モバイルコンピューティングと
ワイヤレス通信 24−21
（2003．3．7）

移動計算機環境におけるグループ抽出プロトコル機構

間 博人[1] 戸辺義人[2] 徳田英幸[1]
[1] 慶應義塾大学大学院 政策・メディア研究科
[2] 東京電機大学 工学部 情報メディア学科

近年の計算機技術の進展による計算機の小型・高速化と無線ネットワーク技術の進展により，我々の身の回りにある情報機器は飛躍的に増加している．こうした情報機器の種類・目的・移動性は様々であり，一元的な可用性の向上を目的としたグループ抽出が必要になる．

本研究は，まず amicus という抽出するグループの概念を定義する．具体的な amicus 適用の対象にモバイルアドホックネットワーク環境を選択し，SoC amicus を定義する．また SoC amicus を抽出するための機構として SoC amicus プロトコル機構を提案する．さらにモバイルアドホックネットワーク上で動作する P2P ファイル共有サービスを想定し，SoC amicus プロトコル機構を利用した SoC amicus 複製アルゴリズムを提案する．定義した 4 種類の移動モデル上で，複数の複製アルゴリズムをシミュレーションで評価し，SoC amicus 複製アルゴリズムを用いることで 検索成功率が向上することを示した．

A Group Abstraction for Mobile Group Communications

Hiroto Aida[1] Yoshito Tobe[2] Hideyuki Tokuda[1]
[1] Graduate School of Media and Governance, Keio University
[2] Department of Information Systems and Multimedia Design, Tokyo Denki University

This paper proposes a new notion of grouping, "amicus" for wireless mobile nodes and defines two protocols to detect an amicus. For applications over mobile ad hoc networks, an amicus-based system provides a hook for enhancing a system-wide performance: reduction of the number of duplicate database, simplification of routing, and coordination of packet scheduling among nodes.

First, we define the amicus in a formal way. An amicus is a collection of nodes that are considered as one group that has the same movement. Thus the nodes in an amicus do not change their relative positions among them.

Then, we describe two schemes by classifying the protocols to detect amicuses depending on whether or not a measurement is used. Finally, we show simulation results to evaluate the stability and agility of the protocols.

1 はじめに

移動計算機環境において，ノードの移動に伴うネットワークの切断はネットワークアプリケーションに悪影響をもたらす．特に，移動可能な無線ノード同士で動的なネットワークを構築するモバイルアドホックネットワーク (MANET) では，この影響は顕著である．MANET 環境および移動計算機環境では，リンク切断の検知による影響の低減は，重要な課題である．

無線リンクの切断は，"ノードの移動" が直接の原因ではない．図 1 は，ノード A とノード B が無線により接続されている様子を表す．ノード A が A' へ ノード B が B' へ移動した場合，ノードが移動したにもかかわらず無線リンクは切断しない．対象ノードとの接続性の強度が変化することで無線リンクは切断する．すなわち対象ノードとの接続接続性の強度の変動からグループを抽出する事で，切断しにくいノードの集合を得る事ができる．

本稿では，MANET 環境において「切断しにくいノードのグループ」を抽出し，その有用性を検証することを目的とする．まず，一般的なグループ抽出の概念として， amicus を示す．次に具体的な amicus 適用の対象に MANET 環境を選択し，SoC amicus を定義する．また SoC amicus を抽出するための機構として SoC amicus プロトコル機構を提案する．さらに SoC amicus プロトコル機構の評価として SoC amicus を複製アルゴリズムに適用し有用性を検証する．

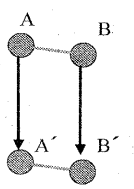

図 1: ノードの移動と無線リンクの切断

2 グループ抽出

グループ抽出とは，抽出条件をもとに母集団からグループを抽出することである．まず，グループ抽出が対象とするグループを分類し，グループ抽出のモデルとして amicus を定義する．さらに amicus を MANET 環境に適用した SoC amicus を示す．

2.1 抽出するグループの分類

グループ抽出が対象とするグループは，Physical Group，Informatics Group，Semantic Group の 3 つに分類することができる．

- Physical Group
  Physical Group とは物理空間上の位置から抽出するグループである．動物や昆虫の群といったものが Physical Group である．
- Informatics Group
  Informatics Group とは，論理空間上の位置から抽出するグループである．ネットワークのトポロジから抽出したグループは Informatics Group である．この他に，ネットワークの遅延を基にグループを構成した場合も同様に Informatics Group であるといえる．
- Semantic Group
  Semantic Group とは，意味空間上の位置から抽出するグループである．友人，家族，会社 といったそれぞれのノードが持つ意味情報をもとに抽出したグループが Semantic Group である．

2.2 Amicus の導入

上述したグループ抽出の概念として amicus を導入する．Amicus はクラスタリングなどのグループ化の概念と違い，時間経過に伴う変動の予測を含む．N_i はノード識別子 i をもつノードを表す．S_{ij} を N_i と N_j 間の強度とする．強度 S とはノード間の関連性を抽象化した指標である．次に 2 台のノードから構成された最小の amicus を定義する．

