2007－AL－111 (2)
2007／3／9

WANET上でのクラスタ及び通信路構築自己安定アルゴリズムについて

宮永 慎太郎▷ 片山 喜章▷ 和田 幸一▷ 高橋 直久▷
小林 基成▷▷ 森田 正範▷▷

▷ 名古屋工業大学大学院 工学研究科 情報工学専攻 〒466-8555 愛知県名古屋市昭和区御器所町
▷▷ 株式会社 NTT ドコモ 総合研究所
E-mail: ▷ miyanaga@moss.elcom.nitech.ac.jp

概要：無線で通信を行なう無線端末のみで構成されるネットワークを無線アドホックネットワークと呼ぶ．本稿では，無線アドホックネットワーク上にクラスタを自己安定に構築するクラスタリングアルゴリズムと，構築されたクラスタ間の効率の良い通信手法を提案する．既存のクラスタリングアルゴリズムでは端末の移動時にクラスタ構造が大きく変化していたのに対し，提案アルゴリズムでは端末の移動に対してクラスタ構造の変化を少なくすることができる．

Self-Stabilizing Algorithms for Constructing Clusters and Communication Paths on WANET

Shintaro MIYANAGA▷, Yoshiaki KATAYAMA▷, Koichi WADA▷, Naohisa TAKAHASHI▷,
Motonari KOBAYASHI▷▷, Masanori MORITA▷▷

▷ Department of Computer Science and Engineering, Graduate School of Engineering,
Nagoya Institute of Technology
Gokiso, Showa, Nagoya 466-8555 Japan
▷▷ NTT DoCoMo,Inc. Research Laboratories
E-mail: ▷ miyanaga@moss.elcom.nitech.ac.jp

Abstract: The network consists only of wireless nodes communicating via radio called wireless ad hoc network(WANET). We proposed self-stabilizing clustering algorithms on WANET and communication method on the clusters. Previous clustering algorithms are not suitable for the case that nodes are moving. Our proposed clustering algorithms work efficiently for such a case.

1 はじめに

通信のためのインフラストラクチャを持たず，無線通信機能を持つ自律的に動作するノードによって構成されるネットワークを無線アドホックネットワーク (WANET) と呼ぶ．WANET では，ノードの能力（電源，通信範囲，通信周波数など）に制限があるため効率のよい通信手法が求められているが，単にノードを起動した状態の WANET 上では，メッセージの衝突や重複により効率の良い通信を行う事ができない．そこで，WANET 上で効率よく通信を行うための手法として，クラスタを利用したものが提案されている [1]．また，クラスタを構築する手法としてクラスタリングアルゴリズムが，WANET に限らず数多く提案されている [1, 2, 3, 4]．

しかし，無線アドホックネットワーク中の各ノードが移動する事を考慮する時，既存のものはそのまま適用しづらいという問題がある．例えば，通常の分散アルゴリズムではトポロジの変化が生じたときにはアルゴリズムの実行を一度リセットし，再び最初からアルゴリズムを実行してクラスタを構築する必要がある．また，自己安定（分散）アルゴリズムでも，トポロジの変化が生じたときには再安定までに多くの時間を必要とする．いずれの場合もトポロジの変化が頻繁に生じる時に問題となるため，ノードの移動が頻繁に生じる WANET にそのまま適用することは難しい．

動的なトポロジの変化に対応したクラスタリングアルゴリズムが内田ら [1] によって提案されている．この手法ではノードの発生によるトポロジの変化に対してクラスタ構造再構築のオーバヘッドを小さくする事が可能となっているが，ノードの移動には明示的に対応しておらず，ノードの消滅と出現の組み合わせで対応されている．また，アルゴリズムの動作中に新たなノードの発生や離脱が起こらないという制限があるため，各ノードの移動を考慮した WANET では，そのまま適用することが難しい．また，Colette ら [2] により WANET を対象とした自己安定クラスタリングアルゴリズムが提案されているが，ノードの移動が頻繁に起こるとクラスタ構造の変化が頻発するため，安定している（クラスタが正しく構築されている）時間が長く続かないという問題点がある．

そこで，本研究ではノードが移動してもクラスタ構造が変化しにくい自己安定クラスタリングアルゴリズムを提案する．さらに，シミュレーションによるクラスタリングアルゴリズムの解析を行い，従来の手法と比較する．

一方，無線アドホックネットワークで効率の良い通信を行う為には，クラスタ構造を構築するだけでは不十分である．クラスタヘッド間の通信を行う際に何の制御も行わないと，ネットワークを構成する全てのノードがメッセージを送信する可能性がある．そこで，本研究では提案アルゴリズムによって構築されたクラスタ構造を利用した，クラスタヘッド間を結ぶゲートウェイを自己安定に決定する手法を提案する．この提案手法により決定されたゲートウェイを利用すると，クラスタ間の通信がゲートウェイであるノードがメッセージ送信するだけで可能となるため，クラスタ間の効率の良い通信が可能となる．

2 モデル

本研究ではノードを節点とし，互いに直接通信が可能なノード間に辺が存在するとして，WANET を無向グラフ $G=(V,E)$(V：ノードの集合，E：無向辺の集合)として扱う．この時，各ノード v の送受信性能及び計算能力は等しく，それぞれ固有の識別子 (ID) 及び固有の重み (w_v) を持つ．また，ノードの移動や出現・消滅を許す．つまり，G はアルゴリズムの実行中にも変化するものとする．

各ノード v は要求に応じてパケットを送信する．ただし，これとは別に hello パケットと呼ばれるメッセージを定期的にやりとりする．hello パケットとはノードの状態を互いに交換するためのパケットであり，G 上で隣接するノード対は hello パケットの送受信を行う事で互いの情報を交換する．

