任意の単一リンク故障を考慮した生成木構成強安定プロトコル

片山　喜章[†a)]　長谷川敏之[††]　高橋　直久[†]

A Super-Stabilizing Spanning Tree Protocol for a Link Failure

Yoshiaki KATAYAMA[†a)], Toshiyuki HASEGAWA[††], and Naohisa TAKAHASHI[†]

あらまし　任意の初期（大域）状況からプロトコルを開始しても，やがて問題を解く（解状況に到達する）分散プロトコルを自己安定プロトコルという．初期状況に一切の仮定をおかないため，プロトコル実行中に生じる任意の一時故障に対して耐性をもつ．自己安定プロトコルのもつ優れた故障耐性を更に高度にしたものに，強安定プロトコルがある．これは，解状況から故障が生じてから再安定するまでの実行に出現するすべてのネットワーク状況が，ある「安全条件」を満たすことを保証する自己安定プロトコルである．本論文では，任意の単一リンク故障に対して耐性をもつ生成木構成のための強安定プロトコルを提案する．

キーワード　自己安定，強安定，リンク故障，生成木，アドホックネットワーク

1. まえがき

近年，無線端末が自律的にネットワークを構成し，互いに情報をやり取りできるアドホックネットワークに関する技術が注目されている．これは，中継局などのインフラを必要とせず，端末さえ用意すれば端末同士で情報を中継することで通信を実現できるため，災害現場などで非常に有効であると考えられるためである．しかし，アドホックネットワークでは，ネットワークへの端末の参加・退出，あるいは移動などによってネットワーク形状が大変流動的で，効率の良い通信を実現するのが困難である．

一方ネットワーク形状が固定された場合には，生成木を構成し，それを用いることで効率良く通信できることが知られている．つまり，形状が流動的なネットワークに対しても生成木を構成・維持できる技術があれば，アドホックネットワークでも効率の良い通信が可能となる．しかし，実際には形状が流動的なネットワークでは，生成木の構成や維持が著しく困難である．最悪，たった一つの端末の退出によって，生成木全体を構成し直す場合も考えられる．

† 名古屋工業大学工学研究科，名古屋市
Graduate School of Engineering, Nagoya Institute of Technology, Nagoya-shi, 466–8555 Japan
†† (株) 日立製作所，横浜市
Hitachi, Ltd., Yokohama-shi, 244–8555 Japan
a) E-mail: katayama@elcom.nitech.ac.jp

アドホックネットワークを含め，自律的に動作する端末（やプロセス）とそれらを相互に接続する通信リンクで構成されるネットワークを，分散システムと呼ぶ．分散システム上で与えられた問題を解く手順を，分散プロトコルという．プロセスやリンクの出現や消滅など，ネットワーク形状の変化に対応した（故障耐性のある）分散プロトコルについて様々研究されているが，特に近年，自己安定プロトコルに関する研究が活発である [1], [2]．自己安定プロトコルとは，プロトコル開始時のネットワーク（全域）状況に一切の仮定をおかず，任意の初期状況からプロトコルを開始しても，やがてあらかじめ決められたネットワーク状況（解状況）に到達するプロトコルである．初期状況に仮定をおかないことより，プロトコル実行中に任意の一時故障（transient failure）が生じてもその状況を初期状況とし，再び解状況に到達する（再安定する）．

自己安定プロトコルは，システム全体の初期化が不要であり優れた故障耐性をもつ一方，解状況から故障が生じた際の再安定実行については一切考慮されない．例えば生成木構成問題を解く自己安定プロトコルにはいくつかあるが [3]〜[6]，いずれの場合も解状況からたった一つのプロセスやリンクの故障により，生成木全体を再構成する可能性がある．そこで，再安定実行においてより優れた性質をもつ自己安定プロトコルとして故障封じ込め（fault-containment）[7], [8] や時間適応（time-adaptive）[9]，更に強安定（super-stabilizing）

プロトコル [10]〜[12] が提案されている．故障封じ込めと時間適応は，解状況で故障が起きた状況からの再安定実行に関して，より早く再安定することを主眼としている．一方，強安定プロトコルは，同じく再安定実行において，ネットワーク状況があらかじめ決められた「安全条件（passage predicate）」を満たしたまま再安定することを保証する．つまり，より安全に再安定することを主眼としている．

アドホックネットワーク上での自己安定アルゴリズムについては，文献 [13]〜[15] などがある．文献 [14] では，アドホックネットワークの流動性を三つに分類しており，最も激しく形状が変化するモデル上でトークンのランダムウォークによる通信を実現している．また文献 [15] ではトークン巡回によって排他制御を実現しているが，頻度の高い形状変化には対応できていない．一方，文献 [13] では，同期ネットワーク上で，想定した範囲での形状変化（リンク故障の間隔を制限）であれば最小生成木を構成しながらトークンを巡回させるプロトコルが提案されている．また，先に述べたように，ネットワーク上に生成木を構成することで多くのサービスが効率良く行えることが知られており，生成木上での自己安定アルゴリズムも数多く研究されてきた（[16]〜[18] など）．特に文献 [16] では，生成木上でのトークン巡回プロトコルが提案されているが，これは故障による生成木の形状変化がトークン巡回に影響を及ぼすため，巡回のインフラである生成木の安定性（形状変化の少なさ）は非常に重要な要素となる．また，文献 [19] では，生成木上でのトークン巡回についての状態数の下界が求められている．

本論文では，（生成木の根となる）特別なプロセスが一つ存在する，形状変化の頻度が比較的低いアドホックネットワークにおいて，ネットワーク形状が変化しても安全に生成木を再構成する（再安定する）生成木構成強安定プロトコルを提案する．具体的には，解状況から任意の単一リンクが消滅した故障状況からの再安定実行において，再安定するまで新たに故障が生じなければある決められた安全条件（消滅リンクと代替リンク間のプロセスのみが木を再構成し，その他のプロセスは木構造を変化させない）を満たしながら再安定する(注1)．提案プロトコルは，既存の生成木構成強安定プロトコル [11] に比べ，対応する故障の範囲が真に広い点で優れている．つまり，[11] では生成木の枝以外のリンク故障のみを強安定の対象としており，生成木の枝が故障した場合は通常の自己安定プロトコルと同様である．それに対し，提案プロトコルでは枝を含めた任意のリンクの消滅に対しても安全条件を満たしながら再安定する．なお，文献 [7], [8] のアルゴリズムは，端末（プロセス）の一時故障に対しては生成木の構成は変化しないが，リンク故障に対しては通常の自己安定アルゴリズムと同様の動作となる．

本論文の構成は，**2.** でモデルについて述べ，**3.** で提案プロトコルを説明し正当性を証明する．最後に **4.** で検討と今後の課題について述べる．