定義： 現在時刻 t から，T_{th} 時間後に $(S_{ij} \geq S_{th})$ が真であるとき，N_i と N_j は amicus を構成する．S_{th} は，amicus として十分な強度を持つ閾値である．

図 2 に S_{ij} と時間の関係を示す．図 2 の実線は測定された S_{ij} を示し，点線は予測される S_{ij} を示す．この際，現在時刻から T_m の間に測定された S_{ij} を用い $t+T_{th}$ 時の S_{ij} を予測する．

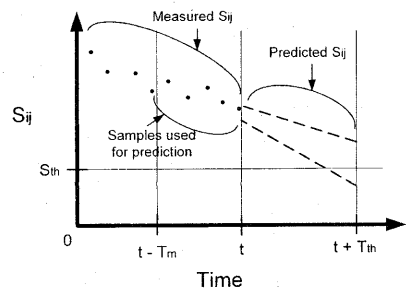

図 2: S_{ij} と時間の関係

$t+T_{th}$ 時における S_{ij} が S_{th} よりも大きい場合，N_i と N_j は amicus である．$N_i \sim N_j$ を N_i と N_j が amicus を構成しているとする．$\sim$ は推移する．すなわち，$N_i \sim N_j$ かつ $N_j \sim N_k$ であるとき，$N_i \sim N_k$ である．従って，N_i の amicus A は以下のように定義できる．

$$A_{N_i} = \{N_j, N_k \mid N_i \sim N_j, N_i \sim N_k\}$$

2.3 モバイルアドホックネットワークへの適用

対象ノードとの接続接続性の強度をもとに amicus を構築する事で，MANET 環境における切断しにくいノードの集合を得る事ができる．本研究では，接続性の強度として SoC (Strength of Connection) を用いる．SoC とは，変動する SNR をソフトウェアで制御可能な時間粒度で平均化した値である．SoC をノード間の強度とする amicus を SoC amicus とする．

3 SoC amicus プロトコル機構

MANET 上で SoC amicus を抽出する方法として，SoC amicus プロトコル機構を提案する．SoC amicus プロトコル機構は，SoC amicus 抽出プロトコル，SoC amicus 識別，SoC amicus テーブルの 3 つのコンポーネントから構成される．SoC amicus プロトコル機構の構成を図 3 に示す．

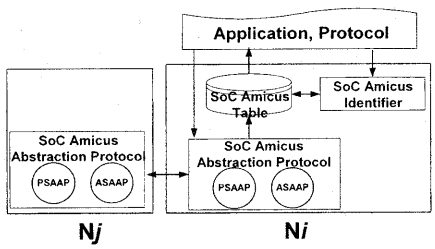

図 3: SoC amicus プロトコル機構

- SoC amicus 抽出プロトコル (SoC amicus abstraction protocol)
  SoC amicus 抽出プロトコルは，N_i，N_j 間の SoC 値を取得するためのプロトコルである．この際，2 種類の取得の方法が考えられる．第 1 に各ノードが持つ情報を隣接ノードに伝搬させ，受動的に amicus 情報を取得する push 型のプロトコルである．第 2 に隣接ノードが持つ情報を能動的に取得する pull 型のプロトコルである．この 2 つには，即応性とオーバヘッドのトレードオフがある．
- SoC amicus テーブル (SoC amicus table)
  SoC amicus テーブルは，SoC amicus 抽出プロトコルが取得した S_{ij} を記録するテーブルである．また，対象ノードが SoC amicus であるか否かの情報を記録する．
- SoC amicus 識別 (SoC amicus Identifier)
  SoC amicus 識別は，SoC amicus テーブルに記録されている S_{ij} から対象ノードが amicus か否かを判別するモジュールである．SoC amicus プロトコル機構を利用するアプリケーションやプロトコルは，SoC amicus 識別に，S_{th}，T_m，T_{th} を与える必要がある．
  SoC amicus 抽出プロトコルにより，SoC amicus テーブルは，対象ノードとの SoC 値および測定した時間を保持する．保持している情報のうち T_m 時

表 1: PSAAP 動作で用いる記号

pm	PSAAP メッセージ
pt	PSA テーブル
ptentry	PSA テーブルエントリ
id	ノード識別子
oid	自ノードの識別子
pmt	PSAAP メッセージタイマー
et	PSA テーブルエントリタイマー

間前までの測定値を用いる．i 番目の測定時間を t_i，測定 SoC 値を soc_i とする．n 個の測定値を，以下のように表す．

$$(t_1, soc_1)(t_2, soc_2) \ldots (t_n, soc_n)$$

この測定値から $(t + T_{th}, psoc)$ の $psoc$ を予測する．$psoc$ は，$t + T_{th}$ における予測 SoC 値を表す．$psoc$ を予測する手法は幾つかあるが，現状の実装では線形最小二乗法を用いる．