本研究では，メッセージの衝突（コリジョン）が発生してもこれを検知し，再送制御等によりそれぞれのメッセージを正しく受信できるとする．

3 クラスタ構築自己安定アルゴリズム

本節では Colette ら [2] が提案したクラスタ構築自己安定アルゴリズム（以下 *SSCL* とする）を紹介し，更に，*SSCL* に対してノードの移動時にクラスタ構造の変化を小さくする変更を加えた 2 つのクラスタ構築自己安定アルゴリズム $SSCL_2$，$SSCL_3$ を提案する．

3.1 動作モデル

各アルゴリズム中であるノード v は，v が持つローカル変数によって評価される論理型変数 **Predicate** と，実行することで自分が持つローカル変数を変更するプログラム **Action** を持つ．

各 **Action** は対応する **Guard** が *True* になる時に実行される．この時，各 **Guard** は v が持つ **Predicate** の組み合わせにより評価される．また，あるノード v においてある **Guard** が *True* であり続けるならば，v は有限時間内に対応する **Action** を実行する．

3.2 構築されるクラスタ構造

提案する各クラスタ構築自己安定アルゴリズムによって構築されるクラスタ構造は以下の通りである．　無向グラフで表現された無線ネットワーク $G=(V,E)$ 上の極大独立点をクラスタヘッドと呼び，それ以外のノードをメンバと呼ぶ．各クラスタヘッドを中心としたスターグラフをクラスタと呼び，一つのクラスタはクラスタヘッドとメンバによって構成される．ただし，メンバであるノードは必ずただ一つのクラスタに所属し，所属するクラスタのクラスタヘッドと隣接している．

3.3 各ノードが保持する情報

G 上の各ノード v は，各クラスタ構築アルゴリズムで利用する以下の情報を保持している．

- Ch_v ：*True* のとき v はクラスタヘッド．
- $Clusterhead_v$ ：v の所属するクラスタのクラスタヘッド ID．v 自身がクラスタヘッドなら v の $ID(v.ID)$ が格納される．
- $id_{neighbor}$ ：v と G 上で隣接するノード ($neighbor$) の ID．
- $w_{neighbor}$ ：ノード $neighbor$ の重み．
- $Clusterhead_{neighbor}$：ノード $neighbor$ のクラスタヘッド ID．
- $Timer$ ：エントリの有効時間．
- $Table\_neighbor$ ：各隣接ノードに対して，$id_{neighbor}$，$w_{neighbor}$，$Clusterhead_{neighbor}$，$Timer$ の四組を 1 つのエントリとして保持する表形式データ (表 1)．

$id_{neighbor}$	$w_{neighbor}$	$Clusterhead_{neighbor}$	$Timer$
≀	≀	≀	≀

表 1: データテーブル：$Table\_neighbor$

このとき，$Clusterhead_{neighbor}$ が $id_{neighbor}$ であるならば，v は G 上で隣接するノード $neighbor$ がヘッドであることが判別できる．また，各ノード v は hello パケットの定期的なやり取りにより $Table\_neighbor$ を更新しているものとする．

3.4 *SSCL*

Colette らが提案したクラスタ構築アルゴリズム [2] を図 1 に示す．各ノード v は自分より重みの大きいクラスタヘッドと隣接する時メンバになり，隣接する一番重みの大きいクラスタヘッドのクラスタに所属する．また，*SSCL* では，各ノード v は自分より重みの大きいノードと隣接しないとき，既に隣接ノードにクラスタヘッドが存在していても新たにクラスタヘッドになる事に注意する．

・*SSCL* の問題点

ノードが移動するネットワークに *SSCL* を適用する事を考えると，全体としてクラスタ構造が安定している時間が長く続かない可能性がある．G 中でもっとも重みが大きいクラスタヘッド h_1 が移動する時を考える．h_1 のクラスタに所属するノードの中に G 中で 2 番目に重みが大きいノードが存在したとすると，このノードは h_1 が移動により隣接しなくなると，既に隣接ノードの中に別のクラスタヘッドが存在していたとしても新たにクラスタヘッドになってしまう．このように，ノードの移動によるトポロジの変化が頻繁に発生する状況では，新たなクラスタヘッドの発生により再構成する必要のない既存クラスタを破壊・再構成することとなり，クラスタ構造の変化が頻繁に発生する可能性がある．

そこで，メンバであるノードは隣接ノードの中にクラスタヘッドが存在する場合にはヘッドにならないように *SSCL* の処理を変更する事で，ノードの移動に対しクラスタ構造の変化を少なくする事が可能なクラスタ構築自己安定アルゴリズム $SSCL_2$ を次小節で提案する．

01: **Predicates**
02: $G_1(v) \equiv (\forall z \in N_v : (Ch_z = F) \vee (w_v > w_z))$;
03: $G_2(v) \equiv (Ch_v = F) \vee (Clusterhead_v \neq v)$;
04: $G_3(v) \equiv (Ch_v = T) \vee (Clusterhead_v \neq max_{w_z}\{z \in N_v : Ch_z = T\})$;
05: **Rules**
06: $R_1(v) : G_1(v) \wedge G_2(v) \rightarrow Ch_v := T$; $Clusterhead_v := v$;
07: $R_2(v) : \neg G_1(v) \wedge G_3(v) \rightarrow Ch_v := F$; $Clusterhead_v := max_{w_z}\{z \in N_v : Ch_z = T\}$;

図 1: Colette らのクラスタ構築アルゴリズム ($SSCL$)