2. モ デ ル

以下では本論文が対象とする分散システム及びプロトコルのモデルについて述べる．

2.1 分散システム

プロセスと通信リンク（以下たんにリンク）で接続された任意の形状の分散システム N を扱う．ただし，N は二連結（任意の一つのリンクを除いても連結性が保たれる）以上とする．

N は，n 個のプロセス集合 $P = p_1, p_2, \ldots, p_n$ と，リンク集合 L からなるものとし，よって $N = (P, L)$ で規定される．分散システムは，プロセスを頂点，リンクを辺としてみなせるので，グラフに対する用語を分散システムに対しても用いる．

各プロセス p_i は，自身の識別子，プログラムコード（プロトコル），隣接プロセス集合 $\mathcal{N}_i$ 及びその他の局所変数をもつ．各プロセスは相異なる識別子をもつものとし，簡単のためプロセス p_i とその識別子を区別せずたんに p_i と呼ぶ．更に，各プロセスは N 中の特別なプロセス $p_r(= p_1)$ の存在とその識別子を知っているとする．

リンク L は，P の相異なる要素の非順序対の集合であり，$(p_i, p_j) \in L$ のとき，プロセス p_i, p_j 間に全二重リンクが存在する．またこのとき p_i, p_j は隣接するといい，$p_i \in \mathcal{N}_j$ かつ $p_j \in \mathcal{N}_i$ である．

各プロセス p_i の隣接プロセス集合 $\mathcal{N}_i$ は，プロトコル外部に存在する隣接プロセス集合保障機構によって任意の時点で正しく維持されているものとする．つまり，ある時点でリンクまたはプロセスが消滅・出現すると，瞬時に $\mathcal{N}_i$ に反映されるものとする．

2.2 通信モデル

プロセス間通信は，レジスタ通信モデルを仮定する．

(注1)：再安定実行中に再び故障が起きると安全条件は保証されず，通常の自己安定アルゴリズムの動きとなる．

レジスタ通信モデルとは，各リンクの各方向に対して一つの通信用レジスタ（以下たんにレジスタ）が存在し，そこに一方が隣接プロセスへのメッセージ（データ）を書き込み他方がそれを読み込むことによって通信する．つまり，リンク $(p_i, p_j) \in L$ には二つのレジスタ R_{ij}, R_{ji} が存在し，p_i から p_j への通信は，p_i が R_{ij} にデータを書き込み p_j がそれを読み込むことで実現する（逆方向も同様）．

各プロセスは，レジスタから（へ）読み込んだ（書き込む）データをレジスタ構造体 R として格納する．また，各プロセス p_i は，以下の通信命令を用いてレジスタへの読み書きを行う．

- **read()**

読み込み命令．$\forall p_j \in \mathcal{N}_i$ に対して，レジスタ R_{ji} から読み込んだデータ（レジスタ構造体）の集合を返す．

- **write**(P', R)

書き込み命令．隣接プロセス集合の任意の部分集合 $P' \subseteq \mathcal{N}_i$ に対して，$\forall p_j \in P'$ に対するレジスタ R_{ij} にデータ（レジスタ構造体）を書き込む．

2.3 システム状況とスケジュール

分散システム N 上でのプロトコルの計算状況（システム状況．以下たんに状況）を，各プロセスの状態を列挙することで表す．各プロセスの状態（すべての変数及びレジスタの値）が q_i であるとき，状況を $c = (q_1, q_2, \ldots, q_n)$ と表す．N の取り得るすべての状況を C，p_i の取り得るすべての状態を Q_i とすると，$C = Q_1 \times Q_2 \times \ldots \times Q_n$ である．

プロセスの部分集合を $S \subseteq P$ とする．ある状況 $c_i \in C$ において，S に属するプロセスが同時にプロトコル $\mathcal{A}$ の 1 原子動作を実行することによって c_{i+1} になったとき，$c_{i+1} = c_i(S, \mathcal{A})$ と表す．ただし，$\mathcal{A}$ が明らかな場合は $\mathcal{A}$ を省略する．

定義 1 (スケジュール)： 空ではないプロセス集合の無限系列をスケジュールと呼ぶ．プロトコル $\mathcal{A}$，スケジュール $T = S(0), S(1), \ldots$ について，状況の無限系列 $E = c_0, c_1, \ldots$ が $c_{i+1} = c_i(S(i), \mathcal{A})$ を満たすとき，E を「初期状況 c_0，スケジュール T に対するプロトコル $\mathcal{A}$ の実行」と呼び，$E(\mathcal{A}, T, c_0)$ と表す．ただし，$\mathcal{A}$ が明らかな場合は $\mathcal{A}$ を省略する．また，T に全プロセスが無限回現れるとき，公平なスケジュールという．

本論文では，公平なスケジュールのみを対象とし，たんにスケジュールという．

スケジュールによって選ばれたプロセスは，1 原子動作のみ行えるものとする．このとき，スケジュール T と 1 原子動作の違いによりいくつかのモデルが考えられるが，本研究では以下のモデルを扱う（Distributed daemon（D-daemon））．

- 任意の $t (t \geq 0)$ について，$|S(t)| \geq 1$
- 1 原子動作：「全隣接プロセスのレジスタからデータを読み込み，自分の状態を遷移させ，必要なら任意の隣接プロセス集合のレジスタに書き込む」

2.4 動的分散システム

分散システム N におけるリンクあるいはプロセスの消滅・出現をトポロジー変化イベントと呼び，ε と表す．ただし，トポロジー変化イベントは 1 原子動作で実行されるとする．動的分散システムとは，トポロジー変化イベントによるネットワーク形状の変化（トポロジー変化）を許容する分散システム N である．

定義 2 (動的分散システムでのスケジュール)： 空ではないプロセス集合の無限系列を $S(0), S(2), \ldots$，トポロジー変化イベントの無限系列を $\varepsilon_1, \varepsilon_3, \ldots$ とするとき，これらの交替列 $T = S(0), \varepsilon_1, S(2), \varepsilon_3, \ldots$ を動的分散システム N におけるスケジュールと呼ぶ．このとき，c_i から c_{i+1} への状況変化は，$c_{i+1} = c_i(S(i), \mathcal{A})$ あるいは $c_{i+1} = c_i(\varepsilon_i, \mathcal{A})$ と表せる．

本論文で対象とする分散システム N は動的分散システムであり，トポロジー変化イベントは，無変化（プロセスやリンクの消滅・出現が起きない）を許すものとする．また，隣接プロセス集合保障機構によって，トポロジー変化イベントによってプロセスやリンクの消滅・出現が生じると，直ちに関係するプロセスの隣接プロセス集合が更新されるものとする(注2)．更に同保障機構は，トポロジー変化イベントが生じるとプロトコルに対してその発生を報告できると仮定する(注3)．