次に，受動 および 能動 SoC amicus 抽出プロトコルの動作詳細を説明する．

3.1 受動 SoC amicus 抽出プロトコル

受動 SoC amicus 抽出プロトコル (PSAAP: Passive SoC amicus abstraction protocol) は amicus 抽出のために明示的な応答を必要としない．まず PSAAP の動作手順を説明し，次に各々の動作について詳述する．

PSAAP の動作

PSAAP の動作手順は 3 つに分割できる．PSAAP 動作の疑似コードを以下に示す．また，この疑似コードで使用する記号を表 1 に示す．

```
PSAAP()
  if (timeout pmt)
    then initialize pmt
      pm ← createMessage(pt)
      send pm

  if (receiving pm)
    then if (id involved in pm is equal to oid)
           then drop pm
           else updateMessage(pm)

  if (timeout et in each ptentry)
    then delete ptentry
```

PSA テーブル (Passive SoC amicus table) は，第 2 章で述べた amicus プロトコル機構を構成するモジュールの一つである，amicus テーブルを PSAAP に適応した物である．PSA テーブルは，複数の PSA テーブルエントリ (Passive SoC amicus table entry) から構成される．ノード識別子は，IP アドレスなど各ノードがもつ一意な識別子である．Amicus プロトコル機構は，ノード識別子の種類に依存しない．

次に PSAAP メッセージフォーマットおよび各々の処理を説明する．

PSAAP メッセージフォーマット

pm は，pmt が切れた時に生成され，ブロードキャストで送信される．図 4 は PSAAP メッセージのフォーマットである．図 4 における hop_j は PSAAP メッセージの送信ノードから対象ノード N_j までのホップ数を表す．id と hop を組み合わせを tupple とする．PSAAP メッセージは，複数の tupple から成る．len は tupple の数を表す．

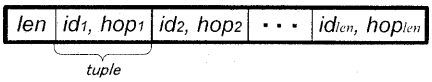

図 4: PSAAP メッセージ

表 2: PSAAP メッセージ生成アルゴリズムに用いる記号

amicusflag	amicus フラグ: 対象ノードが SoC amicus か否かを示すフラグ
lhop	ホップ制限数
tid	PSA テーブルエントリにおける対象ノードの識別子

PSAAP メッセージの生成

PSAAP メッセージを生成する際のアルゴリズムを以下に示す．またこのアルゴリズムで使用する記号を表 2 に示す．

```
createMessage(pt)
  add tuple[oid, 0] to pm
  foreach ptentry in pt
    if (ptentry.amicusflag = true and ptentry.hop ≤
lhop)
      then add tuple[ptentry.tid, ptentry.hop] to pm
  return pm
```

まず，PSAAP メッセージ pm に，送信ノードの識別子 oid とそのホップ数である 0 の tupple を加える．次にノードが持つ PSA テーブル pt のエントリを取得する．

図 5 に PSA エントリの構成を示す．PSA エントリは Index Field と Record Field から成る．Index Field は，*amicusflag*, *hop*, *tid*, *pid* から成る．*amicusflag* は対象ノードが amicus か否かを表し，自ノードの amicus であれば true，amicus でなければ false となる．*hop* は，自ノードから対象ノードへのホップ数を表す．*tid* は，このエントリが対象とするノードの識別子である．*pid* は，自ノードが受け取った PSAAP メッセージの送信ノード識別子である．これは，PSAAP メッセージの重複を防ぐために必要となる．Record Field は，PSAAP メッセージの受信時間である *rtime* と自ノードと対象ノードの SoC 値である *soc* から構成される．

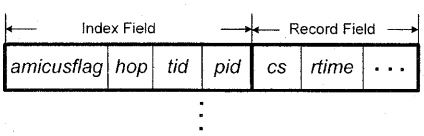

図 5: PSA テーブルエントリ

取得した PSA テーブルエントリ $ptentry$ の $amicusflag$ が true かつ hop がホップ制限の定数 $lhop$ よりも小さい場合 tid と hop の tupple を PSAAP メッセージへ加える．

PSAAP メッセージのの受信と PSA テーブルの更新

PSAAP の第 2 の処理として PSAAP メッセージのの受信および PSA テーブルの更新を説明する．PSAAP

表 3: PSA テーブル更新アルゴリズムに用いる記号

rtime	PSAAP メッセージの受信時間
soc	測定された対象ノードとの SoC 値

メッセージを受信し，PSA テーブルを更新する際のアルゴリズムを以下に示す．このアルゴリズムで未定義の記号を表 3 に示す．

```
updateMessage(pm)
  foreach tuple[tid, hop] in pm
    if tid and pid is already in pt
      then add record field [rtime, soc] to ptentry
      else add ptentry[tid, pid, false, hop+1, rtime, soc]
to pt
```

まず重複して受信することを避けるため，PSAAP メッセージの *id* が自ノードの識別子と等しい場合 PSAAP メッセージは破棄する．PSAAP メッセージに含まれる対象ノードの識別子とホップ数の tupple が PSA テーブルにあるか無いかを調べる．もし PSAAP メッセージの対象ノードの識別子と受信ノードの識別子が PSA テーブルエントリにあれば受信時間および測定した SoC 値を PSA テーブルエントリの Record Field に加える．無い場合は，新しい PSA テーブルエントリを作成し，PSA テーブルに追加する．