01: **Predicates**
02: $G_1(v) \equiv (\forall z \in N_v : Ch_z = F)$;
03: $G_2(v) \equiv (Ch_v = T) \wedge (\exists z \in N_v : (Ch_z = T) \wedge (w_v < w_z))$;
04: $G_3(v) \equiv (Ch_v = F)$;
05: $G_4(v) \equiv (Clusterhead_v \neq v)$;
06: $G_5(v) \equiv (Clusterhead_v \neq max_{w_z}\{z \in N_v : Ch_z = T\})$;
07: **Rules**
08: $R_1(v) : G_1(v) \wedge (G_3(v) \vee G_4(v)) \rightarrow Ch_v := T$; $Clusterhead_v := v$;
09: $R_2(v) : \big((\neg G_1(v) \wedge G_3(v)) \vee G_2(v)\big) \wedge G_5(v)$
$\rightarrow Ch_v := F$; $Clusterhead_v := max_{w_z}\{z \in N_v : Ch_z = T\}$;

図 2: $SSCL_2$

3.5 $SSCL_2$

トポロジの変化に対しクラスタ構造の変化を小さくすることができるクラスタ構築自己安定アルゴリズム $SSCL_2$ を図 2 に示す．$SSCL_2$ では，メンバであるノードは隣接ノードの中にクラスタヘッドが存在する場合にはクラスタヘッドにならないように $SSCL$ の処理を変更する．

つまり，$SSCL_2$ では，メンバであるノード m はクラスタヘッドと隣接しないときにのみクラスタヘッドになる（図 2，08 行）．これにより，新たなクラスタヘッドの発生により既存のクラスタ構造が壊れてしまう問題点を解決する事ができる．

$SSCL$ では各クラスタヘッドが必ず自分のメンバより重みが大きかったのに対し，$SSCL_2$ では自分のメンバより重みの小さなクラスタヘッドが存在する可能性がある．ここでノードの重みを耐久性や信頼性の高さと考えた場合，より重みが大きなノードがクラスタヘッドになる $SSCL$ の方が望ましいアルゴリズムと考えられる．従って，ノードの移動を考えるとき，「トポロジの変化に対するクラスタ構造の変化の少なさ」と「信頼性の高いクラスタヘッドの選択」のトレードオフとなる．

3.5.1 $SSCL_2$ の正しさの証明

定理 1. $SSCL_2$ は，有限時間内に正しくクラスタ構造を構築する自己安定アルゴリズムである．

略証. $SSCL_2$ の $SSCL$ からの変更点は，メンバであるノード m が新たにクラスタヘッドになる条件が m の隣接ノードにクラスタヘッドが存在しない時のみに変更された点である（図 2，08 行）．これは，$SSCL$ から互いに隣接するクラスタヘッドが新たに発生するような動作が無くなっただけであり，隣接するクラスタヘッドが存在しないノードは新たなクラスタヘッドになる事は可能である．ノードがメンバになる条件は $SSCL$ と同様であるため（図 2，09 行），正しいクラスタ構造への収束性は変わらず保証される． □

3.5.2 $SSCL_2$ の改良

$SSCL_2$ では $SSCL$ と同様に，各メンバ m は自分の $Clusterhead_m$ を自分と隣接するクラスタヘッドのうち一番重みの大きいものへと常に更新する為，クラスタ構造の変化が頻繁に発生する可能性がある．そこで，メンバであるノードは自分の所属するクラスタのクラスタヘッドが存在する限り，所属するクラスタの変更を行わないように $SSCL_2$ の処理を変更する．このようにする事で，ノードの出現や移動によるクラスタ構造の変化を $SSCL_2$ より更に小さくすることができる．

以上のアイデアを実現する，トポロジの変化に対し $SSCL_2$ より更にクラスタ構造の変化を小さくすることが可能なクラスタ構築自己安定アルゴリズム $SSCL_3$ を次小節で提案する．

3.6 $SSCL_3$

提案するクラスタ構築自己安定アルゴリズム $SSCL_3$ を図 3 に示す．

$SSCL_2$ に対して，$SSCL_3$ では，メンバであるノードは所属するクラスタのクラスタヘッドが存在する限り，所属するクラスタを変更しないように $SSCL_2$ の処理を変更する（図 3，10 行）．これにより，クラスタヘッドの移動による各メンバの所属するクラスタの変更回数を少なくする事ができる．

3.6.1 $SSCL_3$ の正しさの証明

定理 2. $SSCL_3$ は，有限時間内に正しくクラスタ構造を構築する自己安定アルゴリズムである．

略証. $SSCL_3$ の $SSCL_2$ からの変更点は，メンバであるノード m が行う所属するクラスタの変更が，所属するクラスタが壊れたときにのみ行われるようになった点のみである（図 3，10 行）．従って，各ノードが起こすステータスの変更部分の動作は $SSCL_2$ と同様であるため，正しいクラスタ構造への収束性は変わらず保証される． □

01: **Predicates**
02: $G_1(v) \equiv (\forall z \in N_v : Ch_z = F)$; : ヘッドと隣接しない
03: $G'_1(v) \equiv (\exists z \in N_v : (Ch_z = T) \wedge (w_v < w_z))$; : 自分より重たいヘッドと隣接
04: $G''_1(v) \equiv (\exists z \in N_v : (Ch_z = T) \wedge (w_v > w_z))$; : 自分より軽いヘッドと隣接
05: $G_2(v) \equiv (Ch_v = F) \vee (Clusterhead_v \neq v)$; :
06: $G_3(v) \equiv (Ch_v = T)$; :
07: $G_4(v) \equiv (Ch_{Clusterhead_v} = F)$; : 自分の親がメンバになっている
08: **Rules**
09: $R_1(v) : G_1(v) \wedge G_2(v) \rightarrow Ch_v := T;\ Clusterhead_v := v;$
10: $R_2(v) : \left(G'_1(v) \wedge (G_3(v) \vee G_4(v))\right) \vee \left(\neg\ G'_1(v) \wedge G''_1(v) \wedge G_4(v) \right)$
$\rightarrow Ch_v := F;\ Clusterhead_v := max_{w_z}\{z \in N_v : Ch_z = T\};$