2.5 自己安定プロトコル

$LS \subseteq C$ を，N の状況の任意の集合とする．次の条件 (1)，(2) を満たすとき，「プロトコル $\mathcal{A}$ は，LS に関して自己安定（self-stabilizing）である」といい，$SS(\mathcal{A}, LS)$ と書く．また $SS(\mathcal{A}, LS)$ が成立するとき，LS を「プロトコル $\mathcal{A}$ に関して正当な状況」という．ただし，$\mathcal{A}$ が明らかなときはたんに正当な状況という．

(注2)：無変化でないトポロジー変化イベントが生じると，(大域）状況 c も変化することに注意

(注3)：隣接プロセス集合保障機構については，実ネットワークにおいて，例えば通信タイムアウトやネットワークへの接続通知などで実現可能であると考える．

（1） 到達可能性：任意の状況 $c_0 \in C$ と任意のスケジュール T に対し，c_0 から始まる T によるプロトコル $\mathcal{A}$ の実行 $E(\mathcal{A}, T, c_0)$ に，LS に含まれる状況が現れる．つまり，$E(\mathcal{A}, T, c_0) = c_0, c_1, \ldots$ とするとき，$c_i \in LS$ となる $i \geq 0$ が存在する．

（2） 閉包性：LS 中の任意の状況を c，プロセスの任意の集合を S とする．このとき，$c' = c(S, \mathcal{A})$ も LS に属する．

すなわち，任意の初期状況から開始される任意の公平なスケジュールによるプロトコルの実行は，やがて正当な状況に達し，一度正当な状況に達するとそれ以降の状況は正当な状況である（正当な状況で安定する）．

2.6 強安定プロトコル

自己安定プロトコルであり，かつ正当な状況において特定の故障が生じた状況から再安定するまでの再安定実行中に現れる状況が，新たな故障が生じない限り，ある「安全条件」を満たすものを強安定プロトコルという．

本論文で提案する強安定生成木構成プロトコルにおける安全条件は，直観的には,「生成木を構成するリンクが消滅したとき，木を再構成するために必要な代替リンクをもつプロセスと，消滅したリンクをもつプロセスの間のプロセスのみがその親子関係を逆転させるだけであり，その他のプロセスは親子関係を変更しない」である．なお，提案プロトコルにおける安全条件は **3.** で厳密に定義する．

3. 強安定生成木構成プロトコル

本章では，提案プロトコルの詳細について述べ，その正当性を証明する．

文献 [11] で提案されている強安定生成木構成プロトコルにおける安全条件は「生成木を変化させない」であり，これは生成木を構成する強安定プロトコルにおいては最も強い安全条件である．しかし，対象としているトポロジー変化イベントは「生成木を構成しているリンク以外の消滅のみ」である．

一方本論文で提案するプロトコルは，文献 [11] と同じ強安定性を実現しつつ，枝の消滅時の安全条件を新たに定義し，対象とするトポロジー変化イベントを拡張つまり「任意のリンクの消滅」に対応した強安定生成木構成プロトコルである．

3.1 提案プロトコルのアイデア

提案プロトコルの大まかな動きを説明する．提案プロトコルは，任意の状況から自己安定プロトコルとして生成木を構成する．この状況（正当な状況）から，生成木を構成している単一の枝が消滅した場合を考える．このとき，生成木は図 1 のように，二つの部分木，つまり $p_1(= p_r)$ を根とする $\mathcal{T}_0$，親へのリンクが消滅した p_8 を根とする $\mathcal{T}_1$ に分割される．この状況から部分木 $\mathcal{T}_0, \mathcal{T}_1$ を結ぶリンク（代替リンク）を一つ（(p_7, p_9)）選び(注4)，それを新たに生成木の枝とする．更に p_8, p_9 の親子関係を逆転させることで木を再構成する．ここで問題になるのは，各プロセスが $\mathcal{T}_0, \mathcal{T}_1$ のどちらに属しているかどう判断するかだが，これを根からの経路情報を利用することで解決する．つまり，経路情報中に p_8 を含むプロセスが $\mathcal{T}_1$ に属し，それ以外は $\mathcal{T}_0$ に属すると判断できる．

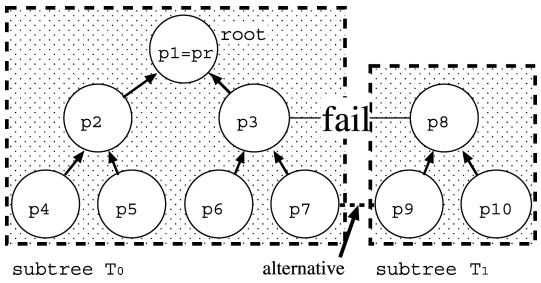

図 1 リンク消滅による木の分割
Fig. 1 2 subtrees with 1 link failure.

3.2 準　　備

提案プロトコルの詳細を説明するにあたり，必要な諸定義を行う．

定義 3 (経路情報)： プロセスの識別子の有限系列（空も許す）を経路情報と呼び，$\mathcal{P} = \langle p_{a_1}, p_{a_2}, \ldots, p_{a_k} \rangle$ で表す．各プロセス p_i は経路情報 $\mathcal{P}_i$ をもつものとし，特に p_r のもつ経路情報は $\mathcal{P}_r = \langle p_r \rangle$ である．また，$\mathcal{P}_j = \langle p_{j_1}, p_{j_2}, \ldots, p_{j_k} \rangle$，$\mathcal{P}_i = \langle p_{j_1}, p_{j_2}, \ldots, p_{j_k}, p_i \rangle$ であるとき，$\mathcal{P}_i = \langle \mathcal{P}_j, p_i \rangle$ と表す．

定義 4 (親子関係)： 任意の二つのプロセス p_i, p_j が以下の全条件を満たすとき，p_j を p_i の親プロセス，p_i を p_j の子プロセスと呼ぶ．また p_i と p_j の間に「親子関係が成立」しているといい，$p_j \leftarrow p_i$ と表す．

条件 *1* $(p_i, p_j) \in L$.

条件 *2* $\mathcal{P}_i = \langle \mathcal{P}_j, p_i \rangle$.