PSA テーブルエントリのタイムアウト

PSAAP の第 3 の処理は，PSA テーブルエントリのタイムアウトである．PSA テーブルエントリに対応する PSAAP メッセージが一定時間届かない場合，PSA テーブルエントリは破棄される．

3.2 能動 SoC amicus 抽出プロトコル

能動 SoC amicus 抽出プロトコル (ASAAP: Active SoC amicus abstraction protocol) は，amicus 抽出のために明示的な応答を必要とする．PSAAP では，全てのノードが定期的にメッセージを送信する必要があった．これに対して，ASAAP は SoC amicus 情報が必要なノードだけが能動的にメッセージを送信する．この SoC amicus 情報が必要なノードをアクティブノードとする．またアクティブノードか否かを判別するために各ノードはアクティブフラグを持つ．アクティブフラグが true のノードはアクティブノードであることを表す．この アクティブフラグは，ASAAP を利用するアプリケーションやプロトコルが設定する．まず ASAAP の動作手順を説明し，次に各々の動作について詳述する．

ASAAP の動作

図 6 に ASAAP の状態遷移図を示す．ASAAP の動作は 4 つの処理に分割できる．ASAAP 動作の疑似コードを以下に示す．また，この疑似コードで使用する記号を表 4 に示す．

```
AADP()
  if activeflag = true and timeout areqtimer
    then initialize areqtimer
      areqm ← createREQ(NULL)
      send areqm

  if receiving areqm
    then if id involved in areqm is equal to oid
      then drop areqm
      else areqtentry ← updateREQ(areqm)
        if areqtentry.amicusflag = true
          then arepm ← createREP(arept, areqm.hop)
          send arepm
        if areqm.hop > 1
          then areqm ← createREQ(areqm)
            send areqm

  if receiving arepm
    then updateREP(arepm)

  if timeout et in each areqtentry and areptentry
    then delete areqtentry and areptentry
```

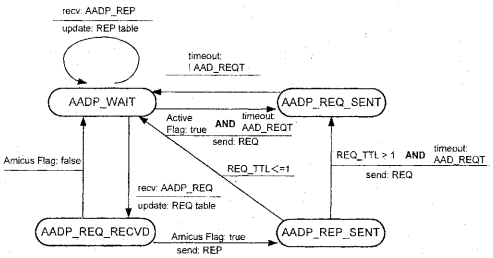

図 6: ASAAP の状態遷移図

表 4: ASAAP 動作で用いる記号

areqm	ASAAP 要求メッセージ
arepm	ASAAP 応答メッセージ
areqt	ASA 要求テーブル
arept	ASA 応答テーブル
areqtentry	ASA 要求テーブルエントリ
areptentry	ASA 応答テーブルエントリ
activeflag	ノードのアクティブフラグ
areqtimer	ASAAP 要求メッセージタイマー

ASAAP は，ASAAP 要求メッセージと ASAAP 応答メッセージの 2 タイプのメッセージを利用する．ASAAP において amicus テーブルは，ASA 要求テーブル (Active SoC amicus reqest table) と ASA 応答テーブル (Active SoC amicus reply table) の 2 つに分割して考える．

次に ASAAP のメッセージフォーマットおよび各々の処理を説明する．

アクティブノードにおける ASAAP 要求メッセージの送信

アクティブフラグが true かつ，ASAAP 要求メッセージタイマーがタイムアウトした場合，図 6 における状態が A_WAIT から A_REQ_SENT へ移行する．A_REQ_SENT では，ASAAP 要求メッセージタイマーを初期化し ASAAP 要求メッセージを生成する．生成した ASAAP 要求メッセージはブロードキャストで送信する．ASAAP 要求メッセージのフォーマットを図 7 に示す．PSAAP メッセージを生成する際のアルゴリズムを以下に示す．またこのアルゴリズムおよびで ASAAP 要求メッセージのフォーマットで新たに使用する記号を表 5 に示す．

```
createREQ(areqm)
  if areqm.hop > 1
    then areqm.hop ← areqm.hop − 1
    else areqm.hop ← reqhop − 1
  add oid to areqm
  areqm.len ← areqm.len + 1
  return areqm
```

表 5: ASAAP メッセージフォーマットおよび生成アルゴリズムに用いる記号

reqhop	SoC amicus を抽出するホップ数
mtype	メッセージタイプ

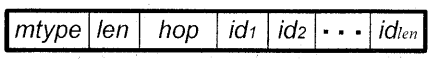

図 7: ASAAP 要求メッセージ

この時の *createREQ* の引数は初期化された ASAAP 要求メッセージである．したがって，ASAAP 要求メッセージの *hop* は定数 *reqhop* になる．*reqhop* は SoC amicus を抽出する範囲をあらわわし，この値は ASAAP を利用するアプリケーション及びプロトコルによって決定される．ASAAP 要求メッセージ の末尾に自ノードの識別子を加えブロードキャスト送信する．