図 3: $SSCL_3$

4 通信路構築アルゴリズム

前節で示した自己安定アルゴリズムによってクラスタ構造が構築された後，クラスタ間の通信を効率良く行うための通信路を構築する自己安定アルゴリズムを提案する．

4.1 必要な性質

前節のアルゴリズムの実行によりネットワークに構築されたクラスタ構造で，あるクラスタ内のノードから隣接する（一方のクラスタ内のノードから他方のクラスタ内のノードへのリンクが G 上に存在する）クラスタ内のノードへ通信を行う事を考える．

この時，2つのクラスタヘッド間に通信路が構築されていれば，各クラスタ内のメンバは自分のクラスタヘッドにメッセージを連絡し，クラスタヘッド間で通信路を用いたメッセージ交換をする事で，互いに隣接するクラスタに所属するノード同士の通信が可能になる．しかし，前節のアルゴリズムを実行するだけではクラスタが構築されるだけであり，互いに隣接するクラスタのクラスタヘッド同士の通信路は用意されていない．

2つのクラスタのクラスタヘッド間の通信を行うためには，2つのクラスタ内に存在するメンバを中継して通信を行う必要がある．この時 2 つのクラスタヘッド間で互いのメッセージを中継するメンバをゲートウェイと呼ぶ．

本節では各クラスタ構築アルゴリズムによってクラスタ構造が構築された後，隣接するクラスタのクラスタヘッド間を連結するゲートウェイを決定する以下の2つの方針による手法を提案する．

- **方針 1** メッセージ送信時に相手クラスタヘッドへのゲートウェイを決定する（2 つのクラスタヘッド間を往復する経路が異なる可能性がある）．
- **方針 2** 予め隣接するクラスタヘッド間を結ぶゲートウェイを一意に決定しておく（2 つのクラスタヘッド間を往復する経路が同一のものになる）．

ノードの移動が発生する状況では，2つのクラスタヘッド間に設定した通信路がメッセージを送信する瞬間にも存在している事が保証されない．そのため，方針 1 では各ノードがメッセージを送信する瞬間にゲートウェイとなるノードを決定する事で，より新しいトポロジ状況に合わせたゲートウェイを決定する（ただし，2 つのクラスタヘッド間を往復する経路が異なる可能性がある）．

2つのクラスタヘッド間を結ぶ経路が同一である事が求められる場合には，方針 2 を利用する．方針 2 の手法は方針 1 の手法に拡張を加える事で実現する．

4.2 方針 1 による通信路構築法

4.2.1 基本方針

各クラスタヘッドは，隣接するクラスタそれぞれに対し，常にゲートウェイの候補を保持しておく．あるクラスタへの送信要求が発生すると，そのノードに対してメッセージを（宛先クラスタのクラスタヘッド ID を添えて）送信する．このメッセージを受信したノード（ゲートウェイ）は添付されている宛先クラスタのクラスタヘッド ID より，自分と隣接するノードの中から転送先を一つ選択してメッセージを転送する．この時，転送されたメッセージを受信したノードが宛先クラスタのメンバであるなら，自分のクラスタヘッドへ更に転送する事で，クラスタヘッド間のメッセージ送信が完了する．

以上を実現するために，各ノード v はどの隣接ノードにメッセージを送信すれば，どの隣接クラスタのクラスタヘッドへメッセージが到達できるかを認識している必要がある．そこで，各ノード v は「自分の隣接ノードの ID のリストと，その各隣接ノードが所属するクラスタのクラスタヘッド ID」を表形式で（*Table_gateway* とする）保持する（詳細は後述）．

また，クラスタヘッド間の通信を正しく行うために，各ノード v は *Table_gateway* を常に最新の状態に更新する必要がある．本研究では，hello パケットにより情報を更新するものとする（詳細は後述）．

その後，実際にクラスタヘッド間で通信を行う際には，メッセージを送信するノードは相手のクラスタヘッドへの最短経路となるゲートウェイを自分が持つ *Table_gateway* から決定しなければならない．

4.2.2 *Table_gateway*

各ノード v はそれぞれクラスタ構築アルゴリズムの為に保持している情報に加え，各エントリに以下の要素を含むデータテーブル *Table_gateway*(表 2) を保持する．

このとき，隣接する 2 つのクラスタに対し，両方のクラスタヘッドとのリンクを持つノードを *DirectNeighbor*(DN)，一方の（自分の）クラスタヘッドとの間にリンクを持ち，かつ他方のクラスタ内のメンバとの間にリンクを持つノードを *InDirectNeighbor*(IDN) と表記する．

$C_{neighbor}$	$gateway$	DN	$Timer$
≀	≀	≀	≀

各エントリにはタイマーが用意されているとする.

表 2: データテーブル : *Table_gateway*

vが保持する *Table_gateway* は，vのステータスによって各フィールドの意味が異なる.

・vがクラスタヘッドのとき

- $C_{neighbor}$: vのクラスタと隣接するクラスタのクラスタヘッド *ID*
- *gateway* : $C_{neighbor}$ のクラスタに所属するノードとリンクを持つ自分のメンバの *ID*，又は v とリンクを持ち $C_{neighbor}$ のクラスタに所属するノードの *ID*
- *DN* : *gateway* が $C_{neighbor}$ への *DN* であるか *IDN* であるかを示す．この値が 1 のとき *DN*，0 のとき *IDN* とする
- *Timer* : エントリの有効時間.