条件 *3* $\mathcal{P}_i, \mathcal{P}_j$ は同じ識別子を二つ以上含まない．

定義 5 (実経路情報)： プロセス p_i の経路情報を

(注4)：N は二連結以上なので必ず見つかる．

$\mathcal{P}_i$=$\langle p_{i_1}, p_{i_2}, \ldots, p_{i_k} \rangle$ とする．$\mathcal{P}_i$ が次の全条件を満たすとき，$\mathcal{P}_i$ は実経路情報であるという．
条件 *1*　$(p_{i_1} = p_r) \wedge (p_{i_k} = p_i)$.
条件 *2*　p_i が根プロセス (p_r) なら，$\mathcal{P}_r = \langle p_r \rangle$．それ以外の場合，$p_{i_1} \leftarrow p_{i_2} \leftarrow \cdots \leftarrow p_{i_k}$ である．

実経路情報は全域状況によって定義される．一方，次に定義する経路情報に関する性質は，各プロセスが局所的に判定可能である．

定義 6 (無矛盾経路情報)： プロセス p_i の経路情報を $\mathcal{P}_i = \langle p_{i_1}, p_{i_2}, \ldots, p_{i_k} \rangle$ とする．$\mathcal{P}_i$ が次の全条件を満たすとき，$\mathcal{P}_i$ は無矛盾経路情報であるという．
条件 *1*　$(p_{i_1} = p_r) \wedge (p_{i_k} = p_i)$.
条件 *2*　$p_{i_{k-1}} \in \mathcal{N}_i$.
条件 *3*　$\mathcal{P}_i$ は同じ識別子を二つ以上含まない．
また，無矛盾経路情報でないものを矛盾経路情報という．

ここで，$\mathcal{P}_i$ が実経路情報の場合，$\mathcal{P}_i$ に出現するすべてのプロセスは無矛盾経路をもつことに注意する．

任意の状況における分散システム N には，定義 4 の親子関係が成立したプロセス集合とそのプロセス間のリンクによって，いくつかの部分木が構成されていると考えることができる．部分木に関して以下のように定義する．

定義 7 (部分木)： どのプロセスとも親子関係が成立していないプロセスを単一部分木と呼ぶ．単一部分木及び親子関係が成立しているプロセスとリンクから誘導される部分グラフ（部分木と呼ぶ）の集合を部分木群と呼び，$\mathcal{T} = \mathcal{T}_0, \mathcal{T}_1, \ldots$ と表す．部分木 $\mathcal{T}_i$ の根（以下，サブルート）を $R(\mathcal{T}_i)$ と表し，$\mathcal{T}_i$ に属するプロセス集合を $P(\mathcal{T}_i)$ と表す．ただし，特に p_r をサブルートとする部分木を $\mathcal{T}_0$ と表す（つまり $R(\mathcal{T}_0) = p_r$）．部分木 $\mathcal{T}_i, \mathcal{T}_j$ において，$p_i \in P(\mathcal{T}_i), p_j \in P(\mathcal{T}_j), (p_i, p_j) \in L$ を満たすようなプロセス p_i, p_j が存在する場合，部分木 $\mathcal{T}_i, \mathcal{T}_j$ は隣接しているという．

3.3 提案プロトコルの詳細

以下では，提案プロトコルの詳細について述べる．提案プロトコルを図 3 に示す．プロトコルの記述が複雑なため，まずプロトコルの動きの概略から説明する．

3.3.1 提案プロトコルの動き

提案プロトコルは，大きく通常の自己安定部分と強安定部分に分けられる．故障が生じない状況では，自己安定部分によって生成木が構成される．つまり，分散システム N 中に p_r を根とする生成木が構成されている状況（正当な状況）に到達する．この状況からプロセス p_i の親への枝が消滅した場合を考える（図 2）．このとき，新たな故障が生じない限り強安定部分によって安全条件を満たしたまま再安定する．これは次のように行われる．

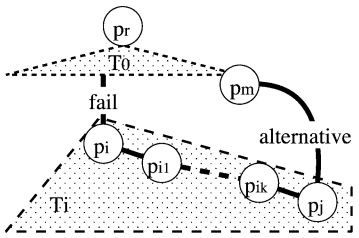

図 2　再安定実行
Fig. 2　Super-stabilizing execution.

N には二つの部分木 $\mathcal{T}_0(R(\mathcal{T}_0) = p_r)$，$\mathcal{T}_i(R(\mathcal{T}_i) = p_i)$ が存在する．もし，p_i が消滅したリンク以外に $\mathcal{T}_0$ に属するプロセス（p_m とする）へのリンク（代替リンク）をもつ場合，p_m を新たな親とするだけで，$\mathcal{T}_0, \mathcal{T}_i$ の木構造（親子関係）は変化しない．ただし，$\mathcal{T}_i$ に属するすべてのプロセスで，経路情報が更新されることに注意する．

一方，p_i に代替リンクが存在しない場合，すべての $p_q \in P(\mathcal{T}_i)$ に対して，代替リンクの存在を確認する必要がある．以下，手順をおって説明する．

（1）　p_i は $\mathcal{T}_i$ に属するすべてのプロセスに対して，代替リンク探索メッセージ `Detour` を放送する．この際，メッセージに p_i がサブルートである旨を付与する．

（2）　p_i からの `Detour` を受信したプロセス p_{i_1} は，自分が代替リンクをもつかどうか確認する．つまり，すべての隣接プロセス $p_s \in \mathcal{N}_{i_1}$ から経路情報を集め，もし経路情報中に p_i を含まない隣接プロセスが存在した場合，そのプロセスへのリンクを代替リンク候補とする．もし自分に代替リンク候補が存在しない場合は，`Detour` をすべての子プロセスに送信する．

（3）　代替リンク候補を発見したプロセス（p_j とする）は，代替リンク候補発見を意味するメッセージ `Discov` を，代替リンクに関する情報とともに p_i 宛（`Detour` が中継されてきた経路を逆方向）に送信する．

（4）　p_i は最初に受信した `Discov` メッセージに対応する代替リンク候補を代替リンクとして選び，その旨を `Decide` メッセージとして，選ばれた代替リンク

をもつプロセス (p_j) に向けて（選ばれた Discov メッセージに付与された経路で）送信する．

（5） Decide メッセージを受信した p_j は，代替リンク先のプロセスを親プロセスとし，経路情報 $\mathcal{P}_j$ を変更する．

（6） 部分木 $\mathcal{T}_i$ 上の経路 p_i から p_j に存在するプロセスは，p_j が親を変更したことをきっかけに，p_j から p_i の順に親子関係を逆転させていく（経路情報を更新する）．親子関係の逆転が p_i まで到達した時点で，生成木が再構成される．

（7） 部分木 $\mathcal{T}_i$ 上の経路 p_i から p_j 以外のプロセスは，p_i の経路情報が変更されることにより，順次各々の経路情報を更新する．この際，経路情報は変更されるが親子関係は変更しない．

