためストレッチは 1 となっている．

復旧時には復旧情報をもとに中継して転送を行うため，全域木上における最短経路ルーティングではなく，以下のような場合にネットワーク上の最短経路と実際に転送される経路の比の最悪な値 $n-1$ をとる．

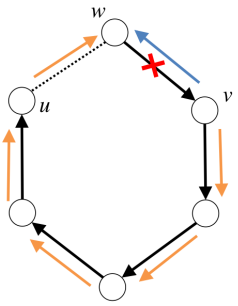

図 8 最悪のケースのストレッチ

上図の青い矢印で構成される経路が最短経路であり，オレンジの矢印で構成される経路が実際に転送される経路である．

6.5 メッセージ複雑性

ノード追加時の処理については Korman[7] と同様であり，メッセージ量が $O(n \log n)$ であることが示されている．

リンク追加時には復旧情報を構築しアップキャストを行う．これは，ノードが一直線に並んだ時に $O(n)$ のメッセージが発生する．

リンク削除時については，全域木の一部のリンクとそうでないリンクとで場合分けを行う．

削除されたリンクが全域木を構成するリンクの一部だった場合には復旧処理が発生する．この際に発生するメッセージ量は切断された部分木 T_v の大きさと復旧後に新たに親となるノード u と w の最も近い共通の祖先 $nca(u,w)$ を根とする部分木 $T_{nca(u,w)}$ の大きさに依存し，メッセージ量は $O(|T_{nca(u,w)}|)$ であるため $O(n)$ である．復旧リンクとして用いているリンクが切断された場合には，両端ノードがアップキャストにより祖先ノードへリンクの切断を通知する．このリンクを復旧情報として用いていたノードでは新たな復旧情報を取得する必要がある．これを探すのには全域木で用いているリンク以外のリンクが必要となるため，$O(\Delta \cdot n)$ となる．ここで Δ は最大次数である．

7. まとめ

Korman による動的木に対するコンパクトルーティングを一般トポロジへと拡張し，構成する全域木の任意リンクに障害が発生した際，ルーティングの整合性を保ちながら全域木を復旧するコンパクトルーティング方式を示した．

この提案コンパクトルーティング方式の評価は以下である．

表 1 ルーティング方式の評価

方式	Korman[7]	提案方式
ラベルサイズ	$O(\log n)$	$O(\log n)$
テーブルサイズ	$O(\log n)$	$O(\log n)$
ストレッチ	1	静的時 1
		復旧時 $n-1$
メッセージ複雑性	$O(n \log n)$	$O(n(\Delta + \log n))$
トポロジ変化	ノード・リンク追加	ノード・リンク追加
		リンク削除（非復旧時）
正確性の保証	すべての時刻	すべての時刻

Δ：最大次数

ただし，基礎グラフ G は常に連結であることを前提としている．ここで，ストレッチは構築している全域木における最短パスに対する実際に転送するパスの長さの比である．

参考文献

[1] Afek, Y., Gafni, E. and Ricklin, M.: Upper and lower bounds for routing schemes in dynamic networks, *Foundations of Computer Science, 1989., 30th Annual Symposium on*, IEEE, pp. 370–375 (1989).
[2] Alstrup, S., Gavoille, C., Kaplan, H. and Rauhe, T.: Nearest common ancestors: A survey and a new algorithm for a distributed environment, *Theory of Computing Systems*, Vol. 37, No. 3, pp. 441–456 (2004).
[3] Dom, M.: Compact Routing, *Algorithms for Sensor and Ad Hoc Networks*, Springer (2007).
[4] Fraigniaud, P. and Gavoille, C.: Routing in Trees, *28 th International Colloquium on Automata, Languages and Programming (ICALP)*, Springer, pp. 757–772 (2001).
[5] Kannan, S., Naor, M. and Rudich, S.: Implicat representation of graphs, *SIAM Journal on Discrete Mathematics*, Vol. 5, No. 4, pp. 596–603 (1992).
[6] Korman, A.: General compact labeling schemes for dynamic trees, *Distributed Computing*, Vol. 20, No. 3, pp. 179–193 (2007).
[7] Korman, A.: Improved compact routing schemes for dynamic trees, *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, ACM, pp. 185–194 (2008).
[8] Korman, A. and Peleg, D.: Compact separator decompositions in dynamic trees and applications to labeling schemes, *Distributed Computing*, Springer, pp. 313–327 (2007).