・vがメンバのとき

- $C_{neighbor}$: 自分が所属するクラスタを除く，v が隣接するクラスタのクラスタヘッド *ID*
- *gateway* : $C_{neighbor}$ のクラスタに所属する v の隣接ノード *ID*
- *DN* : v 自身が $C_{neighbor}$ への *DN* であるか *IDN* であるかを示す．この値が 1 のとき *DN*，0 のとき *IDN* とする
- *Timer* : エントリの有効時間.

各エントリにはタイマー（生存時間）が用意されており，時間経過にあわせて値を 0 から増加させてゆき，一定値になったときそのエントリを廃棄する.

4.2.3 *Table_gateway* の更新法

Table_gateway の更新手順について説明する．各ノード v は自分のステータスに従い，次の情報を hello パケットに乗せて定期的に送信する.

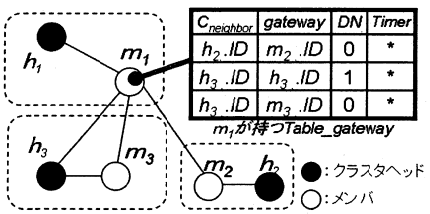


図 4: *Table_gateway* の更新法

vがクラスタヘッドの場合，v 自身の *ID* とステータス，すなわち $(v.ID, v.status)$ を送信する．一方，vがメンバの場合，v 自身の *ID* と所属するクラスタのクラスタヘッド *ID* に加え，v が隣接するその他のクラスタに関する情報を送信する．具体的には，v が自分とは異なるクラスタに所属するノードと隣接している場合に，その隣接ノードが所属するクラスタのクラスタヘッド *ID* および v がそのクラスタに対する *DN* であるかどうかを送信する．つまり，v は *Table_gateway* の全エントリを確認し，各 $C_{neighbor}$ に対して $(v.ID,\ Clusterhead_v, C_{neighbor},\ DN)$ を送信する.

このとき，*DN* が 1 であるエントリが存在する場合には $DN = 1$，それ以外は $DN = 0$ とする.

各メンバが送信する情報について，図 4 を用いて説明する．図 4 で m_1 は，*Table_gateway* に 3 つのエントリを保持しており，この中には 2 種類の $C_{neighbor}$ が存在する．この時，m_1 は h_2 に関しては *DN* が 0 のエントリしか持たないが，h_3 に関しては *DN* が 1 と 0 のエントリが存在する．従って，m_1 は $(m_1.ID, h_1.ID, h_2.ID, 0)$ と $(m_1.ID, h_1.ID, h_3.ID, 1)$ を送信する.

これらのメッセージを受信したノード v が行う動作は以下の 5 通りに分類される．この時，クラスタ構築アルゴリズム上で v が持つ変数 $Clusterhead_v$ を利用する.

(1) vがクラスタヘッドで，自分のメンバ m からメッセージを受信したとき：
受信した $C_{neighbor}$ の各種類に対し，以下の作業を実行する．*Table_gateway* 内にエントリ $<C_{neighbor}, m.ID, *, *>$ が存在するとき，同エントリに $<C_{neighbor}, m.ID, DN, 0>$ を上書きする．存在しない時は，*Table_gateway* にエントリ $<C_{neighbor}, m.ID, DN, 0>$ を追加する.

(2) vがクラスタヘッドで，自分のクラスタ外のメンバ m からメッセージを受信したとき：
Table_gateway 内にエントリ $<Clusterhead_m, m.ID, *, *>$ が存在するとき，同エントリに $<Clusterhead_m, m.ID, 1, 0>$ を上書きする．存在しない時は，*Table_gateway* にエントリ $<Clusterhead_m, m.ID, 1, 0>$ を追加する.

(3) vがメンバで，自分の所属するクラスタ外のメンバ m からメッセージを受信したとき：
Table_gateway 内にエントリ $<Clusterhead_m, m.ID, *, *>$ が存在するとき，同エントリに $<Clusterhead_m, m.ID, 0, 0>$ を上書きする．存在しない時は，*Table_gateway* にエントリ $<Clusterhead_m, m.ID, 0, 0>$ を追加する.

(4) vがメンバで，自分の所属するクラスタ以外のクラスタヘッド h からメッセージを受信したとき：
Table_gateway 内にエントリ $<h.ID, h.ID, *, *>$ が存在するとき，同エントリに $<h.ID, h.ID, 1, 0>$ を上書きする．存在しない時は，*Table_gateway* にエントリ $<h.ID, h.ID, 1, 0>$ を追加する.

(5) 上記以外の場合：
何も実行せず，受信したメッセージを無視する.

一方，*Table_gateway* の各エントリにはタイマーを設定しておき，一定時間更新されなかったエントリを削除する．また，v のステータスが変化したとき，及び $Clusterhead_v$ が変化したとき，v は *Table_gateway* ら全エントリを削除する.

4.2.4 *Table_gateway* を利用した通信手法

各ノードが保持する *Table_gateway* を利用して，実際に通信を行う手法について説明する．すなわち，各クラスタヘッドが自分のクラスタと隣接するクラスタのクラスタヘッドへメッセージを送信する際に実行する手順について説明する．以下，メッセージ送信元クラスタヘッドを *Sender*，メッセージの送信先クラスタヘッドを *Dest* とする.

・クラスタ間通信用メッセージ
隣接するクラスタ間の通信を実現するために用いられるメッセージ構造を定義する．*Sender*は，*Dest*に送信したいメッセージの内容と共に，*Dest*の*ID*とゲートウェイとなるノードの*ID*を指定して送信する．

・通信用メッセージ：(*Dest.ID*, *To.ID*, *Message*)

・*Dest.ID*： *Dest*の *ID*.
・*To.ID*：送信先ノード（ゲートウェイまたは*Dest*）の *ID*.
・*Message*： *Sender*が*Dest*へ送信するメッセージの内容.