3.3.2 提案プロトコルで使用する変数・関数

以下では，提案プロトコル（図 3）で使用する（プロセス p_i の）変数，関数，手続きについて述べる．

- 変数 $\mathcal{P}_i$：経路情報を格納する．
- 変数 $\mathcal{S}_i$：$\mathcal{S}_i = (\mathcal{SI}_i, \mathcal{SP}_i) \mid \perp$ で定義され，プロセスの状態を表す．$\mathcal{SI}_i$ はプロセスの状態を表す値である Detour | Discov | Decide | $\perp$ のいずれかが代入される．また，$\mathcal{SP}_i$ には経路情報が代入される．
- レジスタ構造体 R：レジスタからの情報を格納するための変数の型．$R = (\mathtt{ID}, \mathcal{P}, \mathcal{S})$ で定義され，ID は識別子，$\mathcal{P}$ は経路情報，$\mathcal{S}$ は状態を表す変数（$\mathcal{S} = (\mathcal{SI}, \mathcal{SP}) \mid \perp$）が格納される．
- 変数 $\mathcal{R}_i = \{R_{ji}, R_{ki}, \ldots\}$：関数 **read**() によって，全隣接プロセスに対するレジスタから読み出したレジスタ構造体（レジスタ変数と呼ぶ）の集合を格納する変数（レジスタバッファと呼ぶ）．レジスタ変数 R_{ji} の内容は，$R_{ji}.\mathtt{ID}, R_{ji}.\mathcal{P}, R_{ji}.\mathcal{S}$ として参照可能．
- 変数 t_i：トポロジー変化イベントの発生を表す論理変数．p_i の親へのリンクが消滅すると隣接プロセス集合保障機構によって「真」に設定される．
- 変数 $P_p, \mathcal{P}_p, \mathcal{S}_p$：$P_p$ には p_i の親プロセスの識別子，$\mathcal{P}_p$ には親プロセスのもつ経路情報（つまり $R_{P_pi}.\mathcal{P}$），$\mathcal{S}_p$ には親の状態（つまり $R_{P_pi}.\mathcal{S}$）がそれぞれ格納される．
- 変数 P_c：p_i の子プロセスの識別子の集合が格納される．一つも存在しない場合は $\perp$．
- 関数 $\mathbf{Parent}(\mathcal{P}_i, \mathcal{R}_x)$：レジスタバッファ $\mathcal{R}_x$ の中から，p_i の親プロセスのレジスタ変数を返す．つまり，$\mathcal{P}_i = \langle R_k.\mathcal{P}, p_i \rangle$ なるレジスタ変数 R_k を返す．もし見つからなかった場合は $\perp$ を返す．

```
/* 根プロセス (p_r) はじめ*/
do 無限ループ
P_r := ⟨p_r⟩; S_r :=⊥; M := {(N_r, ⊥)};
write(N_r, (p_r, P_r, ⊥));
od /* 根プロセス (p_r) おわり*/
/* 根以外のプロセス (p_i ≠ p_r) はじめ*/
do 無限ループ
if t_i then{
  if SI_i =⊥ then S_i := (Detour, P_i);(A0)
  t_i := false};
R_i := read(); P_c := Child(P_i, R_i);
if M_1() then SuperRootMode;
else if M_2() then SuperNodeMode;
else NormalMode;
WRITE(M);
od /* 根以外のプロセス (p_i ≠ p_r) 終り*/
/* 手続き定義 */
procedure NormalMode
  R_tmp := Parent(P_i, R_i);
  if R_tmp =⊥ then R_tmp := Random(R_i);
  P_i := ⟨R_tmp.P, p_i⟩; P_p := R_tmp.ID;
  if ¬Consistent(P_i) then {P_i :=⊥; P_p :=⊥}(A0-2)
  S_i :=⊥; M := {(N_i, ⊥)};
procedure SuperRootMode
  R_x := SElect(P_c, R_i);(A0-1)
  if G_1() then{
    R_y := SearchSubLink(P_i, R_i);(A1-1)
    if R_y ≠⊥ then {
      P_i := ⟨R_y.P, p_i⟩; P_p := R_y.ID;(A1-2)
      S_i :=⊥ ; M := {(N_i, ⊥)};
    } else M := {(P_c, S_i), ((N_i − P_c), ⊥)};(A1-3)
  else G_2() then{
    S_i := (Decide, R_x.SP);(A2-1)
    R_y := NextPathNode(P_i, SP_i, R_i);(A2-2)
    M := {(R_y.ID, S_i), ((N_i − R_y.ID), ⊥)};(A2-3)
  else if G_3() then {
    R_y := NextPathNode(P_i, SP_i, R_i);
    if R_y.ID =⊥ |P_p then Error();
    else if (R_y.ID ∉ P_c) then {
      P_i := ⟨R_y.P, p_i⟩ ; P_p := R_y.ID;(A3-2)
      S_i :=⊥; M := {(N_i, ⊥)};
    } else M := {(R_y.ID, S_i), ((N_i = R_y.ID), ⊥)};(A3-3)
  } else Error();
procedure SuperNodeMode
  S_i :=⊥; R_x = SElect(P_c, R_i);
  if G_4() then {
    R_y := SearchSubLink(SP_p, R_i);(A4-0)
    if R_y ≠⊥ then {
      M :={(P_p, (Discov, ⟨P_i, R_y.ID⟩)), ((N_i−P_p), ⊥)};(A4-1)
    } else M := {(P_c, S_p), ((N_i − P_c), ⊥)};(A4-2)
  else if G_5() then {
    M := {(P_p, R_x.S), (P_c, S_p), ((N_i − P_p − P_c), ⊥)};(A5)
  } else if G_6() then {
    R_y := NextPathNode(P_i, SP_p, R_i);
    if R_y.ID =⊥ |P_p then Error();
    else if R_y.ID ∉ P_c then {
      P_i := ⟨R_y.P, p_i⟩; P_p := R_y.ID; M := (N_i, ⊥);(A6-1)
    } else M := {(R_y.ID, S_y), ((N_i − R_y.ID), ⊥)};(A6-2)
  } else Error();
```

図 3 提案プロトコル
Fig. 3 The proposed protocol.