ASAAP 要求メッセージの受信

第 2 に ASAAP 要求メッセージを受信した時の処理を説明する．ASAAP 要求メッセージを受信すると図 6 において A_WAIT から A_REQ_RECVD に状態が移行する．受信した ASAAP 要求メッセージに含まれる *id* が自ノードの識別子と等しい場合，メッセージは破棄する．そうでなければ，ASA 要求テーブルを更新する．更新する ASA 要求テーブルエントリを図 8 に示す．ASA 要求テーブル更新時のアルゴリズムを以下に示す．

```
updateREQ(areqm)
  if areqm.id_len is already in areqt
    then add record field [rtime, soc] to areqentry
    else add areqentry[areqm.id_len, false, rtime, soc] to
areqt
  return areqentry
```

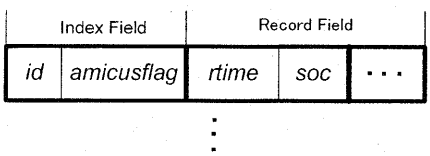

図 8: ASA 要求テーブルエントリ

もし $areqm.id_{len}$ すなわち ASAAP 要求メッセージの送信元ノードの識別子と ASA 要求テーブルエントリの *id* を比較する．もし，一致する ASA 要求テーブルエントリがあれば，対応する ASA 要求テーブルエントリの Record Field に受信時間及び測定された SoC 値を追加する．一致する ASA 要求テーブルエントリがなければ，新しい ASA 要求テーブルエントリを作成し，ASA 要求テーブルに追加する．

ASA 応答メッセージの送信

ASA 要求テーブルの更新後，更新した ASA 要求テーブルエントリの amicus フラグを調べる．もし，amicus フラグが true であれば，状態は A_REP_SENT へ移行する (図 6)．amicus フラグが false であれば，A_WAIT へ移行する．A_REP_SENT では，ASAAP 応答メッセージを作成し ASAAP 要求メッセージの送信元ノードへ送信する．ASAAP 応答メッセージフォーマットを図 9 に示す．ASAAP 応答メッセージ形式は PSAAP メッセージ形式と同様に，ノード識別子とホップ数から成る複数の tupple と tupple の数を表す *len* から構成される．ASAAP 応答メッセージ形式は，ASAAP 要求メッセージとの識別のためフィールドとして *mtype* を持つ．

以下に ASAAP 応答メッセージの作成アルゴリズムを示す．

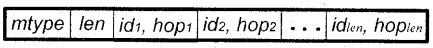

図 9: ASAAP 応答メッセージ

```
createREP(arept, areqm)
  add tuple[oid, 0] to arepm
  foreach areptentry in arept
    if (areptentry.hop < areqm.hop)
      then add tuple[areptentry.tid, areptentry.hop]
  return arepm
```

ASAAP 応答メッセージは，ASA 応答テーブルを参照して作成する．ASA 応答テーブルエントリの形式を図 10 に示す．ASA 応答テーブルエントリは，*tid*, *pid*, *hop*, *utime* の 4 フィールドから構成される．*tid* は，このエントリが対象とするノードの識別子である．*hop* は，自ノードと対象ノード間のホップ数を表す．*pid* および *utime* は，ASAAP 応答メッセージの生成時には使用しない．この 2 フィールドに関しては，3.2 節で解説する．

まず ASAAP 応答メッセージに，自ノード識別子と自ノードまでのホップ数 (即ち 0) から成る tupple を加える．次に ASA 応答テーブルを参照し，各 ASA 応答テーブルエントリの *hop* と対応する ASAAP 要求メッセージの *hop* を比較する．ASA 応答テーブルエントリの *hop* が小さい場合，そのエントリの *tid* と *hop* から成る tupple を ASAAP 応答メッセージに追加する．こうして生成した ASAAP 応答メッセージは，対応する ASAAP 要求メッセージの送信元ノードへユニキャストで送信する．

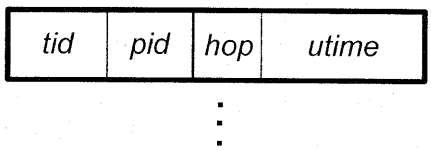

図 10: ASA 応答テーブルエントリ

ASAAP 応答メッセージの再転送

ASAAP 応答メッセージの送信後，受信した ASAAP 要求メッセージの *hop* が 1 未満であれば，A_REP_SENT から A_WAIT へ移行する (図 6)．反対に 1 以上であれば，ASAAP 要求メッセージタイマーのタイムアウトともに A_REP_SENT

表 6: ASA 応答テーブル更新アルゴリズムで使用する記号

tid	対象ノード識別子
pid	受信した ASAAP 応答メッセージの送信元ノード識別子
utime	更新時間

から A_REQ_SENT へ移行する．A_REQ_SENT では，3.2 節と同様に ASAAP 要求メッセージを作成し再度ブロードキャストする．この際，*createREQ* の引数として受信した ASAAP 要求メッセージが要求される．受信した元の ASAAP 要求メッセージに自ノード識別子を追加し，*hop* をデクリメントする．