・ゲートウェイ選択用関数 *selectnode*
*Sender*及び*Sender-Dest*間のゲートウェイとして選択されたノードは，それぞれ自分が持つ*Table_gateway*を利用して*Dest*への最適なゲートウェイとなるノードを一つ選択し，通信用メッセージを送信する．各ノードがゲートウェイ選択の際に実行する関数をここで定義する．

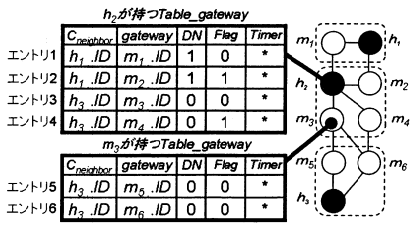
・**int** *selectnode*(**int** *Dest.ID*)
自身が持つ *Table_gateway* の中で $C_{neighbor}$ の値が *Dest.ID* であるエントリ集合の中から，(*DN*,*gateway*) の辞書式順序で最も大きいエントリの *gateway* の値 (*ID*) を出力する．

・クラスタ間通信の手順

・メッセージ送信元クラスタヘッド *Sender* の動作：(*Dest.ID*, *selectnode*(*Dest.ID*), *Message*) を送信．

・メッセージ (*Dest.ID*, *To.ID*, *Message*) を受信したノード v (ゲートウェイ) が実行する動作：
受信した *To.ID* が v 自身の *ID* の時，vは通信用メッセージ (*Dest.ID*, *selectnode*(*Dest.ID*), *Message*) を送信する．ただし，受信した *Dest.ID* が vのクラスタヘッド *ID* であった時は，vは通信用メッセージ (*Dest.ID*, *Dest.ID*, *Message*) を送信する．また，*Dest*が v 自身の *ID* である時は，*Message* を受信して終了する．

図 5 で h_2 が h_3 にメッセージを送信する例を考える．この時，h_2 ではゲートウェイとしてエントリ 3 が選択される (*selectnode*(h_3.*ID*) == m_3.*ID*)．従って，h_2 はメッセージ (h_3.*ID*, m_3.*ID*, *Message*) を送信する．m_3 はゲートウェイとしてエントリ 5 を選択し，(h_3.*ID*, m_5.*ID*, *Message*) を送信する．m_5 は受信した *Dest.ID* が自分のクラスタヘッド *ID* と一致するため，(h_3.*ID*, h_3.*ID*, *Message*) を送信する．



※各メンバの *ID* は m_1.*ID* > m_2.*ID* > ... > m_6.*ID* であるとする．
図 5: クラスタヘッド間の通信手法

4.2.5 通信手法の正しさの証明

補題 1. 提案する通信手法では，2つのクラスタヘッド間の最短経路を用いて通信が可能である．

略証. 互いに隣接するクラスタのクラスタヘッド間の最短経路は *DN* か，*DN* が存在しない時は *IDN* を中継する場合である．
提案する手法では，2 つのクラスタヘッド間に *DN* が存在する時は必ず *DN* を用いて通信を行うため，この時最短経路を用いて通信が可能なのは明らかである．2 つのクラスタヘッド間に *IDN* のみ存在する時，各 *IDN* が持つ *Table_gateway* の中には同じクラスタに所属するメンバへの経路情報は含まれていない為，*IDN* からは必ず他方のクラスタに所属する *IDN* へメッセージが送信される．従って，*IDN* のみ存在する場合も最短経路を用いて通信が可能となる． □

補題 2. 提案する通信手法では，2つのクラスタヘッド間のゲートウェイのみがメッセージを送信する．

証明. アルゴリズムより明らか． □

定理 3. 提案する通信手法を用いる事で，隣接するクラスタのクラスタヘッド間の双方向それぞれの最短経路を用いた通信が正しく行える．

証明. 補題 1，2 より明らか． □

4.3 方針 2 による通信路構築法

本小節では，前節で説明した手法に拡張を加えた，隣接するクラスタヘッド間の往復経路を一意に決定するゲートウェイ決定アルゴリズムについて説明する．

4.3.1 基本方針

方針 2 では方針 1 と同様に，各クラスタヘッドはメッセージを隣接クラスタへ送信したい時，そのクラスタへのゲートウェイ候補の中からノードを一つ選択して，そのノードに対してメッセージを（宛先クラスタのクラスタヘッド *ID* を添えて）送信する．
ただし，方針 1 では両方のクラスタヘッドがそれぞれ自分から見た最適なゲートウェイ（経路）を送信時に決定し，メッセージを送信していたのに対し，方針 2 では「強いクラスタヘッド」からのメッセージを受信した「弱いクラスタヘッド」は，以後「強いクラスタヘッド」の選択した経路を利用する．これにより，互いに隣接するクラスタのクラスタヘッド間の通信路が，双方向で一致する．ここで，より *ID* の小さなクラスタヘッドが「強い」クラスタヘッドとする．
以上を実現するために，ネットワーク中の各ノードは方針 1 と同様に，*Table_gateway* を保持する．この時，各ノードが持つ *Table_gateway* の各エントリに，自分より強いクラスタヘッドが使用した経路を表す為の要素 *Flag* を追加する．

4.3.2 各ノード v が保持する情報

各ノード v はエントリに以下の要素を含むデータテーブル *Table_gateway*(表 3) を保持する．方針 2 で使用する *Table_gateway* は，方針 1 で使用した *Table_gateway* の各エントリに新たな要素 *Flag* を追加したものである．