- 関数 **Child**($\mathcal{P}_i$,$\mathcal{R}_i$)：隣接プロセスの中から，子プロセスに相当するプロセス集合を返す．つまり，レジスタバッファ$\mathcal{R}_i$ 中のレジスタ R_{ji} から，$R_{ji}.\mathcal{P} = \langle \mathcal{P}_i, p_j \rangle$ なるレジスタ変数をすべて見つけ，その識別子（R_{ji}.ID）の集合を返す．存在しない場合は $\perp$ を返す．
- 関数 **Consistent**($\mathcal{P}_i$)：経路情報 $\mathcal{P}_i$ が無矛盾経路（定義 6）のとき真，それ以外は偽を返す論理関数．
- 関数 **Random**($\mathcal{R}_i$)：レジスタバッファ$\mathcal{R}_i$ 内から $R_{ji}.\mathcal{S} = \perp$ であるレジスタ変数 R_{ji} を無作為に一つ返す．
- 関数 **Last**($\mathcal{P}$)：経路情報 $\mathcal{P}$ の最後の要素（プロセスの識別子）を返す関数．もし $\mathcal{P}$ の要素数が 0 の場合は，$\perp$ を返す．
- 関数 **Prefix**($\mathcal{P}_x, \mathcal{P}_y$)：$\mathcal{P}_y = \langle \mathcal{P}_x, \mathcal{P} \rangle$（$\mathcal{P}$ は任意の経路情報）のとき真，それ以外は偽を返す．
- 関数 **SearchSubLink**($\mathcal{P}_x, \mathcal{R}_i$)：代替リンク候補を見つける関数．つまり，レジスタバッファ$\mathcal{R}_i$ の中から，**Prefix**($\mathcal{P}_x, R_{ji}.\mathcal{P}$) が偽であるレジスタ変数 R_{ji} を一つ返す．複数ある場合は，一番最初に見つかったものを返し，見つからなかった場合は $\perp$ を返す．
- 関数 **NextPathNode**($\mathcal{P}_i, \mathcal{P}_x, \mathcal{R}_i$)：与えられた経路情報 $\mathcal{P}_x$ と自分の経路情報 $\mathcal{P}_i$ から，経路情報 $\mathcal{P}_x$ 上での p_i の次のプロセス（親または子プロセスに相当）のレジスタ変数をレジスタバッファ$\mathcal{R}_i$ 中から選んで返す．存在しない場合は $\perp$ を返す．`Detour`，`Discov`，`Decide` を正しく中継するために使用する．
- 関数 **SElect**($P, \mathcal{R}_i$)：与えられたプロセス集合 P とレジスタバッファ$\mathcal{R}_i$ から，$R_{ji}.ID \in P$ なるレジスタ変数 R_{ji} をすべて選出し，その中から優先順位の最も高いレジスタ変数を返す．なお，レジスタ変数の優先順位とは，$R_{ji}.\mathcal{S}$ に含まれる変数 $\mathcal{SI}$ によって，以下のように定義される．$\perp <$ `Detour` $<$ `Discov` $<$ `Decide`．つまり，`Decide` メッセージを含むレジスタ変数が最も高い優先順位をもつ．なお，$\mathcal{SI}$ だけで優先順位が決定されない場合，更に $\mathcal{SP}$ の長さ（短いものが優先），`ID`（大きいものが優先）を加えた 3 項組による辞書式順序で全順序を与える．
- 手続き **WRITE**(M)：与えられた $M = \{(P_1, \mathcal{S}_1), (P_2, \mathcal{S}_2), \ldots\}$ に対して以下を行う．**write**($P_1, (p_i, \mathcal{P}_i, \mathcal{S}_1)$), **write**($P_2, (p_i, \mathcal{P}_i, \mathcal{S}_2)$),$\cdots$．
- 手続き **Error**()：$\mathcal{P}_i := \perp$，$\mathcal{S}_i := \perp$，$M := \{(\mathcal{N}_i, (p_i, \mathcal{P}_i, \mathcal{S}_i))\}$ を実行する．

次に，プロトコル中で用いる述語について述べる．

- $M_1() = (P_p = \perp \wedge$ **Last**$(\mathcal{P}_i) = p_i \wedge$ $\mathcal{SI}_i =$ (`Detour` | `Dicide`)).
- $M_2() = (P_p \neq \perp \wedge$ **Last**$(\mathcal{P}_i) = p_i \wedge$ $\mathcal{SI}_p =$ (`Detour` | `Dicide`)).
- $M_3() = (\forall R \in \mathcal{R}_i, R.\mathcal{P} \neq \perp)$.
- $G_1() = (M_3() \wedge R_x.\mathcal{SI} = \perp \wedge \mathcal{SI}_i =$ `Detour`).
- $G_2() = (M_3() \wedge R_x.\mathcal{S} =$ `Discov` $\wedge$ $\mathcal{SI}_i =$ `Detour`).
- $G_3() = (M_3() \wedge R_x.\mathcal{S} =$ (`Discov` | $\perp$) $\wedge$ $\mathcal{SI}_i =$ `Decide`).
- $G_4() = (R_x.\mathcal{SI} = \perp \wedge \mathcal{SI}_p =$ `Detour`).
- $G_5() = (R_x.\mathcal{SI} =$ `Discov` $\wedge \mathcal{SI}_p =$ `Detour`).
- $G_6() = (M_3() \wedge R_x.\mathcal{SI} =$ (`Discov` | $\perp$)$\wedge$ $\mathcal{SI}_p =$ `Decide`).

3.4 正当性の証明

本節では提案プロトコルの正当性を証明する．なお，本論文では紙面の都合上一部の証明が略証となっている．証明の詳細については，文献 [20] を参照されたい．

まず提案プロトコルの正当な状況及びトポロジー変化イベント Λ を定義する．

定義 8 (正当な状況)： すべてのプロセス p_i において，$\mathcal{P}_i$ が実経路であり，かつ $\mathcal{S}_i = \perp$ であり，かつすべてのレジスタで $\mathcal{S} = \perp$ である状況を，正当な状況という．

定義 9 (トポロジー変化イベントクラス Λ)： 単一リンクの消滅を，トポロジー変化イベントクラス Λ と呼ぶ．

正当な状況から，生成木を構成する枝に対してクラス Λ に属するトポロジー変化イベント ε が生じた状況を考える．このとき，分散システム N には二つの部分木が存在するが，それらを $\mathcal{T}_0(R(\mathcal{T}_0) = p_r)$, $\mathcal{T}_i(R(\mathcal{T}_i) = p_i)$ とする（つまり p_i の親への枝が消滅した（図 2））．このとき，ε が生じてから，再び正当な状況に到達する再安定実行中に満たされるべき条件（安全条件）を以下に定義する．

定義 10 (安全条件)： 親プロセスへのリンクが消滅したプロセス p_i と代替リンクに選ばれたリンクをもつプロセス p_j を結ぶ $\mathcal{T}_i$ の経路上にあるプロセスのみが親子関係を逆転させ，それ以外のプロセスは親子関係を変更しない．

以下，まず提案プロトコルが自己安定プロトコルで

あることを証明し，続いて強安定プロトコルであることを証明する．証明にあたっては，すべてのプロセスが少なくとも 1 度プロトコルの 1 原子動作を行った後の状況(注5)を考える．

補題 1： N の全状態変数 $\mathcal{S}$（局所変数，レジスタ内）が $\perp$ の状況から提案プロトコルを実行すると，十分長い間故障が生じなければやがて正当な状況に到達する．