ASAAP 応答メッセージの受信

ASAAP 応答メッセージを受信した際の処理を説明する．ASAAP 応答メッセージを受信すると，ASA 応答テーブルを更新する．以下に，ASA 応答テーブル更新時のアルゴリズムを示す．また下記アルゴリズムで使用する記号を表 6 に示す．

```
updateREP(arepm)
  foreach tuple[tid, hop] in arepm
    if tid and pid is already in arept
      then update utime to areptentry
      else add areptentry[tid, pid, arepm.hop + 1, utime]
to arept
```

受信した ASAAP 応答メッセージに含まれる *tid* と *hop* の tupple 毎に確認する．各 tupple の対象ノード識別子と ASAAP 応答メッセージの送信元ノード識別子と同じ ASA 応答テーブルエントリがあれば，ASA 応答テーブルエントリの *utime* を更新する．同じエントリが無ければ，新しい ASA 応答テーブルエントリを作成し ASA 応答テーブルに追加する．

テーブルエントリのタイムアウト

ASA 要求テーブルエントリおよび ASA 応答テーブルエントリは，タイマーにより管理されている．一定時間 更新がないエントリは，破棄される．

3.3 SoC amicus 識別

SoC amicus 抽出プロトコルにより，SoC amicus テーブルは，対象ノードとの SoC 値および測定した時間を保持する．保持している情報のうち T_m 時間前までの測定値を用いる．i 番目の測定時間を t_i，測定 SoC 値を soc_i とする．n 個の測定値を，以下のように表す．

$$(t_1, soc_1)(t_2, soc_2) \ldots (t_n, soc_n)$$

この測定値から $(t + T_{th}, psoc)$ の $psoc$ を予測する．$psoc$ は，$t + T_{th}$ における予測 SoC 値を表す．$psoc$ を予測する手法は幾つかあるが，SoC amicus 識別では線形および非線形最小二乗法を用いる．最小二乗法で適用するモデル関数は，SoC amicus 抽出精度に影響する．

4 SoC amicus 複製アルゴリズム

MANET のデータ利用性向上を目的として，MANET 上の複製配置方式が提案されている [3]．この際複製は，切断しやすいノードに優先して配置すべきである．すなわち切断しないノード間で複製を配置しても，その複製が無駄になってしまう可能性が高い．無駄な複製が多いと，各ノードのキャッシュ可能な容量には限界があるため，データ利用性は低下する．

切断しにくいノードのグループである SoC amicus がわかると，無駄な複製を削除する事ができる．無駄な複製を減らす事で，データ利用性は向上する．次に，定周期複製アルゴリズムと 2 つの SoC amicus 複製アルゴリズムを提示する．

Periodic Replication Algorithm (PR)

各ノードは，一定間隔でランダムに選択した複製配置範囲のノードに データの複製を配置する．配置先のノードのキャッシュ容量が一杯の場合，FIFO 方式で複製データを削除する．

PSAAP SoC Amicus Replication Algorithm (PSAR)

各ノードは，PSAAP を用いて SoC amicus を抽出する．各ノードは，一定間隔でランダムに選択した複製配置範囲のノードに データの複製を配置する．配置先のノードの，キャッシュ容量が一杯の場合，配置されている複製データが，抽出した SoC amicus にあるか調べる．SoC amicus にあるデータを優先し削除する複製データを決定する．

ASAAP SoC Amicus Replication Algorithm (ASAR)

各ノードは，ASAAP を用いて SoC amicus を抽出する．各ノードは，一定間隔でランダムに選択した複製配置範囲のノードに データの複製を配置する．配置先のノードの，キャッシュ容量が一杯の場合，配置されている複製データが，抽出した SoC amicus にあるか調べる．SoC amicus にあるデータを優先し削除する複製データを決定する．

5 評価

SoC amicus 複製アルゴリズムを検索ヒット率の観点から評価を行う．まず，評価に使用する移動モデルおよび評価環境について言及する．検索ヒット率の比較と SoC amicus 抽出プロトコルのメッセージ数の比較を評価結果として示す．

5.1 移動モデル

SoC amicus は，移動ノード間で強い接続性を持つノードの集合である．移動モデルにより，集合の生成形態が大きく変わるため，ノードの移動モデルは SoC amicus を評価する上で重要な要素である．

移動モデルは，MANET や人工生命，社会科学などの既存研究により様々な側面から多数のモデルが提案されている [1, 5]．既存の移動モデルは，互いのノードが完全に独立して移動する Entity Mobility Model (EM) とノード間でグループを形成し移動する Group Mobility Model に分類できる．本稿ではさらに Group Mobility Model を，Static Group Mobility Model (SGM) と Dynamic Group Mobility Model (DGM) に分類する．SGM は，移動中に形成するグループのメンバーが固定の移動モデルである．これに対し DGM は，移動中に形成するグループのメンバーが変動する移動モデルである．

本研究では，既存の移動モデルを 3 つに分類し，各々を評価の指標として用いる．以下に各々の移動モデルと実際に評価に利用するアルゴリズムの概要を説明する．

Entity Mobility Model (EM)