新たな要素 *Flag* は，v のステータスによって次の意味を持つ．v がクラスタヘッドのとき，v が受信した v より強い（*ID* の小さい）隣接クラスタのクラスタヘッドからのメッセージを v へ送信したノードを示す．v は自分の隣接クラスタへのゲートウェイ候補であるエントリの中にこの値が 1 であるエントリが存在するとき，そのエントリの *gateway* で指定されるノードへメッセージを送信する．一方，全てのエントリで *Flag* の値が 0 の時は，方針 1 と同じ動作を行う．v がメンバの時，*Flag* の値は利用しないため，全て 0 とする．

Table_gateway の各エントリにはタイマー（生存時間）が用意されており，時間経過にあわせて値を 0 から増加させてゆき，一定値になったときそのエントリを廃棄する．

$C_{neighbor}$	*gateway*	*DN*	*Flag*	*Timer*
≀	≀	≀	≀	≀

各エントリにはタイマーが用意されているとする．

表 3: データテーブル：*Table_gateway*

4.3.3 各ノードが持つ *Table_gateway* の更新法

方針 2 における *Table_gateway* の更新手順について説明する．*Table_gateway* の要素 *Flag* 以外の更新については，方針 1 のものと全く同様のメッセージ・手順を利用する．各クラスタヘッドが持つ *Table_gateway* 内の要素 *Flag* は，実際に *Table_gateway* を利用した通信を行うことで更新される．

つまり，方針 1 では hello パケットの送受信のみで *Table_gateway* の更新が行えるのに対し，方針 2 では hello パケットの送受信に加えて隣接クラスタ間で実際に通信が起こったときに，*Table_gateway* 内の要素 *Flag* が更新される．

そこで，要素 *Flag* の更新手順と共に，方針 2 での隣接クラスタ間の通信手法について説明する．メッセージ送信元クラスタヘッドを *Sender*，メッセージの送信先クラスタヘッドを *Dest* とする．

4.3.4 *Table_gateway* を利用した通信手法

・クラスタ間通信用メッセージ

方針 2 で隣接するクラスタ間の通信を実現するために用いられるメッセージ構造を定義する．

・通信用メッセージ
(*Sender.ID*, *Dest.ID*, *From.ID*, *To.ID*, *Message*)

- *Sender.ID*： *Sender* の *ID*.
- *Dest.ID*： *Dest* の *ID*.
- *From.ID*： 通信用メッセージの送信元ノード *ID*.
- *To.ID*： 通信用メッセージの宛先ノードの *ID*.
- *Message*： *Sender* が *Dest* に送信するメッセージの内容.

・ゲートウェイ選択用関数 *selectnode*
各クラスタヘッドは *Table_gateway* 内に存在する，$C_{neighbor}$ に送信先のクラスタヘッド *ID* が入っているエントリ集合の中に *Flag* が 1 であるエントリが存在する時，そのエントリの *gateway* に入っているノードへ相手の *ID* と共にメッセージを送信する．一方，*Flag* が 1 のエントリが存在しない時は，方針 1 の手順に従う．各メンバが実行する手順は方針 1 の手順と同様である．

方針 2 でゲートウェイ選択の際に実行する関数を定義する．

- **int** *selectnode*(**int** *Dest.ID*)
  自身が持つ *Table_gateway* のエントリの中で，$C_{neighbor}$ が *Dest.ID* であるエントリ集合のうち，(*Flag*,*DN*,*gateway*) の辞書式順序で最も大きいエントリの *gateway* の値 (*ID*) を出力する

・クラスタ間通信の手順
方針 2 では以下のような手順により隣接するクラスタ間の通信を実現する．

- メッセージ送信元クラスタヘッド *Sender* の動作：(*Sender.ID*, *Dest.ID*, *Sender.ID*, *selectnode*(*Dest.ID*), *Message*) を送信．

- メッセージ (*Sender.ID*, *Dest.ID*, *From.ID*, *To.ID*, *Message*) を受信したノード v が実行する動作：
  受信した *To.ID* が v 自身の *ID* の時，v は通信用メッセージ (*Sender.ID* , *Dest.ID*, *v.ID*, *selectnode*(*Dest.ID*), *Message*) を送信する．ただし，受信した *Dest.ID* が v のクラスタヘッド *ID* であった時は，v は通信用メッセージ (*Sender.ID* , *Dest.ID*, *v.ID*, *Dest.ID*, *Message*) を送信する．また，*Dest* が v 自身の *ID* である時は，*Message* を受信して終了する．
  Dest が *Message* を確認した際に，*Sender.ID* が *Dest.ID* より小さければ，*Dest.ID* は自分の *Table_gateway* 内のエントリ < *Sender.ID*, *From.ID*, *, *, *> の *Flag* を 1 にする．また，< *Sender.ID*, *From.ID*以外, *, *, *> の *Flag* を 0 にする．

図 5 で h_2 が h_3 にメッセージを送信する例を考える．この時，方針 1 とは異なり h_2 ではゲートウェイとして *Flag* が 1 であるエントリ 4 が選択される (*selectnode*(h_3.*ID*)==m_4.*ID*)．従って，方針 2 では h_2 はメッセージ (h_2.*ID*, h_3.*ID*, h_2.*ID*, m_4.*ID*, *Message*) を送信し，往復の経路を一意にする．

4.3.5 通信手法の正しさの証明

補題 3. 提案する通信手法では，2 つのクラスタヘッド間の最短経路を用いて通信が可能である．

証明. 補題 1 と同様である． □

補題 4. 提案する通信手法では，2 つのクラスタヘッド間のゲートウェイのみがメッセージを送信する．

証明. 補題 2 と同様である． □

補題 5. 提案する通信手法では，やがて 2 つのクラスタヘッド間の通信経路の往復が一意に決定する．

証明. 隣接する 2 つのクラスタのクラスタヘッドを h_1, h_2 とし，h_1 の方が強いとする．また，h_1 からのメッセージが $h_1 - g_1 - g_2 - h_2$ という経路で h_2 に到達