証明：（略証）すべてのプロセスは，**NormalMode** のみを実行する．部分木 $\mathcal{T}_0$ に属する（無矛盾経路情報をもつ）プロセスは経路情報を更新しない．また，$\mathcal{T}_0$ 以外の部分木に属するプロセスは経路情報の長さは有限であるので矛盾経路情報をもち続けられず，いつか A0-2 が実行され経路情報が $\perp$ となる．結果，やがてすべてのプロセスが $\mathcal{T}_0$ に属する状況になり，補題が成立する． □

補題 2： 任意の状況から提案プロトコルの実行を開始しても，十分長い間故障が生じなければすべての変数 $\mathcal{S}$（局所変数，レジスタ内）は $\perp$ になる．

証明： 各プロセスは，$\mathcal{S} = \perp$ 以外のプロセスを新しい親にしない．また，$\mathcal{S}_i = \perp$ で $\mathcal{T}_0$ に属するプロセスは，経路情報を変更しない．

故障が起きない状況では，フラグ t_i は常に偽なので，$\mathcal{SI}_i :=$ **Detour** は実行されない．また，手続き **SuperNodeMode, NormalMode** では，$\mathcal{S}_i := \perp$ を実行しているため，$\mathcal{S}_i \neq \perp$ なるプロセスは **SuperRootMode** を実行し続ける必要がある．つまり，サブルートのみが $\mathcal{SI}_i \neq \perp$ であり続けられる．$\mathcal{SI}_i =$ **Decide** のプロセス p_i は，提案プロトコルの A3-3 しか実行できない．一方 p_i の子 p_{i_1} を考えると，p_{i_1} の子プロセス p_{i_2} に対するレジスタに `Decide` が書き込まれている．これを繰り返すと `Decide` 以外を返すプロセス p_{i_k} が必ず存在する（経路長は有限）．するとやがて p_i が A3-3 を実行できない状況に達し，$\mathcal{SI}_i = \perp$ になり以後そのままである．$\mathcal{SI}_i =$ **Discov** のプロセス p_i は **SuperRootMode** を実行できない．

よって全プロセスの内部変数 $\mathcal{S}$ はやがて $\perp$ になる．

この状況において，すべてのプロセスは **NormalMode** しか実行できず，したがってすべてのレジスタに $\mathcal{S} = \perp$ を書き込む． □

補題 1，2 より以下の補題がいえる．

補題 3 (到達可能性)： 任意の状況から提案プロトコルの実行を開始しても，十分長い間故障が生じなければ，やがて正当な状況に到達する．

また，提案プロトコルより以下の補題は明らか．

補題 4 (閉包性)： 任意の正当な状況から提案プロトコルを実行しても，故障が起きなければ正当な状況のままである．

補題 3，4 より提案プロトコルが自己安定プロトコルであることがいえたので，続いて強安定プロトコルであることを証明する．

補題 5 (強安定)： 提案プロトコルに関して，正当な状況からトポロジー変化イベントクラス Λ に属するイベント ε が生じた状況からの実行において，故障が生じなければ安全条件（定義 10）を満たしながら正当な状況に到達する．

証明： ε によって，プロセス p_i の親へのリンクが消滅した状況（c_a とする）を考える．以下では，c_a からの提案プロトコルの動作を説明し，安全条件を満たす実行以外起こり得ないことを証明する（図 2 参照）．

c_a において，p_i のみが $\mathcal{SI}_i =$ `Detour` に変更する．p_i では $M_1()$ が成立し，**SuperRootMode** を実行，$G_1()$ が成立するので，まず代替リンクを探索する（図 3 (A1-1)）．見つかれば，代替リンク先のプロセス p_m を親に変更 (A1-2)，見つからなければ子プロセスに `Detour` メッセージを送信する (A1-3)．

親から `Detour` メッセージを送られた子プロセス p_{i_1} では $M_2()$ が成立し **SuperNodeMode** を実行する．$G_4()$ が成立するので，代替リンクを探索 (A4-0) し見つからなければ子プロセスに `Detour` を送信する (A4-2)．この動作が，$\mathcal{T}_i$ 内に伝搬する．もし代替リンクが見つかった場合（p_j とする）は，親プロセスに `Discov` メッセージを送信する (A4-1)．N は二連結であるので，$\mathcal{T}_i$ 中に必ず p_j が存在することがいえる．

p_j から `Discov` を受け取った p_j の親プロセスでは $M_2()$ が成立し **SuperNodeMode** が実行され，$G_5()$ が成立するので，更に親プロセスに `Discov` メッセージを中継する (A5)．

`Discov` メッセージはやがて p_i に到達し，最も早く届いた `Discov` のうち，**SElect** によりただ一つだけ

(注5)：初期状況において存在するレジスタ内や内部変数の不正な値をプロセスによって少なくとも 1 度上書きした状態．

が選ばれる (A0-1)．p_i では $M_1()$ が成立しているので **SuperRootMode** を実行する．更に $G_2()$ が成立するので，$\mathcal{ST}_i =$ `Decide` とし (A2-1)，`Discov` メッセージに付与された経路情報（$\langle \mathcal{P}_j, p_m \rangle$，つまり p_i から p_j までの $\mathcal{T}_i$ 上での経路に代替リンク先のプロセス p_m を加えたもの (A4-1)）に沿って (A2-2) `Decide` を送信し，それ以外の経路上のプロセスには，$\mathcal{S}$ として $\perp$ を送信する (A2-3)．

親プロセスから `Decide` メッセージを受け取ったプロセスは，$M_2()$ が成立し，かつ G_6 が成立するので，p_j まで次々に `Decide` メッセージを中継する (A6-2)．

p_j に `Decide` メッセージが到着すると，A6-1 が実行され，p_m を親プロセスとし，それに準じて経路情報が書き換えられる．またそれと同時に全隣接プロセスに $\mathcal{S} = \perp$ を送信する．

親プロセスから `Decide`，子プロセスから $\perp$ を受け取ったプロセス（例えば p_j の元の親プロセス）は，$M_2()$ が成立，かつ $G_6()$ が成立する．ここで，今まで子プロセスだったプロセスは既に親プロセスを変更しているため，A6-1 が実行され，子プロセスではなくなったプロセスを新たな親プロセスに変更する（親子関係が逆転する）．

このような実行が p_j から p_{i_1} に向かって次々に生じ，p_i では $M_1()$ かつ $G_3()$ が成立するので，A3-2 によって親子関係を逆転させる．