EM は，互いのノードが完全に独立して移動するタイプの移動モデルである．EM に属する移動モデルには，Random Walk Mobility Model, Random Waypoint Mobility Model, Random Direction Mobility Model, Causs-Markov Mobility Model, A Probabilistic Varsion of the Random Walk Mobility Model, City Section Mobility Model などがある．

この中で，最も広範に評価に採用されている Random Waypoint Mobility Model [6] を EM の評価に用いる．Random Waypoint Mobility Model は，ランダムに目的地と速度を設定し，目的地へで移動する．目的地に到達すると一定時間留まり，新しい目的地と速度を再度設定し移動する．

Static Group Mobility Model (SGM)

SGM は，移動中に形成するグループのメンバーが固定の移動モデルである．SGM に属する移動モデルには，Exponential Correlated Random Mobility Model, Column Mobility Model, Nomadic Community Mobility Model, Pursue Mobility Model, Reference Point Group Mobility Model (RPGM) などがある．

SGM での評価には，RPGM [4] を用いる．RPGM はグループの移動を規定するグループ移動ベクトル $\overrightarrow{GM}$ と，各ノードのランダム要素を規定するランダム移動ベクトル $\overrightarrow{RM}$ によりノードを移動を決定する移動モデルである．

Dynamic Group Mobility Model (DGM)

DGM は，移動中に形成するグループのメンバーが変動する移動モデルである．DGM に属する移動モデルには Ant Colony Optimization (ACO) Motion Model [2], Boid Motion Model [7] などが属する．SGM での評価には，Boid Motion Model を用いる．Boid Motion Model は，鳥の群知能のシミュレーションとして考案された移動モデルである．前提として，全ノードが最適距離と視野を持つ．Separation, Alignment, Cohesion の 3 つの基本ルールを元に移動先の座標を決定する．Separation：もっとも近くにいるノードが最適距離の内側にいるとき，そのノードの反対方向へ向かおうとする力が働く．Alignment：視野の中にいる全てのノードと同じ方向を向くような力が働く．Cohesion：視野の中にいる全てのノードの中心（重心）に向かおうとする力が働く．

5.2 シミュレーションモデル

SoC amicus 複製アルゴリズムの性能評価をシミュレーションにより行った．シミュレーションエリアおよびシミュレーション時間に関するパラメータを表 7 のように定めた．フィールドに配置された各ノードは，前節で述べた EM, SGM, DGM で移動する．移動モデルに関するパラメータを表 8 にまとめる．

前提として，各ノード N_i は，それぞれ固定サイズのデータ D_i を持ち，それとは別に固定のキャッシュ容量を持つ．検索するデータは，シミュレーションフィールドに存在するデータのみとする．各ノードは，PR, PSAR, ASAR の複製アルゴリズムを利用し複製の配置を行う．同時に，各ノードは定期的にランダムに選

表 7: 測定パラメータ設定

フィールドサイズ	500 m× 500 m
無線通信範囲	50 メートル
ノード数	50 台
シミュレーション時間	600 sec
測定回数	各条件において 10 回

表 8: 移動モデルパラメータ設定

最大速度	10 m/sec
待機時間	0 sec
グループ数	3
最適距離	40 m
Alignment 率	0.3
Cohesion 率	0.1

択したデータ D_j に対する検索クエリーを検索範囲内にフラッディングする．検索クエリーを受信したノードが D_j のオリジナルデータまたは複製データを持つ場合，検索は成功となる．複製および検索に用いたパラメータを表 9 に示す．また PSAR と ASAR で用いた SoC アミークス抽出プロトコルのパラメータを表 10 に示す．

5.3 検索ヒット率の比較

複製アルゴリズムの性能評価のため，移動モデル毎に検索ヒット率を測定した．図 11 は，DGM model での結果のスナップショットである．横軸は時間を表し，縦軸は累積検索ヒット率を表す．測定開始から 50 秒程度まで急激に検索ヒット率が上昇している．

これは，複製データが次第に配置され検索効率が上昇したためだと考えられる．また，この間の複製アルゴリズムによる差異はあまりないが，これは SoC amicus 複製アルゴリズムは，ノードのキャッシュ容量が一杯になった時に意味があるので，効果が出るまでしばらく時間がかかることが原因だと考えられる．100 秒位から検索ヒット率が安定し，SoC アミークス複製アルゴリズムは PR よりも高いヒット率を達成している．

図 12 は，最終的に達成した検索ヒット率を移動モデル毎にまとめたものである．結果として 3 つの移動モデル全てにおいて SoC アミークス複製アルゴリズムを適用した検索ヒット率が，Simple replication アルゴリズムよりもすぐれていた．このことから SoC アミークス複製アルゴリズムは，動的なグループの生成と消滅に追随することができたといえる．