			$SSCL$	$SSCL_2$	削減率	$SSCL_3$	削減率
802.11normal	静止	36node	901.8	612.4	0.320914	82	0.909071
		100node	4091	1931.4	0.52789	455.8	0.888585
	1-5m/s	36node	1850.8	1525	0.176032	1242.2	0.328831
		100node	5953.6	4821.4	0.190171	3765.4	0.367542
802.11abstract	静止	36node	47	35.4	0.246809	33.2	0.293617
		100node	165.8	112.2	0.323281	104.6	0.369119
	1-5m/s	36node	1761.6	1540.4	0.125568	1383	0.214918
		100node	6828.8	5691.4	0.166559	4867.4	0.287225

表 4: シミュレーション結果

したとする．h_2 は自分より強いクラスタヘッド h_1 からのメッセージが到達した時点で，g_2 に $Flag$ をつける．以後，h_2 が h_1 にメッセージを送信する際，g_2 に $Flag$ が付いている為 h_2 は g_2 にメッセージを送信する．

この時，h_1 が g_1 を選択しているので g_1 は $h_1 - h_2$ 間の最適なゲートウェイである．従って，g_2 も g_1 を選択する（g_2 が選んだ最適なゲートウェイが g_1 以外であったなら，h_1 はそのノードをゲートウェイとして選択しているはずである）．結果，h_2 から h_1 へのメッセージは $h_2 - g_2 - g_1 - h_1$ という経路を辿るため，2 つのクラスタヘッドから送信されるメッセージは同じ経路を使用する． □

定理 4. 提案する通信手法を用いる事で，隣接するクラスタのクラスタヘッド間の双方向それぞれが同一の最短経路を用いた通信が正しく行える．

証明. 補題 3，4，5 より明らかである． □

5 評価

ネットワークシミュレータを用いて，一定のエリア内に状態がメンバであるノードをランダムに配置し，この状況から 3 つの自己安定クラスタリングアルゴリズム $SSCL$，$SSCL_2$，$SSCL_3$ を 1 時間動作させた時に発生するノードの状態変化回数（各ノードのステータスの変更または所属するクラスタの変更回数）をカウントする．この時，状態変化回数が少ないほどノードの移動に対してクラスタ構造の変化が少ないと考えられ，安定性を評価する事ができる．

5.1 シミュレーションの詳細

ネットワークシミュレータ Qualnet3.8 上で 1500m 平方のエリアにノードを 36 個及び 100 個ランダムに配置する．各ノードの無線 LAN 規格を IEEE802.11b(2Mbps) とし (802.11normal)，各ノードが静止している場合と，1m/s ～ 5m/s（平均 2.5m/s）で常時ランダムに移動する場合について 3 つのアルゴリズム $SSCL$，$SSCL_2$，$SSCL_3$ を 1 時間動作させ，発生するノードの状態変化回数をカウントする．

また，同様のシミュレーションを雑音等を考えない理想的な状態の下で行い (802.11abstract)，802.11normal の結果と比較した．

5.2 シミュレーションの考察

シミュレーションの結果を表 4 に示す．表 4 より，2 つの状態のいずれにおいても $SSCL_2$ 及び $SSCL_3$ を適用することで，$SSCL$ よりもノードの状態変化の回数を削減できる事が確認できた．従って，提案アルゴリズムではノードが移動する状況に対してクラスタ構造の安定している時間を従来のものより長くする事が可能である．

ここで，802.11abstract の結果に注目すると，$SSCL_2$ 及び $SSCL_3$ を適用したときの状態変化回数が 802.11normal の場合よりも少なくなっている．これは，802.11normal では 802.11abstract と異なり実際の動作が想定されているため，雑音等により各ノードが持つ隣接ノード情報の更新が不完全になり，無駄な状態変化が発生しているからと考えられる．

また，この無駄な状態変化が $SSCL_2$ 及び $SSCL_3$ により少なくなることで，802.11normal では状態変化回数の削減率が大きくなっているものと考えられる．このことより提案アルゴリズムは，$SSCL$ に比べ実際の WANET 環境下で有効であると考える．

6 まとめと今後の課題

シミュレーションの結果から，提案する自己安定クラスタリングアルゴリズムでは従来のものよりも安定性が増すことが確認できた．今後の課題は，安定な状態に到達するまでの収束時間をより早くすることと，提案方式を用いての通信路を実際のルーティングに利用する事等である．

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費補助金，(基盤 (C)16500028) 並びに平成 16 年度文部科学省科学研究費補助金（特定研究：新世代の計算限界），平成 17 年度日本学術振興会科学研究費補助金（基盤 (C)17500036），電気通信普及財団研究調査助成の研究助成によるものである．

参考文献

[1] J. Uchida, A.K.M. Muzahidul.I, Y. Katayama, W. Chen, K. Wada, Construction and maintenance of a cluster-based architecture for sensor networks, HICSS06, Track9, 2006.

[2] C. Johnen, L.H. Nguyen, Self-Stabilizing Clustering Algorithm for Ad hoc Networks,AlgoSensors 2006, L.R.I., July 2006 (pdf) also: research report no. 1429, January, 2006

[3] S. Basagni, Distributed clustering for ad hoc networks, ISPAN '99, pp.310-315, Washington DC, USA, 1999.

[4] J. Wu, H. Li, On calculating connected dominating set for efficient routing in ad hoc wireless networks, Telecommunication Systems, Vol. 18, no.1/3, pp.13-36, 2001.