一方，p_i から p_j までの経路以外のプロセスは，たとえ `Detour` に対して `Discov` を送信したとしても，p_i から $\perp$ が送信されてくるため，$M_1()$, $M_2()$ が成立せず，**NormalMode** が実行され続ける．もし p_i の経路情報（無矛盾経路情報）が変更されても，親プロセスのもつ経路情報が無矛盾経路情報である限り **NormalMode** では親プロセスを変更しないので，親子関係は変わらない（経路情報は更新される）．

以上より，c_a からの提案プロトコルにおいて，親子関係を逆転させるのは $\mathcal{T}_i$ 上の経路 p_i から p_j に含まれるプロセスのみであり，その他のプロセスは親子関係を変更しないことがいえる． □

補題 3，4，5 より，以下の定理がいえる．

定理 1： 提案プロトコルは，正当な状況からの任意の単一リンク故障に対して，定義 10 の安全条件を満たしたまま再安定する生成木構成強安定プロトコルである．

4. むすび

提案プロトコルでは，ネットワーク上に特別なプロセスの存在を仮定している．一般にモバイルアドホックネットワークではこのような仮定は妥当ではないと考えられるが，いわゆる家庭内 LAN のような比較的流動性が低いアドホックネットワークでは，その中心となる端末を特別なプロセスとして自然に仮定することが可能であると考える．

同時に生じるリンク故障が一つに制限されていること，またプロセスの消滅を考慮していないことについては，実システムへの適用条件としては厳しい．したがって今後の課題としては，複数リンクやプロセス消滅時の扱い，更により高度な木構造（最小生成木など）への対応が挙げられる．

謝辞 本研究の一部は，平成 16 年度日本学術振興会科学研究補助金（基盤研究（C）16500028，若手（B）16700010）の研究助成によるものである．

文　献

[1] E.W. Dijkstra, “Self-stabilizing systems in spite of distributed control,” Commun. ACM, vol.17, pp.643–644, 1974.

[2] S. Dolev, Self-Stabilization, The MIT Press, 2000.

[3] N. Chen, H. Yu, and S. Huang, “A self-stabilizing algorithm for constructing spanning trees,” Inf. Process. Lett., vol.39, pp.147–151, 1991.

[4] G. Antonoiu and P.K. Srimani, “A self-stabilizing distributed algorithm to construct an arbitrary spanning tree of a connected graph,” Computers and Mathematics with Applications, vol.30, pp.1–7, 1995.

[5] G. Antonoiu and P.K. Srimani, “Distributed self-stabilizing algorithm for minimum spanning tree construction,” Euro-Par’97 Parallel Processing, Proc., LNCS:1300, pp.480–487, 1997.

[6] L. Higham and Z. Liang, “Self-stabilizing minimum spanning tree construction on message-passing systems,” Distributed Computing 15th International Symposium, LNCS:2180, pp.194–208, 2001.

[7] S. Ghosh, A. Gupta, and S.V. Pemmaraju, “A fault-containing self-stabilizing algorithm for spanning trees,” Journal of Computing and Information, vol.2, pp.322–338, 1996.

[8] 片山喜章，増澤利光，“重み最小生成木を構成する故障封じ込め自己安定プロトコル，” 信学論（D-I），vol.J84-D-I, no.9, pp.1307–1317, Sept. 2001.

[9] S. Kutten and B. Patt-Shamir, “Time-adaptive self-stabilization,” Proc. 16th PODC, pp.140–158, 1997.

[10] T. Herman, “Superstaibilizing mutual exclusion,” Proc. International Conf. on Parrallel and Distributed Systems, pp.31–40, 1995.

[11] S. Dolev and T. Herman, “Superstabilizing protocols for dynamic distributed systems,” Chicago Journal of Theoretical Computer Science, vol.3, no.4, pp.1–40, 1997.
[12] Y. Katayama, E. Ueda, H. Fujiwara, and T. Masuzawa, “A latency optimal superstabilizing mutual exclusion protocol in unidirectional rings,” JPDC, no.62, pp.865–884, 2002.
[13] H. Kakugawa and M. Yamashita, “A dynamic reconfiguration tolerant self-stabilizing token circulation algorithm in ad-hoc networks,” Proc. OPODIS, pp.179–186, 2004.
[14] S. Dolev, E. Schiller, and J. Welch, “Random wlak for self-stabilizing group communication in ad-hoc networks,” Proc. SRDS, pp.70–79, 2002.
[15] Y. Chen and J.L. Welch, “Self-stabilizing mutual exclusion using tokens in mobile ad hoc networks,” Proc. DIALM, pp.34–42, 2002.
[16] G. Antonoiu and P.K. Srimani, “Self-stabilizing depth-first multi-token circulation in tree networks,” Parallel Algorithms and Applications, vol.16, no.1, pp.17–35, 2001.
[17] A.K. Datta and S. Tixeuil, “Self-stabilizing distributed sorting on tree networks,” Parallel Algorithms and Applications, vol.16, no.1, pp.1–15, 2001.
[18] F. Petit, A. Cournier, A.K. Datta, and V. Villain, “Optimal snap-stabilizing PIF in un-oriented trees,” Proc. OPODIS 2001, pp 71–90, 2001.
[19] S. Ikeda, I. Kubo, N. Okumoto, and M. Yamashita, “Fair circulation of a token,” IEEE Trans. Parallel Distrib. Syst., vol.13, no.4, pp.367–372, 2002.
[20] 長谷川敏之, 経路情報を用いた生成木構成強安定プロトコルに関する研究, 名古屋工業大学大学院修士論文, http://tk-www.elcom.nitech.ac.jp/intro/dist-group.html, 2004.

（平成 17 年 2 月 14 日受付，5 月 11 日再受付）

片山　喜章　（正員）

平 2 阪大・基礎工・情報卒．平 6 同大大学院博士後期課程中退．同年奈良先端科学技術大学院大学情報科学研究科助手．平 7 同大情報科学センター助手．平 14 名工大工学部講師．平 15 年同大大学院講師．現在に至る．分散プロトコル，ネットワーク診断，ユビキタスコンピューティングなどに関する研究に従事．博士（工学）．情報処理学会会員．

長谷川敏之

平 14 名工大・工・電気情報卒．平 16 同大大学院・工学研究科博士前期課程卒．同年（株）日立製作所ソフトウェア事業部に所属，現在に至る．修士（工学）．

高橋　直久　（正員）

昭 49 電通大・応用電子卒．昭 51 同大大学院修士課程了．同年日本電信電話公社（現，NTT）武蔵野電気通信研究所入所．平 13 名工大電気情報工学科教授．平 15 名工大大学院工学研究科教授．この間，並列計算システム，ソフトウェア工学，ネットワークコンピューティングなどの研究に従事．工博（東工大）．情報処理学会，日本ソフトウェア科学会，日本データベース学会，ACM 各会員．