表 9: 複製および検索パラメータ設定

キャッシュ容量	$5 \times sd$
複製配置間隔	3sec
複製配置範囲	1 hop
検索間隔	3sec
最大検索範囲	10 hop

表 10: SoC amicus 抽出プロトコル パラメータ設定

抽出範囲	1 hop
PSAAP メッセージ送信間隔	0.5 sec
ASAAP 要求メッセージ送信間隔	0.5 sec
テーブルエントリタイムアウト間隔	1 sec

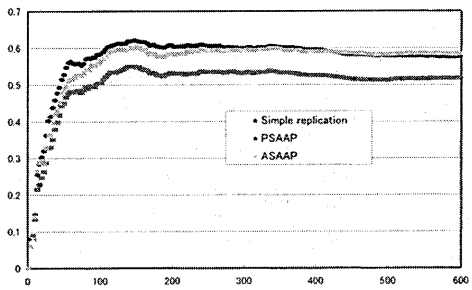

図 11: Dynamic group mobility model における累積検索ヒット率の推移

5.4 SoC amicus 抽出プロトコルのメッセージ数

図 13 は，EM model 上で送信された SoC amicus 抽出プロトコルのメッセージ数である．PSAAP のメッセージ数は一定であるのに対し，ASAAP のメッセージ数はアクティブノードの数により変動する．そこで全ノード中のアクティブノードの比率を 0.02，0.2，0.4，0.6，0.8 と変更し測定した．

アクティブノードの比率が低い場合は，ASAAP の負荷は PSAAP よりも低い．アクティブノードの比率が 0.6 以上になると PSAAP より高負荷となり，全ノードがアクティブノードの場合，ASAAP は PSAAP より 2 倍以上の負荷になる．

この結果から，PSAAP と ASAAP は使用目的により使い分けるべきだと言える．例えば，今回のような複製アルゴリズムへの適用の場合システム全体の負荷の低減が目的なので，PSAAP が適している．しかしながら，ノード単体の負荷低減を目的として SoC amicus を利用する場合 ASAAP の使用が適していると言える．

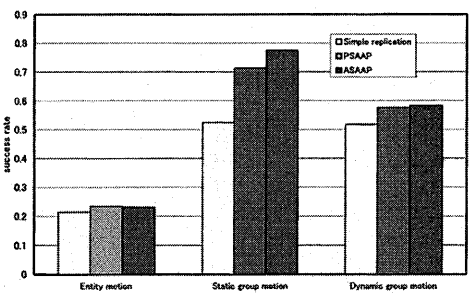

図 12: 検索ヒット率の比較

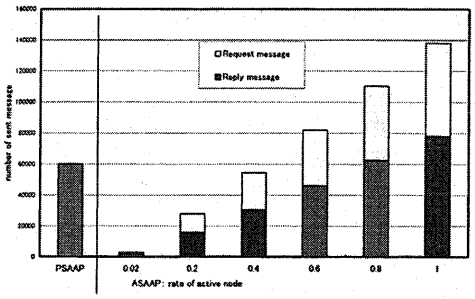

図 13: SoC amicus 抽出プロトコルの送信メッセージ数

6 まとめと今後の課題

本稿では，グループ抽出の概念として amicus を定義し，MANET 上の複製アルゴリズム適用した SoC amicus 複製アルゴリズムを提案した．SoC amicus 複製アルゴリズムを利用する事で，グループメンバが固定の移動モデルにおいて 30% 程度検索ヒット率が向上することがわかった．また，グループメンバが変動する移動モデルにおいて 10% 程度検索率が向上した．このことから，SoC amicus プロトコル機構を用いる事でグループの動的な変動に対応し，接続しにくいノードの集合を発見できたと言える．

現状では，パラメータを固定し検索効率を評価した．今後評価パラメータを変更し，パラメータの検証を行う必要がある．また，今後 Bluetooth を用いた実機による検証も行う予定である．

参考文献

[1] Camp, T., Boleng, J. and Davies, V.: A Survey of Mobility Models for Ad Hoc Network Research, in *Proceedings of Wireless Communications and Mobile Computing (WCMC'02)*, (2002).

[2] Dorigo, M. and Caro, G.: Ant Algorhtms for Discrete Optimization, in *Proceedings of Artificial Life*, Vol.5, No.3, pp. 137-172, (1999).

[3] Hara, T.: Replicating Data with Aperiodic Update in Ad Hoc Networks, in *Proc. of IASTED Int'l Conf. on Communications, Internet and Information Technology (CIIT2002)*, (2002).

[4] Hong, X., Gerla, M., Pei, G. and Chiang, C.: A Group Mobility Model for Ad Hoc Wireless Networks, in *Proceedings of ACM MSWiM'99*, pages 53-60, (1999).

[5] Hu, Y. and Johnson, D.: Caching strategies in on-demand routing protocols for wireless ad hoc networks, in *Proceedings of ACM MOBICOM 2000*, (2000).

[6] Johnson, D. and Maltz, D.: Dynamic source routing in ad hoc wireless networks, in *Proceedings of ACM MOBICOM'96*, (1996).

[7] Reynolds, C.: Flocks, Herds, and Schools: A Distributed Behavioral Model, in *Computer Graphics*, (1987).